Summer 6-15-2022

Enhanced Data Sampling and Feature Generation for Machine Learning-based Lithography Hotspot Detection

Mohamed Ismail
The American University in Cairo AUC, mt.22@aucegypt.edu

Graduate Studies

Enhanced Data Sampling and Feature Generation for Machine Learning-based Lithography Hotspot Detection

A THESIS SUBMITTED BY

MOHAMED TAREK ISMAIL

TO THE

Electronics and Communications Engineering
Graduate Program

February 2, 2022

in partial fulfillment of the requirements for the degree of
Master of Science in Electronics and Communications Engineering

Declaration of Authorship

I, Mohamed Tarek Ismail, declare that this thesis titled, "Enhanced Data Sampling and Feature Generation for Machine Learning-based Lithography Hotspot Detection" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

Abstract

Technology scaling has increased the complexity of integrated circuit design. It has also led to more challenges in the field of Design for Manufacturing (DFM). One of these challenges is lithography hotspot detection. Hotspots (HS) are design patterns that negatively affect the output yield. Identifying these patterns early in the design phase is crucial for high yield fabrication. Machine Learning-based (ML) hotspot detection techniques are promising since they have shown superior results to other methods such as pattern matching. Training ML models is a challenging task due two main reasons. Firstly, industrial training designs contain millions of unique patterns. It is impractical to train models using this large number of patterns due to limited computational and memory resources. Secondly, the HS detection problem has an imbalanced nature; datasets typically have a limited number of HS and a large number of non-hotspots. This requires the use of data sampling techniques to choose the best representative dataset for training. In this thesis, we explore the problem of hotspot detection using machine learning. Specifically, we tackle the problem of data sampling where we introduce a method for dataset selection that enables the reduction of the training dataset size and improves the data balance between hotspot and non-hotspot patterns. In addition, we explore feature engineering using image gradients as a method of improving ML HS detection models.

Acknowledgements

Firstly, I would like to thank my supervisors, Dr. Karim Seddik and Dr. Hossam Sharara, for their continuous support and guidance.

Moreover, I would not have been here without the love and support of my family and friends. First, I would like to specially thank my father, mother, and sister who and were always supporting and encouraging me to do my best. Your advice, patience, and wisdom have pushed me during hard and stressful times. I am sincerely grateful for my wife who was always there to encourage and push me when I was down.

A big shout-out to all my friends who were always there for me: Mostafa Hussein, Ahmed Yehia, Youssef Afifi, Omar Gamal, Ali Atif, Mostafa Ahmed, Mohamed ElBadry, Mohamed Adly, and Hamza Wessam.

I would like also to thank my company, Siemens EDA, for supporting part of the research and to thank my managers, Dr. Sherif Hammouda and Dr. Kareem Madkour, for their guidance and support. Special thanks to Marwa Shafee for her precious thesis writing tips.

Before all, I thank Allah for giving me the inspiration, power, and persistence to do this work.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
BCE	**B**inary **C**ross **E**ntropy
CAD	**C**omputer **A**ided **D**esign
CNN	**C**onvolutional **N**eural **N**etwork
DBSCAN	**D**ensity-**b**ased **S**patial **C**lustering of **A**pplications with **N**oise
DCT	**D**iscrete **C**osine **T**ransform
DNN	**D**eep **N**eural **N**etwork
DRC	**D**esign **R**ule **C**hecking
HOG	**H**istogram **o**f **G**radients
HS	**H**ot**s**pot
HTC	**H**ard **T**o **C**lassify
IC	**I**ntegrated **C**ircuit
ML	**M**achine **L**earning
NHS	**N**on **H**ot**s**pot
PCA	**P**rincipal **C**omponent **A**nalysis
ReLU	**Re**ctified **L**inear **U**nit
SEM	**S**canning **E**lectron **M**icroscope
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
SURF	**S**peeded **U**p **R**obust **F**eatures
SVM	**S**upport **V**ector **M**achine

Chapter 1

Introduction

1.1 Background on Lithography Hotspots

Optical lithography is the method by which an integrated circuit (IC) is printed on silicon. Lithography steps are shown in Figure 1.1. First, a light-sensitive material called the photoresist is placed on the substrate. Then, the mask of the IC design is placed between the substrate and the light source. The substrate is exposed to light with a specific wavelength. This modifies the solubility of the photoresist in certain areas that correspond to the mask. After exposure, etching is carried out. Then, this substrate is immersed in a developer solution. Developer solutions are used to dissolve away areas of the photoresist exposed to light. These areas can then be deposited with metal to create the electrical connections [1].

Like any physical manufacturing process, optical lithography has many process variations that lead to imperfect output. For example, the width of a metal track is usually not equal to the width that was drawn by the designer. Due to process variations, the fabricated width may be close to zero leading to a pinch hotspot (HS) or an open circuit. Other process variations can lead to two metal tracks touching each other causing a short circuit or a bridging hotspot. These two types of hotspots are shown in Figure 1.2.

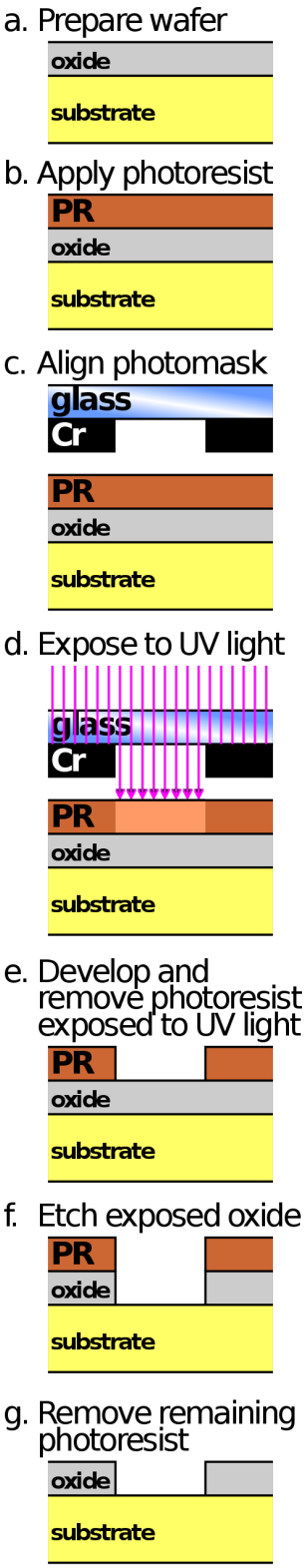


FIGURE 1.1: Optical Lithography Steps

Hotspots cause functional defects that negatively affect the yield of the fabrication process. Yield is defined as the ratio of the working chips to the total number of fabricated chips. In order to maximize yield, it is important to detect hotspots as early as possible, preferably in the design phase. This increases the confidence that the chip would be fabricated successfully.

Traditionally, design rule checking (DRC) was sufficient to ensure high yield fabrication. Design rule checks are geometrical checks that are defined by the manufacturer. The basic checks are shown in Figure 1.3. The first check is the minimum width check. Any metal that has a width smaller than the allowed minimum width is not allowed because it would be vulnerable to process variations. Hence, the designer should ensure that all metal widths are within the requirements defined by the manufacturer. A similar check exists for the spacing between metals and the distance between a contact and the metal layer which is known as the enclosure. A designer should make sure that their layout is DRC-clean before sending it to fabrication.

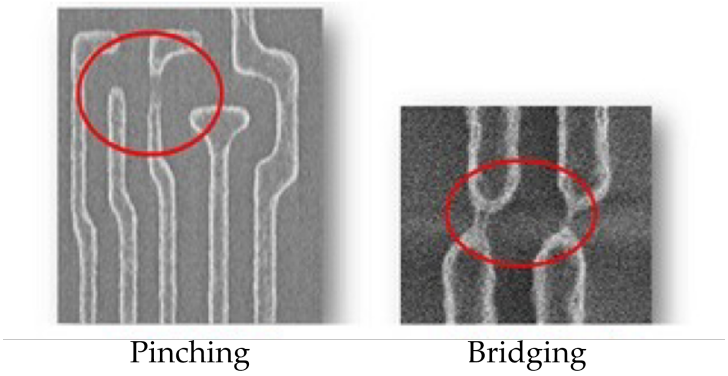


FIGURE 1.2: Scanning Electron Microscope (SEM) Images of Two Types of HS [2]

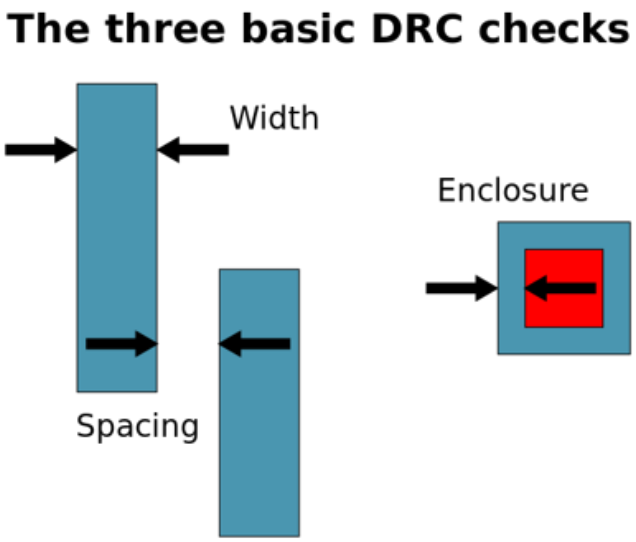


FIGURE 1.3: Basic DRC Checks

As feature sizes decreased due to Moore's Law [3], DRC checks on their own were not enough for high yield, therefore, lithography hotspot detection became a necessity.

1.2 Methods of Hotspot Detection

There are several methods for HS detection. The first method is based on lithography process models [4]. Using these models, simulation can run on a DRC-clean layout to identify hotspot regions. The output of simulation is a process variation (PV) band. This band represents variation in edge placement under different process conditions. From this band the minimum contour can be used to perform measurements and detect pinch hotspots. On the other hand, bridging hotspots can be detected from the maximum contour. Figure 1.4 shows an example of a bridging hotspot. The metal layer is shown in yellow and the PV band in red. It can be seen that the bridging hotspot occurs where the maximum contours touch each other causing a short circuit. Using this information, the designer can then modify the layout to fix the hotspot by moving the two edges highlighted in cyan further apart causing the short circuit to be eliminated. Lithography simulation is the most accurate method for HS detection since it makes use of calibrated process models, however, it is computationally expensive.

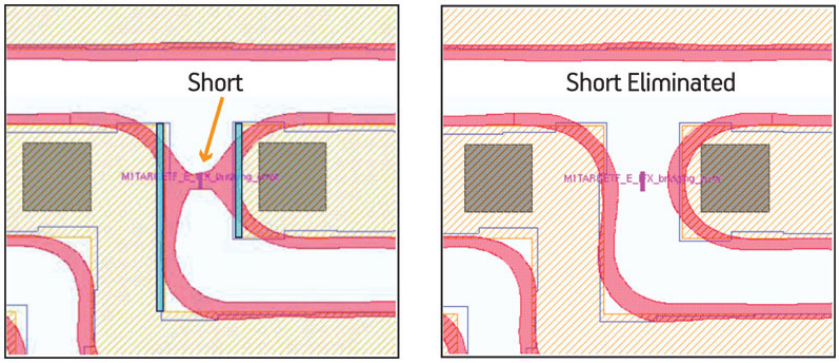


FIGURE 1.4: Example of a Bridging Hotspot and its Fix [5]

The second type is based on pattern matching libraries [6, 7]. This flow is shown in Figure 1.5. First, a pattern matching library of previously seen hotspot patterns is collected from multiple input layouts. This library is then used to run pattern matching on the test design to match the known patterns. Lithography simulation is then run on the matching locations to detect hotspots. The advantage of this method is that it does not require a large computational power like full lithography simulation. This is because the pattern matching step eliminates large areas which do not match the known HS library, so simulation only runs on a subset of the layout. The expensive simulation step is thus reduced. The disadvantage of this method is that it cannot detect new hotspots that are not part of the library.

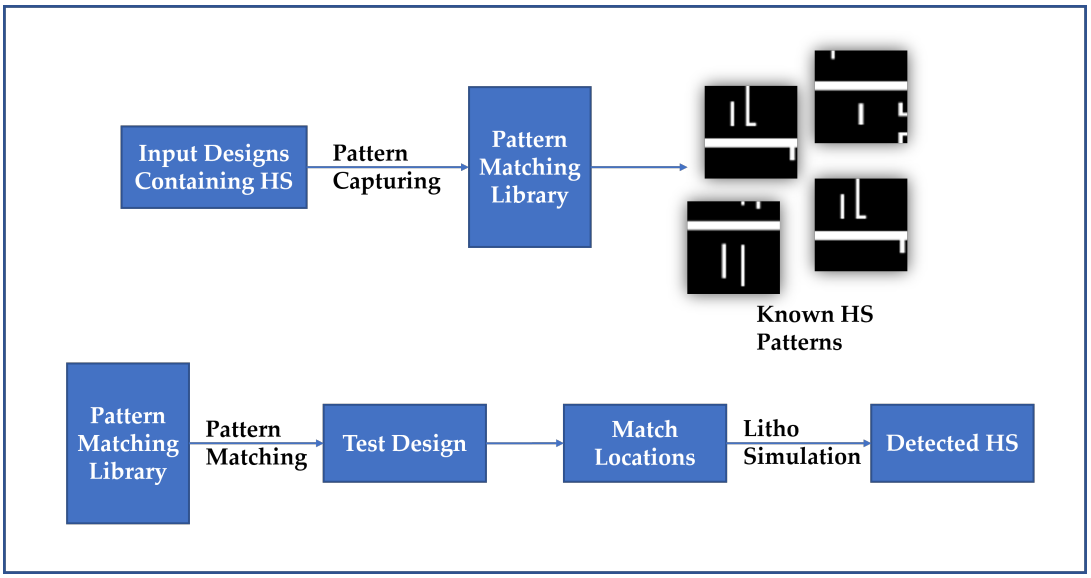


FIGURE 1.5: Pattern Matching HS Detection Flow

The last type is machine learning (ML)-based hotspot detection. In this method, a ML model is trained on a set of known hotspot and non-hotspot patterns. The model is then used to predict hotspots on a new design. This method is highly promising since it does not require as much computational power as lithography simulation. In addition, the inherent fuzziness of ML models can help them predict new hotspots [8]. More details on ML HS detection will be described in the next section. In addition, several machine learning-based hotspot detection techniques will be discussed in Chapter 2.

1.3 Problem Statement

This thesis explores the problem of lithography hotspot detection using machine learning. The process of HS detection using machine learning techniques can be divided into three steps: data generation, model training, and prediction. In the first step, patterns are captured from the input layout and encoded into a set of features that can be used for training. Each feature vector is then labeled as either corresponding to a hotspot (HS) or a non-hotspot (NHS). In the training step, the model iteratively minimizes a pre-defined loss function that measures the deviation from the model predictions during training from the actual labels. The learned model (achieving the minimal loss), is then used to predict patterns on the test layouts to determine whether they correspond to a hotspot or not.

There are three main challenges in the data generation step. The first challenge is that industrial training layouts typically contain millions of unique patterns [9]. It is difficult to use this amount of data to train a ML model due to computing and memory limitations. Hence, data selection is needed to reduce the training set size. The second challenge is the imbalanced nature of the problem; in a typical design, the ratio of hotspot to non-hotspots is very small, which causes a class skew in the generated datasets for training. Machine learning models are sensitive to imbalance in class distribution, which necessitates the employment of a sampling technique to choose a representative dataset for training that is more balanced. The third challenge is the existence of numerous non-hotspot patterns that are structurally and geometrically similar to hotspot patterns, which makes them harder to predict by the machine learning model, and

usually get classified as HS. These patterns are described as hard-to-classify (HTC) [10]. This can lead to an increase in the false alarm rate of the model. Hence, it is important to develop an efficient data sampling technique that addresses this issue, ensuring that the training dataset contains such patterns to help the model learn the subtle differences between HS and NHS.

The problem can be stated as follows: Given a particular training dataset, find the most representative patterns that can reduce the training cycle of any ML model while achieving equal or better performance than the model trained using the full dataset.

1.4 Organization of the Thesis

- Chapter 2 presents a background on the various machine learning methods that are used throughout the thesis. In addition, a literature review of the various ML HS detection models is done to evaluate the current status. Moreover, the different pattern clustering techniques are also studied and compared to determine the gaps and decide on how to proceed with data reduction for ML model training.
- Chapter 3 presents several feature generation methods that are commonly used for ML HS detection as well as pattern clustering. These methods are discussed in order to propose a new method to enhance the performance of ML models.
- Chapter 4 builds on the previous chapters to present our proposed flow for pattern clustering and dataset selection for training ML models.
- In Chapter 5, we explore a feature generation method based on image gradients to improve the performance of ML models.
- In Chapter 6, the outcomes of this thesis are summarized and directions for future work are proposed.

1.5 List of Contributions of this Thesis

- The use of autoencoders for dimensionality reduction is explored. The dimensionality of pattern images that are used to train machine learning-based HS detection models can be reduced by training an autoencoder to learn latent features of a smaller dimension that can be used to reconstruct them.
- The problem of data selection for training ML models is investigated by using encoded features. We use the autoencoder features to run clustering to group similar patterns together. Then, we propose two sampling approaches to prepare the training dataset.
- The use of the encoded features as input for ML-based HS detection models is also explored. This could help create simpler models that are less prone to over fitting.
- Feature engineering using image gradients [11] is investigated as method of improving the performance of ML-based HS detection models.

Chapter 2

Background and Literature Review

2.1 Background on Machine Learning

2.1.1 Basics of Machine Learning

In this section, we describe the basics of machine learning (ML) which is a branch of artificial intelligence (AI) that deals with algorithms that can build models which make predictions without being explicitly programmed [12]. There are three main types of ML algorithms:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

The typical supervised learning framework consists of several elements: train and test data $\mathbf{x}$ and target labels $\mathbf{y}$, where $\mathbf{x}$ and $\mathbf{x}$ are n and m-dimensional vectors respectively. The goal of the framework is to find a function f which converts the input data into the labels, while minimizing the error between the predicted and expected labels. This is expressed by Equation 2.1.

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

In unsupervised learning, the dataset is not labeled. The algorithm tries to discover patterns in the training dataset on its own, without being guided by labels. An example of this is clustering where similar training examples are grouped into different categories [13]. Another

example is Principal Component Analysis (PCA) [14], where the most discriminating features are obtained from the training dataset. These features can be used to reduce the dimensionality of the input features.

Finally, in reinforcement learning, the computer program is known as an intelligent agent (IA). It interacts with an environment in order to reach a certain goal. As the IA interacts with the environment, it performs actions and based on these actions it can receive positive or negative rewards [15]. In this way, it can learn the consequence of each action and update its behavior. The goal of the IA is to maximize its reward. For example, an autonomous car is an example of a reinforcement learning system. The car itself is the IA which interacts with the environment and can learn how to stay inside a lane and how to avoid collisions. This can be done by sensing the environment and learning by doing wrong actions. With several iterations, it can update itself to perform the right actions to achieve the required goal [16].

2.1.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are machine learning models that are inspired by biological neurons. The basic building unit of an ANN is known as a perceptron. A perceptron can be described as: $f(\mathbf{x}) = g(\mathbf{w}\mathbf{x} + b)$, where $\mathbf{w} \in R^n$ is a weight vector, $\mathbf{x} \in R^n$ is an input vector, n is the number of inputs, and b is a bias coefficient. $g(\cdot)$ is an activation function that introduces non-linearity. Examples of activation functions include: sigmoid [17], relu [18], and tanh. An illustration of the perceptron is shown in Figure 2.1 and plots of commonly used activation functions are shown in Figure 2.2.

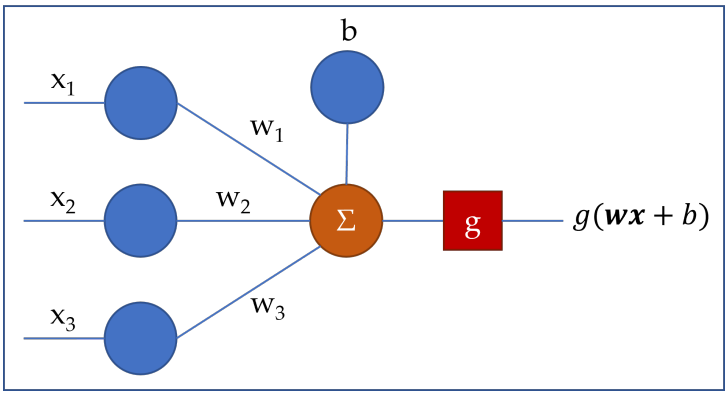


FIGURE 2.1: Illustration of a Perceptron

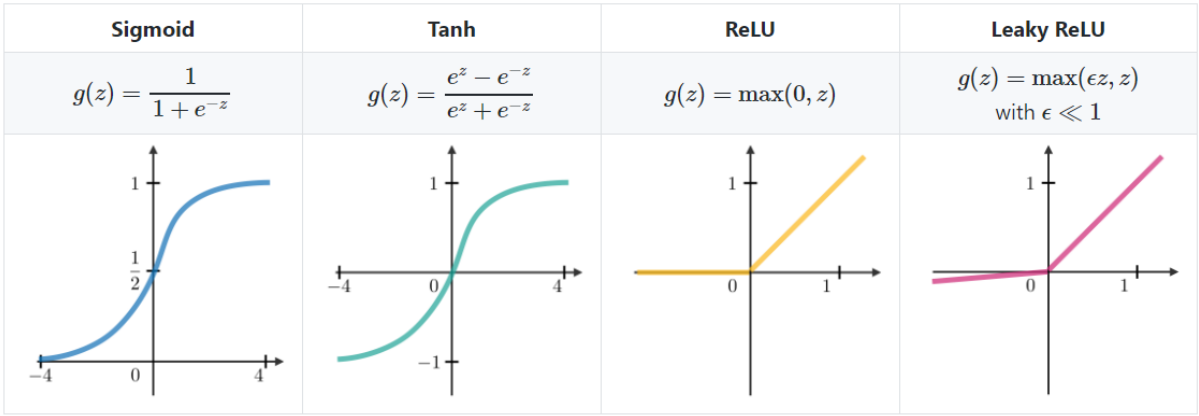


FIGURE 2.2: Common Activation Functions

An ANN consists of several perceptrons stacked together in several layers. Each perceptron is connected to every perceptron from the previous layer. This configuration is known as a fully connected neural network. The ANN can be used for both classification tasks and regressions tasks. In classification, each data point belongs to a class and the model is trained to distinguish between different classes. In regression, the network is trained to predict a continuous value. For example, a network can be trained to predict fuel efficiency of automobiles based on multiple parameters such as the number of cylinders, displacement, and horsepower of the engine [19].

To train an ANN, a loss function is needed to represent the error between the predicted and the expected labels. Then, the backpropagation algorithm runs to iteratively update the weights of the network to minimize the loss [20]. In case of binary classification, binary cross-entropy (BCE) loss is usually used. This is shown in Equation 2.2 where y is the ground truth label, and $\hat{y}$ is a label predicted by a classifier.

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y^i log(\hat{y}^i) + (1 - y^i) log(1 - \hat{y}^i) \tag{2.2}$$

More recently, a loss function known as Focal Sigmoid Cross Entropy was introduced for classification problems on imbalanced datasets [21]. An imbalanced dataset is one where the target classes have an uneven distribution of observations. For a binary classification task, one class has a very high number of observations, while the other has much less. This can cause the classifier to have poor predictive performance, especially on the minority class. This function

gives tuning flexibility for the loss of each class via two parameters: α and γ. α represents a weighting factor to give the minority class a higher loss value. γ changes the gradient of the loss function to reduce the loss for well-classified examples. For a larger value of γ, more focus is put on misclassified examples. These parameters can be tuned to improve the model performance. Focal loss is given by Equation 2.3 where p_t is the probability of the positive class that is output by the model. A graph of Focal Loss is shown in Figure 2.3 where the effect of γ can be observed.

Focal Cross Entropy is important for HS detection which is a typical example of an imbalanced dataset problem. As we will see in the later chapters, this loss will be used to account for the imbalanced nature of the HS detection datasets Tuning its parameters will help improve the performance of the ML model.

$$Focal\ Loss = -\alpha(1-p_t)^{\gamma} log(p_t) \tag{2.3}$$

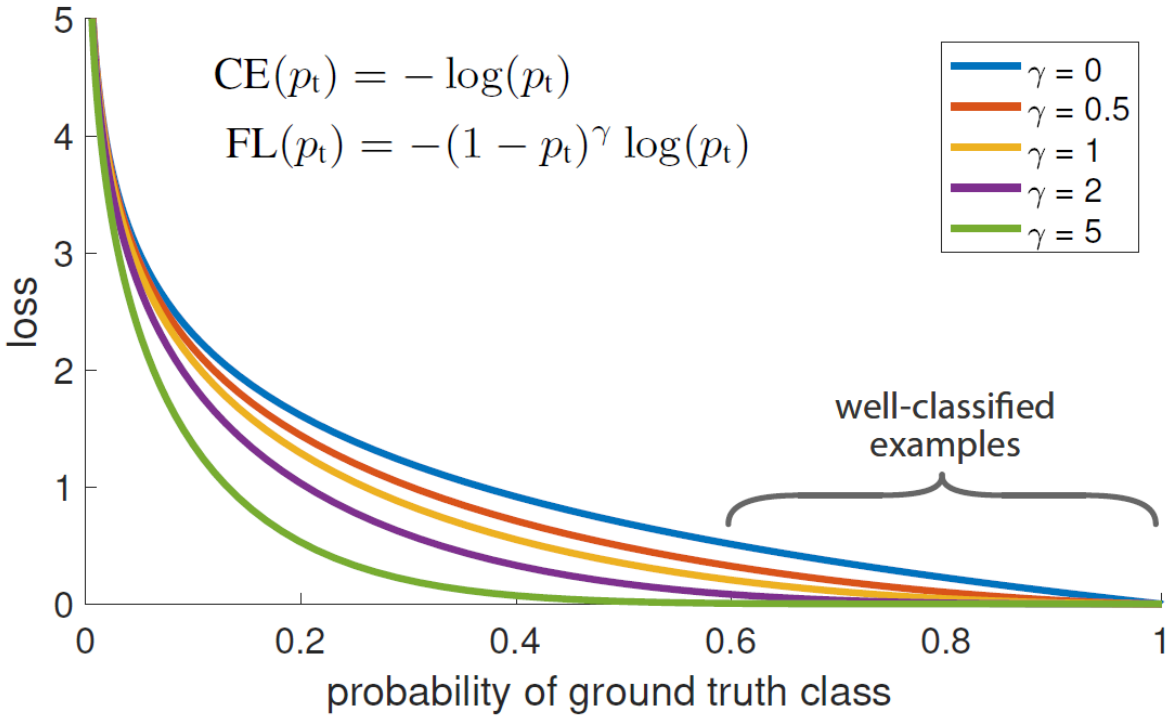


FIGURE 2.3: Sigmoid Focal Cross Entropy [21]

For regression problems, a loss such as mean squared error can be used. It is represented by Equation 2.4. Y_i is the i-th expected label and $\hat{Y}_i$ is the i-th predicted label.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.4}$$

2.1.3 Autoencoders

An autoencoder is a special type of ANN. It is an unsupervised learning method meaning that the data points have no labels. The autoencoder is trained to reproduce its input [22]. The loss function for the autoencoder is a reconstruction loss. For example, it can be the mean square error between the input and output. The hidden layers of the autoencoder usually have a smaller number of nodes than the input itself. This is shown in Figure 2.4 where the input and output layers have 10 nodes and the latent layer in the middle has 4 nodes. This creates a bottleneck in the autoencoder and forces the network to learn a compressed representation of each data point. Hence, it learns an encoding for each data point.

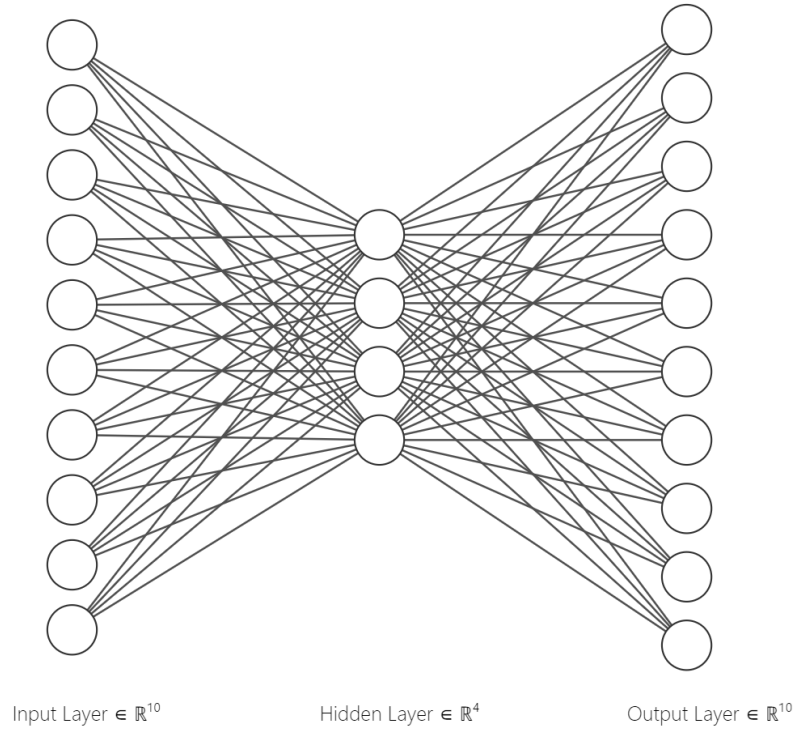


FIGURE 2.4: Example of Autoencoder Architecture

Autoencoders are powerful because they can learn key latent features that would enable it to reconstruct the input. The non-linear activation functions in the autoencoder can generate features that are more generic than other methods such as Principal Component Analysis (PCA).

2.1.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep neural network which takes 2-D objects (usually images)[1] as input, and then updates its weights and biases based on the input features. The difference between CNNs and fully connected neural networks is that the weights and biases are in the form of convolution filters. This allows the network to capture spatial features in the images. In addition, the number of trainable parameters in CNNs is much lower than fully connected architectures because not every node is connected to all other nodes. The reduced number of parameters helps reduce over fitting and has a smaller computational cost. In addition, flattening images and using a fully connected neural network would lead to a huge number of parameters and loss of spatial information. Hence, CNNs are highly suitable for image datasets. CNNs typically contain of three types of layers:

- Convolutional Layers
- Pooling Layers
- Fully Connected Layers

A convolutional layer consists of a number of filters. Each filter is a small matrix which is convolved with the image resulting in another image where each element is a weighted combination of the entries of a region or patch of the original image; an example is shown in Figure 2.5. As shown, the dot product between the image patch and the kernel ($1 * 1 + 0 * 2 + 1 * 3 + 0 * 4 + 1 * 5 + 1 * 6 + 1 * 7 + 0 * 8 + 1 * 9 = 31$) gives the value of the top left pixel of the output. The kernel is then shifted and the same calculation is repeated on the next patch. Based on the size of the kernel as well as the stride, the size of the output image can be calculated. While training a CNN, these filters are updated based on the loss function in order to learn the key features to help predict a certain output.

[1] By abuse of notations in this section and throughout the thesis, we refer to these 2-D objects as images.

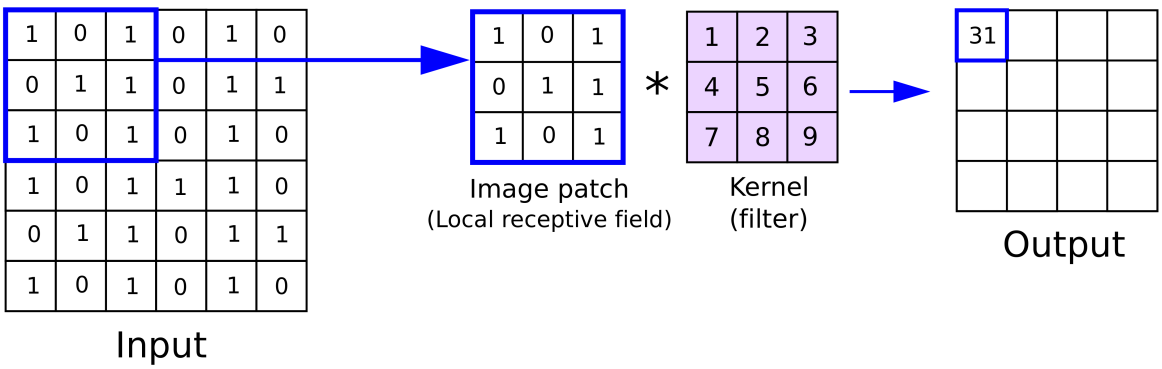


FIGURE 2.5: The Convolution Operation [23]

Pooling layers reduce the size of the images, keeping only the more important features. Pooling is done by also sliding a matrix across the input images with a certain stride. The most important difference between pooling and convolution is that pooling does not have learnable parameters. This means that the same operation is done regardless of back propagation. The most common type of pooling is Max Pooling. In Figure 2.6 a 2-by-2 filter is used with a stride of 2. In this case, the maximum value of each 2x2 block of the original image is calculated as the output.

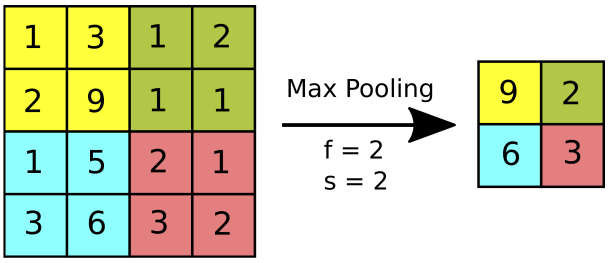


FIGURE 2.6: Max Pooling [23]

Fully connected layers usually come before the output of a CNN. The outputs from the previous layers are flattened and passed to one or more fully connected layers. These can then be used to calculate the final output of the network whether it is a classification or a regression network.

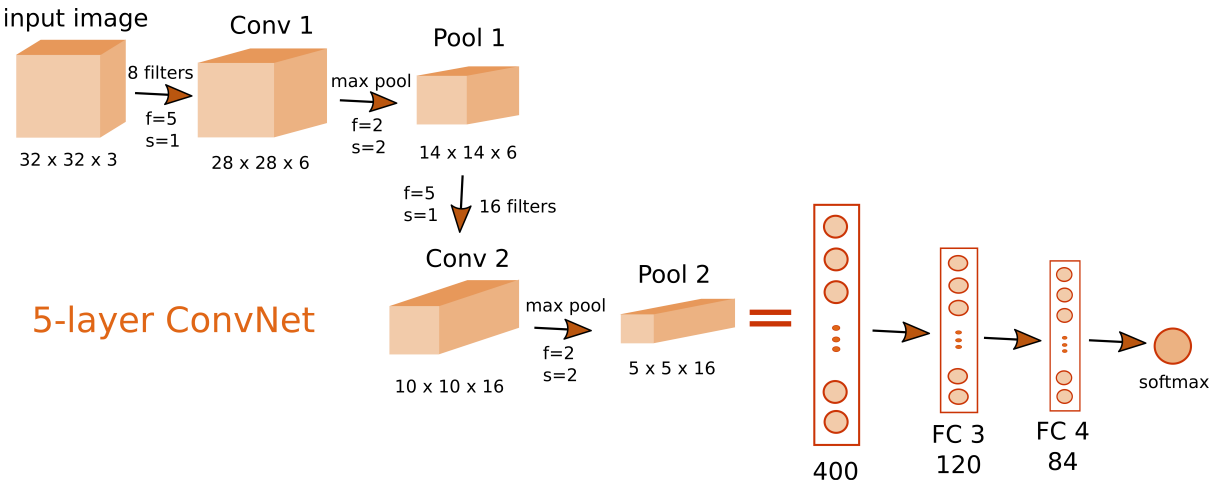


FIGURE 2.7: Example of Convolutional Neural Network [23]

Figure 2.7 shows a simple CNN that consists of two convolutional, two pooling and two fully connected layers. We can observe that the size of the image decreases while the number of filters and number of channels increase as we go deeper into the network. We can also see that towards the end of the network, the output of the second pooling layer is flattened and then passed into two fully connected layers. Finally, the last layer is a softmax layer which is used for classification problems.

2.1.5 Clustering

Overview

Clustering is a type of unsupervised learning where the data points are divided into a number of groups, where each group contains points that are very similar to each other. In simple terms, the objective is to segregate groups with similar features and assign them into clusters. Figure 2.8 shows a simple example of clustering, where we first start with a group of unorganized shapes. By using the right features for each data point and having a distance measure, a clustering algorithm can separate the data points into clusters where each cluster would contain similar points. Note that each data point does not have a label i.e. circle, square, triangle, but rather, each point is described by features such as the number of vertices and edges, and based

on that it can calculate distances and assign cluster IDs. Hence, clustering is considered to be an unsupervised machine learning algorithm.

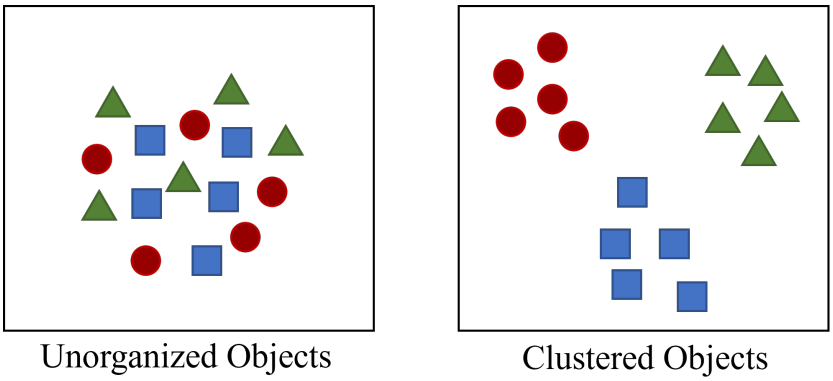


FIGURE 2.8: Simple Clustering Example

k-means Clustering

k-means is one of the simplest and popular clustering algorithms. The input to the algorithm is k which is the required number of clusters. The algorithm begins by initializing k centroids. Then, each data point is assigned a cluster ID using the closest centroid based on some distance measure. Next, the mean of the points is calculated for each cluster to update the location of the k centroids. Then, the cluster IDs are updated based on the new centroids. This is repeated until the centroids stop changing or a maximum number of steps is reached.

k-means is the basic clustering algorithm which can be used for many applications. However, it has two issues:

- The number of clusters k is an input parameter. It can be difficult to choose the right value and a wrong one can yield misleading results.
- It does not always converge to the global minimum, hence the result might be counter-intuitive.
- It assumes that the clusters are spherical with similar sizes. This might not be suitable for all datasets.
- It is sensitive to outliers due to the mean calculation.

Hierarchical Clustering

Hierarchical clustering is an algorithm that builds a hierarchy of clusters. This algorithm starts with each data point assigned to a cluster on its own. Then, the two nearest clusters are merged into the same cluster. This process is repeated until the whole data is combined into one cluster. Alternatively, the process can be halted when the required number of clusters is obtained.

Hierarchical Clustering is usually represented as a dendrogram as the one shown in Figure 2.9. At the bottom each point is in a cluster on its own. Then, the points are merged based on a distance measure between clusters. This algorithm follows the agglomerative or bottom-up approach. It can also be done in a top-down or divisive approach where we start with a single cluster containing all the points.

Hierarchical clustering has a time complexity of $\mathcal{O}(n^3)$ and space complexity of $\mathcal{O}(n^2)$. This makes it difficult to use for large datasets [24]. Hence, other methods are usually used in practical situations.

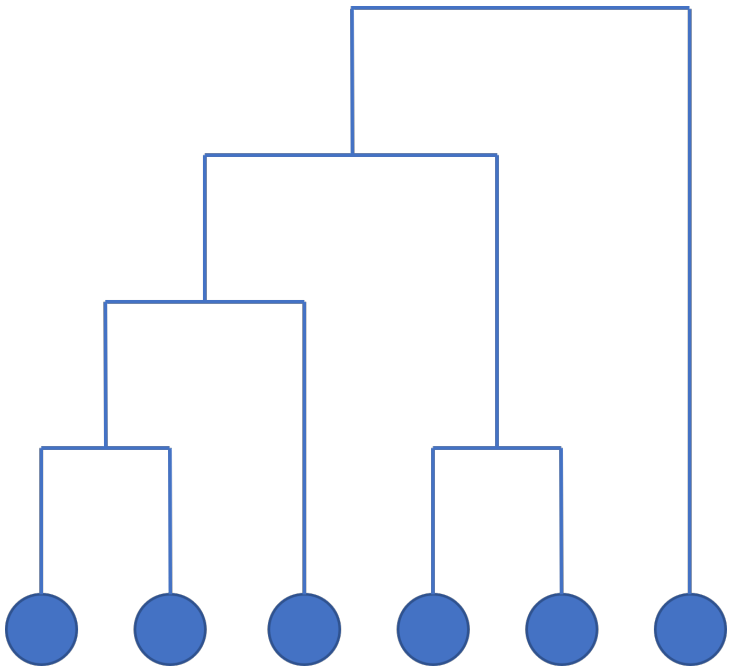


FIGURE 2.9: Dendrogram Representing the Result of Hierarchical Clustering

Density-based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is a clustering algorithm that was proposed in 1996 [25]. It is described as "density-based" because it can group points that are closely packed together. In addition, it can identify outliers that are far away from dense regions.

DBSCAN requires two input parameters: radius of neighborhood around a point (ϵ) and the minimum number of points required to form a dense region ($minPts$). Hence, the number of clusters does not need to be specified as an input. It is automatically determined based on the values of ϵ and $minPts$.

DBSCAN classifies points into one of the following:

- Core Point: this is a point which has at least $minPts$ data points (including itself) within radius ϵ
- Density-Reachable: this is a point that has at least one core point within its radius but is not a core point itself
- Noise: this is point that is neither a core point nor density-reachable

This classification is shown in Figure 2.10.

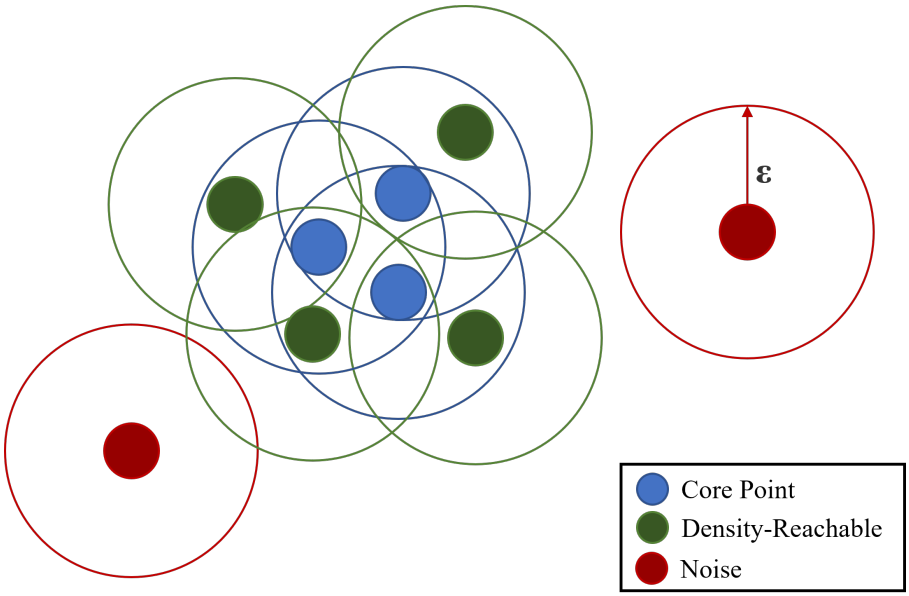


FIGURE 2.10: DBSCAN with $minPts = 4$ and ϵ of Each Point Shown

DBSCAN runs as follows: First, a random point is selected. If the neighborhood of the point contains at least $minPts$ points a cluster is started, otherwise the point is labeled as noise. If the point is part of a dense cluster, then it is added to the cluster as well as all the points in its neighborhood. Noise points can later on be added to a cluster if they are found to be part of a dense neighborhood. This process is done until a connected cluster is found. Then, another unvisited point is obtained and the same process is repeated. An abstract version is described in Algorithm 1 [26].

Algorithm 1 Abstract DBSCAN Algorithm

1: Compute neighbors of each point and identify core points
2: Join neighboring core points into clusters
3: **foreach** non-core point **do**
4: Add to a neighboring core point if possible
5: Otherwise, add to noise

2.2 Machine Learning-Based Hotspot Detection

A major part of our thesis work is focused on lithography hotspot detection using machine learning as will be presented later in Chapters 4 and 5. Therefore, in this section, we will introduce the available benchmark datasets for hotspot detection. Then, the literature be examined to analyze and compare the different ML-based HS detection methods.

2.2.1 Benchmark Datasets for Hotspot Detection

ICCAD-2012 Benchmark

The ICCAD-2012 dataset was introduced as part of the computer-aided design (CAD) contest of the ICCAD conference. Since its inception, this dataset has been the standard for comparing the various machine learning HS detection models. As will be seen in the subsequent sections, this dataset has been highly cited and is used almost exclusively for comparing machine learning models. The benchmark statistics are shown in Table 2.1.

TABLE 2.1: ICCAD-2012 Benchmark Statistics

	Training Dataset		**Testing Dataset**	
	HS Count	**NHS Count**	**HS Count**	**NHS Count**
Benchmark1	99	340	226	319
Benchmark2	174	5285	498	4146
Benchmark3	909	4643	1808	3541
Benchmark4	95	4452	177	3386
Benchmark5	26	2716	41	2111

ICCAD-2019 Benchmark

The ICCAD-2019 dataset [10] was introduced to tackle the shortcomings of the earlier ICCAD-2012 dataset. The ICCAD-2012 dataset was shown to have some underlying structural issues that led to inaccuracies in reporting the machine learning models performance. The dataset showed that the false alarm rates for many of the machine learning models presented in the literature were higher than what they were believed to be. In addition, it was shown that the

ICCAD-2012 does not contain "truly not seen before" (TNSB) hotspots. Hence, measuring HS accuracy might be misleading using this dataset.

The ICCAD-2019 dataset addresses these issues by introducing a single training dataset and two test datasets. Test Dataset I addresses the issue of TNSB hotspots and Test Dataset II addresses the problem of false alarm rates. The statistics for ICCAD-2019 dataset is shown in Table 2.2.

TABLE 2.2: ICCAD-2019 Benchmark Statistics

	HS Count	**NHS Count**
Training Dataset	467	17758
Testing Dataset I	1001	14621
Testing Dataset II	64310	65523

2.2.2 SVM-based Methods

Early adoption of ML techniques for hotspot detection relied on feature engineering and the use of support vector machines (SVM). In [27] the authors proposed a feature engineering method where various layout pattern features are generated:

- Corner Information (convex or concave)
- External distances between polygon edges
- Internal distances between polygon edges

These features are shown in Figure 2.11.

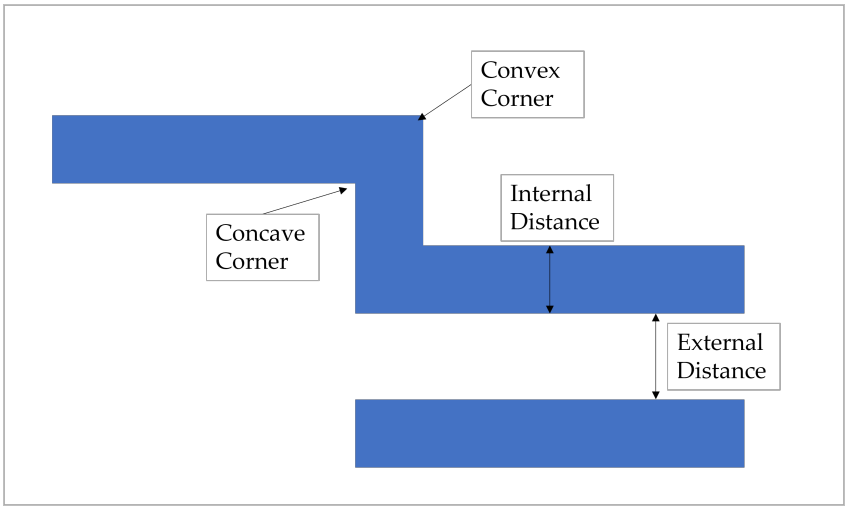


FIGURE 2.11: Examples of Layout Features

Each edge is divided into fragments and given a certain radius, the features outlined above are calculated. These features are then used to train a support vector machine (SVM) and an artificial nerual network (ANN). The two approaches are then compared. In addition, they also introduce a process for iteratively refining the models using different classification thresholds. Both ML models were tested using industrial layouts and have shown high HS recall (> 85%) and high precision (99.9%) [27].

In [28], the same fragment-based layout features are used. They also use SVM as a classifier. However, they employ principal component analysis (PCA) to reduce the dimensionality of the feature vector. Moreover, due to the high imbalance between hotspots and non-hotspots which can negatively affect the SVM, k-means clustering is used to group similar patterns. Then, the center of each cluster can be used as a representative. The center here is defined as the mean of the patterns in each cluster. In this paper, the ICCAD'12 dataset is used as a benchmark [29]. The results on the test set are shown in Table 2.3. The results are shown for the features with and without PCA. It can be seen that using PCA improves precision, and therefore, improves the F1 score of the SVM.

In [30], the input patterns are grouped into hotspot and nonhotspot clusters according to the polygon topologies in their core areas. Then, an SVM is trained to classify patterns in each cluster. Finally, the false alarms are used to train an additional SVM to fine tune the results by

TABLE 2.3: Prediction Results on ICCAD'12 Test Set [28]

Without PCA			With PCA		
Recall	Precision	F1 Score	Recall	Precision	F1 Score
83.6%	6.9%	12.7%	83.0%	8.5%	15.5%

making use of the geometries outside the core area. This approach has shown to improve HS accuracy when compared to training a single SVM. A recall of 92.7% and precision of 7.4% was achieved.

In [31], the authors used a very similar approach to that used in [30]. The difference was in the usage of an industrial pattern matching tool for classifying the patterns [32]. In addition, this method uses fewer inputs features to describe the patterns. This method achieved 87.9% recall and 10.7% precision.

Table 2.4 compares the recall, precision, and F1 scores of the various SVM-based models on the ICCAD-2012 dataset. Note that the model in [27] does not have results on this dataset because it was published before the dataset was introduced.

TABLE 2.4: Comparison of SVM-Based Models on the ICCAD'12 Test Set

Method	**Recall (%)**	**Precision (%)**	**F1 Score (%)**
Hierarchical Approach [27]	N/A	N/A	N/A
PCA-SVM [28]	83	8.5	15.5
Topological Classification [30]	92.7	7.4	13.7
SVM and Clustering [31]	87.9	10.7	19.1

2.2.3 Deep Learning Methods

As integrated circuits became increasingly complex due to technology scaling, detection of hotspots became more difficult. Hence, more sophisticated methods based on deep learning were introduced. A comparison between the different models is shown in Table 2.5.

Shin and Lee were one of the first authors to use convolutional neural networks for HS detection [33]. Their method is based on a sliding window to scan the input layout to obtain various patterns. Then a CNN classifier is trained using images of the patterns from the training layout. Their CNN model consists for four convolutional and pooling layers followed by two fully-connected layers. For prediction on a test layout, the same sliding window approach is

used to predict the probability of HS. The coordinates of the center of each image is used to create HS probability map. DBSCAN clustering is then performed on this map to filter the results to extract HS coordinates and remove outliers. This method has shown a recall of 95.5% and precision of 22.2% on the ICCAD-2012 test dataset.

Another CNN based model was introduced by Borisov and Scheible [34]. In this paper, they have used a model with two "basic blocks". Each basic block consists of three convolutional layers, a batch normalization layer, followed by activation and then max pooling. They also introduced used a skip connection between layers to improve the convergence of the model and reduce the effect of vanishing gradients. This method was also tested on the ICCAD-2012 dataset. This method achieved a recall of 98.9% and precision of 97.1%.

Moreover, Matsunawa et al. have introduced a HS detection method using fully-connected deep neural networks (FC-DNN) [8]. The input to the network is a density-based encoding of each pattern. The network is trained in two steps: pre-training and then fine tuning. Pre-training is an unsupervised training method was introduced by Hinton [35]. Instead of training a sequence of fully-connected layers at once, the layers are trained iteratively as an autoencoder to reconstruct its input. Then, in the fine-tuning phase, supervised training is performed to train the network as a classifer to differentiate between hotspots and non-hotspots. This method was also tested on the ICCAD-2012 dataset and has shown superior results with recall greater than 95% and false alarms very close to zero.

More recently, Yang et al. proposed a method for feature generation using discrete cosine transform (DCT) [36, 37]. At first, each layout pattern image is divided into $n - by - n$ blocks. For each block, DCT is used to convert it into the frequency domain. Then, the upper-left coefficients of each block are used to represent the block. These coefficients are low-frequency components which are usually sufficient to reconstruct the block. In addition, the number of coefficients needed is much smaller than the original block size, hence a high compression ratio is achieved. The authors call this feature representation a "feature tensor". The feature tensors are then used to train a convolutional neural network. This method was tested on the ICCAD-2012 dataset and achieved a recall of 97.7% and a precision of 99.9%.

In a very recent paper, Huang et al. proposed two improvements in HS detection [38]. First,

TABLE 2.5: Comparison of Deep Learning Models on the ICCAD'12 Test Set

Method	Recall (%)	Precision (%)	F1 Score (%)
CNN [39]	95.5	22.2	36.0
FC-DNN [8]	95.0	99.0	97.0
CNN with Skip Connection [34]	98.9	97.1	98.0
CNN with DCT Features [36]	97.7	99.9	98.8
CNN with Ensemble Learning [38]	97.4	94.8	96.1

a multi-input deep learning model is proposed, which includes a CNN model that takes input images and a fully connected DNN model that takes input physical features such as the number of polygons and number of corners. Second, an ensemble learning method is proposed based on multiple submodels. This is done by doing a second round of training, combining the weights of models that achieve highest recall, precision, and F1 score respectively into another network to remove the redundant weights. In addition, they also used sigmoid focal loss [21] which is more suitable for imbalanced datasets than the traditional binary cross entropy.

2.3 Layout Pattern Classification

2.3.1 Overview

Layout pattern classification is required for integrated circuit design. There are different applications for pattern classification such as design space analysis, design rule generation, and systematic yield optimization. On the other hand, this functionality is provided by very few vendors, hence, there exists a need for an open source or academic solution. The pattern classification contest in the ICCAD 2016 conference was an important enabler of pattern classification solutions, as it provided a benchmark dataset that enabled fair companions between various methods. In addition, it formulated the problem with clear inputs, outputs, and objectives [40].

The input consists of a layout in GDS format. There are marker polygons that indicate the hotspots on the layout. There are two types of clustering that are required: area-constrained clustering (ACC) and edge-constrained clustering (ECC). Clustering can be performed using either ACC or ECC but not both simultaneously [40].

In ACC, the pattern similarity is determined based on the difference in area between two patterns. Pattern similarity increases as the differnce in area decreases. ACC is given by Inequality 2.5 where A and B are two patterns, w and h are the width and height of the pattern extent, and α indicates the ratio of area match. α is a number between 0 and 1 where $\alpha = 1$ indicates an exact match [40].

$$\frac{Area(XOR(A, B))}{w \cdot h} \leq 1 - \alpha \tag{2.5}$$

In ECC, the similarity is measured by how much the pattern edges are allowed shift. This is indicated by a parameter e which is a positive floating-point number. $e = 0$ indicates an exact match. Two patterns are considered to be part of the same cluster if the shifts between their corresponding edges are $\leq e$ [40].

The objective is to group the patterns into clusters based on ACC or ECC while at the same time minimizing the number of clusters.

2.3.2 Pattern Classification in the Literature

In this section, we will explore several pattern classifications methods in the literature and compare them. All of the methods use the ICCAD-2016 contest dataset as a benchmark.

Ishino et al. proposed that the pattern classification problem can be converted into a minimum dominating set problem [41, 42]. This problem can be solved using Interger Linear Programming (ILP) [43]. As the size of the problem increases, it would be difficult to solve in a practical time, hence, an initial clustering step is done to divide the large problem into smaller problems. Then, each cluster is fed into an ILP solver to obtain the final result. For the initial clustering step, a feature extraction step is required. Two methods are used: density based, and Fourier Transform based. The density based feature vector is basically a flattened vector of pixels. The authors claim that this would be suitable for both ACC and ECC. The Fourier Transform method is suitable for ECC. In addition, two clustering algorithms are explored: k-means++ [44] and Agglomerative Clustering [24].

Oliveira et al. proposed a greedy clustering algorithm that outperforms most of the ICCAD-2016 contest winners [45]. They also incorporated a method to divide complex shapes into rectangles to speed up comparisons between patterns.

Another clustering method was introduced by Woo et al. in 2017 [46]. This method is based on transforming the classification problem into a Set-Covering Problem (SCP) [47]. Then, a metaheursitic algorithm known as Greedy Randomized Adaptive Search Procedures (GRASP) [48] is used to solve the SCP.

Moreover, Chang et al. introduced another clustering technique based on Markov clustering [49]. In this method, a graph is constructed where the nodes represent a pattern and the edges represent the similarity between two clips. The Markov Clustering algorithm is a stochastic process which works based on the assumption that a random walk that reaches a dense cluster with likely not leave the cluster until most of its nodes are visited. This process will discover a partial grouping of the patterns. A post processing step is then performed on each cluster to assign representative patterns and make sure that the clustering constraints (ACC or ECC) are satisfied.

In addtion, Wu et al. proposed a MapReduce-based method that reduces the size of the pattern classification problem [50]. Using this method they were able to minimize the number of clusters and maximize the size of the largest cluster.

Recently, Xu et al. introduced a method to find the lower bound of the cluster count [51]. The lower bound calculation is converted into a Maximum Clique Problem (MCP) [52] which is then solved to find the minimum number of clusters. Similar to the method in [46], the authors formulate the problem as a SCP, however, they use the result of MCP to initialize the SCP solution. They showed that this greatly improves the performance and the convergence of the SCP. The final results show that this technique has a smaller run time when compared to [46]. In addition, the results show that the cluster count obtained via this method is equal to the lower bound that was determined via MCP which shows that the optimal value is reached.

Table 2.6 compares the different clustering approaches. n is the number of clusters, s_{max} is the size of the largest cluster, and t is the run time in seconds. Note that the run time is reported for reference only because they were tested on different machines.

TABLE 2.6: Comparison Between Different Pattern Classification Approaches*
*The reported run times are for reference only because they were not tested on identical machines.

Benchmark	Parameters	Reference [29]			iClaire [49]			GRASP [46]			Maximum Clique [51]			Greedy [45]		
		n	s_{max}	t (s)	n	s_{max}	t (s)	n	s_{max}	t (s)	n	s_{max}	t (s)	n	s_{max}	t (s)
Case1	Default	8	5	0.903	8	5	0.001	8	5	0.001	8	5	5.00E-04	8	5	0.001
	α = 0.95	4	6	1.808	3	9	0.005	3	9	0.001	3	9	0.001	4	6	0.001
	e = 4	5	5	1.324	5	5	0.005	5	5	0.001	5	5	0.003	5	5	0.001
Case2	Default	26	104	1.167	26	104	0.004	26	104	0.017	26	104	0.003	26	104	0.004
	α = 0.95	13	106	0.854	11	106	0.011	11	106	0.014	11	106	0.007	12	106	0.006
	α = 0.9	10	114	1.168	7	112	0.01	7	112	0.015	7	115	0.006	8	112	0.006
	e = 4	18	104	0.874	18	104	0.011	18	104	0.017	18	104	0.02	18	104	0.011
Case3	Default	70	792	1.314	70	792	0.06	70	792	0.149	70	792	0	70	792	0.154
	α = 0.85	26	1,344	1	13	2,608	0	13	2672	0.159	13	2,672	0.191	20	2,176	0.243
	e = 8	52	1,048	1.311	37	1,056	0.1	47	1048	0.161	37	1,104	0.184	76	792	0.484
Case4	Default	72	193,370	5.279	72	193,370	4.17	72	193,370	2.298	72	193,370	2.913	72	193,370	410
	α = 0.99	31	197,660	4.74	24	197,830	4.27	24	197,830	2.352	22	198,000	2.987	26	197,830	410
	e = 2	57	193,540	4.364	46	193,710	4.48	52	193,540	2.428	46	198,000	3.376	184	192,240	26

2.4 Chapter Summary

In this chapter, we have presented a background on various machine learning algorithms that would be used in later chapters of this thesis. These algorithms include: artificial neural networks, convolutional neural networks, and autoencoders. Then, a summary of several clustering algorithms were discussed as this is one of the major themes related to our problem statement.

Moreover, the problem of HS detection using machine learning was discussed. The well-known ICCAD benchmarks were presented. Then, various ML methods were discussed and compared. These methods include SVM-based methods as well as deep learning methods.

Finally, layout pattern classification was discussed to determine the state of the art techniques and evaluate their performance. This would help us propose our own method for clustering and sampling to prepare training datasets for ML HS detection.

2.5 Conclusion

The literature review showed that numerous efforts have been made to explore various techniques to improve machine learning HS detection models. In addition, several methods were proposed to tackle the problem of pattern clustering. However, there are many areas that require further exploration:

- The benchmark datasets that are available to compare models are limited. The ICCAD-2012 dataset [29] is the one that is most widely used to compare models. However, it has been shown that it is not suitable for that purpose. The recent ICCAD-2019 dataset [10] which was introduced to tackle the ICCAD-2012 dataset issues has not yet gained enough traction in the literature.

- Most of the literature is focused on introducing new models or enhancing existing models, but there is very little information on how to select the right samples for training the models. Due to the imbalanced nature of the problem, dataset selection is highly important. In addition, industrial training designs contain billions of patterns and thus, data sampling

is required since it would not be feasible to train a model using this large number of patterns.

- There are many pattern classification algorithms available in the literature. However, the effect of pattern clustering and sampling on machine learning models has not been studied.

Based on the the identified gaps, the following ideas can be explored:

- Investigate methods for pattern clustering and sampling to prepare datasets for training machine learning models.
- Use autoencoders to learn latent features that can be used for unsupervised clustering and sampling of layout patterns.
- Explore the effect of sampling on the performance of machine learning HS detection models and use the ICCAD-2019 dataset as a benchmark for the study.

Chapter 3

Feature Generation Methods

3.1 Introduction

In Chapter 2, we have discussed various machine learning algorithms for hotspot detection. These methods included traditional ML approaches like SVM and more sophisticated methods like deep learning. In this chapter, we explore the most commonly used feature generation techniques that enable these algorithms to be leveraged. The first method to be discussed is rasterization, where a layout pattern is converted to an image. This enables the use of convolutional and fully-connected neural networks as in [33, 34, 8, 38]. The second method is using discrete cosine transform (DCT) which transforms images into the frequency domain and enables the dimensions of the feature vectors to be reduced. This has been explored in [36]. The third and last method is the use of image gradients to describe spatial properties of images. This was discussed in [53] and will be further explored in Chapter 5 as an attempt to improve model performance.

In the following sections, these three methods will be explained in more detail to give the reader enough background. These methods will then be used in Chapters 4 and 5 to generate input features for ML models.

3.2 Rasterization

The first method for feature generation is rasterization. This is basically converting a layout pattern into an grey scale image. The layout pattern is usually represented in the form of

vertices, edges, or polygons. The rasterization process converts an $m - by - m$ nanometer (nm) window into an $n - by - n$ pixels image. There are numerous algorithms for performing this task. One of the early techniques that is the basis of modern rasterization techniques was presented by Juan Pineda in 1988 [54]. Assume that we have a triangle shape that needs to be rasterized. There are two steps that should be performed:

- Determine which pixels overlap the rectangle
- Assign pixels with the intensity values based on whether they overlap the triangle or not

The main method to determine which pixels overlap the rectangle is known as the edge function. This is illustrated in 3.1. To determine whether point P is inside the triangle, we loop over the edges in a clockwise manner starting from v_0. For each edge, if point P lies of the right of the edge, then it is assigned a "positive" label, otherwise it is assigned a "negative" label. We can see that point P has a positive label for all three edges. If the point P is the center of a pixel then we can determine whether the pixel overlaps the triangle. In this manner, we can determine which pixels overlap the triangle and then proceed to assign intensity values [54].

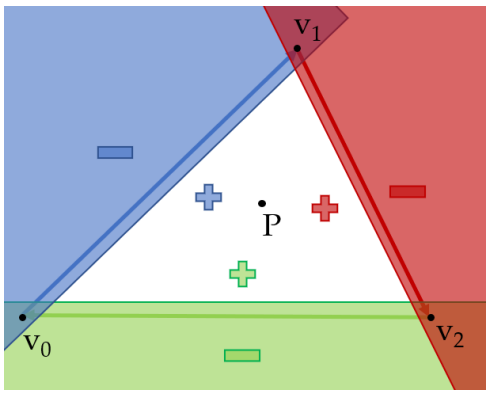
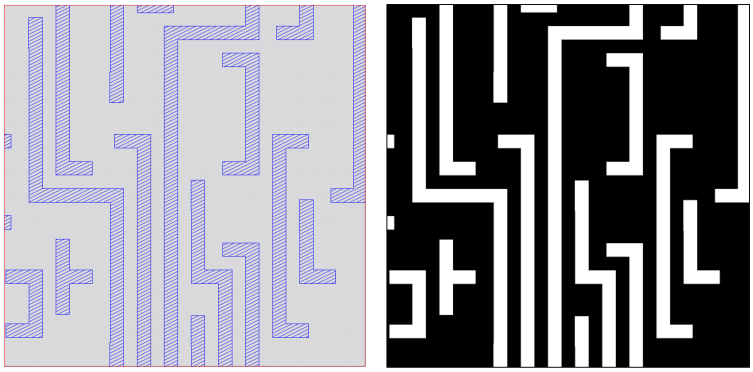


FIGURE 3.1: Illustration of the Edge Function

This method is the basis for more sophisticated algorithms that make use of graphics processing units (GPUs). These algorithms leverage the parallelism provided by GPUs for high performance graphics [55, 56].

For layout patterns, a square window of size $m-by-m$ nm is converted into an $n-by-n$ grey scale image. As n increases, the resolution and the number of features increases. Depending on

the technology node and the size of the window, the image resolution can be adjusted to obtain a clearer representation of the pattern. Figure 3.2a shows the layout pattern and Figure 3.2b shows the corresponding pattern image.



(a) (b)

FIGURE 3.2: Example of a Layout Pattern and Its Corresponding Image

Pattern images can be used for HS detection purposes. They are suitable for use with fully-connected and convolutional neural networks. However, their disadvantage is that the feature size grows quadratically as the image size increases.

3.3 Discrete Cosine Transform

The second method for feature generation is DCT. It was first introduced by Nasir Ahmed in 1974 [37]. DCT transforms a sequence of data points into a summation of cosine functions of different frequencies. Essentially, the sequence of points is converted from the spatial domain into the frequency domain. This has allowed the development of data compression standards for digital media. An example of this is the Joint Photographic Experts Group (JPEG) format which is widely used to store digital images [57].

Given a sequence of N real numbers $x_0, x_1, ..., x_{N-1}$, DCT transforms it into a sequence of N real numbers $X_0, X_1, ..., X_{N-1}$ given by Equation 3.1. The inverse DCT is given by Equation 3.2.

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \quad \textit{for } k = 0, 1, ..., N\text{-}1 \tag{3.1}$$

$$x_n = \frac{1}{2}X_0 + \sum_{k=1}^{N-1} X_k \, cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \quad for \;\; n = 0, 1, ..., N\text{-}1 \tag{3.2}$$

For an image, DCT can be applied in a two dimensional way by applying the one dimensional version along the rows and then along the columns or vice versa. For an $n - by - n$ image, a corresponding $N - by - N$ matrix of DCT coefficients can be obtained. These coefficients can be used to reconstruct the image by performing inverse DCT. In many cases, the DCT components have different magnitudes thus, not all of them have the same effect on the reconstructed image. In the case of JPEG compression, the high frequency components can be discarded without affecting how the user sees the image, therefore, giving room for data compression [57].

The nature of DCT components can be helpful for dimensionality reduction, where a high dimensional feature vector can be represented in a lower dimensional space. This can be used for the application of HS detection using ML. Pattern images can be divided into $m - by - m$ blocks and the DCT coefficients for each block can be calculated. The components with small magnitude can be discarded, leaving only the relevant components which can then be used as input features to a ML algorithm. This has been explored by Yang et. al and has shown good results on the ICCAD-2012 dataset [36].

3.4 Image Gradients

The third method for feature generation is based on image gradients. Gradients represent the rate of change in pixel intensity. A gradient vector has two components: magnitude and orientation. The magnitude represents the rate of change and the orientation represents the direction of the change. Figure 3.3a shows a small patch of a pattern image where the metal polygon is represented by white pixels. The gradient vectors are shown in grey. It can be seen that the vectors point to the left where the intensity increases from black to white. The length of the vector indicates the magnitude of the change; arrows closer to the metal polygon have a bigger length and vice versa.

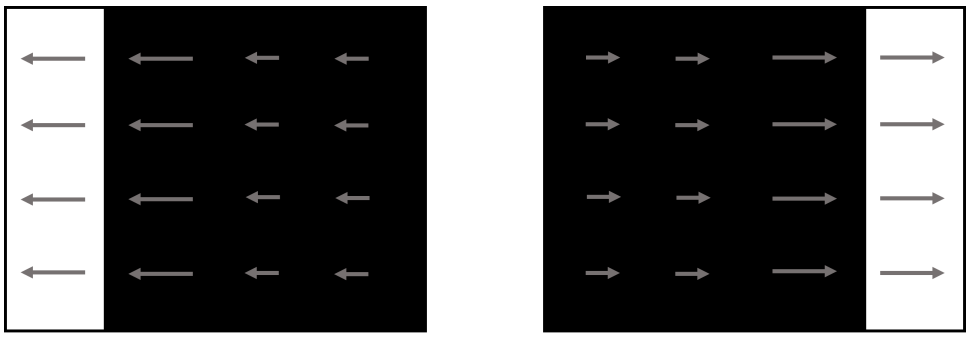


FIGURE 3.3: Illustration of Image Gradients

Pixel intensities on their own lack spatial information. Figures 3.3a and 3.3b have equal average densities. However, the first one has a metal polygon on the left and the other one has the metal on the right. Using gradient vectors, these two patterns would be differentiated since they have opposite gradient orientations. Hence, image gradients provide additional spatial information that would be useful for HS detection applications.

The most common way to calculate gradients is using the Sobel filter [58]. The input image is convolved with the Sobel kernel in two directions to obtain the horizontal and vertical components of the gradients. There are two kernel sizes: 1x3 and 3x3. These are represented by the matrices in 3.3 and 3.4. G_x is used to calculate the gradient in the x direction and G_y is used for the y direction.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{3.3}$$

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.4}$$

The overall magnitude and orientation for each pixel can then be calculated using Equations 3.5 and 3.6. The OpenCV library for image processing contains several functions that can easily calculate these quantities [59]. cv::Sobel() can be used for the convolution operation and

cv::cartToPolar() can be used to calculate the overall magnitude and orientation from the x and y components.

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3.5}$$

$$\theta = atan2(\frac{G_y}{G_x}) \tag{3.6}$$

Based on image gradients, several image descriptors were proposed. These methods include: Scale Invariant Feature Transform (SIFT) [60], Speeded Up Robust Features (SURF) [61], and Histogram of Gradients (HOG) [11]. These methods have many applications including object recognition, image classification, and feature matching.

3.5 Conclusion

In this chapter, the most commonly used feature generation methods for ML-based hotspot detection were discussed. These methods are important because they enable the transformation of layout patterns into features that could be used as input to various ML algorithms. In addition, methods such as DCT can enable dimensionality reduction by only keeping the most representative features. In the following chapters, some of these methods would be used to generate features that will be fed into ML algorithms for hotspot detection.

Chapter 4

Data Sampling for ML-based Hotspot Detection

As mentioned in Section 1.3, our goal is to reduce the size of the training dataset and at the same time tackle the issue of data imbalance in ML-based hotspot detection systems. Industrial datasets typically contain millions of unique patterns [9]. This makes it difficult to train ML models in a feasible amount of time, therefore, data reduction is needed. This will be done via clustering and data sampling techniques. To start this chapter, the next section will define the ML system that will be used. Then, a sampling method will be introduced to enhance the training dataset selection. This method is divided into three steps: dimensionality reduction, clustering, and then sampling. Each step will be explained in detail in the following sections.

4.1 ML System for HS Detection

Figure 4.1 shows the different blocks of a general machine learning HS detection system. It is divided into three steps. The first step is data generation, where we have an input training layout in OASIS or GDS format. From this layout, we can obtain $n - by - n$ images of patterns. These patterns are then fed into the training step. In this step, the model is trained to minimize a certain loss function and then it outputs the probability of HS. In the prediction step, we obtain $n - by - n$ images from the test layout and feed it into the trained model. The model then outputs a probability for each pattern and if the probability is greater than 0.5, it is considered as a HS.

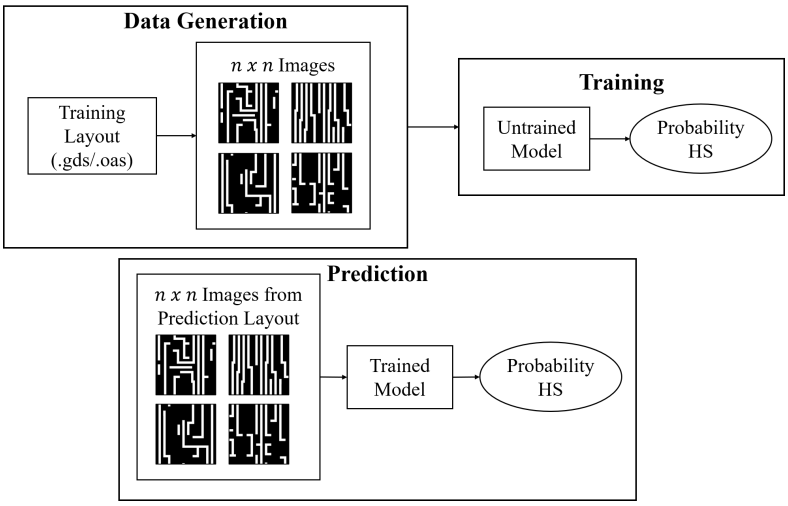


FIGURE 4.1: ML-based Hotspot Detection System

This system will be used in the following sections to describe the proposed sampling approach.

4.2 Proposed Data Sampling Approach

4.2.1 Feature Reduction using Autoencoder

In our ML system the input feature vector has a size of n^2 which can quickly become large, making clustering difficult. Therefore, a dimensionality reduction step is required before clustering of the input patterns can be performed.

An autoencoder-based feature reduction method is proposed. An autoencoder can be trained to reconstruct the input images using a lower dimension feature vector. It learns different relations between the data points and can encode the high dimensional data into a lower dimensional space without a significant loss of information.

Figure 4.2 shows the general autoencoder architecture. The input and output layers have equal sizes. While the layer in the middle has a smaller number of nodes. This layer is latent space where the autoencoder would learn non-linear features that would allow it to reconstruct the input feature vector.

In our proposed flow, we will use an autoencoder with a single layer between the input and output. The size of this layer will be less than the input feature size. This is the simplest autoencoder that can be used. However, it would be shown that this would give us useful features that can be used for clustering. Deeper networks can be used depending on the complexity and size of the input dataset.

The size of the latent layer is determined empirically. In our proposed flow, we will train an autoencoder with a latent size of n^2 until the loss is minimized. Then, another autoencoder is trained with half the latent size. This process repeated for several iterations until the latent size approaches zero. A curve of reconstruction loss against latent size is then plotted. $d_{encoded}$ is selected to minimize the loss. The dimension of each data point is thus reduced from $n^2$to $d_{encoded}$. Each pattern from the training dataset is then encoded using the trained autoencoder. This encoded dataset can then be used in the next step of the flow which is clustering of the patterns using the DBSCAN algorithm [25].

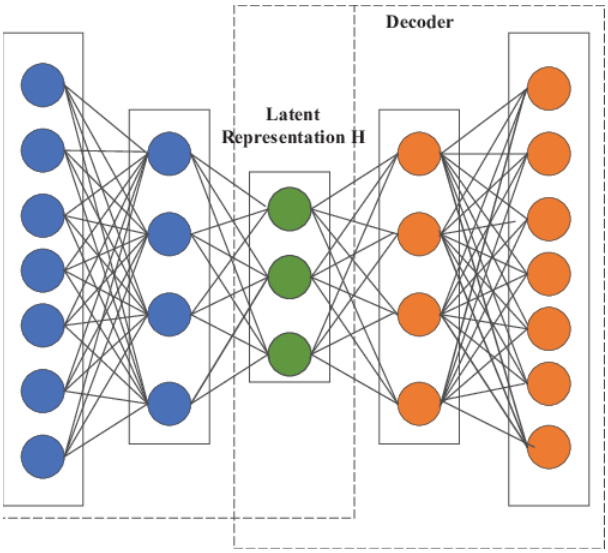


FIGURE 4.2: Autoencoder Architecture

4.2.2 DBSCAN Clustering

The next step of the proposed flow is to reduce the size of the dataset, by clustering the encoded dataset. The DBSCAN algorithm is used [25]. It required two input parameters: radius of neighborhood around a point (ϵ) and the minimum number of points required to form a dense

region ($minPts$). Figure 4.3 shows an illustration of DBSCAN parameters. In the proposed flow, $minPts$ will always be equal to one. This means that any point that does not have neighbors will be assigned a cluster on its own. This is done to simplify the flow where only a single parameter (ϵ) is varied. Clusters which have single points will then be handled in the sampling module of the flow. They can either be included in the sampled data or discarded based on the user's preference. The distance measure used is the Euclidean Distance defined over the autoencoder feature space [62].

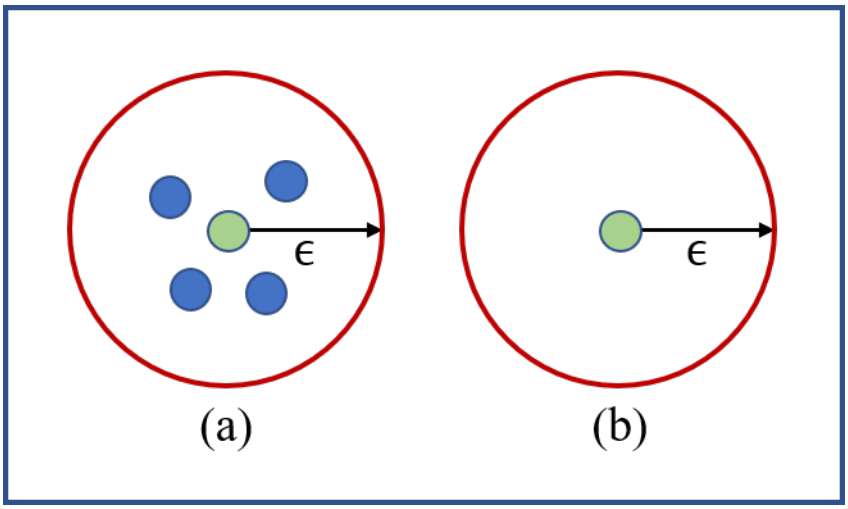


FIGURE 4.3: DBSCAN Illustration where $minPts = 1$. (a) These points form a dense region beacause the number of points is greater than $minPts$ (b) This is also considered as a dense region because $minPts = 1$.

Due to the binary nature of the HS detection dataset, there exists three different types of clusters:

- Clusters containing HS only (pure cluster)
- Clusters containing NHS only (pure cluster)
- Clusters containing both HS and NHS (hybrid cluster)

Each type of cluster will be handled in a different way when performing the sampling step. This will be discussed in the next section.

4.2.3 Sampling from Clustered Data

After clustering is done at a certain value for ϵ, sampling of the data is performed. For this purpose, another parameter is introduced to the flow which is called the sampling percentage per cluster (P). This allows the user to control how much data is sampled from each cluster. The goal of the sampling step is to select patterns that maximize the cluster coverage.

The proposed sampling technique starts by obtaining the median point of the cluster. Next, the furthest point from the median is obtained. Then, the furthest point from the previous two points is obtained. This is done by summing the distances to all points from these two points and then getting the point which has the largest distance. This process is repeated until the required number of points is obtained. This gives good cluster coverage because sample contains a pattern from the center of the cluster (median) and then several patterns at the boundary of the cluster. This method can be used for pure clusters. For hybrid clusters, the algorithm is slightly modified to ensure that hard-to-classify (HTC) patterns are included in the output sample. For each point sampled, the closest pattern with the opposite label is obtained. This enhances the sampling by including HS and NHS that are close to each other to allow the model to differentiate between them.

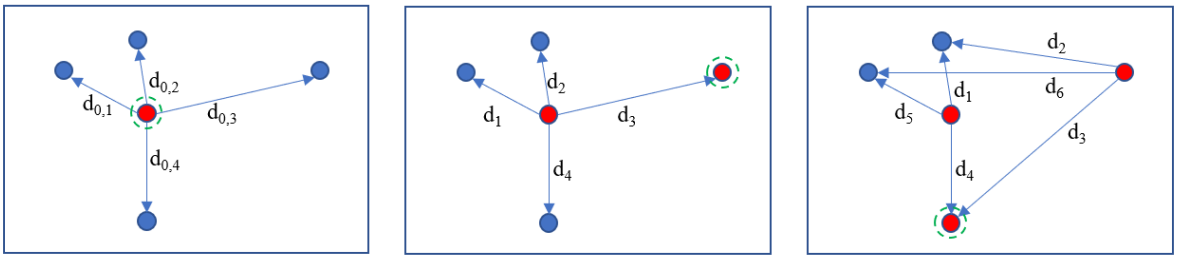


FIGURE 4.4: 2D Illustration of the Sampling Algorithm. (a) Median Selection. (b) and (c) Selection of Next Furthest Point.

Fig. 4.4 shows an illustration of the algorithm using 2D points. The sampling method for pure clusters is explained in Algorithm 2 while the one for hybrid clusters is explained in Algorithm 3.

Algorithm 2 Sampling Pure Clusters

Input: (Required Number of Samples)
Output: (List of Patterns)
- Get the Median Pattern of the Cluster
- Add Median to Output
while there are still unprocessed patterns and the required number of samples is not obtained **do**
    - Get the furthest pattern relative to the currently selected set of patterns so far
    - Add Pattern to Output
end while

4.3 Experimental Procedure and Results

4.3.1 Experiment Description

In this subsection, the experimental procedure used to evaluate the proposed method will be outlined. The first step is to perform feature reduction as described in Section 4.2.1. An autoencoder with a single hidden layer is used. The activation function is the rectified linear unit (relu) [18] and the cost function is binary cross-entropy. To find the size of the encoded features, several autoencoders with varying hidden layer sizes are trained using the training dataset dataset. The size of the hidden layer starts at 3600 and is reduced by half on each iteration. A graph of loss against the hidden layer size is then plotted. An example of this plot is shown in Figure 4.5. It can be seen that the loss starts to increase rapidly after a layer size of 225. By examining the reconstructed images, we have decided to use an encoded feature size ($d_{encoded}$) of 150 to give us a compressed representation at the expense of a slightly larger loss.

The next step is to generated the encoded dataset to be used for clustering. Then, different reduced datasets would then be generated by running DBSCAN and sampling using various combinations of ϵ and P. For each dataset a ML model would be trained for 100 epochs using a 90-10 training-validation split. The model with the highest validation F1 score would then be used to run prediction on the test dataset. In order to have a baseline for comparison, reduced datasets will also be generated using random sampling. Each random sampling experiment would be repeated 50 times and the average statistics would be calculated. Finally, the models trained using the clustered dataset versus random sampling are then compared to evaluate how the two sampling methods affect the HS recall, precision and F1 score.

Algorithm 3 Sampling Hybrid Clusters

Input: (Cluster Size (n), Sampling Percentage (P), HS/NHS ratio in the Training Dataset (hs_nhs_ratio), #HS ($input\_hs\_count$), #NHS ($input\_nhs\_count$))
Output: (List of Patterns)
- Calculate the total number of sampled patterns: $total\_count = P \times n$
- Calculate the number of HS:
  $out\_hs\_count = min(hs\_nhs\_ratio \times total\_count,\ input\_hs\_count)$
- Calculate the number of NHS:
  $out\_nhs\_count = min(total\_count - out\_hs\_count,\ input\_nhs\_count)$
- Get the Median Pattern of the Cluster
- Add Median to Output
- Get the furthest pattern relative to the Median
- Add Pattern to Output

while there are still unprocessed patterns and the required number of samples is not obtained **do**
  - Get the furthest pattern relative to the currently selected set of patterns so far
  - Get the closest pattern with the opposite label
  if out_hs_count is not exceeded **then**
    - Add HS pattern to Output
  end if
  - Add NHS pattern to Output
end while

The model used for the experiments is the Mobilenet CNN model which is known for its efficiency in Mobile Vision applications [63]. In addition, it has been recently used for HS detection on the ICCAD-2012 dataset and has shown good results [38]. The loss function used for training is the Focal Loss which is suited for problems with data imbalance [21]. The optimizer used during training is the Adam Optimizer [64].

The experiments described above will be performed on two datasets: ICCAD'19 and ICCAD'12 datasets. The results will be described in the following subsections.

4.3.2 Results using ICCAD'19 Dataset

Table 4.1 shows how the size of the reduced data as a percentage of the full dataset size varies with ϵ and sampling percentage per cluster. For example, at ϵ = 35 and P = 20, 54.5% of the full dataset is used for training. From the table, it can also be seen that the size of the sampled dataset decreases as ϵ increases and P decreases. Moreover, as ϵ approaches infinity, all the patterns are assigned to the same cluster. Hence, at large values of ϵ, the size of the dataset

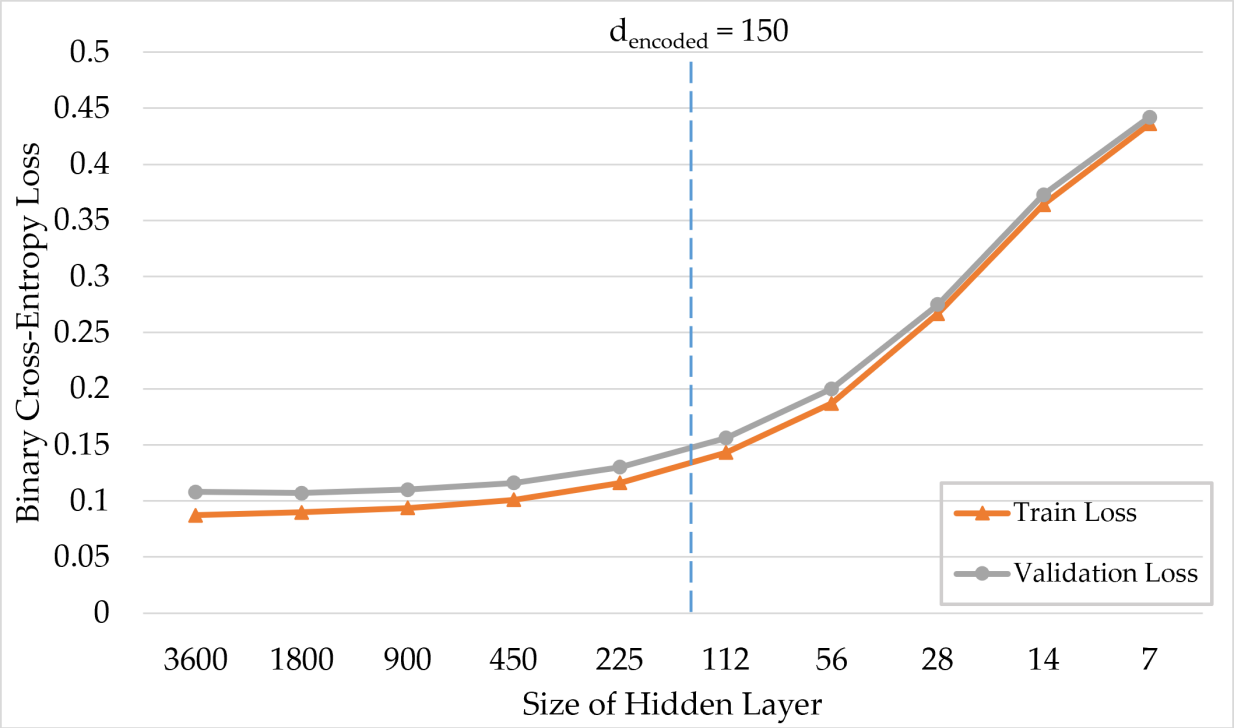


FIGURE 4.5: Plot of Binary-Cross Entropy against Hidden Layer Size of the Autoencoder

approaches the value of sampling percentage. This is observed at $\epsilon = 50$ where the dataset size is very close to the value of P.

An important point to note is that at small values of ϵ, most of the clusters will contain a single pattern. In that case, it was decided to include the pattern output sample. Hence, at small values of ϵ (10-15), the achieved data reduction will be small. Higher reduction can be observed as ϵ increases.

To compare the proposed sampling method against random sampling, Test Set I is examined first. The difference in F1 score between the models trained with clustered data and the ones trained with random data is plotted in a contour plot as shown in Fig. 4.6. The plot indicates an increase in F1 score relative to the random sampling method. The model with the largest positive difference of 5.7% is at (80,50). This corresponds to a dataset size of 81.1% and F1 score of 68.2. The second-best model at (10, 35) has a difference in F1 score of 5.4%. The dataset size is 48.8% and the F1 score is 64.6.

TABLE 4.1: Dataset Size (%) as a Function of ϵ and P

ϵ/P	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
10	93.2	93.8	96.1	96.3	96.7	96.9	97.0	99.4	99.9
15	89.0	90.2	93.4	93.8	94.7	95.0	95.5	98.6	99.8
20	83.7	85.6	89.8	90.6	92.1	92.7	93.5	97.7	99.6
25	76.2	79.1	84.2	85.8	88.1	89.3	91.0	96.1	98.9
30	63.9	68.0	73.9	77.2	80.9	83.8	87.2	93.1	97.2
35	48.8	54.5	61.8	67.0	72.3	77.1	82.2	89.6	95.3
40	32.6	40.2	49.1	56.1	63.2	69.9	77.0	85.9	93.4
45	18.8	27.9	37.6	46.4	55.2	63.9	72.7	82.4	91.5
50	13.5	23.1	33.2	42.7	52.2	61.5	71.0	81.1	90.7

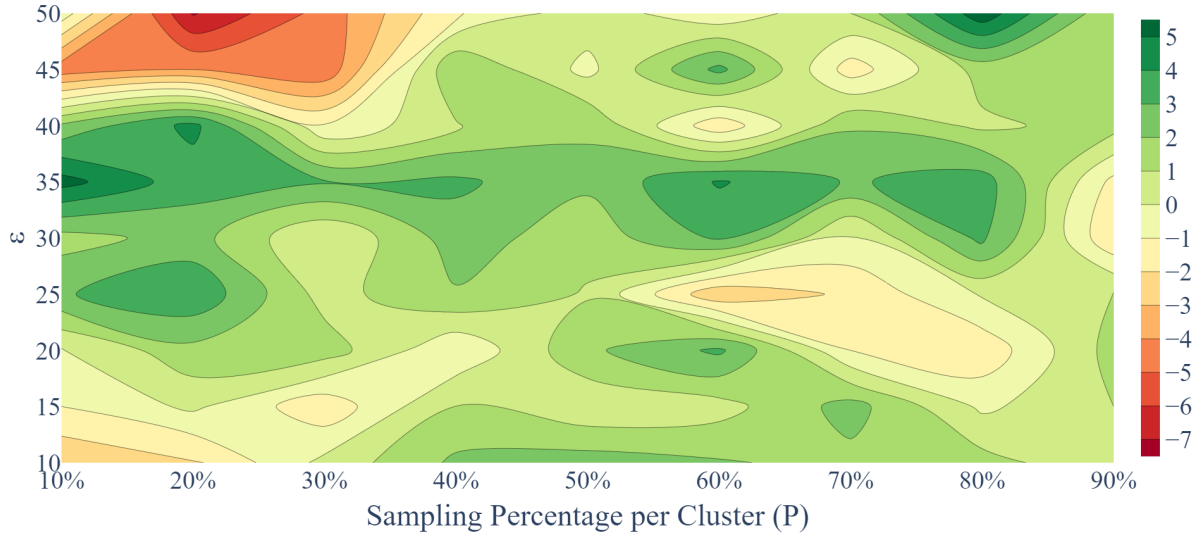


FIGURE 4.6: Plot of the Difference in F1 Score Between Clustering and Random Sampling on Test Set I.

Fig. 4.7 and Fig. 4.8 show the recall and precision values obtained on Test Set I for each combination of ϵ and P. These plots provide a way to choose different models that suit various applications. For example, when the detection of HS is more important (e.g., defect analysis), models having the highest recall can be used such as the one at (90, 25) which has a recall of 87%. For applications where the cost of false alarms is high (e.g., design), then models having high precision can be selected. An example of this is the model at (70, 15) which has the highest precision of about 60%. A trade-off between recall and precision can also be made by maximizing F1 score. Hence, clustering and sampling enable customization of the training data

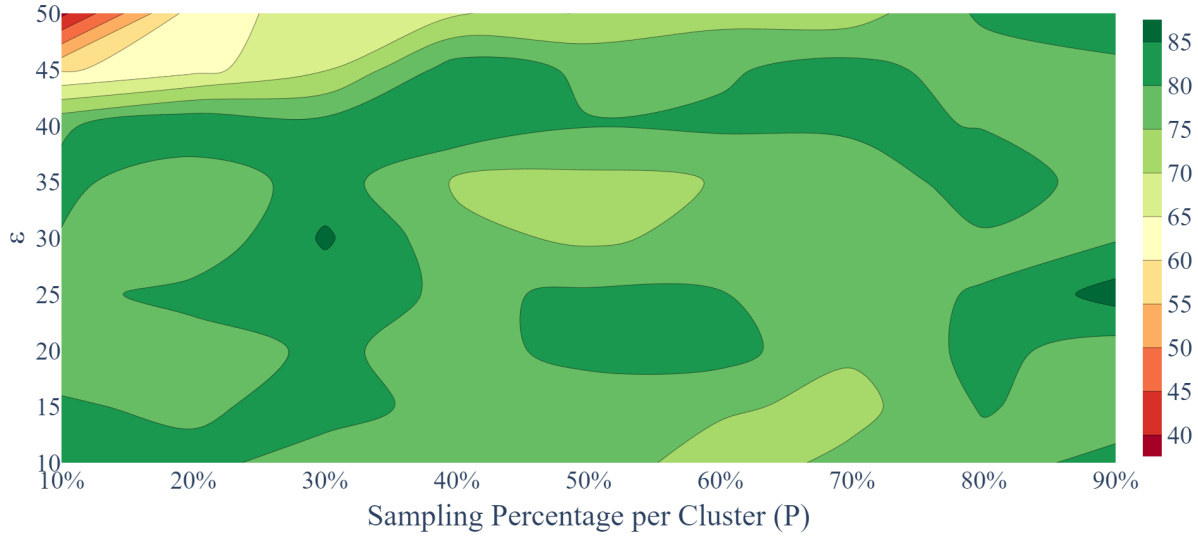


FIGURE 4.7: Recall on Test Set I

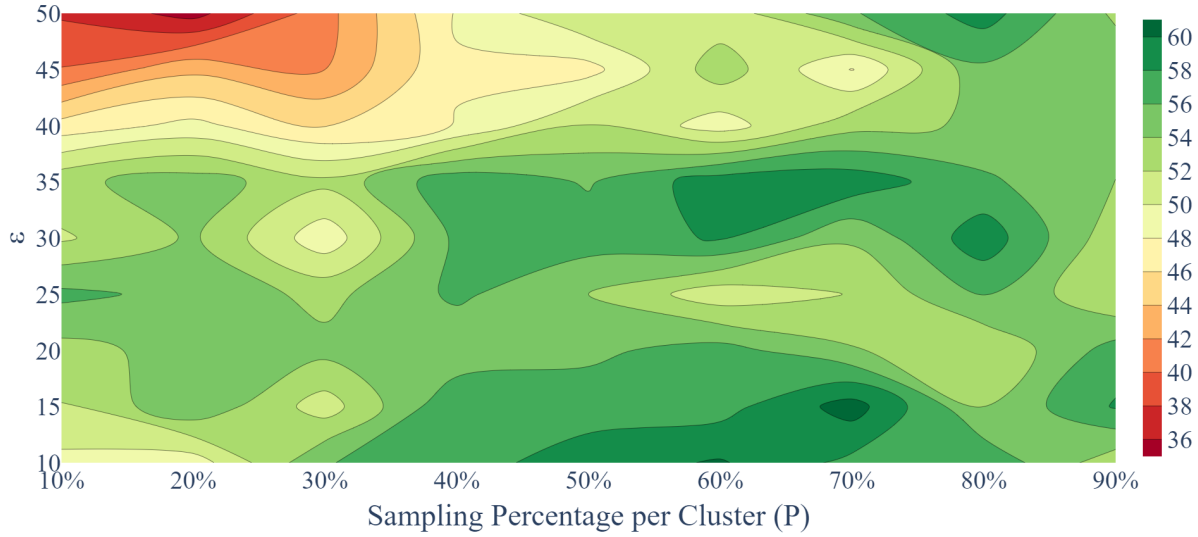


FIGURE 4.8: Precision on Test Set I

to achieve different outcomes.

Tables 4.2 and 4.3 show the time taken for DBSCAN clustering and sampling steps. Note that the sampling step was implemented using a single thread only, so it is possible to obtain a smaller run time by making use of parallelism. The overhead introduced by this step might not be trivial, however, this step is only done once for any given training dataset. It will also

TABLE 4.2: DBSCAN Time as a Function of ϵ Using ICCAD-2019 Training Dataset

ϵ	**Time (s)**
10	4.8
15	5.4
20	6.3
25	7.0
30	7.2
35	8.9
40	9.0
45	9.7
50	10.3

allow us to obtain a reduced dataset that is representative of the full dataset, hence shortening the training cycle significantly as seen in Table 4.4. Training a ML model is an iterative process that requires multiple steps of tuning, so any run time reduction in this cycle would be highly beneficial.

Table 4.4 lists several models trained using different datasets. The first model was trained using the full ICCAD-2019 data set. The following models were trained using data obtained via our proposed approach in addition to the random sampling approach. The datasets for Models 2 and 3 are 20% smaller than the full dataset. This reduces the training time to around 66% of the full dataset time. For models 4 and 5, the dataset size is more than 50% smaller and the training time is reduced to 33%. Model 2 achieves a higher F1 score than the baseline model, while model 4 almost achieves the same F1 score with half the dataset size. When examining models 3 and 5, the same data reduction is obtained but the F1 score is lower than the baseline. This shows that the clustering and sampling steps result in choosing better representatives of the training dataset, thus, reducing the time needed for training and improving the data balance.

For Test Set II, the difference in F1 score between clustering and random sampling is not significant as shown in the contour plot in Fig. 4.9. This proves that the lack of coverage in the training dataset is the dominating factor, hence our proposed method of data selection will not influence the final trained model. Therefore, data enrichment of the training dataset is needed to improve the model and show the value of dataset selection. Synthetic pattern generation can be used to enrich the training dataset. In [65], a new method was introduced where subtle

TABLE 4.3: Sampling Time (s) as a Function of ϵ and P

ϵ/P	10%	20%	30%	40%	50%	60%	70%	80%	90%
10	4	14	53	53	53	53	53	13	4
15	5	24	77	77	77	77	77	24	5
20	8	35	103	103	103	103	103	35	8
25	10	45	117	117	117	117	117	45	10
30	17	45	103	105	109	110	113	58	35
35	21	52	113	124	136	147	161	124	116
40	16	45	142	177	204	236	258	242	247
45	18	43	82	105	118	140	162	174	385
50	20	42	78	95	127	140	177	184	211

TABLE 4.4: Comparison of Mobilenet Models Trained with Different Sampled Datasets using ICCAD-2019 Test Set I

#	Description	Dataset Size	Relative Training Time	Recall	Precision	F1
1	Baseline	100%	100%	86.7	52.2	65.2
2	P = 80, ϵ = 50	81.10%	66.67%	80.9	58.9	68.2
3	Random Sampling	81.10%	66.67%	80.4	51.2	62.5
4	P = 10, ϵ = 35	48.80%	33.33%	81.3	53.5	64.6
5	Random Sampling	48.80%	33.33%	84.8	45.4	59.2

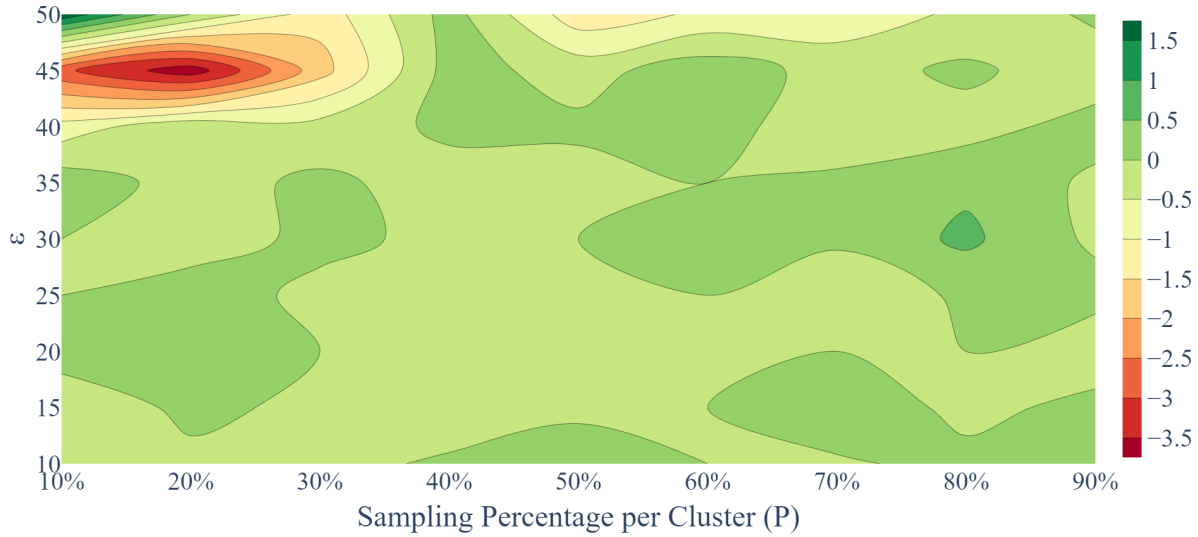


FIGURE 4.9: Plot of the Difference in F1 Score Between Clustering and Random Sampling on Test Set II

differences between patterns were generated by randomly moving pattern edges. Exploring similar techniques is needed to create richer datasets for ML-based hotspot detection models.

4.3.3 Results using ICCAD'12 Dataset

TABLE 4.5: Dataset Size (%) as a Function of ϵ and P

ϵ/P	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
10	93.2	93.7	96.2	96.3	96.3	96.8	97.0	99.4	99.9
15	89.0	90.1	93.6	93.9	94.0	95.0	95.3	98.7	99.8
20	83.7	85.5	90.1	90.7	91.0	92.6	93.2	97.8	99.6
25	76.8	79.5	85.1	86.3	87.1	89.6	90.8	96.4	99.1
30	67.1	71.0	77.4	79.8	81.6	85.3	87.7	94.1	98.0
35	52.3	57.6	65.0	69.4	73.5	78.7	83.1	90.5	95.8
40	36.6	43.8	52.6	58.9	64.8	71.7	78.0	86.8	94.0
45	22.2	31.1	40.8	48.9	56.7	65.5	73.5	83.2	92.1
50	15.3	24.8	34.9	43.9	52.9	62.3	71.4	81.5	91.0

Table 4.5 shows the dataset size as a function of ϵ and P for the ICCAD'12 dataset. We can see a similar behavior to the ICCAD'19 dataset. The dataset size decreases as ϵ increases and P decreases. We can also see that the size of the dataset approaches P as ϵ tends to infinity.

Figure 4.10 shows the difference in F1 score between models trained using clustered data and the ones trained with random data. Unlike the ICCAD'19 results, we can see no significant difference between the two methods. This can be explained by the fact that the ICCAD'12 training and test datasets are very similar as described in [10]. It has been shown that the training and test patterns are close to each other when examining the first three PCA components. Hence, random sampling was enough to obtain a representative training sample. From Table 4.6, it can be seen that even at high reduction ratios using random sampling, the F1 score of the model is greater than 90% for most cases. This shows that the data coverage is good enough, so clustering would not show further improvement.

TABLE 4.6: F1 Score of the Model using Random Sampling on the ICCAD'12 Dataset

ϵ/P	10%	20%	30%	40%	50%	60%	70%	80%	90%
10	98.4	98.5	98.5	98.6	98.4	98.5	98.7	98.5	98.3
15	98.2	98.5	98.1	98.5	98.6	98.3	98.4	98.7	98.5
20	98.7	98.5	98.4	98.1	98.6	98.2	98.4	98.5	98.5
25	98.2	98.2	98.5	98.5	98.4	98.4	98.3	98.4	98.6
30	98.3	98	98.3	98.3	98.4	98.5	98.5	98.3	98.4
35	98.1	97.7	98.1	98.2	98.1	98.1	98	98.4	98.4
40	96.4	96.9	97.5	97.8	97.8	97.8	98.2	98.5	98.5
45	92.7	93.7	95.5	97	97.2	97.1	98	98.3	98.4
50	69.6	87.9	93.1	94.5	95.8	96.5	97.7	97.9	98

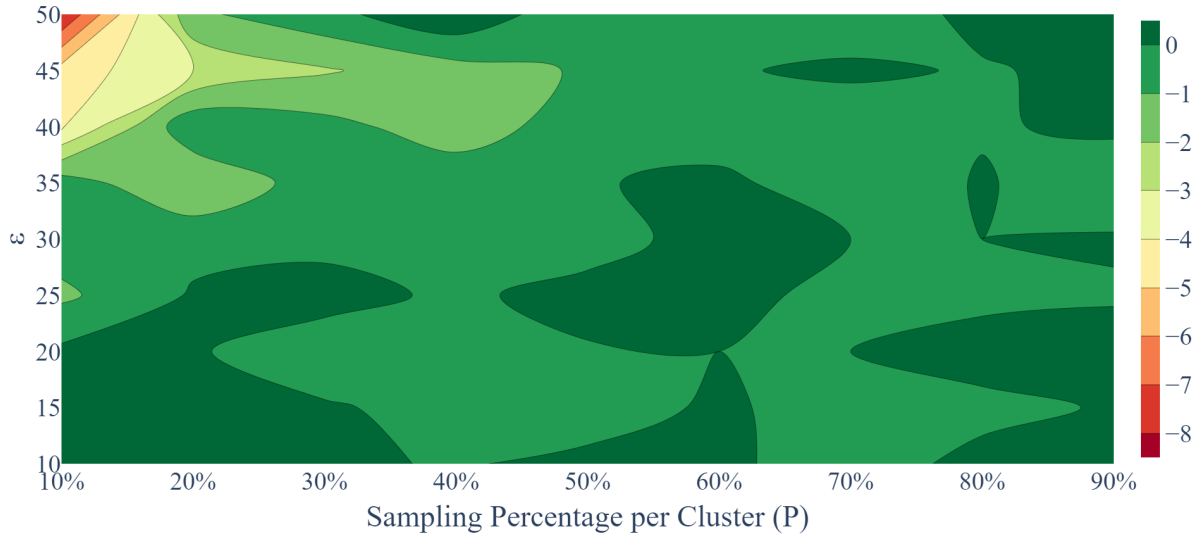


FIGURE 4.10: Plot of the Difference in F1 Score Between Clustering and Random Sampling on the ICCAD'12 Test Set

4.4 Conclusion

Based on the results described in the previous sections, we have seen that clustering and sampling can reduce the dataset size while maintaining or improving the F1 score of the model. This shortens the training cycle and enables training using a representative dataset when the full dataset is too large for training in a feasible amount of time. For datasets such as the ICCAD'12 where random sampling does not show deteriorating results, clustering would not give additional benefits in terms of model metrics. The proposed clustering and sampling

method provides two parameters (ϵ and P) which can be varied to tune the performance of the model to cater to the needs of different applications. For example, if the application gives higher importance to recall, sampling can help maximize the recall of the model. The same is also true for precision. A trade-off can also be made by maximizing F1 score. This fine tuning cannot be done via random sampling which shows the benefit and strength of our proposed method.

Chapter 5

ML Hotspot Model Enhancement

In Chapter 4, we discussed a data sampling technique based on autoencoder features and DBSCAN clustering. In this chapter, we extend our work on data sampling and explore other options for improving ML-based hotspot detection models. Two methods will be explored:

- Using the previously generated autoencoder features for training HS detection models. The benefit of this method is that the latent feature vectors have a smaller number of dimensions which would make the models simpler. We will also explore whether this can improve the model performance in terms of recall, precision and F1 score.
- In addition to density-based features, image gradients will be used as a descriptor which can help maintain the spatial information of the pattern image.

For each method, hyperparameter optimization would be done to choose the best model architecture and training parameters in an attempt to maximize the F1 score of the model. A fully connected neural network will be used and compared to another model from the literature that uses raw image features for HS detection [8]. The models will be compared for their performance on the test set and the number of trainable parameters which represent the model complexity. For hyperparameter tuning, the popular Optuna framework will be used [66]. These experiments will be done using the ICCAD-2019 dataset [10].

5.1 HS Detection using Autoencoder Features

5.1.1 Experimental Procedure

In this section, the encoded features that were used for pattern clustering in Chapter 4 will be used as an input to train a hotspot detection model. These features are a compressed representation of each pattern and can be used to reconstruct it. Using these features to train a hotspot classifier could enable us to build simpler models with fewer trainable parameters. Moreover, the effect of using these features on the model performance metrics should be analyzed.

Based on the autoencoder loss in Figure 4.5, it can be seen that the loss value is almost constant from 3600 features to 900 features and then starts increasing gradually. From this observation, two experiments would be performed. The first one would use 900 encoded features in an attempt to use the smallest number of features without significant loss of information. The second experiment would use 300 features which is the point at which the loss starts to increase. Two fully connected neural networks would be trained using these features and then compared to the model in [8] which uses raw images. The model used for comparison is a fully connected neural network that consists of 4 hidden layers. The sizes of the layers are: 196-196-144-144. The input to the network is a 60x60 pixels flattened image (3600 features).

As mentioned earlier, Optuna framework will be used for hyperparameter optimization [66]. Table 5.1 lists the different hyperparameters considered for optimization and the range of values for each parameter. $d_{encoded}$ represents the size of the encoded input features.

TABLE 5.1: List of Hyperparameters Considered for Optimization

Hyperparameter	**Range**
Batch Size	128, 256, 512, 1024, 2048
Optimizer	RMSprop, Adam, SGD
Learning Rate	[1e-5, 1e-1]
Number of Hidden Layers	[1,4]
Number of Nodes for Layer 1	[$d_{encoded}$, $d_{encoded}/2$]
Number of Nodes for Layer 2	[$d_{encoded}/2$, $d_{encoded}/4$]
Number of Nodes for Layer 3	[$d_{encoded}/4$, $d_{encoded}/8$]
Number of Nodes for Layer 4	[$d_{encoded}/8$, $d_{encoded}/16$]
Weight Decay (L2 Regularization)	[1e-10, 1e-3]

As for the loss function for training, the Sigmoid Focal Loss would be used. In addition, the initialization method described in [21] is also used so that the starting loss is dominated by the hotspot patterns which can help improve the convergence of the model. The training would be performed with the ICCAD-2019 dataset and 10% of the data would be used for validation.

5.1.2 Experimental Results

Tables 5.2, 5.3, 5.4, and 5.5 summarize the results of experiments using the encoded features. Different values for α and γ for the Focal Loss function were used to tune the F1 score of the model. Note that the F1 Scores on Test Set II in Tables 5.3 and 5.5 are very similar to the baseline model. This confirms the findings in [10], where the authors have concluded that in order to improve the F1 Score, the training dataset should be enriched with more patterns so that the precision can be improved. Hence, for all of our comparisons we will only focus on Test Set I (Tables 5.2 and 5.4).

TABLE 5.2: Results of Models Trained Using 300 Encoded Features on ICCAD'19 Test Set I

Model	α	γ	Trial	Number of Layers	Nodes per Layer	Batch Size	Optimizer	Number of Parameters	Recall	Precision	F1
Baseline [8]	N/A	N/A	N/A	4	196-196-144-144	256	Adam	793k	74.2	56.2	63.9
1	0.6	2	97	3	219-83-70	128	Adam	90k	79.1	40.6	53.7
2	0.6	3	17	3	258-93-41	512	Adam	106k	67.9	47.9	56.2
3	0.7	2	83	4	217-77-34-25	128	Adam	86k	77.7	42.7	55.1
4	0.7	3	98	4	168-98-32-23	128	Adam	71k	73.8	44.2	55.3

TABLE 5.3: Results of Models Trained Using 300 Encoded Features on ICCAD'19 Test Set II

Model	α	γ	Trial	Number of Layers	Nodes per Layer	Batch Size	Optimizer	Number of Parameters	Recall	Precision	F1
Baseline [8]	N/A	N/A	N/A	4	196-196-144-144	256	Adam	793k	98.1	49.8	66.1
1	0.6	2	97	3	219-83-70	128	Adam	90k	97.2	50.7	66.6
2	0.6	3	17	3	258-93-41	512	Adam	106k	91	51.3	65.6
3	0.7	2	83	4	217-77-34-25	128	Adam	86k	97.4	50	66
4	0.7	3	98	4	168-98-32-23	128	Adam	71k	95.9	50.4	66.1

Table 5.2 shows the results of models using 300 encoded features. For each model, the values of α and γ of the Sigmoid Focal Loss is shown. In addition, the architecture is outlined in terms of the number of hidden layers, number of nodes per layer, and the number of trainable parameters. Moreover, the batch size and optimizer that were selected by Optuna are also listed. Finally, the performance metrics on the validation and Test Sets are shown. The four best

TABLE 5.4: Results of Models Trained Using 900 Encoded Features on ICCAD'19 Test Set I

Model	α	γ	Trial	Number of Layers	Nodes per Layer	Batch Size	Optimizer	Number of Parameters	Recall	Precision	F1
Baseline [8]	N/A	N/A	N/A	4	196-196-144-144	256	Adam	793k	74.2	56.2	63.9
1	0.55	1	68	4	400-332-143-93	128	Adam	554k	66.5	52.8	58.9
2	0.55	1.5	41	3	544-358-114	128	Adam	726k	71	45.7	55.6
3	0.55	2	47	4	423-237-158-91	512	Adam	534k	73.2	46.6	56.9
4	0.6	2	44	3	451-341-117	2048	Adam	601k	77.8	44.9	56.9
5	0.6	3	34	4	766-250-147-81	128	Adam	930k	77	44.9	56.7

TABLE 5.5: Results of Models Trained Using 900 Encoded Features on ICCAD'19 Test Set II

Model	α	γ	Trial	Number of Layers	Nodes per Layer	Batch Size	Optimizer	Number of Parameters	Recall	Precision	F1
Baseline [8]	N/A	N/A	N/A	4	196-196-144-144	256	Adam	793k	98.1	49.8	66.1
1	0.55	1	68	4	400-332-143-93	128	Adam	554k	91.7	51.3	65.8
2	0.55	1.5	41	3	544-358-114	128	Adam	726k	92.5	51.3	66
3	0.55	2	47	4	423-237-158-91	512	Adam	534k	96.1	51	66.7
4	0.6	2	44	3	451-341-117	2048	Adam	601k	97.7	50.6	66.7
5	0.6	3	34	4	766-250-147-81	128	Adam	930k	96.6	50.8	66.6

models from our experiments are shown in the table. Models 1 and 2 have 3 hidden layers with sizes: 219-83-70, and 258-93-41 respectively. Models 3 and 4 have 4 hidden layers with sizes: 217-77-34-25, and 168-98-32-23 respectively.

By examining Table 5.2, the following observations can be made:

- The number of trainable parameters for Models 1-4 is much smaller than the baseline. The smallest model was about 11x smaller than the baseline. This is because the size of the input features was reduced from 3600 to 300.
- For Models 1-4, the precision on Test Set 1 is around 10% lower than the baseline while recall was comparable to the baseline. This low precision caused the F1 score to drop below 60% which shows that these features have some information loss.
- Even though the encoded features helped reduced the size of the model, the performance was not improved. This set of features is not suitable enough to be used for HS detection.

Table 5.4 shows the results of models using 900 encoded features. The fives best models from our experiments are shown. Models 1, 3, and 5 have 4 hidden layers with sizes: 400-332-143-93, 423-237-158-91, and 766-250-147-81 respectively. Models 2 and 4 have 3 hidden layers with sizes: 544-358-114, and 451-341-117 respectively.

By examining Table 5.4, the following observations can be made:

- The number of trainable parameters for Models 1-4 is at the same order of magnitude as the baseline model.
- Models 1-5 have slightly higher precision than the ones which used 300 features, while recall was similar. This has caused the F1 score to increase slightly but it did not reach the levels of the baseline.

Overall, even though the encoded features have allowed smaller models to be built, these models were not able to achieve the F1 scores of the baseline model. Hence, it can be concluded that these features cannot be used for HS detection. However, they are still suitable for pattern clustering purposes as we have seen in Chapter 4.

5.2 HS Detection using Image Gradients

5.2.1 Introduction

In this section, image gradients based on the Sobel filter [58] will be used as input features to train ML-based hotspot detection models. Gradient vectors have two components: magnitude and angle. The magnitude gives the rate of change and the angle shows the direction of the largest change. When using pixel values as input to fully-connected DNNs, the network tries to learn correlations between the individual pixels to determine the features that cause hotspots. However, using a fully-connected network causes the number of parameters to increase rapidly. In addition, the flattened input to the network causes loss of spatial information, therefore image gradients would provide a better way of describing the patterns by including this spatial information in the form of gradient vectors.

To test this method, we use 60x60 images to represent the patterns. Then, the image is divided into $m - by - m$ windows and for each window the average pixel density is calculated. In addition, the Sobel filter is applied in the x and y directions for each window giving us two gradient matrices g_x and g_y. Then, the overall magnitude and direction are calculated using Equations 5.1 and 5.2. In Equation 5.2, θ is in degrees between 0 and 360. For each window,

we now have the density as well as the gradient magnitude and angle for each pixel. This information will be used to generate different feature representations for the input patterns.

$$|g| = \sqrt{g_x^2 + g_y^2} \tag{5.1}$$

$$\theta = atan2(\frac{g_y}{g_x}) \quad where \quad 0° \leq \theta < 360° \tag{5.2}$$

5.2.2 Experimental Procedure

Four experiments will be performed with different input feature configurations:

1. Window size = 3x3. For each window, 2 features would be calculated: average density, orientation of the middle point of the window.

2. Window size = 5x5. For each window, 2 features would be calculated: average density, orientation of the middle point of the window.

3. Window size = 5x5. For each window, 2 features would be calculated: average density and the orientation of the middle point of the window. Then, four 3x3 sub-windows are created and the orientation of the middle point of each sub-window is added to the features. This results in a total of 6 features for each window.

4. Window size = 5x5. As in the previous configuration, four 3x3 sub-windows are created. For each sub-window, the orientation of the middle point of the window and the average density are added to the features. Then, the orientation of the middle point of the 5x5 window is added. This results in a total of 9 features for each window.

Table 5.6 gives a summary of the inputs of each experiment. Figures 5.1, 5.2, 5.3, and 5.4, show the different features that are generated from each window. $D_{i,j}$ and $\theta_{i,j}$ are the density and orientation at row i and column j. The red and green rectangles shows the regions where average density is calculated. The blue circles show the orientations used as input features.

TABLE 5.6: Summary of Image Gradient Experiments Using 60x60 Pattern Images

#	Window Size	Number of Windows	Features per Window	Total Number of Features
1	3x3	400	2	800
2	5x5	144	2	288
3	5x5	144	6	864
4	5x5	144	9	1296

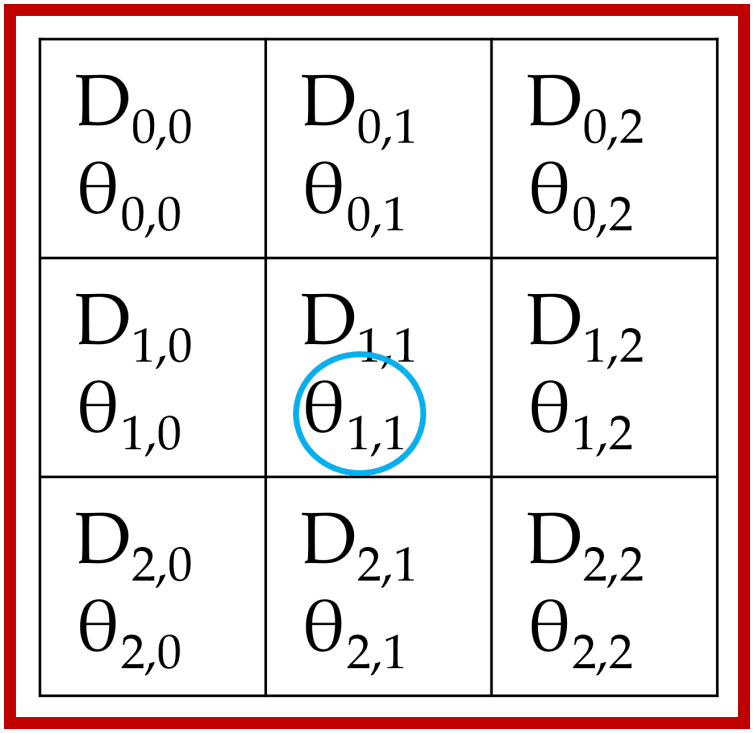


FIGURE 5.1: Experiment 1: The average density is calculated for the whole window and the orientation of the middle pixel is used.

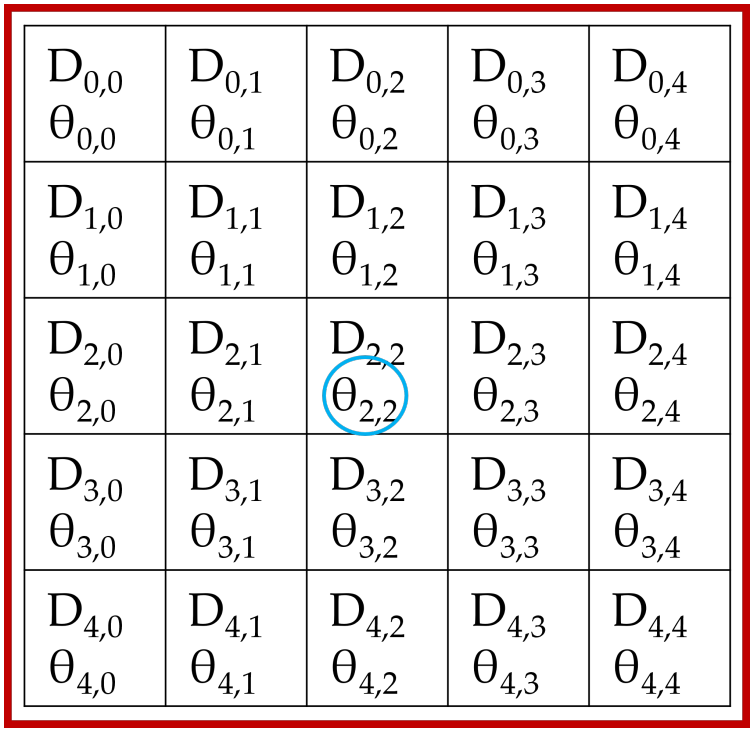


FIGURE 5.2: Experiment 2: The average density is calculated for the whole window and the orientation of the middle pixel is used.

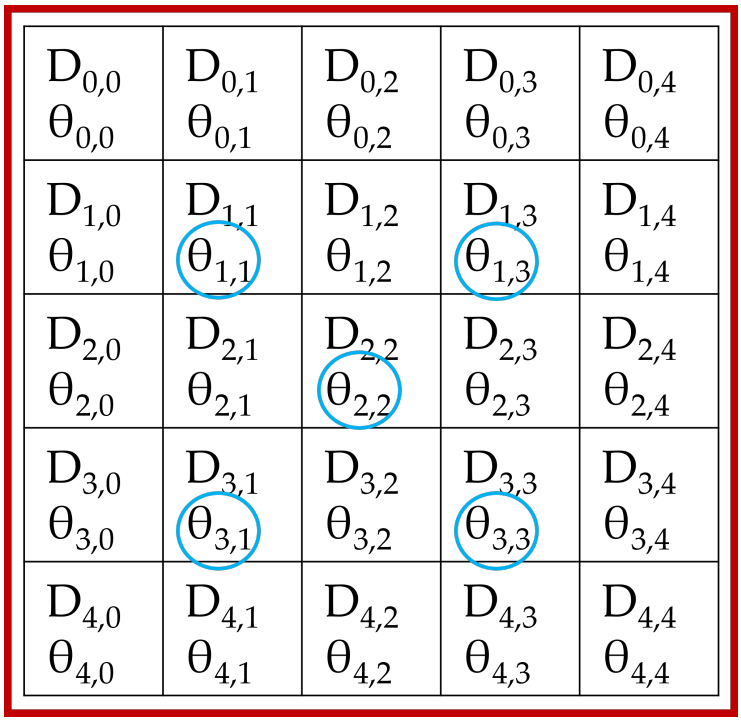


FIGURE 5.3: Experiment 3: The average density is calculated for the whole window and the orientations of the pixels circled in blue are used.

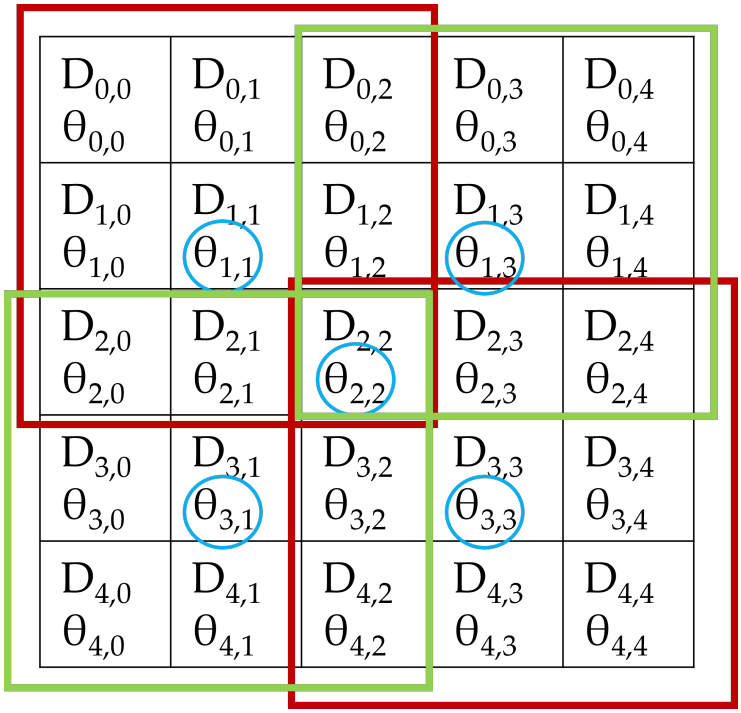


FIGURE 5.4: Experiment 4: Four average densities are calculated for the sub-windows shown in red and green. Five orientations are used as indicated by the blue circles.

Similar to Section 5.1, Optuna would be used for hyperparameter optimization [66]. The loss function used is also the Sigmoid Focal Loss. Different values for α and γ were used in order to tune the F1 score for each model.

5.2.3 Results

Table 5.7 and 5.8 show the results of the best models using different input configurations. Models 1-4 have a single hidden layer with sizes: 156, 701, 538, and 738 respectively, while Model 5 has 2 hidden layers with sizes: 505-322.

By examining Table 5.7, the following observations can be made:

- Models that use image gradients as input features show that a smaller network depth is needed. This is seen in Models 1-4 which only consist of a single hidden layer, and Model 5 which consists of two hidden layers. On the contrary, the baseline model which uses raw images consists of four hidden layers.

TABLE 5.7: Results of Using Image Gradients on ICCAD'19 Test Set I

Model	Number of Features	α	γ	Trial	Number of Layers	Nodes per Layer	Batch Size	Optimizer	Number of Parameters	Recall	Precision	F1
Baseline [8]	3600	N/A	N/A	N/A	4	196-196-144-144	256	Adam	793k	74.2	56.2	63.9
1	288 (5x5)	0.6	2	99	1	156	1024	Adam	45k	71.2	53.5	61.1
2	800 (3x3)	0.6	2	47	1	701	2048	Adam	560k	73.3	55.5	63.2
3	864 (5x5)	0.6	2	73	1	538	256	Adam	465k	71.6	56.5	63.1
4	864 (5x5)	0.6	3	93	1	738	128	Adam	639k	66.8	61.9	64.3
5	1296 (5x5)	0.6	1	59	2	505-322	128	Adam	818k	73.3	54.9	62.8

TABLE 5.8: Results of Using Image Gradients on ICCAD'19 Test Set II

Model	Number of Features	α	γ	Trial	Number of Layers	Nodes per Layer	Batch Size	Optimizer	Number of Parameters	Recall	Precision	F1
Baseline [8]	3600	N/A	N/A	N/A	4	196-196-144-144	256	Adam	793k	98.1	49.8	66.1
1	288 (5x5)	0.6	2	99	1	156	1024	Adam	45k	94	50.9	66
2	800 (3x3)	0.6	2	47	1	701	2048	Adam	560k	97.5	50.4	66.5
3	864 (5x5)	0.6	2	73	1	538	256	Adam	465k	96	51.3	66.9
4	864 (5x5)	0.6	3	93	1	738	128	Adam	639k	95.7	51	66.6
5	1296 (5x5)	0.6	1	59	2	505-322	128	Adam	818k	96.7	51	66.8

- The model size, which is represented by the number of trainable parameters, can be significantly reduced when using image gradients. This is seen in Model 1 where the number of parameters was reduced from 793k to 45k, and the F1 score on Test Set 1 only decreased by about 3%.

- Model 2 which uses a 3x3 window performs as good as the baseline model. However, it uses a much smaller feature size (4.5x smaller) which would speed up the training time significantly and can allow more data to be used. Model 3 also shows a similar behavior where the feature size is around 4x smaller.

- Models 3,4, and 5 used larger windows and more features as attempts to improve the model results. However, the final results on Test Sets I are similar to the baseline model. This indicates that more training data is needed which aligns with the findings in [10].

From Table 5.8, it can be seen that the results on Test Set II for all models are almost identical to the baseline. This also confirms the same finding in [10] that data enrichment is needed to improve the precision on Test Set II.

5.3 Conclusion

In this Chapter, two methods were explored to improve ML-based hotspot detection models. The first method made use of autoencoder features. The following outcomes were found:

- The autoencoder features were useful to reduce the size of the model.
- The encoded features were not enough to obtain a good model. The precision was negatively affected.
- Autoencoder features are suitable for pattern clustering purposes. For hotspot detection, better performance is obtained when the neural network learns the features from raw images.

The second method made use of pixel-based features in addition to image gradients. The following outcomes were found:

- Model size can be significantly reduced by augmenting pixel values with image gradients.
- Pixel densities and image gradients can obtain the same performance as models that use raw images, however, this is done with a fewer number of input features which can ease the training process.
- This method did not improve the results on the ICCAD-2019 Test Sets. More training data is required and this can be done by exploring various data enrichment techniques.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we have explored the problem of lithography hotspot detection. It is a critical step that needs to be performed to ensure high yield. We have also discussed various methods for hotspot detection like lithography simulation, pattern matching, and machine-learning based methods. As integrated circuits became larger and more complex, traditional HS detection methods became difficult. The community has now steered towards ML based methods.

There are two main issues that occur while training ML-based HS detection models:

- Training datasets have millions of patterns and it is difficult to use all of the data for training due to computational and memory limitations.
- Data imbalance between HS and NHS patterns makes training more difficult since models can tend to predict the majority class or suffer from high false alarm rate.

We have introduced a data sampling method to tackle these issues based on using autoencoder features and the DBSCAN algorithm. We have shown that this method can reduce the dataset size while maintaining the model performance metrics.

Moreover, we have explored the two methods for enhancing HS detection models. The first method made use of the encoded features to train ML models. However, the performance was not satisfactory. The second method made use of image gradients to provide additional spatial information to the ML models. This experiment showed that the model size can be significantly

reduced while maintaining performance. In addition, the input feature size was reduced which can speed up the training phase.

6.2 Future Work

As part of future work, the proposed methods in Chapters 4 and 5 should be tested on larger benchmark datasets. The difference between random sampling and clustering methods should become more apparent with a larger dataset. In addition, feature engineering using image gradients can be further explored by using larger window sizes and more features.

Another problem that is worth exploring is the idea of data enrichment. In some cases, even though the training data has a larger number of patterns, the variations might not be enough to train a model that can generalize well. In this case, the training dataset would require enrichment to improve results. One of the possible ways of doing this is to generate DRC-clean patterns from existing patterns by introducing random shifts in the pattern edges [65]. Another way is to generate random patterns based on a set of defined unit patterns while making sure that the generated patterns follow the design rules [67].

Bibliography

[1] C. Mack. *Fundamental Principles of Optical Lithography: The Science of Microfabrication*. Wiley, 2008. ISBN: 9780470723869. URL: https://books.google.de/books?id=nHm4e7rFNfgC.

[2] Vadim Borisov and Jürgen Scheible. "Research on data augmentation for lithography hotspot detection using deep learning". In: *34th European Mask and Lithography Conference*. Ed. by Uwe F.W. Behringer and Jo Finders. Vol. 10775. International Society for Optics and Photonics. SPIE, 2018, pp. 204 –209. DOI: 10.1117/12.2326563. URL: https://doi.org/10.1117/12.2326563.

[3] Gordon E. Moore. "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (2006), pp. 33–35. DOI: 10.1109/N-SSC.2006.4785860.

[4] F.H. Dill. "Optical lithography". In: *IEEE Transactions on Electron Devices* 22.7 (1975), pp. 440–444. DOI: 10.1109/T-ED.1975.18158.

[5] Kai Peter and Reinhard März. *Quantifying returns on litho-friendly design*. Aug. 2011. URL: https://www.techdesignforums.com/practice/technique/quantifying-returns-on-litho-friendly-design/ (visited on 01/01/2022).

[6] Jie Yang et al. "DRCPlus in a router: automatic elimination of lithography hotspots using 2D pattern detection and correction". In: *Design for Manufacturability through Design-Process Integration IV*. Ed. by Michael L. Rieger and Joerg Thiele. Vol. 7641. International Society for Optics and Photonics. SPIE, 2010, pp. 205 –211. DOI: 10.1117/12.846650. URL: https://doi.org/10.1117/12.846650.

[7] Wan-Yu Wen et al. "A Fuzzy-Matching Model With Grid Reduction for Lithography Hotspot Detection". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.11 (2014), pp. 1671–1680. DOI: 10.1109/TCAD.2014.2351273.

[8] Tetsuaki Matsunawa, Shigeki Nojima, and Toshiya Kotani. "Automatic layout feature extraction for lithography hotspot detection based on deep neural network". en. In: ed. by Luigi Capodieci and Jason P. Cain. San Jose, California, United States, Mar. 2016, 97810H. DOI: 10.1117/12.2217746. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2217746 (visited on 09/01/2021).

[9] Aliaa Kabeel et al. "Machine Learning using retarget data to improve accuracy of fast lithographic hotspot detection". In: *Design-Process-Technology Co-optimization for Manufacturability XIV*. Ed. by Chi-Min Yuan. Vol. 11328. International Society for Optics and Photonics. SPIE, 2020, pp. 1 –7. DOI: 10.1117/12.2551827. URL: https://doi.org/10.1117/12.2551827.

[10] Gaurav Rajavendra Reddy, Kareem Madkour, and Yiorgos Makris. "Machine Learning-Based Hotspot Detection: Fallacies, Pitfalls and Marching Orders". en. In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Westminster, CO, USA: IEEE, Nov. 2019, pp. 1–8. ISBN: 978-1-72812-350-9. DOI: 10.1109/ICCAD45719.2019.8942128. URL: https://ieeexplore.ieee.org/document/8942128/ (visited on 09/01/2021).

[11] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[12] Arthur L. Samuel. "Some studies in machine learning using the game of Checkers". In: *IBM JOURNAL OF RESEARCH AND DEVELOPMENT* (1959), pp. 71–105.

[13] Charu C. Aggarwal and Chandan K. Reddy, eds. *Data Clustering: Algorithms and Applications*. en. 1st ed. Chapman and Hall/CRC, Sept. 2018. ISBN: 978-1-315-37351-5. DOI: 10.1201/9781315373515. URL: https://www.taylorfrancis.com/books/9781315362786 (visited on 11/13/2021).

[14] Ian Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. New York: Springer-Verlag, 2002. ISBN: 978-0-387-95442-4. DOI: 10.1007/b98835. URL: http://link.springer.com/10.1007/b98835 (visited on 11/12/2021).

[15] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4 (May 1996), pp. 237–285. ISSN: 1076-9757. DOI: 10.1613/jair.301. URL: https://www.jair.org/index.php/jair/article/view/10166 (visited on 11/13/2021).

[17] Jun Han and Claudio Moraga. "The influence of the sigmoid function parameters on the speed of backpropagation learning". In: *From Natural to Artificial Neural Computation*. Ed. by José Mira and Francisco Sandoval. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 195–201. ISBN: 978-3-540-49288-7.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[19] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[21] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". en. In: *arXiv:1708.02002 [cs]* (Feb. 2018). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002 (visited on 09/01/2021).

[22] Geoffrey E. Hinton and Richard S. Zemel. "Autoencoders, Minimum Description Length and Helmholtz Free Energy". In: *Proceedings of the 6th International Conference on Neural*

Information Processing Systems. NIPS'93. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 3–10.

[23] Anh Reynolds. *Convolutional Neural Networks (CNNs)*. English. 2019. URL: https://anhreynolds.com/blogs/cnn.html (visited on 01/11/2021).

[24] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer US, 2006. ISBN: 9780387254654. URL: https://books.google.com.eg/books?id=S-XvEQWABeUC.

[25] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: AAAI Press, 1996, pp. 226–231.

[26] Erich Schubert et al. "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". In: *ACM Trans. Database Syst.* 42.3 (July 2017). ISSN: 0362-5915. DOI: 10.1145/3068335. URL: https://doi.org/10.1145/3068335.

[27] Duo Ding et al. "High performance lithographic hotspot detection using hierarchically refined machine learning". In: *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*. 2011, pp. 775–780. DOI: 10.1109/ASPDAC.2011.5722294.

[28] Jhih-Rong Gao, Bei Yu, and David Z. Pan. "Accurate lithography hotspot detection based on PCA-SVM classifier with hierarchical data clustering". In: *Advanced Lithography*. 2014.

[29] J. Andres Torres. "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite". In: *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2012, pp. 349–350.

[30] Yen-Ting Yu et al. "Machine-Learning-Based Hotspot Detection Using Topological Classification and Critical Feature Extraction". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.3 (2015), pp. 460–470. DOI: 10.1109/TCAD.2014.2387858.

[31] Kareem Madkour et al. "Hotspot detection using machine learning". In: *2016 17th International Symposium on Quality Electronic Design (ISQED)*. 2016, pp. 405–409. DOI: 10.1109/ISQED.2016.7479235.

[32] Siemens EDA. *Calibre Pattern Matching*. Wilsonville, Oregon. URL: https://eda.sw.siemens.com/en-US/ic/calibre-design/physical-verification/pattern-matching/.

[33] Moojoon Shin and Jee-Hyong Lee. "CNN Based Lithography Hotspot Detection". en. In: *The International Journal of Fuzzy Logic and Intelligent Systems* 16.3 (Sept. 2016), pp. 208–215. ISSN: 1598-2645, 2093-744X. DOI: 10.5391/IJFIS.2016.16.3.208. URL: http://www.ijfis.org/journal/view.html?doi=10.5391/IJFIS.2016.16.3.208 (visited on 09/01/2021).

[34] Vadim Borisov and Jurgen Scheible. "Lithography Hotspots Detection Using Deep Learning". en. In: *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. Prague: IEEE, July 2018, pp. 145–148. ISBN: 978-1-5386-5153-7. DOI: 10.1109/SMACD.2018.8434561. URL: https://ieeexplore.ieee.org/document/8434561/ (visited on 09/01/2021).

[35] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647. URL: https://www.science.org/doi/abs/10.1126/science.1127647.

[36] Haoyu Yang et al. "Layout Hotspot Detection With Feature Tensor Generation and Deep Biased Learning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.6 (2019), pp. 1175–1187. DOI: 10.1109/TCAD.2018.2837078.

[37] N. Ahmed, T. Natarajan, and K.R. Rao. "Discrete Cosine Transform". In: *IEEE Transactions on Computers* C-23.1 (1974), pp. 90–93. DOI: 10.1109/T-C.1974.223784.

[38] Xuanyu Huang et al. "Enhancements of Model and Method in Lithography Hotspot Identification". en. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Grenoble, France: IEEE, Feb. 2021, pp. 102–107. ISBN: 978-3-9819263-5-4. DOI: 10.23919/DATE51398.2021.9473963. URL: https://ieeexplore.ieee.org/document/9473963/ (visited on 09/01/2021).

[39] Moojoon Shin and Jee-Hyong Lee. "Accurate lithography hotspot detection using deep convolutional neural networks". en. In: *Journal of Micro/Nanolithography, MEMS, and*

MOEMS 15.4 (Nov. 2016), p. 043507. ISSN: 1932-5150. DOI: 10.1117/1.JMM.15.4.043507. URL: http://nanolithography.spiedigitallibrary.org/article.aspx?doi=10.1117/1.JMM.15.4.043507 (visited on 09/01/2021).

[40] Rasit O. Topaloglu. "ICCAD-2016 CAD contest in pattern classification for integrated circuit design space analysis and benchmark suite". en. In: *Proceedings of the 35th International Conference on Computer-Aided Design*. Austin Texas: ACM, Nov. 2016, pp. 1–4. ISBN: 978-1-4503-4466-1. DOI: 10.1145/2966986.2980073. URL: https://dl.acm.org/doi/10.1145/2966986.2980073 (visited on 11/15/2021).

[41] Shuhei Ishino, Mitsuru Hasegawa, and Kunihiro Fujiyoshi. "A Method of Layout Pattern Classification Using Clustering". en. In: (), p. 6.

[42] S.T. Hedetniemi and R.C. Laskar. "Bibliography on domination in graphs and some basic definitions of domination parameters". en. In: *Discrete Mathematics* 86.1-3 (Dec. 1990), pp. 257–277. ISSN: 0012365X. DOI: 10.1016/0012-365X(90)90365-O. URL: https://linkinghub.elsevier.com/retrieve/pii/0012365X90903650 (visited on 11/17/2021).

[43] C.H. Papadimitriou, K. Steiglitz, and Prentice Hall. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982. ISBN: 9780131524620. URL: https://books.google.ie/books?id=DvBQAAAAMAAJ.

[44] David Arthur and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: vol. 8. Jan. 2007, pp. 1027–1035. DOI: 10.1145/1283383.1283494.

[45] Andre Oliveira et al. "Clip Clustering for Early Lithographic Hotspot Classification". en. In: *2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)*. Armenia, Colombia: IEEE, Feb. 2019, pp. 149–152. ISBN: 978-1-72810-453-9. DOI: 10.1109/LASCAS.2019.8667548. URL: https://ieeexplore.ieee.org/document/8667548/ (visited on 09/01/2021).

[46] Mingyu Woo, Seungwon Kim, and Seokhyeong Kang. "GRASP based metaheuristics for layout pattern classification". In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017, pp. 512–518. DOI: 10.1109/ICCAD.2017.8203820.

[47] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9. URL: https://doi.org/10.1007/978-1-4684-2001-2_9.

[48] Thomas A. Feo and Mauricio G. C. Resende. "Greedy Randomized Adaptive Search Procedures". en. In: *Journal of Global Optimization* 6.2 (Mar. 1995), pp. 109–133. ISSN: 0925-5001, 1573-2916. DOI: 10.1007/BF01096763. URL: http://link.springer.com/10.1007/BF01096763 (visited on 11/18/2021).

[49] Wei-Chun Chang et al. "iClaire: A fast and general layout pattern classification algorithm". In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2017, pp. 1–6. DOI: 10.1145/3061639.3062263.

[50] Yan-Shiun Wu et al. "MapReduce-based pattern classification for design space analysis". In: *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 2018, pp. 1–4. DOI: 10.1109/VLSI-DAT.2018.8373275.

[51] Xu He et al. "Maximum Clique Based Method for Optimal Solution of Pattern Classification". In: *2020 IEEE 38th International Conference on Computer Design (ICCD)*. 2020, pp. 304–311. DOI: 10.1109/ICCD50377.2020.00058.

[52] Immanuel M Bomze et al. "The maximum clique problem". In: *Handbook of combinatorial optimization*. Springer, 1999, pp. 1–74.

[53] Yoichi Tomioka et al. "Lithography hotspot detection by two-stage cascade classifier using histogram of oriented light propagation". In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2017, pp. 81–86. DOI: 10.1109/ASPDAC.2017.7858300.

[54] Juan Pineda. "A parallel algorithm for polygon rasterization". en. In: *Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH '88*. Not Known: ACM Press, 1988, pp. 17–20. ISBN: 978-0-89791-275-4. DOI: 10.1145/54852.378457. URL: http://portal.acm.org/citation.cfm?doid=54852.378457 (visited on 12/30/2021).

[55] Samuli Laine and Tero Karras. "High-performance software rasterization on GPUs". en. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics - HPG '11*. Vancouver, British Columbia, Canada: ACM Press, 2011, p. 79. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018337. URL: http://dl.acm.org/citation.cfm?doid=2018323.2018337 (visited on 12/30/2021).

[56] N. Gharachorloo et al. "A characterization of ten rasterization techniques". en. In: *Proceedings of the 16th annual conference on Computer graphics and interactive techniques - SIGGRAPH '89*. Not Known: ACM Press, 1989, pp. 355–368. ISBN: 978-0-201-50434-7. DOI: 10.1145/74333.74370. URL: http://portal.acm.org/citation.cfm?doid=74333.74370 (visited on 12/30/2021).

[57] G.K. Wallace. "The JPEG still picture compression standard". In: *IEEE Transactions on Consumer Electronics* 38.1 (1992), pp. xviii–xxxiv. DOI: 10.1109/30.125072.

[58] Irwin Sobel. "An Isotropic 3x3 Image Gradient Operator". In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[59] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[60] D.G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[61] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.

[62] John Tabak. *Geometry: the language of space and form*. English. OCLC: 62879193. New York: Facts On File, 2004. ISBN: 978-0-8160-6876-0 978-0-8160-4953-0. URL: http://site.ebrary.com/id/10128663 (visited on 01/12/2021).

[63] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". en. In: *arXiv:1704.04861 [cs]* (Apr. 2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861 (visited on 09/01/2021).

[64] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[65] Gaurav Rajavendra Reddy, Constantinos Xanthopoulos, and Yiorgos Makris. "On Improving Hotspot Detection Through Synthetic Pattern-Based Database Enhancement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), pp. 1–1. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2021.3049285. URL: https://ieeexplore.ieee.org/document/9313011/ (visited on 09/17/2021).

[66] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[67] Jong-hyun Lee et al. "A random approach of test macro generation for early detection of hotspots". In: ed. by Luigi Capodieci and Jason P. Cain. San Jose, California, United States, Mar. 2016, 97810J. DOI: 10.1117/12.2218806. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2218806 (visited on 12/31/2021).