NOVEL DEEP LEARNING METHODS FOR OPTICAL RECOGNITION

A thesis submitted for the degree of Doctor of Philosophy

by:

Richárd Bence Rádli

DOI:10.18136/PE.2026.1002

Supervisor: László Czúni, PhD

University of Pannonia

Faculty of Information Technology

Doctoral School of Information Science

2026

Novel Deep Learning Architectures for Optical Recognition

The thesis was prepared for the award of a doctoral degree (PhD) within the framework of the Doctoral School of Information Science and Technology at University of Pannonia

in the discipline of Computer Sciences

Written by Richárd Bence Rádli

Supervisor: László Czúni, PhD

I recommend the dissertation for acceptance: yes / no.

.....

supervisor

I recommend the dissertation for peer review.

.....

chair of the DDHC

The PhD-candidate has achieved ..... % at the public debate.

The composition of the Final Examination Committee:

chair: .....

reviewers: .....

members: .....

Veszprém,

.....

chair of the committee

Qualification of degree .....

Veszprém,

.....

chair of the UDHC

I wish to honour the memory of my father with this dissertation, who, sadly, could not see its completion.

ACKNOWLEDGEMENTS

I acknowledge the financial support of the projects 2018-1.3.1-VKE-2018-00048 under the ÚNKP-19-3 New National Excellence Program, 2020-4.1.1-TKP2020 under the Thematic Excellence Program This work has been partly supported by the 2020-1.1.2-PIACI-KFI-2021-00296 and the TKP2021-NVA-10 project of the National Research, Development and Innovation Fund. I acknowledge the financial support of the Hungarian Sceintific Research Fund grant OTKA K-135729. I am grateful to the NVIDIA corporation for supporting my research with GPUs obtained by the NVIDIA GPU Grant Program.

I would like to express my deepest gratitude for the support and guidance of my supervisor, László Czúni, PhD, whose dedication greatly contributed to the success of my studies and research.

Cannot express my gratefulness enough for my colleague and friend, Zsolt Vörösházi, PhD, whose support, both mentally and emotionally, has been a cornerstone for me throughout these years. Also, thank you for your guidance whenever and in whatever I needed, and thank you for led me to this path all those years ago.

I appreciate my colleagues in the Department of Electrical Engineering and Information Systems for their unwavering support and invaluable assistance.

I would like to point out to the pharmacy colleagues at the University of Pécs for their help in completing my studies.

I am also immensely grateful to my dear friends, Gergely Kápli and Péter Tóth, for their continuous support. I extend my heartfelt thanks to my former colleagues, Attila Raschek and József Bene, who remain cherished friends.

I want to extend my deepest thanks to my family for their support and patience.

Finally, I would like to dedicate this thesis work for my parents. Without their loving support, I would not have reached this milestone. This achievement is as much theirs as it is mine.

ABSTRACT

Deep learning (DL), a subset of machine learning (ML) and artificial intelligence (AI), is now considered a fundamental technology driving the Fourth Industrial Revolution (4IR or Industry 4.0). Due to its powerful data learning capabilities, DL technology has become a prominent topic in computer science. It is extensively applied across diverse domains such as healthcare, visual recognition, text analytics, and many others. Nevertheless, constructing an effective DL model remains a complex task due to the inherent dynamism and variability of real-world problems and data. The integration of Industry 4.0 and deep learning technologies plays a crucial role in automating and optimising various industrial processes, such as anomaly detection and pill recognition.

First, I proposed multiple frameworks and solutions that improved the overall performance of an already existing SSIM-Autoencoder (SSIM-AE). These developments demonstrate that although higher fidelity outputs can be achieved, they do not necessarily translate into improved anomaly detection outcomes. Additionally, a novel framework was developed by integrating a Spatial Transformer Network (STN) with the SSIM-AE, revealing that an STN can be effectively trained to normalise image orientations. This was accomplished by designing a custom loss function that minimises the image gradient, optimising the model's ability to standardise input orientations and improve detection accuracy.

My following proposal is two novel Deep Randomised Neural Network (DRNN) frameworks, each introducing significant innovations in network architecture. The first framework's primary contribution lies in its phased construction, where the network is progressively built-in sequential stages. The second framework extends this architecture by strategically modifying it, wherein a subset of network neurons — identified as having minimal impact on the final accuracy — is iteratively replaced with neurons derived from smaller auxiliary networks. This iterative replacement process is applied throughout the entire network. These novel approaches demonstrated superior accuracy and faster training times on multiple datasets, compared to widely used backpropagation optimisers, such as Adam and SGD.

Finally, I introduced a novel framework for pill recognition by modifying a multi-stream, two-phase neural network. This modification involved updating the backbone network in the first phase and including a multi-head attention mechanism in the second phase.

Additionally, I constructed a Dynamic Margin Triplet Loss (DMTL) function designed to inject high-level cues — derived from free text or visual appearance — into the training of multi-stream metric models. The proposed metrics-learning solution was evaluated on two benchmark datasets, where it demonstrated superior recognition performance, outperforming the other competitive network.

Furthermore, the multi-stream, two-phase neural network was evaluated against popular automated machine learning (AutoML) solutions and the leading object detection model YOLO11 in a one-shot learning scenario, where it demonstrated superior performance.

KIVONAT

A mélytanulás (DL), amely a gépi tanulás (ML) és a mesterséges intelligencia (AI) egyik alága, napjainkra az Ipar 4.0 (4IR) mozgató rugójává vált. Erőteljes adatfeldolgozási képességeinek köszönhetően a DL kulcsfontosságú szerepet tölt be a számítástudományban, és számos területen, mint például az egészségügyben, a vizuális felismerésben és a szöveganalitikában. Ennek ellenére egy hatékony mélytanulási modell megalkotása továbbra is kihívást jelent, a dinamikusan változó valós problémák és adatok miatt. Az Ipar 4.0 és a mélytanulási technológiák integrációja kulcsfontosságú az ipari folyamatok automatizálásában és optimalizálásában, például az anomáliadetektálás és a gyógyszerfelismerés területén.

Elsőként több olyan keretrendszert és megoldást javasoltam, amelyek javították egy már létező SSIM-Autoencoder (SSIM-AE) teljesítményét. E fejlesztések rávilágítottak arra, hogy bár a kimeneti képek hűsége növelhető, ez nem feltétlenül vezet jobb anomáliadetektálási eredményekhez. Továbbá egy új keretrendszert fejlesztettem ki a SSIM-AE és a Spatial Transformer Network (STN) integrálásával. Ez megmutatta, hogy az STN hatékonyan betanítható a képek orientációjának normalizálására. Ehhez egy egyedi veszteségfüggvényt terveztem, amely minimalizálja a képgradiens értékét, így javítva a modell képességét az orientációk egységesítésére és a detektálási pontosság növelésére.

A következő javaslatom két új Deep Randomised Neural Network (DRNN) keretrendszer volt, amelyek mindegyike jelentős újításokat vezetett be a hálózati architektúrában. Az első keretrendszer legfőbb újdonsága a fázisokban való építkezés, amely során a hálózat fokozatosan, egymást követő lépésekben épül fel. A második keretrendszer ezt fejlesztette tovább azáltal, hogy egyes, a végső pontosság szempontjából kevésbé jelentős neuronokat iteratívan helyettesített kisebb segédhálózatokból származó neuronokkal. Ez az iteratív cserefolyamat az egész hálózatra kiterjeszthető. Az új módszerek több adathalmazon is jobb pontosságot és gyorsabb tanulási időt mutattak, mint a széles körben alkalmazott visszaterjesztéses optimalizálók, például az Adam és az SGD.

Végül egy új gyógyszerfelismerési keretrendszert mutattam be egy többcsatornás, kétfázisú neurális hálózat módosításával. A módosítás során az első fázisban kicseréltem a gerinchálózatot, a második fázisban pedig multi-head attention mechanizmust vezettem be. Emellett kifejlesztettem egy Dynamic Margin Triplet Loss (DMTL) veszteségfüggvényt,

amely a többcsatornás metrikus modellek tanításába magas szintű információkat — szabad szövegből vagy vizuális jellemzőkből származó kulcsfontosságú jellemzőket — injektál. A javasolt metrikus tanulási megoldást két benchmark adathalmazon értékeltem ki, ahol kimagasló felismerési teljesítményt ért el, felülmúlva a többi versenytárs hálózatot.

Továbbá a többcsatornás, kétfázisú neurális hálózatot népszerű automatizált gépi tanulási (AutoML) megoldásokkal, valamint az egyik vezető objektumdetektáló hálózattal – YOLO11 – vetettem össze egy one-shot learning teszt során, ahol a általam javasolt modell jobb teljesítményt ért el ezeknél a megközelítéseknél.

Contents

List of Figures

List of Tables

List of Abbreviations

AAE	Adversarial Autoencoder
Adam	Adaptive Moment Estimation
AE	Autoencoder
ANN	Artificial Neural Networks
AUC	Area Under Curve
CNN	Convolutional Neural Network
DAE	Denoising Autoencoder
DEV-DRNN	Developmental Deep Randomized Neural Networks
DEV-DRNN AUX	Developmental Deep Randomized Neural Networks Auxiliary
DM-MSFN	Dynamic Margin Multi-Stream Fusion Network
DMTL	Dynamic Margin Triplet Loss
DNN	Deep Neural Network
DrELM	Deep representations learning via Extreme Learning Machine
DTD	Describable Textures Dataset
ELM	Extreme Learning Machine
ELM-AE	Extreme Learning Machine - Autoencoder
FISTA	Fast Iterative Shrinkage-Thresholding Algorithm
FM-AE	Feature Matching - Autoencoder
FNN	Feedforward Neural Network
FPR	False Positive Rate
FSL	Few-Shot Learning
GAN	Generative Adversarial Networks

GPU	Graphical Processing Unit
HE-ELM	Hierarchical Ensemble of Extreme Learning Machine
H-ELM	Hierarchical - Extreme Learning Machine
H-ELM-E	Hierarchical Ensemble of Ensemble
H-LR21-ELM	L_{21} Norm Loss Function and Regularization ELM auto-encoder
KL	Kullback-Leibler divergence
LBP	Local Binary Pattern
LTH	Lottery Ticket Hypothesis
Mask R-CNN	Mask Region-based Convolutional Neural Network
ML-ELM	Multilayer-Extreme Learning Machine
MM-LRF-ELM	Multi-Modal Local Receptive Field Extreme Learning Machine
MSE	Mean Squared Error
MSFN	Multi-Stream Fusion Network
NIH	National Institute of Health
NLM	National Library of Medicine
NLP	Natural Language Processing
OCR	Optical Character Recognition
OP-ELM	Optimally Pruned - Extreme Learning Machine
PCA	Principal Component Analysis
P-ELM	Pruned-ELM
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
RVFL	Random Vector Functional Links
SAE	Sparse Autoencoder
SGD	Stochastic Gradient Descent
SLFN	Single-layer feed-forward neural network
SLP	Single Layer Perceptron
SSD	Single Shot MultiBox Detector
SSIM-AE	Structural Similarity Index Measure - Autoencoder

SSIM-AEe	Structural Similarity Index Measure - Autoencoder Extended
SSIM-DAE	Structural Similarity Index Measure - Denoising Autoencoder
SSIM-DAEe	Structural Similarity Index Measure - Denoising Autoencoder Extended
STN	Spatial Transformer Networks
SVM	Support Vector Machines
TPR	True Positive Rate
t-SNE	t-Distributed Stochastic Neighbor Embedding
VAE	Variational Autoencoder
VAE	Variational autoencoder
V-ELM	Voting-based ELM
YOLO	You Only Look Once

Chapter 1

Introduction

In the following pages, I will elaborate on the key ideas and fundamentals essential for understanding the concepts in this study. Other important terms and concepts that are related to this chapter, but not presented here can be found in Section A.1 of the Appendices.

1.1 | Artificial neural networks

The roots of artificial neural networks (ANNs) (*Goodfellow et al., 2016*) can be found in biology. Specifically, ANN models simulate the electrical activity observed in the brain and nervous system. While they emulate a biological neural network, ANNs employ a simplified set of concepts.

The biological neuron constantly gathers electrical impulses from its surroundings or other neurons through its slender extensions called dendrites. The cell body then processes these impulses, generating an action potential determining whether the neuron will transmit an electrical signal along its axon. The neuron will only propagate the signal if the cumulative electrical input received by the dendrites surpasses a specific threshold.

Since the concept of an artificial neuron is inspired by the biological neuron, many of its features have analogous counterparts. As illustrated in Figure 1.1, the inputs received by the artificial neuron are akin to the impulses collected by the dendrites of a biological neuron. The weights in the artificial neuron play a similar role to synaptic strength, assigning importance to each input by multiplying the input value by its corresponding weight.

1.1.1 | Single layer perceptron

Single layer perceptron (SLP) (*Singh et al., 2019*), (*Dongare et al., 2012*) is the essential form of neural network architecture, including no hidden layers, particularly used for the classification of linearly separable patterns. It contains a single McCulloch-Pitts neuron that has adjustable weights and bias. The signals entering a neuron are weighted, simulating the electrical excitation of a nerve cell, which facilitates the transfer of information within the network. The input values $\mathbf{x_i}$ to a processing element are multiplied by a connection weight $\mathbf{w_i}$, representing the strengthening of neural pathways in the brain. Learning in ANNs is achieved by adjusting these connection strengths or weights.

The weight-adjusted input values are then aggregated using a function such as summation, averaging, input maximum, or mode value to produce a single input value for the neuron. Once this input value is calculated, the neuron applies a transfer/activation function to produce its output, which serves as the next layer's input signal. Common activation functions include the sigmoid, hyperbolic tangent, or other non-linear functions. This process is repeated across layers of neurons until the network produces a final output value or a vector of values.

In the case of a threshold (sign) activation function, a neuron's output y can be described by the following formula, where the neuron fires based on the input signals it receives:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} \mathbf{w_i x_i} + b > 0, \\ -1 & \text{otherwise} \end{cases} \tag{1.1}$$

where $\sum_{i=1}^{n} \mathbf{w_i x_i} + b$ is the weighted sum of the input signals plus a bias term. This criterion defines whether the neuron fires (produces an output of 1) or not (produces an output of -1). The transformation of the neuron's input value is crucial for introducing non-linearity into the model, enabling the network to learn complex patterns.

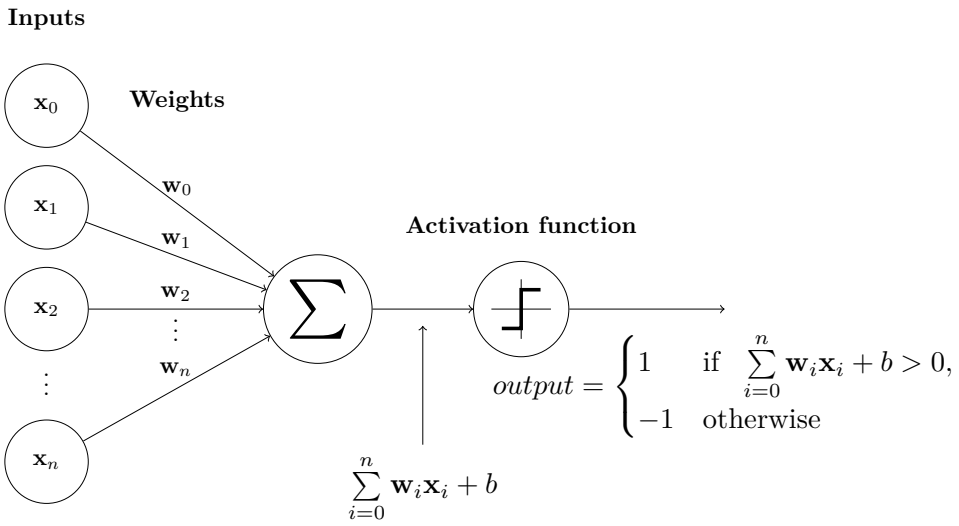

Figure 1.1: Architecture of a single layer perceptron.

1.1.2 | Feedforward neural networks

A feedforward neural network (FNN) (*Goodfellow et al., 2016*) is one of the simplest yet most crucial types of artificial neural networks. The primary objective of an FNN is to approximate a function f^*. For instance, given a classification task, $y = f^*(\mathbf{x})$ maps an input $\mathbf{x}$ to a category y. An FNN defines this mapping as $y = f(\mathbf{x}, \theta)$, and it learns the parameters θ that produce the best approximation of the function.

These models are termed feedforward because the information flows strictly in one direction: from the input $\mathbf{x}$, through a series of intermediate computations, to the output y. There are no feedback loops where the output is sent back into the network.

The architecture of an FNN consists of an input layer, one or more hidden layers, and an output layer. Each layer consists of units known as neurons, and the layers are connected by weights.

- **Input layer**: This layer contains neurons that receive inputs and pass them to the next layer.
- **Hidden layer(s)**: These layers serve as the computational core of the neural network. Neurons in a hidden layer compute the weighted sum of the inputs from the previous layer, apply an activation function, and pass the result to the next layer. For example, in the case of a network with a single hidden layer, the activations can be formulated as:

$$\mathbf{h} = \phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \tag{1.2}$$

where $\mathbf{h}$ denotes the activations of the hidden layer, $\mathbf{W}_1$ is the weight matrix connecting the input layer to the hidden layer, $\mathbf{b}_1$ is the bias vector, and ϕ_1 is the activation function.

- **Output layer**: The final layer produces the output for the given inputs. The number of neurons in the output layer depends on the number of possible outputs/classes the network is designed to generate. In the single hidden layer case, the output can be computed as:

$$y = \phi_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2) \tag{1.3}$$

where y represents the output vector, $\mathbf{W}_2$ is the weight matrix connecting the hidden layer to the output layer, and $\mathbf{b}_2$ is the bias vector.

Each neuron in one layer is typically connected to every neuron in the next layer, making this a fully connected network. Weights represent the connection strengths between neurons, and learning in a neural network involves updating these weights based on the error in the output.

Formally, for a feedforward neural network with a single hidden layer, the overall mapping can be expressed as:

$$y = \phi_2(\mathbf{W}_2 \cdot \phi_1(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \tag{1.4}$$

When multiple hidden layers are used, the model is referred to as a deep neural network (DNN), where the input is successively transformed through several layers before producing the final output.

1.1.3 | Convolutional neural networks

Before convolutional neural networks (CNNs) (*LeCun et al., 2015*) became part of the public consciousness, object identification in images relied on manual and often time-consuming feature extraction methods. CNNs revolutionized this process by offering a more scalable solution for image classification and object recognition tasks. These networks are based on

the principles of linear algebra, mainly on matrix multiplication, to perceive patterns within images. CNNs stand out from other neural networks due to their powerful performance with image, speech, and audio signal inputs. However, despite their effectiveness, CNNs can be computationally heavy and often require graphical processing units (GPUs) for model training.

They consist three main types of layers:

- **Convolutional layer**: This layer applies convolutional operations to the input, effectively detecting patterns and features.
- **Pooling layer**: It reduces the spatial dimensions of the input.
- **Fully-connected layer**: Finally, this layer connects every neuron in one layer to every neuron in another, aiding the final decision-making process in the network.

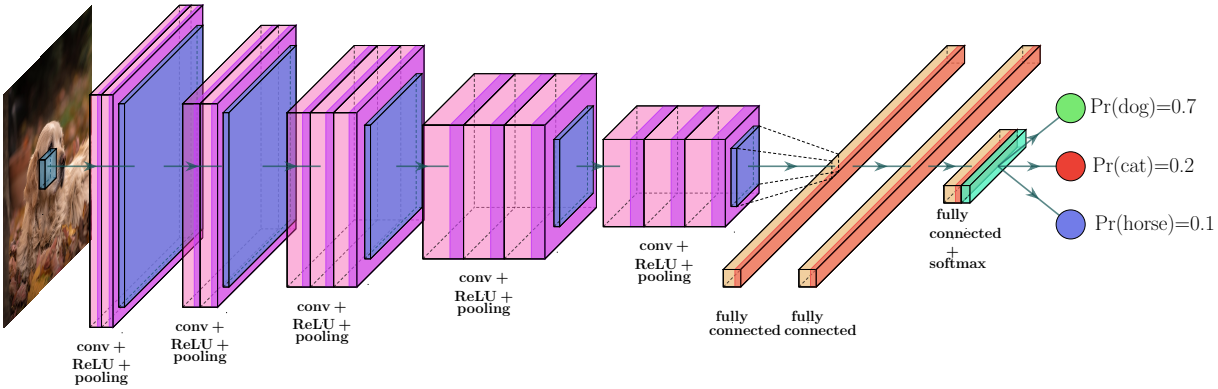

Figure 1.2: General structure of a convolutional neural network.

The **convolutional layer** is the core building block of a CNN, where most computations occur. It requires input data, a filter (kernel), and produces a feature map.

Assuming the input is a colour image, it is represented as a 3D matrix with dimensions corresponding to the image's height, width, and depth (RGB channels). The filter, typically defined with the same depth as the input, moves across the image to detect features. This process, called convolution, can be expressed mathematically as:

$$(f * g)(m, n) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(k, l) \cdot g(m - k, n - l), \tag{1.5}$$

where $(f * g)(m, n)$ is the result of the convolution. For colour images, the convolution is performed across all channels, meaning that $f(k, l)$ and $g(m-k, n-l)$ represent multi-channel

values, and the operation involves summation over both spatial dimensions and channels.

The filter size determines the receptive field size and is applied to image sections. A dot product is calculated between the input pixels and the filter values, and this result is fed into an output array. The filter then shifts by a stride, repeating this process across the entire image. The final output from these dot products is the feature map.

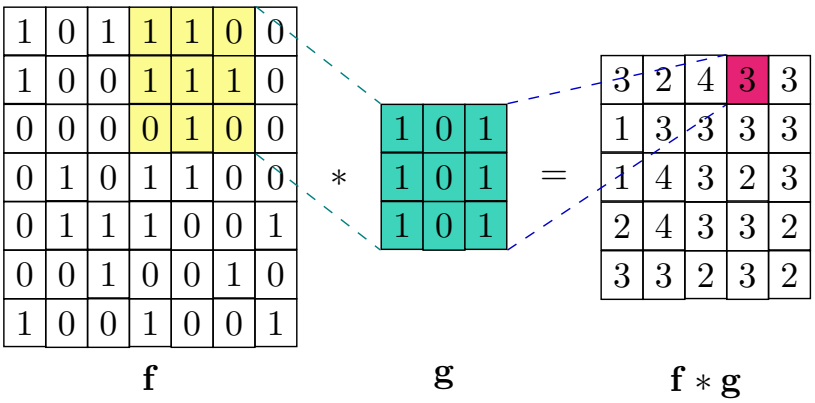

Figure 1.3: Example for a convolution operation.

During training, some parameters, like weight values, are adjusted through backpropagation and gradient descent. However, three hyperparameters affecting the size of the output need to be set before the training starts:

1. **Number of filters**: This affects the depth of the output.
2. **Stride**: This is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values greater than two are rare, a larger stride produces a smaller output.
3. **Padding**: This is used when the filters do not fit the input image. Padding sets elements outside the input matrix to zero, producing a larger or equally sized output. There are three types of padding:
   - **Valid padding**: Also known as no padding, where the last convolution is dropped if dimensions do not align.
   - **Same padding**: Ensures the output layer has the same size as the input layer.
   - **Full padding**: Increases the output size by adding zeros to the input border.

After each convolution operation, a CNN applies an activation function called Rectified Linear Unit (ReLU) to the feature map, and by this, it introduces non-linearity to the

model. Non-linearity is important; otherwise, the CNN could not learn meaningful, complex patterns. The initial convolutional layers in a CNN typically learn to detect basic, low-level features. Some common types of features that the early convolutional layers might detect are:

- **Edges**: Detecting edges is fundamental because they represent the boundaries between different regions or objects in an image. Early filters often respond to vertical, horizontal, and diagonal edges.
- **Corners**: Corners are points where two edges meet, and detecting them helps recognise shapes and objects within the image.
- **Textures**: Basic textures or patterns, such as stripes or blobs, can be detected by initial convolutional filters.
- **Colour patterns**: Early layers might learn to detect different colour patterns or gradients in colour images.

Another convolution layer can follow the initial convolution layer, creating a hierarchical structure. Later layers can view the pixels within the receptive fields of previous layers. Ultimately, the convolutional layer converts the image into numerical values, enabling the neural network to interpret and extract relevant patterns.

Before going any further, it is vital to introduce the operation called transposed convolution or deconvolution (*Dumoulin et al., 2016*). Deconvolution can be understood as the reverse operation of convolution. It expands the feature map by applying a kernel so that the output spatial dimensions are larger than the input dimensions. It spreads the values of the input feature map to a larger space. Furthermore, in generative AI models like generative adversarial networks (GANs) or autoencoders, deconvolution layers generate high-resolution images from lower-dimensional representations.

Pooling layers, also known as downsampling, introduces dimensionality reduction, thereby reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation moves a filter across the entire input. However, unlike convolutional layers, the pooling layer filter has no learnable parameters. Instead, the kernel applies an aggregation function to the values within the receptive field, filling the output array. There are two main types of pooling:

- **Max pooling**: As the filter moves across the input, it selects the pixel with the

maximum value of the designated area to send it to the output array. This approach tends to be used more often than average pooling.

- **Average pooling**: As the filter moves across the input, it calculates the average value within the receptive field to move it to the output array. Both pooling methods are visualised in Figure 1.4.

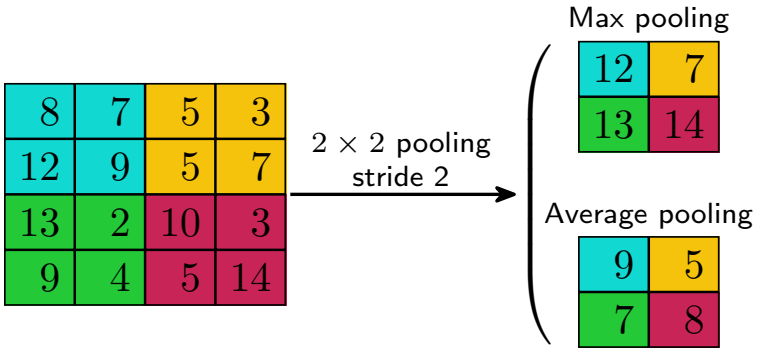

Figure 1.4: Pooling operations; max- and average pooling.

While the pooling layer results in a loss of information, it also provides several benefits to CNN. Pooling layers help to reduce complexity, improve efficiency, and limit the risk of overfitting.

The last layer in a CNN is the **fully-connected layer**. As the name suggests, it connects each node in the output layer directly to a node in the previous layer. First, we "flatten" the data from the last max-pool layer and only then transfer the data to the first fully connected layer. This layer performs classification based on the features extracted by the previous layers and their various filters. While convolutional and pooling layers typically use ReLU/leaky ReLU functions, fully-connected layers employ linear and softmax activation functions - softmax is only applied in the last fully-connected layer - to classify inputs appropriately, producing a probability distribution over the possible output classes, ranging from 0 to 1.

1.2 | Learning paradigms

Learning paradigms (*Goodfellow et al., 2016*), (*Nasteski, 2017*) in machine learning refers to the techniques or algorithms used by models to understand patterns and relationships within data and make predictions or decisions. These methods can be broadly categorised into three main types: supervised learning, unsupervised learning, and reinforcement learning.

The first two will be elaborated, however reinforcement learning is out of the scope of this study.

Unsupervised learning algorithms analyse a dataset containing numerous features to distinguish and learn the underlying structure of the data. Within deep learning, the objective is often to model the entire probability distribution that generated the dataset, either explicitly, as in density estimation, or implicitly, for applications such as synthesis or denoising. Some unsupervised learning algorithms also serve other functions, such as clustering, which involves partitioning the dataset into clusters of similar instances.

Supervised learning algorithms process a dataset that includes features and corresponding labels or targets for each example. Generally, unsupervised learning focuses on analysing multiple instances of a random vector x to implicitly or explicitly learn the probability distribution $p(x)$ or to uncover significant properties of that distribution. Conversely, supervised learning involves examining multiple instances of a random vector x and an associated value or vector y to predict y from x, typically by estimating $p(y \mid x)$.

Other variants of the learning paradigm are possible. For example, in **semi-supervised learning**, some examples include a supervision target while others do not in order to predict y from x.

To summarise, supervised learning originates from the perspective that the target y is provided by an instructor or teacher who guides the machine learning system. On the contrary, unsupervised learning lacks an instructor or teacher, requiring the algorithm to distinguish patterns in the data independently. Although unsupervised and supervised learning differences are not entirely formal, these categories are helpful for broadly classifying machine learning tasks. Traditionally, regression, classification, and structured output problems fall under supervised learning, whereas dimensionality reduction or clustering is considered unsupervised learning. Anomaly detection is a prime example of semi-supervised learning.

Few-shot learning (FSL) is the ability of machine learning models to generalise from a few training examples. When there is only one example to learn from, FSL is called one-shot learning. The motivation for FSL is that models excelling at this task would have many practical applications (such as pill recognition, as we will see later). First, we would not be required to collect hundreds or thousands of labelled examples for a reasonable performance on a new task. This would help relieve the data-gathering procedure and reduce

the computation costs and time spent in model training. Furthermore, data is hard, if not impossible, to acquire in many fields due to privacy and safety. Models that generalise from a few examples would be able to capture this type of data effectively.

To tackle the problem of limited data, several techniques are employed in FSL:

- **Metric learning**: Approaches such as Siamese Networks and Prototypical Networks learn similarity metrics to facilitate classification with few examples.
- **Transfer learning**: Utilises pre-trained models and fine-tunes them on new tasks with limited data.

Additionally, several recent approaches exploit **in-context learning**, wherein models generalise to previously unseen tasks by conditioning on a small number of input–output examples provided directly in the prompt, without requiring any parameter updates. This paradigm enables models, such as large language models (LLM), to adapt to new tasks dynamically at inference time, making them particularly useful when labelled data is limited.

1.3 | Types of computer and machine vision tasks

Plenty of tasks can be solved with the aid of machine learning (*Goodfellow et al., 2016*). Some of the most frequent tasks in the field of computer and machine vision are listed and elaborated below.

Classification is a task where a program must determine which of k categories a given input belongs to. In order to solve this, the learning algorithm typically creates a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$. When $y = f(\mathbf{x})$, the model assigns an input, represented by the vector $\mathbf{x}$, to a category identified by the numeric label y. There are also other implementations of the classification task, such as those where f outputs a probability distribution over the classes.

Regression is a task that aims to predict a numerical value given some input. To achieve this task, the learning algorithm has to output a function $f : \mathbb{R}^n \to \mathbb{R}$. This is quite similar to classification. However, the output format differs, being a continuous numerical value rather than a discrete category.

Anomaly detection involves moving through a set of events or objects and detecting some as unusual or abnormal. This task is crucial in various industrial systems applications such as network security or fault detection. The goal is to identify patterns in the data

that deviate significantly from the expected behaviour, which could indicate critical and actionable insights.

Object detection is a task in which a model identifies and localises objects within an image. Unlike classification, which assigns a single label to the entire image, object detection provides both the categories and the locations of the objects, typically in the form of bounding boxes.

The learning algorithm produces a function $f : \mathbb{R}^n \to \{(c_i, \mathbf{b}_i)\}_{i=1}^m$, where c_i is the class label and $\mathbf{b}_i \in \mathbb{R}^4$ denotes the bounding box coordinates (e.g., center coordinates, width, and height) for the i-th object in the image.

Object detection is widely used in autonomous driving, surveillance, and image retrieval applications.

Segmentation can be further divided into semantic segmentation and instance segmentation. In **semantic segmentation**, the task involves classifying each pixel in an image into a category, producing a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$ for each pixel, where each output corresponds to a class label. On the other hand, in **instance segmentation**, the task is to classify each pixel and differentiate between different instances of the same class. This means producing a function that outputs the class label and a unique identifier for each object instance, allowing the segmentation of individual objects within the same category. Finally, **amodal segmentation** extends instance segmentation by predicting the full extent of objects, including the parts that are occluded or not visible in the image. These tasks are critical in medical imaging, autonomous driving, and scene understanding applications.

1.4 | Development environment

The computational experiments were carried out on a dedicated workstation with the following hardware configuration: an AMD Ryzen 5 5600X processor, an NVIDIA RTX 5000 GPU with 16 GB of VRAM, and 32 GB of DDR4 RAM. The utilized operating systems were Windows 11 Pro and Ubuntu 24.04 LTS. GPU acceleration was involved in handling the computational workload in the experiments described in **Chapter 2** and **Chapter 4**. Due to the nature of the experiments in **Chapter 3** only the CPU was involved.

The software environment involved Python versions 3.8 through 3.11 and incorporated libraries for numerical computation (NumPy), deep learning (PyTorch), and computer vision

(OpenCV). RayTune was employed for parameter search. Other useful libraries, such as Pandas, Matplotlib, Scikit-Image and Scikit-Learn, were also involved. The development was done with PyCharm IDE Professional and Visual Studio Code.

1.5 | Code repositories and datasets

For the sake of reproducibility, the source code developed and used throughout this work is publicly available via GitHub repositories (see clickable links below). Each repository corresponds to a specific paper and is referenced accordingly.

- About the Application of Autoencoders for Visual Defect Detection
- Deep Randomized Networks for Fast Learning
- Faster and Accurate: Developmental Deep Randomized Networks
- Multi-stream pill recognition with attention
- Pill Metrics Learning with Multihead Attention
- Word and Image Embeddings in Pill Recognition
- Metric-based pill recognition with the help of textual and visual cues
- Code-Based Versus AutoML Methods for Pill Recognition in Clinical Settings: Comparative Performance Study

The datasets used in this study are summarized in Table 1.1. The table offers an overview of their names, origins, and the chapters in which they are applied.

Table 1.1: Overview of datasets used in this study, including sources, associated chapters.

Dataset	Source	Chapter	Notes
Texture I & II	Link to datasets	2	More on Section 2.3
CPU	Link to dataset	2	More on Section 2.3
MVTec Screw	Link to dataset	2	More on Section 2.3
UCI Repository	Link to datasets	3	More on Section 3.3.1
CURE	Link to dataset	4	More on Section 4.2
OGYEIv2	Link to dataset	4	More on Section 4.2
Lab. & Exh. Set	Not publicly available	4	More on Section 4.2
PTE Pill	Not publicly available	4	More on Section 4.2
PTE-PE Pill	Not publicly available	4	More on Section 4.2

1.6 | The structure of the thesis

My dissertation is divided into five main chapters, with **Chapter 2**, **Chapter 3**, and **Chapter 4** dedicated to presenting my three thesis groups. At the beginning of the dissertation, a general introduction was provided.

Chapter 2 explores autoencoder networks (AEs) that do not require defect examples for training. I investigate the performance of modified and enhanced AEs across various datasets and examine the use of spatial transformer networks (STNs) to improve the efficiency of AEs in reconstruction and defect detection tasks.

Chapter 3 presents two innovative algorithms that show new building approaches for dense hierarchical randomised deep neural networks. The first algorithm employs a multi-phase optimisation process to build deep randomised networks. In contrast, the second iteratively replaces the least contributing neurons with more effective ones sourced from smaller auxiliary networks. Both techniques improve classification accuracy while maintaining rapid training times.

I introduce a novel architecture designed for visual feature extraction and processing in **Chapter 4**. This architecture employs a multi-stream approach combined with multi-head attention layers to estimate the similarity of pills. A key enhancement is made to the traditional triplet loss function by incorporating word and visual embeddings, enabling the integration of textual pill similarities into the model. This improvement refines the learning ability of the model by yielding finer granularity compared to conventional triplet

loss models, resulting in a more accurate visual model. The proposed method is evaluated on two benchmark datasets, demonstrating its effectiveness.

Finally, **Chapter 5** summarises my work and presents new scientific results and contributions.

Chapter 2

Enhancing Autoencoder Architectures to Improve Visual Defect Detection Abilities

Visual defect detection is a key technology in modern industrial manufacturing systems. Since anomalous items are often missing during the training process, unsupervised anomaly detection is the most suitable approach to building a defect detection system. Many possible appearances of product defects include distortions in colour, shape, contamination, and missing or superfluous parts. For the detection of those, besides traditional image processing techniques, convolutional neural network-based methods have also appeared to avoid the usage of hand-crafted features and to build more efficient detection mechanisms. In this chapter, I deal with autoencoder networks (AEs), which do not require examples of defects for training. Unfortunately, the manual and/or trial-and-error design of AEs is still required to achieve good performance since many unknown parameters of AEs can greatly influence detection abilities. My purpose is to find ways to improve the anomaly detection accuracy of a baseline AE.

First, I will give a brief introduction to AEs, and then I will elaborate on my contributions and practical applications. Some additional material can be found in Section B.1 of the Appendices.

2.1 | Overview of autoencoders

A primary goal of unsupervised learning is to uncover data representations that are useful for various applications (*Bishop et al., 2024*). One widely used method for learning internal

representations is the auto-associative neural network, commonly known as the AE. An AE is designed to replicate its input at the output, with the number of input units equal to the number of output units. The network typically consists of three main components:

- The encoder network, $E : \mathbb{R}^{k \times h \times w} \to \mathbb{R}^d$,
- The latent space, $\mathbb{R}^d$,
- The decoder network, $D : \mathbb{R}^d \to \mathbb{R}^{k \times h \times w}$.

Formally, this is expressed as:

$$\hat{\mathbf{x}} = D(E(\mathbf{x})) = D(\mathbf{z}), \tag{2.1}$$

where $\mathbf{x}$ is the input image, $\mathbf{z}$ is the latent space representation, and $\hat{\mathbf{x}}$ is the reconstructed output.

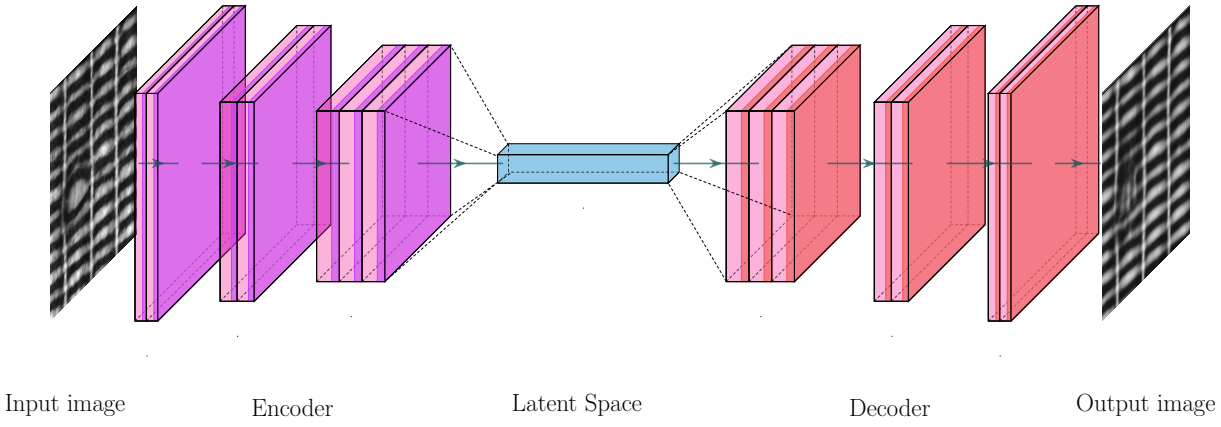

Figure 2.1: Structure of an autoencoder.

The encoder consists of layers that reduce the dimensionality of the input data, creating a compressed representation. In a typical AE, the hidden layers gradually decrease the number of neurons compared to the input layer. As the data passes through the encoder, it is "squeezed" into a more compact form.

The bottleneck, or latent space, holds the most compressed representation of the input and acts as both the encoder's output and the decoder's input. A critical design goal for the AE is to identify the minimum set of features (or dimensions) needed to reconstruct the input accurately. This compressed representation is passed to the decoder.

The decoder consists of layers with progressively increasing neurons, which decompress the latent representation to reconstruct the original data. The reconstruction is then

compared to the "ground truth"—typically the original input—allowing evaluation of the AE's performance. The reconstruction error, which quantifies the difference between the output and the ground truth, is given by the sum-of-squares error:

$$J(\mathbf{x}, \theta) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{2.2}$$

where $\theta = (w_E, b_E, w_D, b_D)$ are the weights and biases of the encoder and decoder networks, and n is the number of elements.

If an AE merely learns to replicate its input exactly, it becomes less valuable. Therefore, AEs are intentionally designed to avoid perfect copying. Typically, they are constrained to approximate the output and replicate only input data similar to the training examples. By prioritising which aspects of the input to replicate, the model often uncovers meaningful properties of the data. One such constraint could be achieved by restricting the dimensionality of the latent space $\mathbf{z}$ relative to that of the input $\mathbf{x}$ or by requiring $\mathbf{z}$ to have a sparse representation. Alternatively, the network can be encouraged to discover non-trivial solutions by modifying the training process to learn to undo corruptions to the input vectors, such as additive noise or missing values.

The similarity of principal component analysis (PCA) and AEs is well known; moreover, in many scenarios, AEs outperformed linear or kernel PCA (*Sakurada et al., 2014*). One of the primary advantages of using AE over other dimensionality reduction techniques is their ability to capture complex non-linear correlations. Consequently, the activation functions used in AE are typically non-linear, such as the sigmoid function.

2.2 | Types of autoencoders

Most types of AEs are employed in artificial intelligence tasks related to feature extraction, such as data compression, image denoising, anomaly detection, and facial recognition. Specific types of AEs, such as variational autoencoders (VAEs) and adversarial autoencoders (AAEs), adapt the AE architecture for generative tasks, including image generation and generating time series data (*Goodfellow et al., 2016*), (*Bishop et al., 2024*).

2.2.1 | Undercomplete autoencoders

Copying the input to the output may seem trivial, but the primary interest lies not in the output of the decoder but in the properties of the latent space z. By training the AE to perform the input copying task, we aim for z to acquire beneficial properties. One way to obtain valuable features from the AE is to constrain the latent space to have a smaller dimension than the input image. An AE with a code dimension smaller than the input dimension is called undercomplete. Learning an undercomplete representation forces the AE to capture the most salient features of the training data.

If the encoder and decoder have too much capacity, the AE can learn to copy the input to the output without extracting useful information about the data distribution. Theoretically, an AE with a one-dimensional code but a very powerful non-linear encoder could learn to represent each training example x^i with the code i. The decoder could then map these integer indices back to the specific training examples. While this scenario does not occur in practice, it illustrates that an AE trained to copy the input can fail to learn anything meaningful about the dataset if it has too much capacity.

2.2.2 | Regularised autoencoders

Regularised AEs aim to overcome the limitations of undercomplete autoencoders by incorporating regularisation techniques. These techniques constrain or modify the calculation of the reconstruction error within the model. By introducing regularisation terms, overfitting is reduced, and the AE can learn valuable features or functions.

2.2.2.1 | Sparse autoencoders

Sparse autoencoders (SAEs) are a type of autoencoder designed to learn efficient representations by encouraging sparsity in the hidden layer. Unlike standard autoencoders, which can trivially learn the identity function, SAEs are forced to capture distinct statistical features of the training data. This is achieved by adding a regularisation term to the loss function that penalises hidden unit activations, effectively reducing the dimensionality and producing a sparse internal representation.

Formally, the loss function of a sparse autoencoder can be written as:

$$J(\mathbf{x}, \theta) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_1 + R(\mathbf{z}), \tag{2.3}$$

where the first term is the reconstruction loss and $R(\mathbf{z})$ is the sparsity-inducing regularisation term applied to the hidden activations $\mathbf{z}$. Common choices for $R(\mathbf{z})$ include:

- **L1 regularisation:**

$$R(\mathbf{z}) = \lambda \sum_{j=1}^{h} |z_j|,$$

  which encourages most hidden units to remain near zero for any given input.

- **Kullback-Leibler (KL) divergence:**

$$R(\mathbf{z}) = \sum_{j=1}^{h} \mathrm{KL}(\rho \,\|\, \hat{\rho}_j),$$

  where $\hat{\rho}_j$ is the average activation of hidden unit j over the training data, and ρ is the desired sparsity level. This formulation penalises deviations from the target sparsity.

By applying these regularisation terms, the network learns a compact and informative representation of the input data, where most hidden units remain inactive for any single input. This sparsity often improves the performance of the learned features on downstream tasks such as classification.

2.2.2.2 | Denoising autoencoders

As mentioned earlier, constraining the dimensionality of the latent space layer in a simple AE is crucial to prevent the model from merely learning the identity mapping. An alternative approach, which also encourages the model to uncover interesting internal structures in the data, is denoising autoencoders (DAEs) (*Vincent et al., 2008*). A DAE takes a corrupted data point as input and is trained to reconstruct the original, uncorrupted data point as its output. In (*Vincent et al., 2010*), Gaussian noise, salt-and-pepper, and zero-masking noise were investigated.

By learning to denoise the input data, the network is compelled to learn aspects of the underlying structure of that data. For instance, in the case of image data, learning that nearby pixel values are strongly correlated enables the model to correct noise-corrupted pixels.

Equation 2.1 now becomes:

$$\hat{\mathbf{x}} = D(E(C(\mathbf{x}))) = D(E(\tilde{\mathbf{x}})) = D(\mathbf{z}), \tag{2.4}$$

where C denotes the corrupting function generating $\tilde{\mathbf{x}}$. Furthermore, the applied loss puts an emphasis on the corrupted areas by proper weighting:

$$\mathbf{J}(\mathbf{x}, \tilde{\mathbf{x}}, \theta) = \alpha \sum_{i \in \mathcal{C}} (\mathbf{x_i} - \hat{\mathbf{x}}_\mathbf{i})^2 + \beta \sum_{i \notin \mathcal{C}} (\mathbf{x_i} - \hat{\mathbf{x}}_\mathbf{i})^2, \tag{2.5}$$

where $\mathcal{C}$ denotes the indexes of corrupted components.

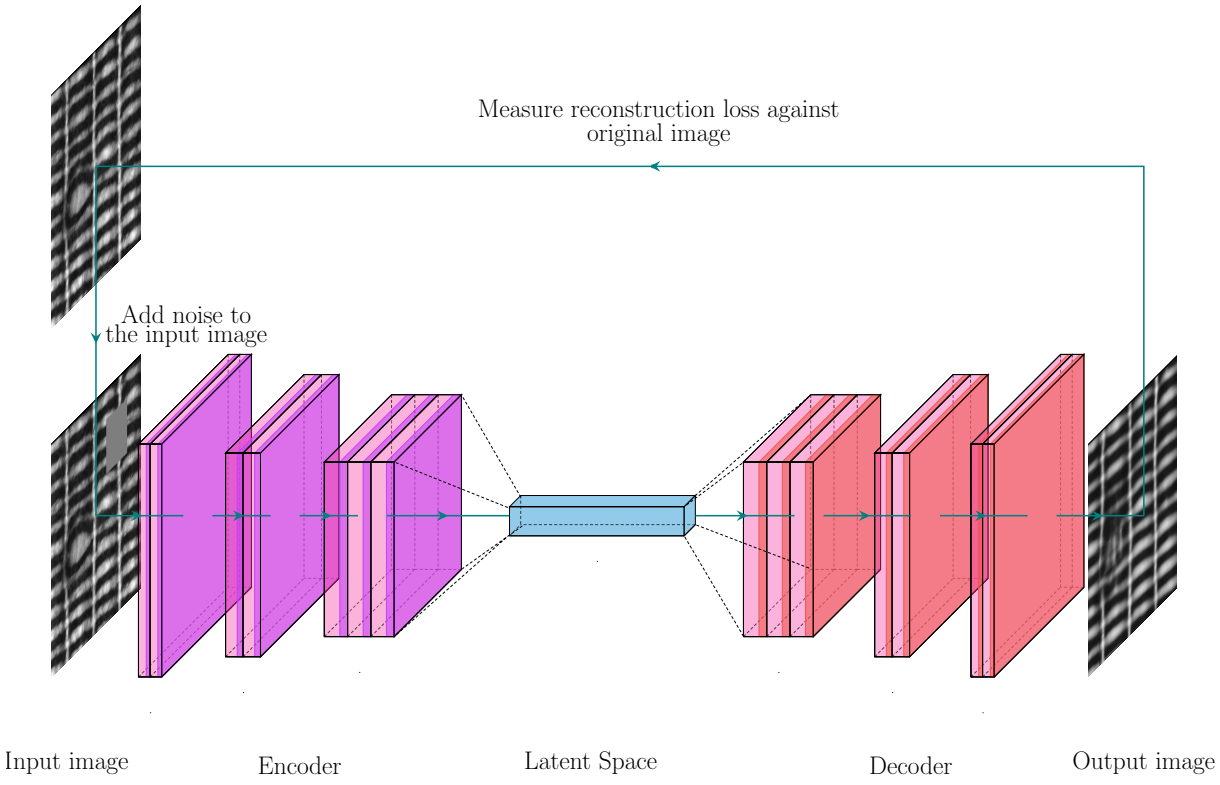

Figure 2.2: Schematic display of a denoising autoencoder.

As a point of interest, there is a variant called masked autoencoder (*He et al., 2022*). In this AE, a deep network is employed to reconstruct an image from a corrupted version of the image, akin to DAE. However, in this scenario, the corruption is masking or dropping out parts of the input image.

2.2.2.3 | Feature matching autoencoders

(*Dosovitskiy et al., 2016*) proposed an enhancement to standard AE that improves the quality of generated reconstructions. This approach involves extracting features from both

the input image $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$ and enforcing the equality of these features. In other words, a feature-matching autoencoder enforces the features of the input and its reconstruction to be equal.

Consider $F : \mathbb{R}^{k \times h \times w} \to \mathbb{R}^f$ to be a feature extractor that obtains an f-dimensional feature vector from an input image. Then, a regularizer can be added to the loss function of the AE, yielding the feature-matching autoencoder (FM-AE) loss:

$$\mathbf{J}(\mathbf{x}, \tilde{\mathbf{x}}, \theta) = w_{im} \sum (\mathbf{x} - \hat{\mathbf{x}})^2 + w_{ft} \sum (F(\mathbf{x}) - F(\hat{\mathbf{x}}))^2, \tag{2.6}$$

where F is a feature extractor which can be fixed or also trained. In (*Dosovitskiy et al., 2016*), besides using features for evaluation, adversarial discriminator was also utilized. For evaluation, we compute a residual map R_{FM} by calculating the per-pixel L2-distance between $\mathbf{x}$ and $\hat{\mathbf{x}}$. The underlying assumption is that the enhanced, more accurate reconstructions provided by this method will result in superior residual maps compared to those produced by a conventional AE.

2.2.3 | Variational autoencoders

Variational autoencoders (VAEs) (*Kingma et al., 2013*) introduce constraints on the latent variables, forcing them to follow a specified distribution $z \sim P(z)$. Commonly, this distribution is selected as a unit-variance Gaussian for simplicity. This transforms the framework into a probabilistic model, facilitating efficient posterior inference and enabling the generation of new data by sampling from the latent distribution.

The approximate posterior distribution $Q(z|x)$ obtained by encoding an input image can be leveraged to define additional anomaly measures. One approach is to calculate the distance between the posterior and prior distributions, such as the KL-divergence:

$$\mathrm{KL}(Q(z|x) \| P(z)). \tag{2.7}$$

Large deviations from the prior $P(z)$ can then be used to identify defects (*Soukup et al., 2018*).

For pixel-accurate anomaly segmentation, the deviations in the latent space or reconstruction errors can be mapped back to the input image, typically using a single forward pass through the encoder-decoder network rather than a separate pass for each pixel.

2.3 | Datasets

In the upcoming sections, I will turn to the practical use of AEs. To this end, I will first present the datasets I used.

The visually observable defects include faulty colour, contamination, geometrical distortion, missing parts, or superfluous parts. One can assume that objects are almost 2D (like the surface of a rectangular object without significant bulges or holes), and images have variations in translation, orientation, and lighting conditions.

2.3.1 | Texture and Screw datasets

The Texture datasets (1 and 2), provided by (*Bergmann et al., 2018*), contain regular patterns of woven fabric textures. Each of these sets includes 100 defect-free images and 50 test images with various defects. Ground truth images were also provided in binary maps. I have chosen these textures since they represent roughly regular repetitive patterns. Since I focus on convolutional AEs, the radius of convolutions and the number of layers can significantly impact the generation abilities depending on the texture size.

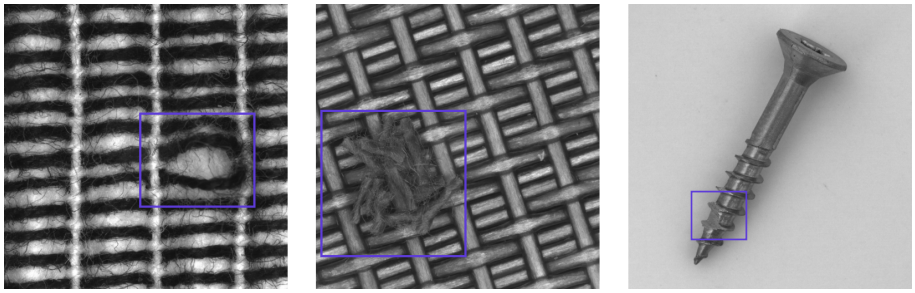

Figure 2.3: Samples from Texture 1, 2, and Screw with a defect.

The original texture sets included 100-100 defect-free images and 50-50 defective images with various faults. The authors also provided pixel-wise ground truth images. All of the original texture images are of the size of 512×512 pixels. These datasets were used both in Section 2.4.1 and Section 2.5.5.

The Screw dataset is part of the MVTec dataset (*Bergmann et al., 2019*), and has 320 defect-free images of size 1024×1024; the defective class Thread-top - used in my defect detection tests in Section 2.5.5 - has 23 images.

2.3.2 | Custom CPU dataset

My custom CPU dataset involves 60 genuine images of the backside of a CPU of size 37.50 mm × 37.50 mm. I separated 48 images for training purposes and the remaining 12 for various tests. Three different test cases were generated for the CPUs:

- CPUa: I added extra components with a visually realistic appearance;
- CPUc: the test images are loaded with synthetic contamination;
- CPUm: some components (such as pins or capacitors) are removed.

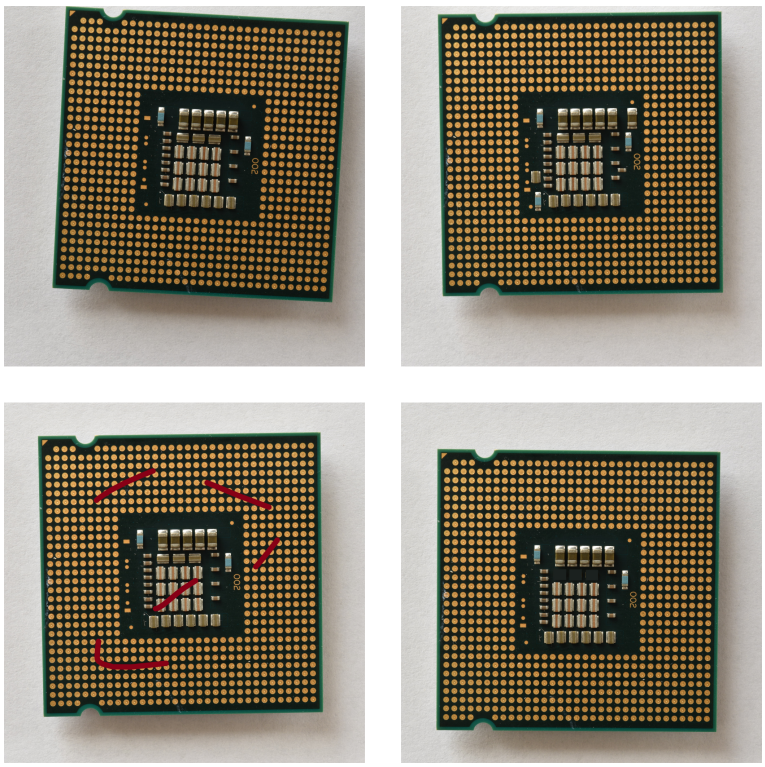

Figure 2.4: Samples from the CPU dataset. The top row (from left to right) shows a CPU without defects and a CPU with added extra components. The bottom row (from left to right) shows a CPU with synthetic contamination and a CPU with missing capacitors.

These operations are carried out on all 12 test images.

The selection of the CPU images is twofold: first, such and similar images are typical in the production of electronic components; second, the image of the CPU contains regularly repetitive (outward areas with contacts) and regular but much less repetitive regions (central capacitor area). Besides, the CPU images contain more finer details than the textures.

All images are of size 512 × 512; their illustration with defects is in Figure 2.4. Ground

truth images were also provided as binary maps. Results on the CPU dataset can be found in Section 2.4.1.

2.4 | Application of autoencoders for visual defect detection

As mentioned previously, trained AEs can generate very similar outputs to their inputs if the latter fits the trained model, or, in other words, the image models could learn the inherent features properly. That is, if the difference between the two is above a threshold Th, we set the detection map to 1:

$$\mathbf{m} = \begin{cases} 1, & \text{if } ||\mathbf{x} - \hat{\mathbf{x}}|| > Th, \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

The applied $||.||$ can be various (e.g. L1, L2, SSIM (*Bergmann et al., 2018*)).

In (*Beggel et al., 2020*), they deal with the problem of training AEs when the training set is contaminated with a small fraction of outliers. Their proposed adversarial AE imposes a prior distribution on the latent representation, placing anomalies into low-likelihood regions; thus, potential anomalies can be identified and rejected during training.

(*Shvetsova et al., 2021*) proposes perceptual deep AEs where relative-perceptual-L1 loss (*Tuluptceva et al., 2019*), robust to low contrast and noise, is applied for training and also to predict the abnormality for new inputs. The applied progressive, growing technique helped the efficiency of training: at the beginning of the training, the loss function computes the dissimilarity between the low-resolution images using the features from the coarse layers of the network, whereas, as the training advances, the "level" of this information is increased by including deeper and deeper features. The addition of the new layers to the AE with the gradual increase of the depth of the features entailed in calculating the perceptual loss was synchronised.

We can assume that SSIM-AE (*Bergmann et al., 2018*) is a kind of feature matching AE with the following loss function:

$$\mathbf{J}(\mathbf{x}, \theta) = \frac{1 - SSIM(\mathbf{x}, \hat{\mathbf{x}})}{2} = \frac{1 - l(\mathbf{x}, \hat{\mathbf{x}})^{\alpha} c(\mathbf{x}, \hat{\mathbf{x}})^{\beta} s(\mathbf{x}, \hat{\mathbf{x}})^{\gamma}}{2}, \tag{2.9}$$

where l stands for the luminance measure, estimated by comparing the patches' mean

intensities, c for the contrast responsible for the local variance, and s evaluates the covariance of patches. SSIM is not only utilised in the loss but also in the comparison of the input and outputs:

$$\mathbf{m} = \begin{cases} 1, & \text{if } \frac{1-SSIM(\mathbf{x},\hat{\mathbf{x}})}{2} > Th, \\ 0, & \text{otherwise.} \end{cases} \tag{2.10}$$

Since SSIM already includes the luminance of pixel values, $w_{im} = 0$ in Equation 2.6, colours are neglected. The SSIM index was initially designed to create accurate perceptual quality metrics for comparing two $K \times K$ image patches (*Wang et al., 2004*). In my case, it is not the perceptual information that makes it appealing to use it in the loss function but its efficient extraction of structural information: SSIM-AE is forced to represent, besides luminance, local variance and covariance. According to (*Bergmann et al., 2018*), SSIM-AE could significantly outperform the FM-AE of (*Dosovitskiy et al., 2016*) and a tested VAE.

In my experiments, I investigated several questions:

- Can the results be improved by increasing the number of layers but keeping the size of the latent space?
- How does SSIM-AE perform in the case of less periodic structures?
- What happens if some components are added or removed from the original images?

I made three modifications to the original SSIM-AE:

1. In one modification (named SSIM-AEe), I increased the number of convolutional layers as depicted in Table 2.2 and Table B.2 in the Appendices. I kept the latent layer untouched.
2. In SSIM-DAE, I followed Equation 2.4 to build a denoising AE trained with masked blocks. My SSIM-DAE aimed to implement an implicit regularisation of the network to reconstruct patches from a larger vicinity. There is a low probability that defects affect both the target and source areas during decoding. But either one is distorted $||\hat{x} - \tilde{x}||$ will have a high value.
3. In SSIM-DAEe, I merged the previous two modifications to leverage both strengths.

The structure of the encoder of SSIM-AE is given in Table 2.1, and the decoder has the opposite structure and is given in Table B.1 of the Appendices. Input images have the

following size: w × h × c, where w and h are 128, c is 1 in the Texture datasets, and 3 in the case of the CPU dataset. The size of the latent space z varies along the datasets.

Table 2.1: Architecture of the encoder of SSIM-AE (*Bergmann et al., 2018*).

Layer	**Output Size**	**Kernel**	**Stride**	**Padding**
Input	128×128×c	-	-	-
Conv1	64×64×32	4×4	2	1
Conv2	32×32×32	4×4	2	1
Conv3	32×32×32	3×3	1	1
Conv4	16×16×64	4×4	2	1
Conv5	16×16×64	3×3	1	1
Conv6	8×8×128	4×4	2	1
Conv7	8×8×64	3×3	1	1
Conv8	8×8×32	3×3	1	1
Latent Space	1×1×z	8×8	1	0

Table 2.2: Architecture of the encoder of SSIM-AEe.

Layer	**Output Size**	**Kernel**	**Stride**	**Padding**
Input	128×128×c	3×3		
Conv1	64×64×32	3×3	2	1
Conv2	64×64×32	3×3	1	1
Conv3	64×64×32	3×3	1	1
Conv4	32×32×64	3×3	2	1
Conv5	32×32×64	3×3	1	1
Conv6	32×32×64	3×3	1	1
Conv7	16×16×128	3×3	2	1
Conv8	16×16×128	3×3	1	1
Conv9	16×16×128	3×3	1	1
Conv10	8×8×256	3×3	2	1
Conv11	8×8×256	3×3	1	1
Conv12	8×8×256	3×3	1	1
Latent Space	1×1×z	8×8	1	0

2.4.1 | Training and evaluation

During training, I followed the method and hyper-parameters described in (*Bergmann et al., 2018*) with some improvements. First, I resized the input images to 256 × 256; then, image cropping was applied to the size of 128 × 128. Networks were trained for 200 epochs with Adam optimiser, and the initial learning rate was 2×10^{-4}. The latent space was set to $z = 100$ and $z = 500$ for Texture 1 and Texture 2, and strictly $z = 500$ for the CPU, considering its structural complexity and finer details. The window size of the SSIM was adjusted to 11. Furthermore, the models were trained on grayscale images for the Texture datasets and RGB images for the CPU dataset.

In (*Bergmann et al., 2018*), for both texture datasets, they increased the variability of training data by generating 10,000 images using various augmentation procedures such as rotation, cropping, and flipping (both vertically and horizontally). In my study, I did the same, except in the case of the training on the CPU dataset, where no flip was involved to keep the asymmetric structure. Additionally, I generated 2500 more images for validation purposes for each dataset.

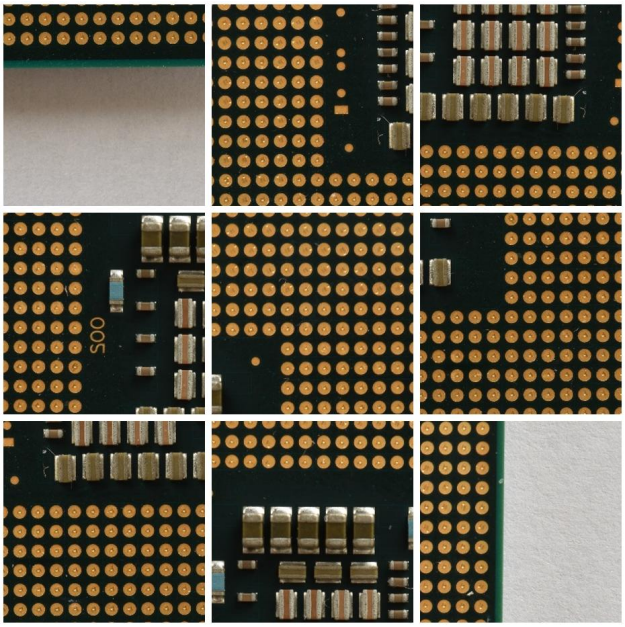

Figure 2.5: 3 by 3 mosaic picture, containing samples from the augmented train set of the CPU dataset.

To train the SSIM-DAE, I added noise to the input images: small, homogeneous masks were added to the original training dataset. The masks were set to 20×20, more prominent

than the size of one capacitor in the middle of the CPU. The same mask size was applied to the Texture datasets. The colour was set to the dominant colour of the actual (128×128) crop. Each training image contained one mask, $\alpha = \beta = 1$ in Equation 2.5.

I have chosen the standard receiver operating characteristic (ROC) and the area under curve (AUC) values for evaluation. I defined the true positive rate (TPR) as the rate of pixels correctly classified as a defect and the false positive rate (FPR) as the ratio of pixels misclassified as a defect. The testing process was accomplished by reconstructing image patches of 128×128 by moving over the test images (with a stride of 32). Residual maps were created using SSIM (Equation 2.10), then ROC curves and AUC values were computed thresholding with increasing SSIM values, from the range of [0, 1] with 50 steps. ROC AUC values are solely computed on test images.

To monitor the reconstruction abilities of the different networks, I also computed the SSIM and mean squared error (MSE) of reconstructions on defect-free training images.

As described before, four kinds of AEs (SSIM-AE, SSIM-AEe, SSIM-DAE, and SSIM-DAEe) were compared in 5 test cases (Texture 1, Texture 2, CPUa, CPUc, CPUm). Results are summarised in Table 2.3, Table 2.4 and Table 2.5 (best values are in bold).

In the case of Texture 1, neither increasing the number of convolutions (SSIM-AEe) nor increasing the size of the latent space (from $z = 100$ to $z = 500$) could make (significant) achievements.

For the Texture 2 dataset, the increased latent space dimension was beneficial since it enhanced the AUC scores of all four models. Additionally, with a higher latent dimension value set, the enlarged network could surpass the original base model architecture.

In general, Textures 1 and 2 are missing fine details (see Figure 2.3); therefore, in cases when the dimension of latent space is not high enough, deeper convolutions are useless.

Table 2.3: ROC AUC values when testing AE methods on dataset Texture 1.

	ROC AUC	**SSIM**	**MSE**
SSIM-AE, z=100	0.9500	**0.8903**	**78.0980**
SSIM-AEe, z=100	0.9471	0.8856	78.6884
SSIM-DAE, z=100	0.9552	0.8849	79.4235
SSIM-DAEe, z=100	**0.9558**	0.8661	82.3891
SSIM-AE, z=500	0.8866	**0.9366**	**66.0230**
SSIM-AEe, z=500	**0.9234**	0.9042	75.1607
SSIM-DAE, z=500	0.9073	0.9262	70.3116
SSIM-DAEe, z=500	0.9232	0.9022	76.4824

Table 2.4: ROC AUC values when testing AE methods on dataset Texture 2.

	ROC AUC	**SSIM**	**MSE**
SSIM-AE, z=100	0.9623	**0.8965**	**80.8407**
SSIM-AEe, z=100	0.9573	0.8705	85.3222
SSIM-DAE, z=100	**0.9626**	0.8901	81.6996
SSIM-DAEe, z=100	0.9599	0.8805	82.2689
SSIM-AE, z=500	0.9641	**0.9514**	**63.8048**
SSIM-AEe, z=500	0.9705	0.9263	74.1510
SSIM-DAE, z=500	0.9674	0.9437	66.7457
SSIM-DAEe, z=500	**0.9705**	0.9231	74.9931

In the case of the CPU tests, I got different results. Comparing SSIM-AE to SSIM-AEe, one can see that increasing the number of convolutional layers could increase the accuracy of reconstruction, but it could not help the ability to detect anomalies (except for CPUm). That is, if a network is better at reconstructing defect-less items, it does not mean it is also better to mishandle defective inputs (thus to detect anomalies).

Since the CPU images had more details, SSIM-AEe could learn the image models more accurately than SSIM-AE (even with the same latent space): it is shown by MSE and SSIM values, measured between error-less inputs and their reconstructions, in Table 2.5. Since CPUa contained minor anomalies, detection went quite well (AUC is between 0.9822 and 0.9952), while CPUc had the worst results. Finally, I found that the larger number of layers and the denoising type training of SSIM-DAEe resulted in the best performance for all CPU test cases (bold in the table).

Table 2.5: Testing AE methods for different distortions on the CPU dataset. AUC values are given in the first three columns, SSIM and MSE is measured as the reconstruction error of error free train samples. SSIM is on the [-1,1] scale, MSE stands for Mean Squared Error.

	CPUa	**CPUc**	**CPUm**	**SSIM**	**MSE**
SSIM-AE	0.9930	0.9071	0.9262	0.7986	94.3443
SSIM-AEe	0.9822	0.8939	0.9557	0.8632	69.8586
SSIM-DAE	0.9914	0.9059	0.9651	0.8216	92.1458
SSIM-DAEe	**0.9952**	**0.9499**	**0.9815**	**0.9510**	**58.9940**

2.5 | Improving the efficiency of autoencoders for visual defect detection with orientation normalisation

The hypothesis is the following; reducing the variability of incoming structures can result in more efficient representation in latent space and better reconstruction quality for defect-free inputs. In this section, I investigate the utilisation of spatial transformer networks (STN) to improve the efficiency of AEs in reconstruction and defect detection. I found that the simultaneous training of the convolutional layers of the AEs and the weights of STNs does not result in satisfactory reconstructions by the decoder. Instead, the STN can be trained to normalise the orientation of the input images. I evaluate the performance of the proposed mechanism on three classes of input patterns by reconstruction error and standard anomaly detection metrics.

The improvement of VAEs with STNs was published in (*Bidart et al., 2019*). However,

questions were raised regarding the simultaneous learning of VAE and STN weights, and its effect on anomaly detection was not investigated. These are my main questions in this section, but I use AEs (instead of VAEs) as the baseline model.

2.5.1 | Spatial transformer networks

STNs were proposed by (*Jaderberg et al., 2015*), designed for the spatial transformation of data within existing network architectures. It was shown that using the spatial transformers results in models that learn invariance to translation, scale, rotation and more generic warpings, leading to state-of-the-art performance on several benchmarks. STN consists of three main parts:

- **Localisation subnetwork**: Its task is to figure out the right transformation parameters θ. It can have any form but should include a final regression layer to produce the transformation parameters. (Typically, affine and perspective transformations are in focus, but I concentrate only on rotations in my study.)
- **Sampling grid generator**: To perform the geometric transformation of the input, each output value should be computed by applying a sampling kernel centred at a particular location in the input. The sampling grid defines how to compute the locations of values on the input.
- **Sampler**: This part computes the result of transforming the input according to the sampling grid.

STNs have been found to be useful in improving many classical image processing tasks. For example, in (*Li et al., 2018*), the recognition of jersey numbers in football videos was improved by adding STN to the front end of a classification network. A semi-supervised mechanism was proposed to help the learning phase, where the localisation error was part of the loss function, besides softmax classification loss. In (*Lee et al., 2019*) the goal was to learn a complex function that maps the appearance of input image pairs to the parameters of the spatial transformation in order to register anatomical structures. To guide the registration of the moving image to the fixed one, alternatives of the loss function contained prepared structures-of-interest regions (segmentations or landmarks) beside the pixel-wise registration error.

Discussing all applications is beyond this dissertation. Nevertheless, there are interesting

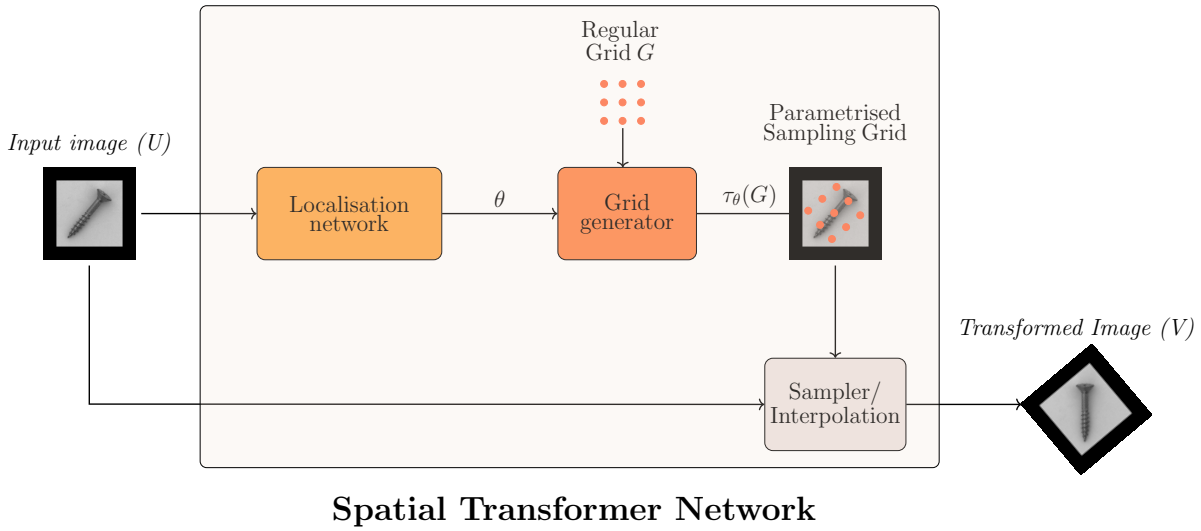

Figure 2.6: The base STN architecture, where U is the input image or image features, and V is the transformed version of them.

ones, such as grasp detection (*Park et al., 2018*), image retrieval (*Ding et al., 2020*), or change detection (*Chianucci et al., 2016*). Other applications include traffic sign classification (*Tiryaki et al., 2025*) or agricultural object detection (*Zambre et al., 2024*). The most similar approach to mine is (*Bidart et al., 2019*), where VAE networks were extended with STNs. Their findings are discussed in the following subsection.

2.5.2 | STNs with autoencoders

It seems straightforward that inserting an STN at the front end of an AE's encoder and the inverse transformation after the AE's decoder can result in better reconstruction since the STN might learn the best spatial transformation warping images to similar scale, position, and orientation. This way, the AE meets less variability of input patterns, can more easily learn the necessary convolutions, and can finally make a better reconstruction. Figure 2.7 illustrates this configuration.

Unfortunately, to simultaneously find the optimal weights for either the localisation network and the AE is not guaranteed. The same structure was used in (*Bidart et al., 2019*), but instead of AE, a VAE was used. I quote the description of this problem from (*Bidart et al., 2019*):

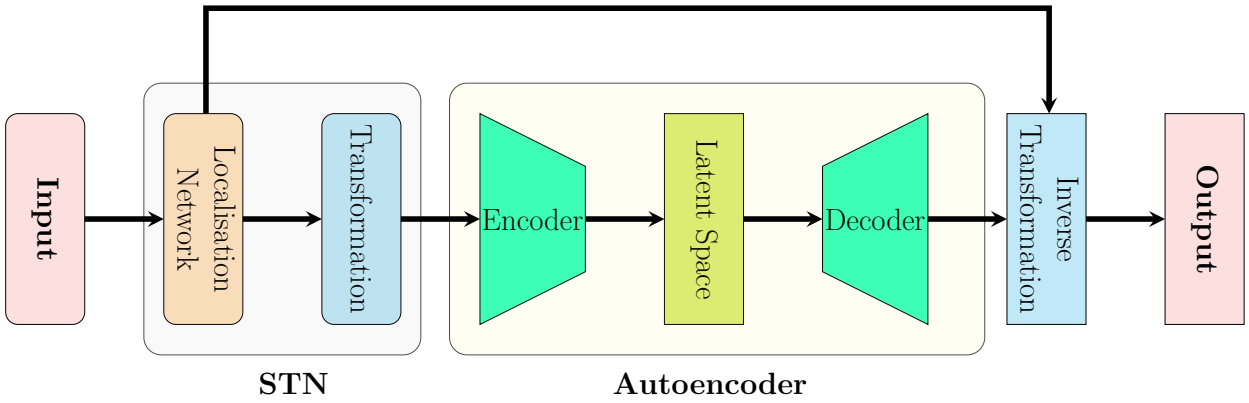

Figure 2.7: STN inserted into an AE with the inverse spatial transformation.

> *"We find in practice there are issues with the optimiser being caught in local minima, so we use multiple random restarts, where we first try the loss at a set number of affine parameters, and only perform gradient descent on the best performing parameters."*

Moreover, they investigated only images from the MNIST database of handwritten digits, which are almost binary patterns from a small size set of classes. Also, it is a question of how the combination of STNs and AEs perform in the case of more complicated structures (such as grayscale textures, for example) and what effect all these have on anomaly detection. Since my AEs are fully convolutional and test images have the same scale, only the orientation of input patterns should be compensated by the STN. I can avoid the learning of translation and scaling parameters.

To investigate the simultaneous optimisation of AE and STN weights, I tested three strategies:

- Both the STN and AE were initialized with random weights, and the loss function used the squared version of Equation 2.3, with no regularization ($R = 0$).
- I separated the STN from the AE during training. STN was pre-trained to normalise the orientation of input patterns, which is detailed in the following section. During the training of the AE, the localisation network was frozen.
- The same as above but allowed the training to refine the weights of the STN.

2.5.3 | Normalisation with STNs

The purpose of normalisation STN is to rotate the inputs to the same orientation. To achieve this, I built an STN with a loss function based on image gradients (see Figure 2.8 for visualisation).

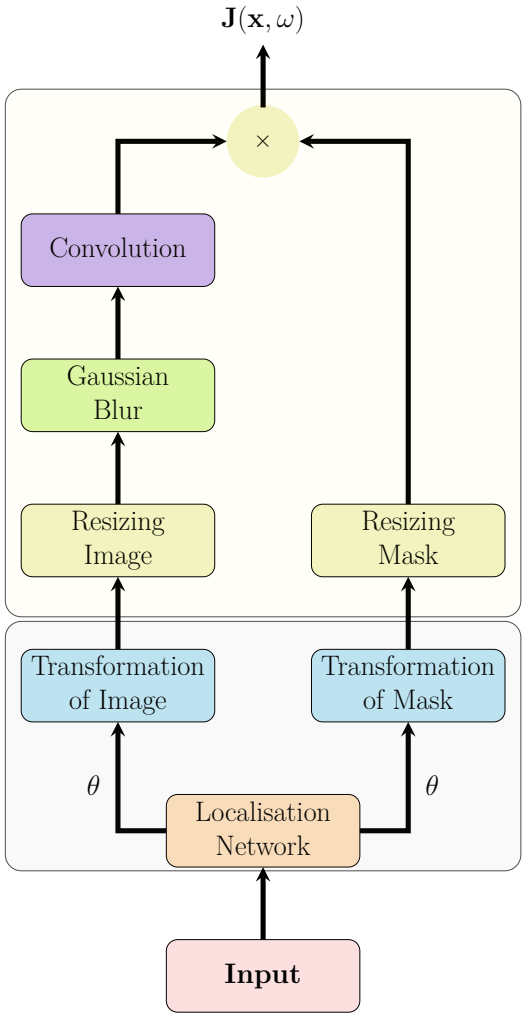

Figure 2.8: STN with custom loss function as specified by Equation 2.12.

Local image gradients were computed with a convolution kernel on the re-scaled variant (64×64) of the image. To sub-press image noise and small details, Gaussian blur was applied before computing the horizontal gradients by the vertical Sobel filter:

$$\mathbf{G}_{\mathbf{x}} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{2.11}$$

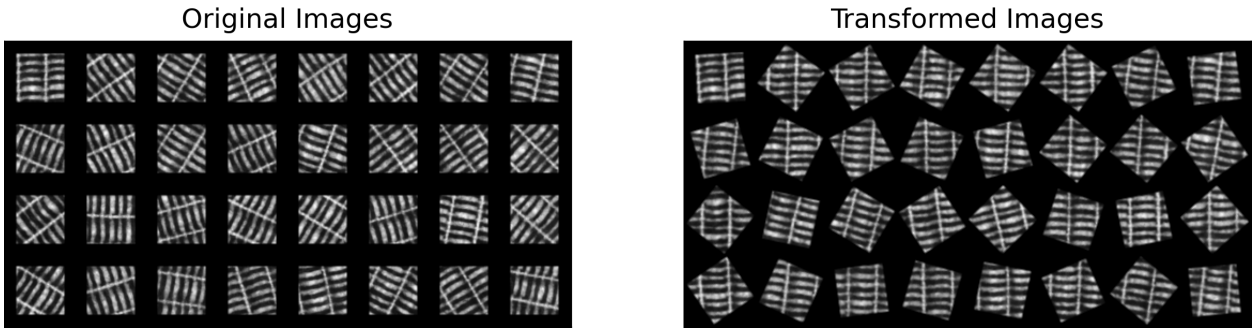

Figure 2.9: Input images from class Texture 1 and their STN rotated counterparts.

The loss function of the STN is defined as:

$$\mathbf{J}(\mathbf{x}, \omega) = \frac{1}{n} \sum |(\mathbf{M} \circ \hat{\mathbf{x}}) \circledast \mathbf{G}_{\mathbf{x}}|, \tag{2.12}$$

where $\hat{\mathbf{x}}$ is the output of the STN network, and $\mathbf{M}$ is a rotated mask for the central region of the zero-padded image. The zero padding is necessary to avoid losing information when some parts of the rotated image lay outside the original frame. See illustration in Figure 2.9. The element-wise product with $\mathbf{M}$ eliminates possible high gradient areas where the padding region meets the central content.

Figure 2.10 and Figure 2.11 illustrate the loss values for two images from the Texture 1 class. In the latter example, we can see two minima of the function with approximately 10 degrees difference, but both functions are pretty smooth. This means that we do not expect always to find exactly the same orientation, but the function's smoothness and the differentiability of STNs (*Jaderberg et al., 2015*) can result in almost the same orientation.

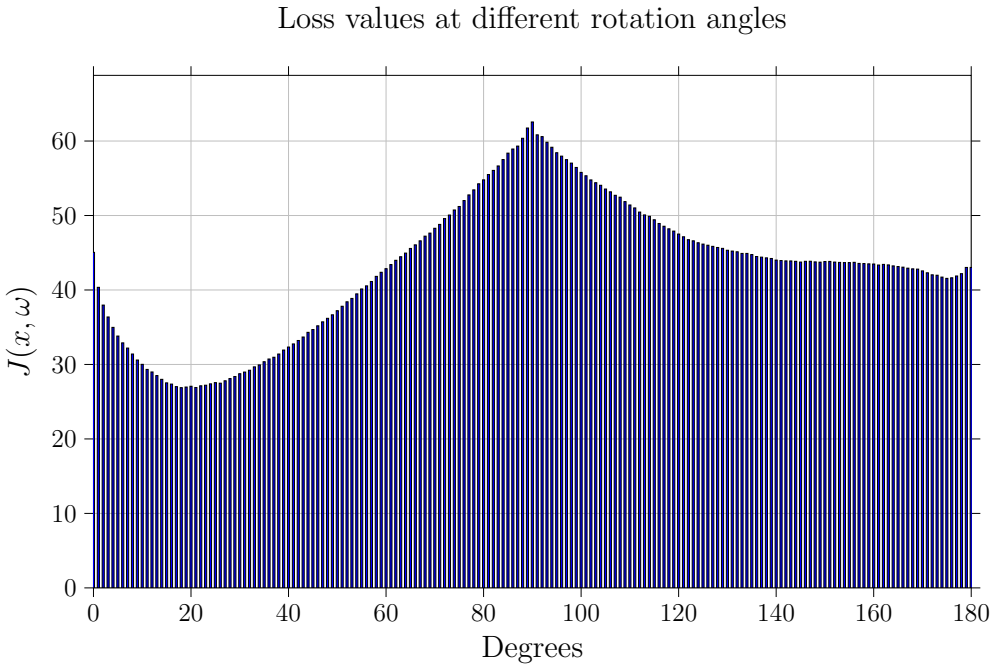

Figure 2.10: Illustrations for the loss value for two images from class Texture 1 at different rotation angles.

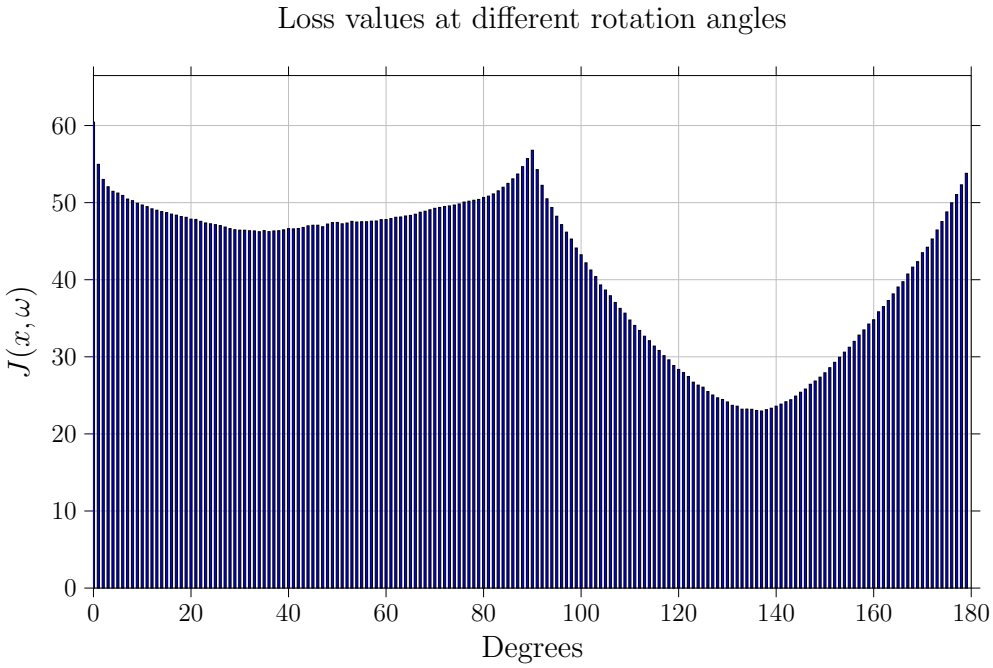

Figure 2.11: Illustrations for the loss value for two images from class Texture 1 at different rotation angles.

2.5.4 | Localisation network

There is no specific rule of thumb regarding the architecture of a localisation network (except for its final regression layer); fundamentally, it can be either a convolutional neural network or a fully connected network. The architecture of my localisation network can be seen in Table 2.6.

Table 2.6: Localisation network layers.

Layer	Output size	Kernel	Stride	Padding
Conv1	$8 \times 58 \times 58$	7×7	1	0
MaxPool 1	$8 \times 29 \times 29$	2×2	2	-
Conv2	$10 \times 25 \times 25$	5×5	1	0
MaxPool 2	$10 \times 12 \times 12$	2×2	2	-
Conv3	$10 \times 8 \times 8$	5×5	1	0
MaxPool 3	$10 \times 4 \times 4$	2×2	2	-
Flatten	160	-	-	-
Fully Connected 1	160×32	-	-	-
Fully Connected 2	32×1	-	-	-

Its task is to obtain the θ matrix, which holds the spatial transformation parameters. As explained above, in my implementation, I limited θ to rotations, that is:

$$\theta = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \end{bmatrix} \tag{2.13}$$

As for the θ^{-1}, I extend the transformation matrix with a row of $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$, calculate the inverse, and then crop the unnecessary row.

For training the STN, I fed 10,000 images to the network. The image size was set to 128×128. Using the Adam optimiser, the network was trained for 10 epochs, where the learning rate was set to 1×10^{-2}, and no weight decay was applied. At the end of the training, I saved the network weights for later utilisation and checked the quality of the training on 2500 test images.

2.5.5 | Testing the reconstruction accuracy and defect detection of STN with AE

In this section, I first investigate the reconstruction abilities of the differently trained variants of the structure given in Section 2.5.2 and illustrated by Figure 2.7.

Four cases are under investigation:

1. For reference, I trained the plain AE defined in Table 2.1.
2. Both STN and AE are initialised with random weights.
3. STN is pre-trained to normalise the orientation of input patterns. During the training of the AE, the localisation network is frozen.
4. The same as above but allowing the refinement of STN weights.

These four cases correspond to the four columns in Table 2.7, and in the Appendices, Table B.3 Table B.4, where SSIM and mean squared error (MSE) values are computed as the mean of 2000 test images. Lower MSE and higher SSIM mean better quality. Hyperparameters were the same in all cases: Networks were trained on NVIDIA RTX 5000 GPU for 200 epochs - except where early stopping was intervened - with Adam optimiser, the initial learning rate was 2×10^{-4} with 10^{-5} weight decay. The latent space was set to $z = 100$. All experiments were repeated 5 times; the data given are average values. (Abbreviations in Table 2.7, Table 2.8 and Table 2.9 and in the Appendices, Table B.3, Table B.4, Table B.5 and Table B.6 are the following: S: STN; rnd. in. S w.: randomly initialised STN weights; pre-tr. frz. S w.: pre-trained frozen STN weights; pre-tr. n. frz. S w.: pre-trained not frozen STN weights.)

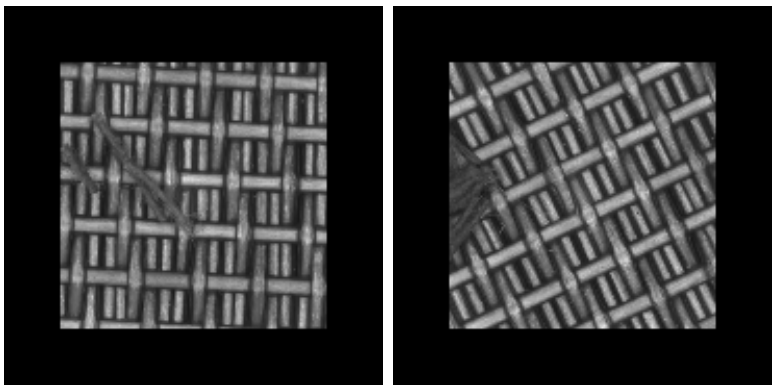

Figure 2.12: Test images from the Texture 2 dataset.These examples shows some defects to be detected.

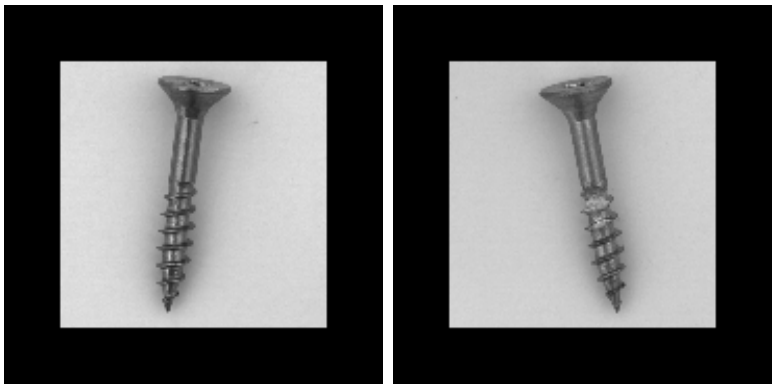

Figure 2.13: Train and test images from the Screw dataset.

First, images were resized to 256×256 pixels, and then patches of size 128×128 were created. At this point, I applied various augmentation procedures such as rotations and cropping. It was vital in order to enrich my dataset. Then, I added zero-padding with the minimally necessary size to avoid clipping the content when rotating them by the STN: this resulted in an image size of 181×181 pixels. Finally, these images were resized to 128×128. For the illustration of the three classes, see Figure 2.9, Figure 2.12, and Figure 2.13; each class contained finally 10,000 images used for training and validation.

Table 2.7: Reconstruction quality of images of the class Texture 2. Abbreviations are given in the text.

	Texture 2			
	AE	**S+AE pre-tr. frz. S w.**	**S+AE rnd. in. S w.**	**S+AE pre-tr. n. frz. S w.**
MSE	88.3534	**86.1341**	89.9087	87.9534
SSIM	0.7915	**0.8174**	0.7675	0.7911

Following the observation of (*Bidart et al., 2019*), quoted in Section 2.5.2, I found that in all three image classes, the insertion and random initialisation of the STN (and the insertion of the corresponding inverse transformation) could not improve reconstruction quality, in all cases I got worse results. Pre-training the STN with the method introduced in Section 2.5.3 improved quality compared to the plain AE. Moreover, allowing the fine-tuning of STN weights, simultaneously with the training of the AE, resulted in lower quality in two of the three cases.

To further evaluate the proposed method, I tested defect detection using the ground truth defect images of the sources named in Section 2.3. Three well-known pixel-level metrics were used for comparisons: receiver operating characteristic (ROC) curves, precision-recall curves (PRC), and intersection over union (IoU). In the case of all curves, the area under the curve (AUC) was calculated. For more details about these metrics, see Section A.1 of the Appendices. Residual maps (between the inputs and the decoded and inverse transformed outputs) were created using SSIM and MSE with different thresholding, resulting in the above-mentioned curves. The differences were computed only on the central regions of the padded test images (excluding the zero-padding areas).

Table 2.8: Defect detection using SSIM and MSE metrics for the Texture 2 class. Abbreviations are given in the text.

		Texture 2			
		AE	**S+AE pre-tr. frz. S w.**	**S+AE rnd. in. S w.**	**S+AE pre-tr. n. frz. S w.**
SSIM	**ROC**	0.8379	**0.8409**	0.8216	0.8253
	PRC	0.2785	**0.3228**	0.2648	0.2683
	IoU	0.1148	**0.1242**	0.1113	0.1141
MSE	**ROC**	0.5826	**0.6087**	0.5833	0.5879
	PRC	0.1795	**0.1976**	0.1710	0.1808
	IoU	0.0661	**0.0702**	0.0647	0.0656

Table 2.9: Rankings of different defect detection approaches. Abbreviations are given in the text.

		Ranking			
		AE	**S+AE pre-tr. frz. S w.**	**S+AE rnd. in. S w.**	**S+AE pre-tr. n. frz. S w.**
Texture 1	1st	1	**4**	0	2
	2nd	1	1	0	**3**
	3rd	1	1	**3**	1
	4th	**3**	0	**3**	0
Texture 2	1st	0	**6**	0	0
	2nd	**4**	0	0	2
	3rd	1	0	1	**4**
	4th	1	0	**5**	0
Screw	1st	0	**6**	0	0
	2nd	**3**	0	0	**3**
	3rd	2	0	**5**	0
	4th	1	0	1	**3**

Results regarding the three test classes can be seen in Table 2.8, and in the Appendices, Table B.5 and Table B.6. Best values are highlighted in bold. As can be seen, in all metrics, the proposed S+AE (pre-tr.frz. S w.) configuration outperformed the plain AE in the Texture 2 and Screw classes. On the other hand, in the Texture 1 class, the plain AE scored the highest IoU value when the SSIM metric was selected. However, since the SSIM detection mechanism showed significantly better results, using MSE as the error detection metric would not be reasonable. The overall ranking of the methods can be seen in Table 2.9: for two test classes, the proposed technique was better in all measures; for the Texture 1 class, it was best in 4 cases, once second and once third.

2.6 | Summary

In this chapter, I investigated the performance of different SSIM AEs in defect detection. I implemented several modifications of the proposed AE of (*Bergmann et al., 2018*), including

constructing a DAE, enlarging the existing network and combining these two steps. I also dealt with applying STNs to boost the performance of AEs for defect detection. STNs have shown promising results in many applications, such as classification, change detection, and image registration. However, it was not clear how the requirement of accurate image encoding/decoding could be achieved or whether the simultaneous learning of AE and STN weights is applicable. The answer to the latter is negative.

To summarise my observations, I found that:

1. If an AE can reproduce (anomaly-free) images with higher fidelity, it does not strictly induce it to be better in anomaly detection. It is an important observation since AE's main application area is unsupervised anomaly detection, where no examples of defects are available to design and train optimal neural networks.

2. Unsupervised AEs are unreliable for (at least minor) contamination or superfluous parts; DAEs may better fit many defect detection tasks.

3. I also observed that the optimal dimensionality of the latent space is based on the characteristics of the given dataset. Lower dimensional latent spaces tend to be more effective for datasets with simpler structures (such as Texture 1). Conversely, higher-dimensional latent spaces generally yield superior performance for datasets comprising more complex and structured images (for instance, the Texture 2 dataset).

4. Furthermore, in cases where the latent space is sufficiently selected, larger networks with more layers and parameters often achieve better results compared to networks with fewer layers and parameters.

5. A pre-trained STN, to normalise the orientation of input patterns, can improve the reconstruction and defect detection, even if the input patterns produce multiple minima in the loss ($\mathbf{J}(\mathbf{x}, \omega)$) when applying the orientation normalising filter mechanism.

To underlie the above statements, I tested the models on repetitive and semi-repetitive structures on four different datasets. The proposed DAE model outperformed the base SSIM-AE network in all the experiments executed on the Texture and CPU datasets in terms of ROC AUC. My proposed normalisation approach could outperform the base AE method in almost all cases and metrics.

Chapter 3

Fast and Scalable Deep Randomized Neural Networks

In the era of the heavy use of optimisation techniques, it may seem strange that random weights can play a valuable role in constructing efficient ANN models. First, however, we should remember that the famous pioneering Mark I Perceptron machine (*Rosenblatt, 1958*) from 1958, based on the idea of McCulloch and co-authors in (*McCulloch et al., 1943*) of using neural models to solve computation problems, used random weights to connect input photocells and hidden layer neurons. On the other hand, random neural networks are inspired by the brain's plasticity, which can efficiently adapt and learn from various inputs. This adaptability, the robust ability supported by random connections to handle diverse and noisy data, allows for the creation of robust classifiers. Published more than three decades ago in (*Schmidt et al., 1992*), then in (*Pao et al., 1994*), and also reinvented as extreme learning machines (ELMs) (*Huang et al., 2006*) in the 2000s, random networks get some popularity periodically. Although we know that even the most popular gradient optimizers rely on randomness (shortly discussed in Section 3.1.1), since their first appearance, different new approaches have emerged (e.g. (*Zhu et al., 2005*), (*Rong et al., 2008*), (*Miche et al., 2009*), (*Cao et al., 2011*), (*Han et al., 2014*), (*Tang et al., 2015*), (*Rádli et al., 2023b*)) where random weights play a significant role in solving classification or regression problems. In addition to the seemingly unexpected efficiency of using random connections, these networks' fast learning ability was very tempting initially.

On the other hand, the biggest limitation of the first random networks was their shallow depth. For years, only single hidden layer models were available, which prevented their use

for more complex tasks.

In general, the improvement in the problem-solving accuracy of ANNs stems from the increasing number of neurons and the deeper and more sophisticated structures. Compared to the best (shallow) network in the ImageNet challenge, the 8-layer AlexNet (*Krizhevsky et al., 2012*) network reduced the Top-5 error by 9.4% in 2012. In 2015, ResNet (*He et al., 2016*), ahead of even human performance, with a Top-5 error rate of only 3.6% (a further 12.8% drop), already had 152 layers.

Shallow neural networks are not competitive with deep neural networks (DNN), except in some cases; for example, the head of a deep convolutional network often consists of only a few fully connected layers. Training these shallow heads with random techniques can be done in a very short time, which in some application domains (e.g. (*Nagy et al., 2022*)) can give an important advantage over traditional training methods, which are typically trained over long epochs by backpropagation.

In this chapter, I will first review the most relevant literature on the subject, the motivations behind the method and the method itself. Finally, I will present the experiments I have carried out. Furthermore, I encourage the reader to read the additional material related to this chapter in the Appendices, Section C.2, Section C.4 and Section C.5.

3.1 | Related works

For finding the appropriate weights of networks, besides the widely used gradient optimisation methods, there is a large number of alternatives, such as genetic algorithms, differential evolution, particle swarm optimisation, artificial bee colony, biogeography-based optimisation (also others motivated by biological habitant), and metaheuristics.

My purpose is to revisit and improve such solutions, which apply randomly generated weights in trained ANNs. My overview will focus on those, especially on DNNs. I deal only with supervised learning and do not discuss the possibilities of applying my proposed ideas to un-, semi-, weakly–, or self-supervised or reinforcement learning methods.

To emphasize the difference between traditional network training approaches and randomized networks, I first briefly review the two most popular gradient techniques and then discuss the different randomized approaches.

3.1.1 | A few words about the randomness of gradient optimizers

At first glance, one may wonder how gradient algorithms can be useful for non-convex, high-dimensional optimisation problems. In short, the answer can be summarized in three points:

- In high-dimensional spaces, local minima are rare, while most critical points are saddle points. The curvature is negative in at least one direction at saddle points. As a result, the gradient optimisation can proceed further out of these locations (*Dauphin et al., 2014*).
- Randomness (if present) helps us not to get stuck in shallow or bad minima. Noise constantly "jolts" the network, causing the algorithm to explore the search space and avoid awful solutions.
- Deep neural networks are not random, non-convex optimisation problems but have a specific structure that can benefit optimisation. DNNs are typically over-parameterized, and if a network has many parameters (more than the number of samples in the dataset), there may be several equivalent optimal solutions (*Allen-Zhu et al., 2019*).

Thus, randomness is crucial for high-dimensional convex problems such as DNNs. The widely used stochastic gradient descent (SGD) optimisation (*Robbins et al., 1951*), while based on the gradient descent optimisation approach of Augustin-Louis Cauchy from 1847 (*Lemaréchal, 2012*), randomly selects batches of input data. This avoids processing the entire dataset during each iteration; only a few samples from the dataset are considered at a time, resulting in efficient and computationally feasible optimisation. Since typically a small part of data is used in each step, updates are noisier, which can be beneficial to avoid local minima. The Adam (adaptive moment estimation) algorithm (*Kingma, 2014*), an extension of the SGD method, combines the advantages of two other popular optimisation techniques, AdaGrad and RMSProp. It is designed to adjust the learning rate for each network weight adaptively. Unlike SGD, which maintains a single learning rate throughout training, Adam dynamically computes individual learning rates based on past gradients and their second moments. This adaptivity results in faster convergence and improved performance in some neural networks, while SGD has better generalisation properties in most cases. For the comparison and deeper understanding of SGD and Adam I refer to (*Zhou et al., 2020*),

(*Gupta et al., 2021*), while recent advances of these traditional optimizers can be found in (*Tian et al., 2023*).

However, there are important specific problems with SGD and Adam, such as slow convergence, oscillation (SGD), difficulty in setting learning rate (SGD), bad performance for rare data (SGD), bad generalisation, overfitting (Adam), low learning rate (Adam), lazy training (Adam) (*Chizat et al., 2019*), the problem of vanishing or exploding gradients can easily happen to both, and the training is time-consuming in both cases.

Please note that while in these (implicitly randomized) methods, network parameters are converged, the network structure is unchanged, contrary to my approach, where the structure develops in time. Also, while these methods involve implicit randomness, my method explicitly applies random weights for the final solution of models. In addition, the reader is reminded that random initialisation of weights is often used in gradient methods, which is very different from the situation where a large part of the weights remain random throughout the whole learning process. In the former, the aim is to avoid the so-called weight symmetry problem and ensure a proper gradient flow; in the latter case, fixed random weights are considered a kind of priors.

3.1.2 | Randomized networks

As mentioned in the Introduction, the Mark I Perceptron machine (*Rosenblatt, 1958*) already used random weights to connect input photocells and hidden layer neurons to recognise optical signals. Almost 40 years later, published in (*Schmidt et al., 1992*), a single-layer feed-forward neural network (SLFN) was introduced, where the input data were assigned random weights. In the output layer, a fully connected network with bias was applied, and its weights were determined by solving a system of linear equations using numerical methods. A few years later, in (*Pao et al., 1994*), the random vector functional links (RVFL) network was proposed, which, in addition to processing the original input patterns, also generated and fed randomised versions of the inputs to the output layer through weighted connections. Here, the weights were determined by a gradient method. Both architectures demonstrated strong predictive performance in various experiments. Published more than a decade later in (*Huang et al., 2006*), the so-called extreme learning machines (ELMs) exhibit subtle differences from these randomised networks: the absence of direct connections between inputs and outputs (as seen in RVFL) and a different approach to bias compared

to (*Schmidt et al., 1992*). ELMs have gained greater popularity in recent years, despite that RVFL outperformed ELM in certain cases (*Suganthan et al., 2021*). In any case, these models performed relatively well. However, their main practical advantage was that they did not need to use the relatively slow error backpropagation technique to determine the weights and were, therefore, able to provide a very fast solution.

The main idea of ELM methods is that for a sufficiently large hidden layer, any function can be approximated well by transforming the input into a higher dimensional space. If the randomly generated weights are well distributed over the inputs in a higher dimensional space, then the output layer (computed with a pseudo-inverse technique or the FISTA algortihm (*Beck et al., 2009*)) will be able to learn the task by relatively simple linear combinations. It is mathematically similar to the kernel trick (*Aizerman, 1964*) applied in Support Vector Machines (SVM), where data is projected into a non-linear space (*Boser et al., 1992*).

There are strong theoretical foundations and validations to support the success of randomised network models. It is vital to mention the background work analysing the function approximation abilities of randomised networks such as in (*Gelenbe et al., 1999*), (*Huang et al., 2011*), (*Neufeld et al., 2023*), and (*Needell et al., 2024*). It is worth noting that the randomisation of features is not only relevant in the field of neural networks but also plays a significant role in training kernel machines on large datasets, such as SVMs, as demonstrated in Rahimi and Recht's paper (*Rahimi et al., 2007*). For a more profound overview of randomised neural networks, I encourage the reader to read the work of (*Gallicchio et al., 2020*) and (*Cao et al., 2018*).

In (*Liu et al., 2018*), a multi-modal extreme learning machine was introduced with local receptive fields on RGB and depth images (Multi-Modal Local Receptive Field Extreme Learning Machine: MM-LRF-ELM) for object recognition. While it maintains ELMs' advantages of training efficiency, the MM-LRF-ELM is relatively shallow. Since it is adopted for images, feature extraction is randomised by the convolutional kernels, while the final classification weights are computed by matrix inversion as in (*Huang et al., 2006*).

There were doubts about how good the random weights were, so different approaches have emerged. I focus on two of them, most relevant to my study:

- Pruning: Since too few hidden nodes would lead to under-fitting while too many to overfitting in (*Rong et al., 2008*), the pruned-ELM (P-ELM) algorithm is proposed.

P-ELM uses statistical methods to measure the relevance of hidden nodes. From an initially large number of hidden nodes, irrelevant nodes are pruned by considering their relevance to the class labels. The method optimally pruned extreme learning machine (OP-ELM), proposed by Mich et al. (*Miche et al., 2009*) selects relevant nodes using least angle regression ranking of the nodes. The selection of the final model structure is achieved through leave-one-out validation in the last step of OP-ELM, which selects the number of neurons.

- Evolutionary algorithms: Both in (*Zhu et al., 2005*) and (*Cao et al., 2011*), a differential evolutionary algorithm is used to search for the optimal network parameters and to reduce the number of hidden nodes. Unsurprisingly, both require maintaining a population of networks and applying evolutionary principles across generations, significantly increasing the learning time.

While these later methods use biological analogies, I would rather avoid the very long processes of evolution and focus on development processes, which more rapidly result in complex structures known for better problem-solving:

- the neural model incrementally expands and develops in stages, similar to biological development (e.g. embryonic development),
- it can be characterised by self-organisation and formation of new connections,
- while it has a kind of random nature, it follows rules and processes.

In the Appendices, Section C.1 details my motivations related to biological analogies.

3.1.3 | Deep randomized networks

Despite the initial success of random models, by the early 2010s, it was clear that deep learning networks would be much more successful, so attempts were made to deepen random networks. Their single hidden layer structure was not satisfactory for more complex problems; there was a definite need for a kind of hierarchy in the structure. In this discussion, I start with top-down approaches, where the possibility of random weights in larger networks is considered. Bottom-up approaches act opposite from smaller to larger networks by using different construction techniques.

3.1.3.1 | Top-down approaches

The Lottery Ticket Hypothesis (LTH) proposed by Frankle and Carbin (*Frankle et al., 2018*) states that a randomly initialized dense neural network contains sparse subnetworks—called "winning tickets"—which, when trained in isolation from their original initialization, can achieve performance comparable to the full network. This indicates that larger networks foster training by providing a higher chance of having subnetworks that can be successfully optimized from their original initialization.

In (*Giryes et al., 2016*), the properties of deep networks under the assumption of random weights are studied. It is proved that random DNNs preserve the metric structure of the data as it propagates along the layers, allowing for the stable recovery of the original data from the features calculated by the network.

A few years later, in 2019, as an interesting study, the authors in (*Rosenfeld et al., 2019*) studied the learning of a deep network in which a large percentage of weights were kept at their original random values. They showed that for modern deep CNNs, it is possible to fix up to nine-tenths of the parameters and adjust only the remaining 10%, in many cases suffering only little loss of accuracy. Moreover, authors of (*Zhou et al., 2019*) made an analysis of the LTH (*Frankle et al., 2018*) and found something even more interesting: they observed the existence of *supermasks*. These masks can be applied to the nodes of an untrained, randomly initialized network to produce a model with performance far better than chance. To find supermasks, the authors of (*Ramanujan et al., 2020*) proposed an " edge-popup " method. Here, all the weights in the network are set randomly and never updated. Scores are assigned to edges, and then only a subset of the highest-scoring edges remain active; the rest are deleted. Scores are learned (not weights) based on the gradient obtained from the loss function. If the score of a deleted edge increases, it can be recovered; if it decreases, it can disappear again. Ultimately, the best edges remain active, and the network becomes functional with random weights. This approach was tested on different datasets with inspiring results. While this technique is interesting, its stability raises some questions, and the solution also uses the iterative backpropagation technique.

3.1.3.2 | Bottom-up approaches

To increase the depth of ELMs, multilayer ELMs (ML-ELM) were introduced in (*Cambria et al., 2013*) using a sequence of basis transformations where the new basis vectors were

generated by AE called ELM-AE. The encoder network $E : \mathbb{R}^h \to \mathbb{R}^d$, the latent space $\mathbb{R}^d$, and the decoder network $D : \mathbb{R}^d \to \mathbb{R}^h$ form a computation mechanism for the compact representation of input information by $\mathbf{z}$:

$$\hat{x} = D(E(x)) = D(z) \tag{3.1}$$

where $\mathbf{x}$ is the input data, $\mathbf{z}$ is the latent information, and $\hat{\mathbf{x}}$ is the output (approximated input) of the ELM-AE network. Randomness plays a role as the weights and the bias of the encoder are randomly generated. These random weights were chosen to be orthogonal, but this condition was omitted in later variations (*Tang et al., 2015*). The weights of the decoder were computed as in (*Huang et al., 2006*), and then the transposed weights of the decoder were used as weights for the hidden layer representation of input data in the final ELM model. Replacing the random weights of ELMs with these means a more data-oriented random transformation. After a few such consecutive steps (two in the published article), the obtained representation was connected to the output with weights computed again in the same way as above.

While the ML-ELM approach was tested only in one dataset (MNIST), a slightly modified version, called hierarchical ELM (H-ELM) (*Tang et al., 2015*), was evaluated on several datasets and could outperform previous variants. In addition to leaving the orthogonality condition for the random weights of the ELM-AE, the L_2 norm was replaced by L_1, and the FISTA algorithm (*Beck et al., 2009*) was used to compute the output weights instead of the matrix inversion technique. Here, I note that the obtained representation of input data was finally connected to a normal ELM in this case, as opposed to the ML-ELM above, where the random input weights of ELM were omitted.

In the hierarchical approach called H-LR21-ELM (*Li et al., 2021*), a specific combination of L_{21} norm-based loss function and regularisation is used to improve the robustness and the sparsity of H-ELMs. While in most optimisations, the L_2 norm is used, we have already seen (*Tang et al., 2015*) that the L_1 norm can improve results. The proposed L_{21} norm in (*Li et al., 2021*) is claimed to make the H-ELM model less sensitive to outliers and impulsive noises pervasive in real-world data. Moreover, the L_{21} regularisation can learn the most relevant sparse representations to reduce the intrinsic complexity of models.

Autoencoders are often used for the generation of compact data representation; in my context, it is noteworthy to mention the method published in (*Tissera et al., 2016*), where

the labels for the classification are themselves encoded into the images by some dedicated pixels to label the classes.

Further interesting approaches are elaborated in Section C.3 of the Appendices.

3.2 | Implementation

My developmental model is based on stacking SLFN networks to refine the previous stage. For a better understanding, I start with a description of the formulation of an SLFN, then with the pruning method, and finally, I will discuss how to stack new layers to a previous stage. The steps of the full process of my methodology is given in Algorithm 2.

3.2.1 | The initial phase of creating the SLFN as a building block

First, let us formalise the basic idea behind SLFN random networks according to (*Huang et al., 2006*). Consider a set of N distinct training samples $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]^\top \in \mathbf{R}^n$ and $\mathbf{y}_i = [y_{i,1}, y_{i,2}, \ldots, y_{i,m}]^\top \in \mathbf{R}^m$.

Then, a SLFN with L hidden neurons has the following output equation:

$$\sum_{i=1}^{L} \beta_i \phi(\alpha_i \mathbf{x}_j + b_i) = \mathbf{o}_j, \quad j = 1, \ldots, N \tag{3.2}$$

where ϕ is an activation function, $\alpha_i = [\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,n}]$ is the weight vector connecting the i_{th} hidden node and the input nodes, b_i are the biases, $\beta_i = [\beta_{i,1}, \beta_{i,2}, \ldots, \beta_{i,m}]$ are the weight vectors connecting the i_{th} hidden node and the output node, and $\mathbf{t} = [o_{i,1}, o_{i,2}, \ldots, o_{i,m}]$ is the target vector (the outputs for the different classes). A conventional SLFN with L hidden nodes and activation function $\phi(x)$ can achieve zero training error for these N samples if appropriate weights and biases are chosen. In other words, there exists a weight and bias configuration such that $\sum_{i=1}^{L} ||\mathbf{o}_i - \mathbf{y}_i|| = 0$ such that,

$$\sum_{i=1}^{L} \beta_i \phi(\alpha_i \mathbf{x}_j + b_i) = \mathbf{y}_j, \quad j = 1, \ldots, N \tag{3.3}$$

We can shorten the equation using matrices as:

$$\mathbf{H}\beta = \mathbf{Y}, \tag{3.4}$$

where $\mathbf{H}$ is called the hidden layer output matrix of the neural network, and can be written as:

$$\mathbf{H} = \begin{bmatrix} \phi(\alpha_1 \mathbf{x}_1 + b_1) & \cdots & \phi(\alpha_L \mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ \phi(\alpha_1 \mathbf{x}_N + b_1) & \cdots & \phi(\alpha_L \mathbf{x}_N + b_L) \end{bmatrix}_{N \times L}, \tag{3.5}$$

$$\beta = \begin{bmatrix} \beta_1^\top \\ \vdots \\ \beta_L^\top \end{bmatrix}_{L \times m}, \quad Y = \begin{bmatrix} y_1^\top \\ \vdots \\ y_N^\top \end{bmatrix}_{N \times m} \tag{3.6}$$

Matrix inversion is not straightforward for arbitrary matrices (even not defined for matrices that are not square). The Moore-Penrose pseudoinverse allows us to make some headway in these cases. To find the "best fit" (least squares) solution to the system of linear equations the pseudoinverse is computed (*Moore, 1920*):

$$\mathbf{H}^\dagger = \lim_{\alpha \to 0} (\mathbf{H}^\top \mathbf{H} + \alpha \mathbf{I})^{-1} \mathbf{H}^\top. \tag{3.7}$$

Finally, we get the weights:

$$\tilde{\beta} := \mathbf{H}^\dagger \mathbf{Y}. \tag{3.8}$$

Alternatively, to apply regularisation on β:

$$\mathbf{H}^\dagger = \begin{cases} (C^{-1}\mathbf{I} + \mathbf{H}^\top \mathbf{H})^{-1} \mathbf{H}^\top & if N > L \\ \mathbf{H}^\top (C^{-1}\mathbf{I} + \mathbf{H}\mathbf{H}^\top)^{-1} & if N < L \end{cases} \tag{3.9}$$

Where C is a scalar regularisation constant for the control of magnitude of weights (*Huang et al., 2011*) and $\mathbf{I}$ is the identity matrix.

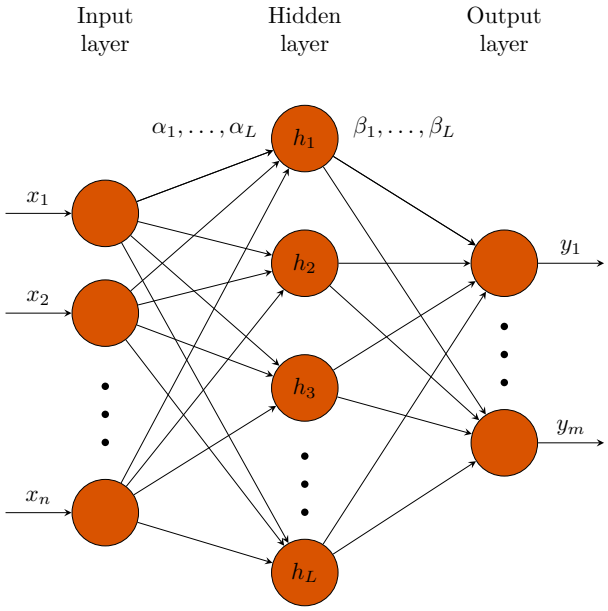

Figure 3.1: Illustration of a single-layer feed-forward neural network, the building block of my developmental algorithm, and the used notations.

3.2.2 | Scoring and replacing low-score neurons

According to an interpretation of the lottery ticket hypothesis (*Frankle et al., 2018*), the large number of neurons increases the probability of finding a good solution for the given task. Furthermore, in the human neural system, during childhood synaptic pruning, redundant connections are removed while relevant connections are strengthened (*Mahajan et al., 2012*). I applied this concept to keep the good random connections and prune then replace the bad ones between the two layers specified above (α_i). New temporal auxiliary networks are generated to find good connections, and neurons are scored and evaluated. Since the generation and evaluation of randomized networks is very fast, this trial-and-error mechanism is useful for filtering and replacing random neural connections.

Please note that my unstructured pruning is not to get a smaller network with fewer connections but to replace the pruned connections with new ones. The three steps of this procedure:

1. Identify less important hidden neurons (h_i, where i is the index of a neuron) by

magnitude:

$$Score(h_i) = \sum_{j=1}^{m} |\beta_{i,j}|, \quad i = 1, \ldots, L \tag{3.10}$$

Rank all h_i and choose the worst p percent of hidden nodes to be replaced. Please note that since the early work of Hanson et al. in (*Hanson et al., 1988*), to prune neurons, the input weights are generally considered in magnitude-based pruning. In my case, since the input weights are random, there is little point in doing this rather the usefulness of neurons (and implicitly their random input connections) is to be scored based on their output weights $\beta_{i,j}$ - which are not random but results of optimization.

2. Generate and evaluate temporal auxiliary networks for the same task and with the same structure but with different random weights. Compute $\beta_{i,j}$ and evaluate hidden neurons with Equation 3.10. Now, choose the best p percent of hidden nodes and replace low-scoring neurons in the initial network with these neurons.

3. Recompute the $\beta_{i,j}$ weights of the updated network which has now p percent of hidden nodes replaced.

This procedure can be repeated several times, with new auxiliary networks always being created. Algorithm 1 formalizes these steps:

Algorithm 1 Pruning and Neuron Replacing Method

Input: Initial model, pruning percentage $p \in [0, 1]$

Output: Updated model

Identify least important hidden neurons in initial model
for each hidden neuron h_i **do**
    Compute importance score as given in Equation 3.10
end for
Rank all h_i and select the lowest p percent as L
Train auxiliary network(s)
for k auxiliary network(s) **do**
    Initialize and train the network on the same task
    Compute $\beta_{i,j}$ and importance scores using Equation 3.10
    Rank hidden neurons and select the top p percent as M
end for
Replace neurons in L with corresponding neurons from M
Recompute $\beta_{i,j}$ for the updated network

Two issues may give rise to concern about this strategy:

- What is the chance that new neurons, which are to replace less efficient ones, are similar to existing ones? If the weights are reduced (quantized) to a scale of 5 values (which is a generous simplification) and L is 200, the probability of matching is $1/(5^{200})$, which is negligible.

- Why use auxiliary networks, why not simply increase L, then get rid of the necessary number of low-scoring neurons? First, increasing the number of hidden neurons increases the computation time of the pseudo-inversion method since its computation complexity is $O(NL^2)$. This is illustrated in Figure C.1 of the Appendices and indirectly in Figure 3.4. Second, large L can increase the instability when computing the pseudo-inverse. Third, the chance of overfitting increases, since the model can adapt better to training data but still lacks generalisation by hierarchical structures.

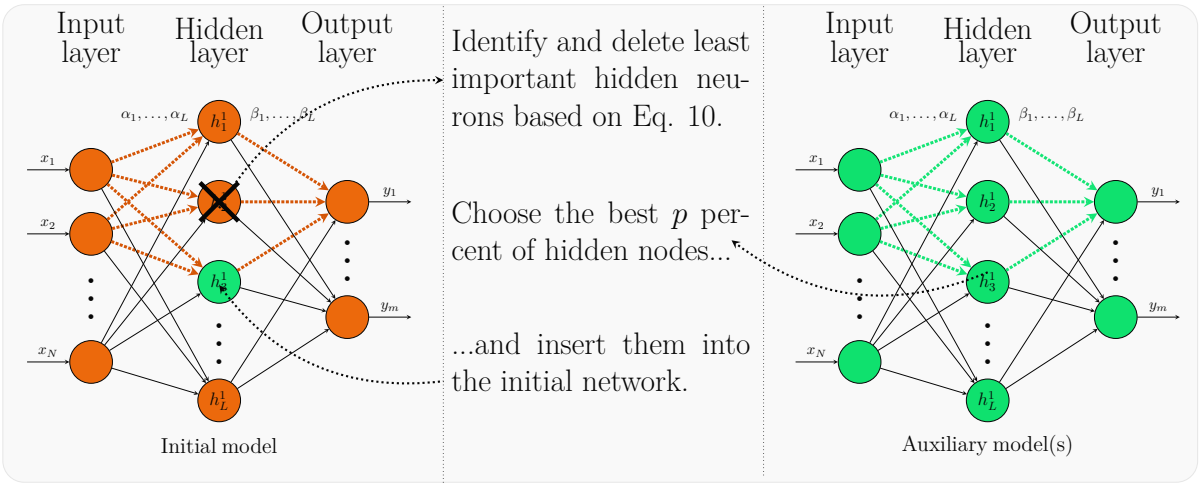

Figure 3.2: Pruning and replacing neurons in the proposed DEV-DRNN AUX framework. The initial model is improved by replacing the least important neurons (i.e. their connections) proved to be important in auxilary model(s).

3.2.3 | Increasing depth by adding new layers

Since the problem-solving ability of SLFNs is limited, increasing the depth of the model has proven to be the right strategy. I propose to add new neurons to the structure hierarchically, hoping that useful high-level features can be extracted with their help. I follow the theory of evolutionary complexification or the principle of incremental adaptation by introducing new layers to the existing structure. I apply three strategies to introduce new hidden neurons (denoted as Hidden layer 2 in Figure 3.3):

1. *Random sampling*: neurons (h_i^2, where the upper index denotes the layer number) with random connections to the previous layer, similar to α_i, are generated to follow the fundamental concept of randomisation. I am generating orthogonal random weights as found more efficient in (*Rádli et al., 2023b*).

2. *Random sampling around already computed weights*: Neurons denoted by $\tilde{y}_i^1$ are the clones of the output neurons from the previous phase (y_i^1), with their weights perturbed by adding Gaussian noise with zero mean and η variance. These neurons are intended to explore the parameter space around the optimal weights computed in the previous phase using Eq. 3.8. Small η might introduce some redundancy, but it is not necessarily a problem - think of the low-rank approach of (*Ouyang, 2021*) mentioned in Section 3.1.3.2.

3. *Output neurons of the previous phase*: neurons denoted by y_i^1 produce output values trained previously.

The weights of the new output layer (y_i) are finally computed with the matrix inversion technique described earlier. Arbitrary number of layers can be added to the network with these steps.

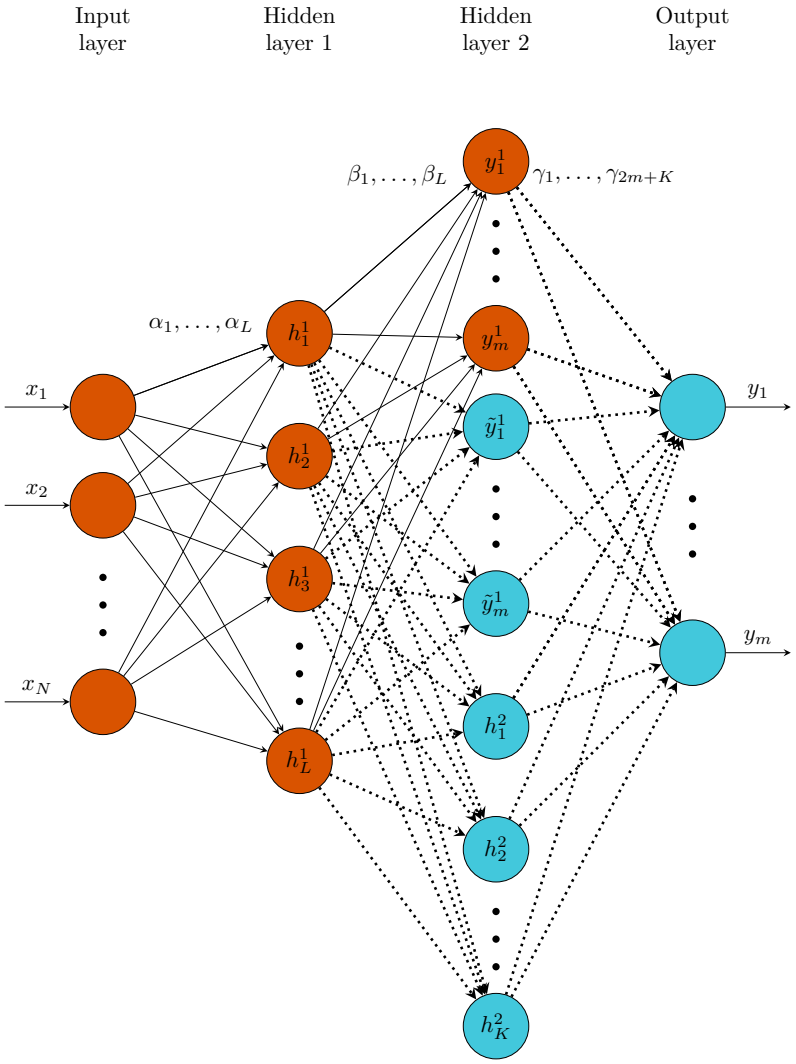

Figure 3.3: Building of DEV-DRNN: The first extension step of the development process for layer addition. Rust coloured neurons belong to the first phase neurons, light blue nodes denote the new neurons of the second phase. Dotted lines indicate the new weights as incremental structures added to the previous phase.

Algorithm 2 Building and training of DEV-DRNN AUX

Input: Training set of $(\mathbf{x}_i, \mathbf{y}_i)$

Output: Optimized DEV-DRNN AUX O structure with weights

1: Randomly generate weight matrix $\boldsymbol{\alpha}$ between input and hidden layer
2: Calculate the hidden layer outputs $\mathbf{H}^1 = \phi(\boldsymbol{\alpha}_L \mathbf{x}_N + b_L)$
3: Calculate the Moore-Penrose pseudoinverse $\mathbf{H}^{1\dagger}$ of $\mathbf{H}^1$
4: Calculate the weight matrix $\boldsymbol{\beta} := \mathbf{H}^{1\dagger}\mathbf{Y}_N$ between hidden and output layer
5: Identify less important hidden neurons by Equation 3.10
6: Create O arbitrary auxiliary networks
7: **for** each O auxiliary model **do**
8: Repeat steps 1, 2, 3, and 4 for the auxiliary model
9: Choose the best p percent of hidden nodes
10: **end for**
11: Replace low-scoring neurons in the initial network with these neurons from the auxiliary network(s)
12: Recalculate steps 2, 3, and 4 for the updated model
13: **for** (i=1; i≤ number of layers; i=i+1) **do**
14: Copy the output neurons y^i from the previous phase, add noise $\mathcal{N}(0, \sigma^2)$ to their weights
15: Add more neurons H^{i+1} with random weights
16: Concatenate y^i, $\tilde{y}_i^1$ and H^{i+1} to form the $(i+1)^{\text{th}}$ hidden layer
17: Calculate the $(i+1)^{\text{th}}$ hidden layer output $\mathbf{H}^{(\mathbf{i+1})^{\text{th}}}$
18: Calculate the Moore-Penrose generalized inverse $\mathbf{H}^{(\mathbf{i+1})^{\text{th}}\dagger}$
19: Calculate the weight matrix $\mathbf{H}^{(\mathbf{i+1})^{\text{th}}\dagger}\mathbf{y}_i$
20: **end for**

3.3 | Experiments, results, and evaluations

3.3.1 | Datasets

I used 13 datasets from the UCI Machine Learning Repository (*Asuncion et al., 2007*). The selection included five image datasets and seven traditional real-world, non-image datasets, each addressing classification tasks. The datasets were partitioned into train, validation,

and test sets following a 70-15-15 split ratio. In all network configurations, the validation set was used for hyperparameter tuning (see in Section 3.3.3). Additionally, for the Adam and SGD, the validation set was employed to support an early stopping mechanism during training. As part of the preprocessing procedure, each dataset was normalised by scaling the features between 0 and 1. A description of the datasets can be seen in Table 3.1, where I present the most important properties.

Table 3.1: Datasets and their most important properties

Dataset	**Training samples**	**Validation samples**	**Test samples**	**Features**	**Classes**
Connect4	47,290	10,134	10,133	42	3
Isolete	5,458	1,170	1,169	617	26
Letter	14,000	3,000	3,000	16	26
MNIST	49,000	10,500	10,500	784	10
Fashion-MNIST	49,000	10,500	10,500	784	10
Musk2	4,619	990	989	168	2
Optdigits	3,934	843	843	64	10
PageBlocks	3,831	822	820	10	5
Satimages	4,504	966	965	36	6
Segment	1,617	347	346	19	7
Shuttle	40,600	8,700	8,700	9	7
Spambase	3,220	691	690	57	2
USPS	6,509	1,395	1,394	256	10

3.3.2 | Models of the experiments

My experiments aim to investigate the performance of my proposal compared to other competitive approaches concerning accuracy, F1 Score, and training time evaluated mainly on fully connected networks. Six approaches compete on fully connected networks: SGD, Adam, the autoencoder-based H-ELM (*Tang et al., 2015*), and the proposed DEV-DRNN with zero, one, and two auxiliary models. In the case of the last two models, the sum of neurons of the auxiliary networks is equal.

First, the number of neurons per hidden layer for the Adam and SGD methods was determined using the hyperparameter tuning tool, Ray Tune (*Liaw et al., 2018*). I used a grid search method, looking at the wide range of neuron numbers per layer: 266, 500, 866, 1000 and 2000. While for Adam and SGD, neuron quantities were the same in all hidden layers, for DEV-DRNNs, a decreasing manner was applied, as given in Table 3.2, giving more opportunities for the random sampling approach. (I have also tested decreasing neuron counts for Adam and SGD networks, but since I did not find any significant differences, I do not present them) For H-ELM, the distribution followed the pattern used in (*Rádli et al., 2023b*) and (*Li et al., 2021*). Please note that while the number of neurons in the layers of models running Adam and SGD were systematically tuned, only "hand tuning" was applied for the other approaches, and the total amount of neurons was equal for all methods.

Table 3.2: Neurons' distribution along the datasets, layers, and network variants. Lx indicates the actual layer. Please note, that the number of layers and neurons are the same for all competitors.

	Adam/SGD	**DEV-DRNN**			**H-ELM**			
Datasets	L1-2-3	L1	L2	L3	L1	L2	L3	*SUM*
Connect4	866	2,046	452	100	400	198	2,000	*2,598*
Isolete	1,300	2,480	1,010	410	600	300	3,000	*3,900*
Letter	1,500	3,000	1,000	500	1,000	500	3,000	*4,500*
MNIST	2,000	4,000	1,500	500	1,000	2,000	3,000	*6,000*
Fashion-MNIST	2,000	4,000	1,500	500	1,000	2,000	3,000	*6,000*
Musk2	866	2,046	452	100	400	198	2,000	*2,598*
Optdigits	600	1,000	500	300	700	100	1,000	*1,800*
PageBlocks	600	1,000	500	300	100	50	1,000	*1,800*
Satimages	1,200	2,600	700	300	100	50	3,450	*3,600*
Segment	300	600	200	100	100	50	750	*900*
Shuttle	216	400	200	48	100	48	500	*648*
Spambase	500	800	400	300	900	200	400	*1,500*
USPS	1,000	1,600	1,000	400	800	200	2,000	*3,000*

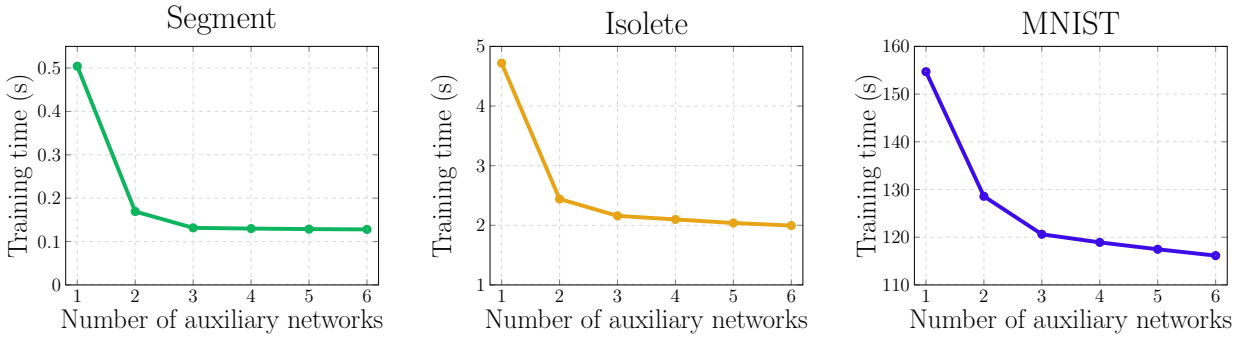

Figure 3.4: Training time on three different datasets as a function of the number of auxiliary networks. Training time is the sum of all auxiliary models. As the number of auxiliary networks increases, the total sum of hidden neurons is kept the same, but training time decreases as a consequence of smaller layers.

3.3.3 | Network setups

3.3.3.1 | Models for Adam and SGD

The fully connected neural network models had one input layer, three hidden layers, and one output layer. As mentioned above, all hidden layers had the same number of neurons (for a control test, I made a variant where the number of neurons descended similarly to DEV-DRNNs models, with results quite similar to the original architecture). In the hidden layers, I applied leaky ReLU activation functions with a slope set to 0.2; for the loss function, cross-entropy was selected. Ray Tune was used to optimize the learning rate for the Adam and SGD optimizers and the batch size for each dataset. The learning rate was tested within the range of 4×10^{-4} to 1×10^{-1}, while batch sizes were evaluated across the values of 32, 64, and 128. For the SGD, I also identified optimal momentum values selected within the range of 0.5 to 0.9. Other parameters were manually configured: for instance, the number of epochs was set to 1000 for all configurations, with the early stopping mechanism and patience parameter 10.

3.3.3.2 | Setting up the H-ELM

My Python implementation was based on the MATLAB code provided at (*Huang, 2013*) for the H-ELM. I developed my own implementation to fairly compare training times by providing the same running environment. The network parameters C (regularisation) and s (the scaling factor of weights and bias values) were optimized for each dataset using Ray

Tune to achieve the best accuracy. The value of C was tested within the range 1×10^{-30} to 1×10^{-5}, while the scaling factor ranged from 0.0 to 1.0.

3.3.3.3 | Configuring DEV-DRNNs

I determined the penalty term C, the parameter for relative tolerance R_t (for the pseudoinverse calculation), and the percentage of replaced neurons p, similar to that of Ray Tune. The number of layers and the activation function were the same as used in the Adam and SGD optimized models. R_t was investigated in a range from 1×10^{-1} to 1×10^{-30}, the penalty term within the range of 0.1 to 30, and the neuron replacement was determined by adjusting the percentage of replaced neurons from 10% to 100%.

Table 3.3: Hyperparameters for different methods.

Method	**Hyperparameters**
Adam	Learning rate, number of hidden neurons, batch size, loss function
SGD	Learning rate, momentum, number of hidden neurons, batch size, loss func.
H-ELM	regularisation, penalty value (C), scaling factor (s)
DEV-DRNN	penalty term (C), relative tolerance (R_t), percentage of replaced neurons (p)

3.3.4 | Experimental results

I executed ten tests for all networks on every dataset and averaged the results. The metrics used were training and testing accuracy, training and testing F1-score, and training time. The testing accuracy and training time results are presented in Table 3.4 and Table 3.5, while the testing F1-score results are provided in the Appendix (Table C.2).

To accumulate the results, rankings were computed for all metrics and are shown in Table 3.6 and in the Appendix (Table C.3 and Table C.4).

Table 3.4: Testing accuracy of the different methods as the average of ten experiments. Bold numbers denote the highest values.

Dataset	**Adam**	**SGD**	**H-ELM**	**DEV-DRNN**	**DEV-DRNN AUX 1**	**DEV-DRNN AUX 2**
Connect4	0.8080	**0.8169**	0.7978	0.7800	0.7904	0.7910
Isolete	0.9168	0.9340	0.9405	**0.9500**	0.9424	0.9467
Letter	0.9649	**0.9709**	0.9156	0.9599	0.9674	0.9534
MNIST	0.9707	0.9718	**0.9781**	0.9728	0.9729	0.9751
Fashion-MNIST	0.8781	0.8898	0.8745	0.8945	**0.8959**	0.8944
Musk2	0.9334	0.9514	**0.9666**	0.9608	0.9642	0.9655
Optdigits	0.9801	0.9705	0.9919	0.9909	**0.9947**	**0.9947**
PageBlocks	0.9668	0.9538	0.9588	0.9668	**0.9709**	**0.9709**
Satimages	**0.8838**	0.8787	0.8703	0.8750	0.8740	0.8789
Segment	**0.9798**	0.9642	0.9728	0.9688	0.9705	0.9737
Shuttle	0.9943	0.9666	0.9762	0.9919	**0.9957**	0.9950
Spambase	0.9310	**0.9401**	0.9162	0.9280	0.9316	0.9310
USPS	0.9445	0.9596	**0.9758**	0.9672	0.9684	0.9695

Table 3.5: Training time (sec) of the methods as the average of ten experiments. Bold values are for the shortest training time.

Dataset	Adam	SGD	H-ELM	DEV-DRNN	DEV-DRNN AUX 1	DEV-DRNN AUX 2
Connect4	144.2423	224.0359	**3.4750**	13.7913	42.8242	17.5582
Isolete	30.3428	36.4111	2.8593	**1.6432**	4.7192	2.4407
Letter	46.7726	209.7468	**5.1331**	12.2867	37.4548	15.4112
MNIST	449.4071	390.2254	**21.8655**	53.8329	154.6978	128.5774
Fashion-MNIST	449.4071	1978.5518	**22.1634**	50.6276	154.6421	124.5941
Musk2	18.2375	18.6694	**0.6480**	0.9615	2.9333	1.3636
Optdigits	7.8805	7.4449	**0.5341**	0.6300	1.9528	0.6788
PageBlocks	3.9931	10.7080	**0.0997**	0.1176	0.3638	0.2823
Satimages	33.5461	33.9377	**0.8365**	5.1942	16.4739	6.0766
Segment	11.1918	18.8388	**0.0643**	0.1569	0.5042	0.1692
Shuttle	28.7526	21.5538	**0.7647**	1.4111	4.2336	3.0767
Spambase	4.4404	36.2790	1.0109	**0.3371**	1.2297	0.4470
USPS	18.5247	22.1755	**1.6981**	2.0160	6.3233	2.1225

Table 3.6: Average ranking of the investigated methods along the evaluated metrics for the 13 datasets. Bold values are for the network with lowest average rank.

Method	Train accuracy	Test accuracy	Train F1-score	Test F1-score	Training time
Adam	4.38	3.92	4.46	3.92	5.23
SGD	4.23	4.15	4.38	4.08	5.77
H-ELM	5.08	3.85	5.08	3.92	**1.31**
DEV-DRNN	3.00	3.85	2.92	4.00	1.85
DEV-DRNN AUX 1	**1.92**	2.62	**1.69**	2.85	4.00
DEV-DRNN AUX 2	2.38	**2.38**	2.46	**2.23**	2.85

3.4 | Discussion

Based on the data presented above, the proposed DEV-DRNN methods can reach high performance regarding accuracy and F1-score. Considering the large number of datasets, the different metrics, and the nature of results, including rankings, helps better understand the performance hidden in data. While in Table 3.6, we can see that DEV-DRNN AUX 1 and DEV-DRNN AUX 2 have the highest average ranking, - in the Appendices - Table C.3 and Table C.4 reveal more details: consistently one of those had the highest number of first positions, and they were never the worst.

Considering the training time, we see a slightly different picture. Notably, the H-ELM model has unrivalled performance; however, the training times of my DEV-DRNN models are significantly shorter than those utilizing backpropagation algorithms. From the last column of Table 3.6, we see that DEV-DRNN has the second best value and DEV-DRNN AUX 2 is third. However, what does it mean in practice? From Table 3.5, we can get that in the case of DEV-DRNN, the speedup is in the range of 3.8-71.3, while in the case of DEV-DRNN AUX 2, it is in the range of 3.03-66.1 when compared to the faster of the two traditional methods (Adam and SGD).

Finally, I provide a plot in Figure 3.6 for a more straightforward interpretation of the relative performance of the approaches. While in Figure 3.5, the superiority of DEV-DRNN against Adam was plotted as of training time vs. accuracy for the 13 datasets, in this figure, the ranking in time and accuracy is given as a global view on the performance. The fastest time is achievable with H-ELM and DEV-DRNN, while the extensions with auxiliary networks can balance speed and accuracy. This picture is slightly nuanced by the fact that although H-ELM and DEV-DRNN are ranked on average the same by accuracy (see average rankings in Table 3.6), DEV-DRNN was ranked at the bottom of the accuracy list only once. However, H-ELM was the least accurate method three times (out of 13), as the numbers show in Appendices, Table C.4. The performance of the AUX versions of DEV-DRNN is clearly above all in average accuracy ranking and could always avoid the worst ranking positions.

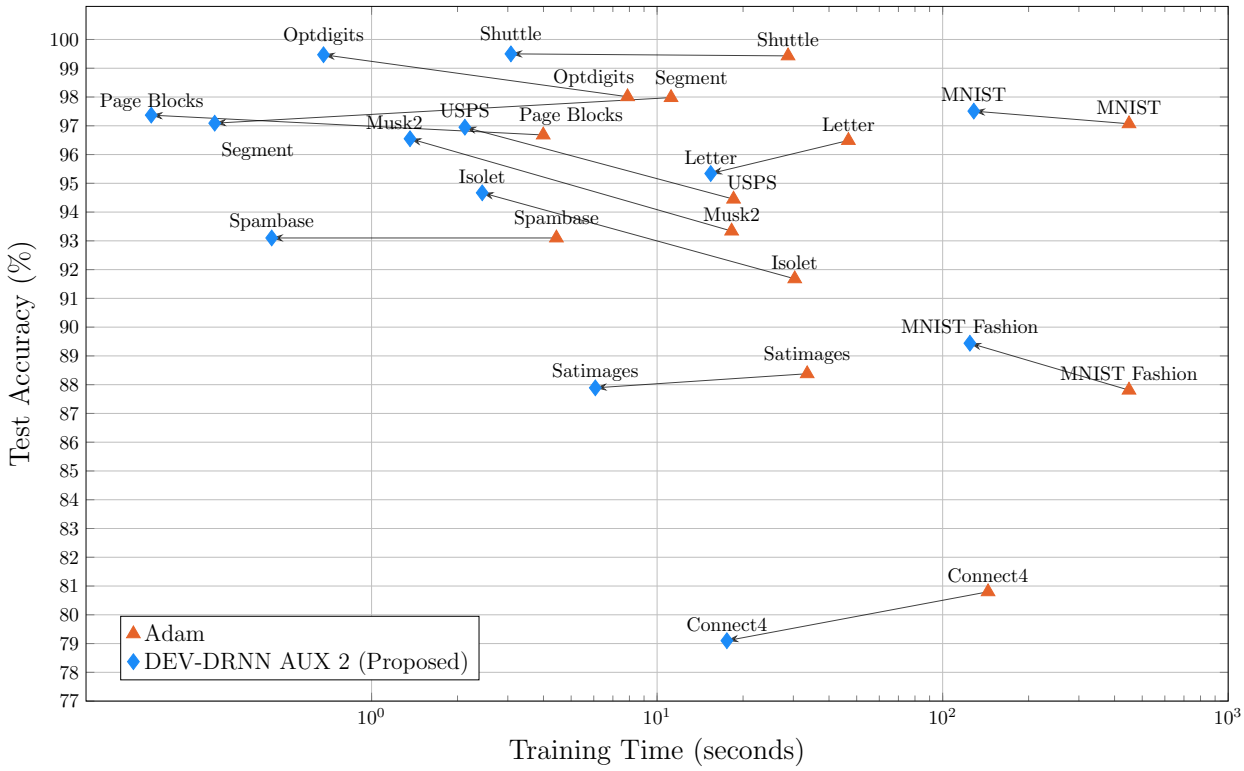

Figure 3.5: Comparison of the performance of fully connected networks trained by Adam backpropagation and the proposed DEV-DRNN AUX 2 method on 13 datasets (as the average of 10 experiments). The training time is shorter in all cases, while in 9 out of 13 cases, the test accuracy is equal or higher with the proposed solution. It should be noted that the training time is on the logarithmic scale, which shows the significant advantage of the proposed method.

3.5 | Summary

In this chapter, I reviewed different efforts to improve random neural networks to be competitive in accuracy with other deep neural architectures. To generate deep randomized networks as an alternative to backpropagation training, I introduced a new developmental approach with such biological motivations as the iterative growth of biological systems, brain plasticity, and synaptic pruning. In contrast to many other biology-motivated approaches (such as genetic algorithms, differential evolution, particle swarm optimisation, and artificial bee colony), my developmental algorithm does not need large population sizes. While traditional deep learning relies on gradient-based optimizers like Adam and SGD, which can be slow and computationally expensive, the proposed DEV-DRNN eliminates backpropagation by using random weights, pseudoinverse matrix inversion, iterative growth, pruning ineffective neurons, and replacing them with optimized connections.

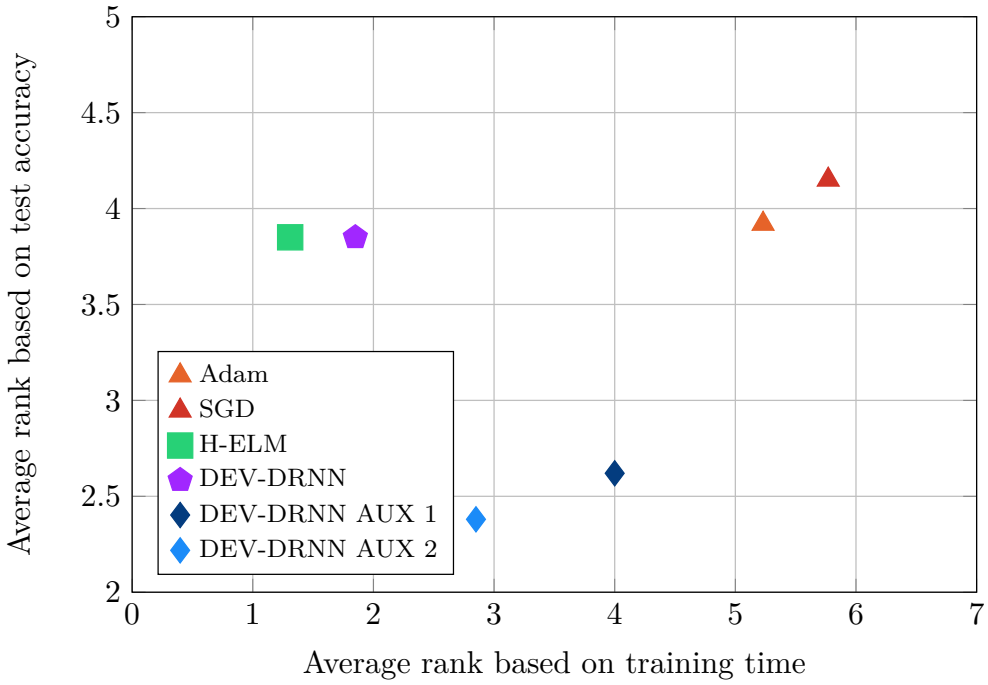

Figure 3.6: Average ranking of test accuracy and training time for the evaluated methods based on 13 datasets. Smaller rank is better.

DEV-DRNN achieves significantly shorter training times than backpropagation-based methods and is faster than the otherwise efficient algorithm of (*Gao et al., 2020*). The improvement in learning speed ranges from 3.8 to 71 times compared to gradient-based optimizers, making it ideal for large-scale applications with comparable or better classification accuracy. Besides the faster training speed, the accuracy is also higher or comparable, as supported by a large number of experiments involving four competing approaches, as illustrated in Figure 3.5, Figure 3.6, and Table C.6 of the Appendices.

The new method was mainly tested on more than a dozen datasets with fully connected networks. However, it is straightforward to use with other mechanisms such as convolution (as can be found in Section C.5.2). Additionally, I already included a brief but successful experimental evaluation with PCA as a preprocessing step for dimensionality reduction (likewise, it is available in Section C.5.1).

Chapter 4

Metric-Based Pill Recognition with the Help of Textual and Visual Cues and Multi-Head Attention

Accurate recognition of prescription pill images based on their visual characteristics can play a crucial role in ensuring patient safety and optimizing modern healthcare systems, particularly for elderly patients. Since medication errors are the most common mistakes in healthcare (*Cronenwett et al., 2007*) recognition technology has the potential to prevent errors throughout the pharmaceutical supply chain, enhance the expertise of poison control professionals, improve medication adherence, mitigate losses of medications and prescriptions during evacuation scenarios, and drive advancements in remote and self-diagnosis technologies as well as smart healthcare applications. Pill recognition systems can significantly enhance the quality of medication dispensing, either for home usage or in large-scale automated pill dispensing systems.

Strictly theoretically, medication pills are designed with distinctive features, including size, colour, shape, engravings, and imprints. However, various factors contribute to the challenges of accurate recognition, such as:

- Pill photographs are captured under diverse conditions, such as varying illumination, viewing angles, distances, and camera settings.
- Due to their small size, local features on pills are often not clearly visible or may be distorted.
- The number of possible classes can be very large (tens of thousands of possible pills),

while few-shot learning is a requirement for many applications.

4.1 | Related works

Pill recognition problem is very similar to other recognition tasks, such as the recognition of stamps, coins, or other small objects with large number of classes. Below is a short overview of such techniques designed for and tested on drug datasets. Broadly, these approaches can be roughly grouped into two main categories, though there is overlap and ongoing development in the field:

1. **Object detection and recognition frameworks:** Methods that rely on generic or adapted object detectors, often combined with feature extraction and classification techniques. These approaches aim to localise and identify pills in images and can benefit from advances in general-purpose detection models.
2. **Metric learning and similarity-based approaches:** Techniques that focus on learning embeddings or similarity metrics to distinguish between classes with limited examples. This includes traditional approaches such as Siamese networks, as well as more recent multimodal methods that leverage additional information (e.g., text descriptions or shape/color attributes) to enhance representations.

Many generic object detectors are applied to similar problems as pill recognition, so it is obvious to check their performance on pills. In (*Ou et al., 2018*), a two-stage approach was applied: during the detection stage, a ResNet-50 (*He et al., 2016*) based deep convolutional neural network was applied to extract features and construct a feature pyramid. Then, still in the first phase, the subsequent regression and classification layers were responsible for determining the position of the pills. In the second stage, based on these positions, the Xception (*Chollet, 2017*) deep convolutional neural network is used to classify the pill types. This approach could reach 79.4% Top-1 accuracy on the authors' dataset. Naturally, handling the problem of few-shot recognition, an essential aspect due to the large number of possible classes, is far from this approach.

Since the large number of pill classes is a costly problem when training recognition systems, the solution in (*Kwon et al., 2021*) proposes an algorithm which can improve the detection performance based on limited training data and an effective database expansion method. For the detection and recognition, Mask R-CNN (*He et al., 2017*) was utilised; thus, the

recognition method is not new, but the focus was only on improving the training procedure.

In (*Tan et al., 2021a*) three widely used single-stage detectors, namely YOLOv3 (*Redmon et al., 2018*), RetinaNet (*Lin et al., 2017*), and SSD (*Liu et al., 2016*), were compared on a custom pills dataset, resulting only in minor (circa 2%) differences, but all above 0.80, in mAP. As these detectors are constantly and rapidly evolving, I also use the YOLOv8 model (in Subsection 4.4.1), but only for segmentation purposes. The reason for such detectors' limited usage is an important limitation of generic classification methods when used in few-shot problems. When an instance of a new class arrives, which has a non-negligible probability in the case of pill dispensing applications, knowing that the possible number of pill classes is over 10,000 in a typical country, the new classes should also be trained and recognised, preferably without costly efforts. Moreover, DNNs can lead to catastrophic forgetting (*Pfülb et al., 2018*), although there are techniques to eliminate this effect, such as discussed in papers (*Nagy et al., 2022*) and (*Doan et al., 2023*). Furthermore, the challenges posed by the unbalanced class distribution when dealing with a low number of instances of new-coming classes cannot be overlooked. This issue, coupled with the time and/or GPU facilities needed for the retraining process, underscores the complexity of the problem.

As an alternative to generic classification approaches, metric learning approaches create an embedding of input information that can easily be used to compare queries and candidates. If the metric model is trained on a large enough dataset representing the possible features of possible objects well, even newly arrived class instances can be classified without new training procedures, using comparisons with features of reference images.

To foster the development of reliable solutions the United States National Library of Medicine (NLM) announced an algorithm challenge on pill recognition in 2016 (*Yaniv et al., 2016*). After the evaluation of results, the accuracy of the top three submissions seemed not to be sufficient for the development of a mobile online service for matching consumer quality images. My proposed method is inspired by the winner model architecture of the competition (*Zeng et al., 2017*) and its descendant (*Ling et al., 2020*), (*Pastor et al., 2023*).

The enhanced model of (*Ling et al., 2020*) applied a two-phase training approach: in the first phase, separate data streams, responsible for the embedding of RGB, texture, and contour information, were trained with triplets (inputs of anchor, positive, and negative samples), then in the second phase the output vectors of the streams were fused with a unique OCR stream. The OCR stream was responsible for text localisation, geometric

normalisation, and feature embedding with the Deep TextSpotter (*Busta et al., 2017*). This model showed better accuracy tested on the NIH (National Institute of Health) and the CURE datasets (*Ling et al., 2020*) (the description of datasets is in Section 4.2) than any previous model.

My basic model uses a strategy similar to (*Ling et al., 2020*) but with significant modifications and improvements. First, I replaced the separate stream's OCR method with an LBP (local binary pattern) (*Ojala et al., 1994*) descriptor stream, making a unified architecture that can be trained more straightforwardly without any language or symbol dependency. I have chosen LBP because local intensity changes can describe and distinguish characters or symbols. LBP is often applied in different vision tasks, including handwritten or printed OCR (*Liu et al., 2010*), (*Hassan et al., 2015*). I also added attention mechanisms into my models; details are available in Subsection 4.4.2. Moreover, besides using a different pill detection model, I use more enhanced backbones in my proposed solutions. Some of the consequences of these modifications are already investigated in (*Rádli et al., 2023c*). Besides these improvements, the main difference in my approach is the inclusion of different high-level (mostly explicit) cues, which were completely missing from the articles mentioned above.

Compared to general optical object recognition tasks, some things could be explicitly considered when dealing with medical pills:

1. Each pill is strictly designed to have a different appearance.

2. The distinguishing features can be described by high-level textual or visual features.

A straightforward way to utilise semantic distinguishing features is to extract imprinted characters. In (*Heo et al., 2023*), the authors trained not only RGB images of the pills but also imprinted characters for a separate neural network. In the pill recognition step, the different modules separately extract the features of pills and their imprints while correcting the recognised imprint to fit the actual data of other features. A trained language model was applied for imprint correction. It was shown through an ablation study that the language model could significantly improve the system's ability to identify pills. The drawback of this solution is that a specific language model (including an OCR module) is required for the application, which might differ from country to country.

In addition to the above-listed unique factors, (*Nguyen et al., 2022*) proposed a novel method, which is based on a hypothesis that all the medicines, in an actual capture of a

set of pills, are prescribed to cure or alleviate some diseases or symptoms. Thus, there must be implicit relations between pills and diagnoses, which can be exploited during the recognition of the tablets. The proposed solution used a prescription-based knowledge graph, representing the relationship between pills. A graph embedding network extracted the relational features of pills, and a framework was applied to fuse the graph-based relational information with the image-based visual features for the final classification. While we can generally accept the above-described assumption, there are several conditions which could weaken this approach: the images should contain several pills at a time; a large number of prescriptions are needed to train the graph model; adding new pills to the task requires a (long) process of collecting prescriptions.

In (*Nguyen et al., 2023*), the multi-pill detection problem in real-world settings is tackled by constructing heterogeneous a priori graphs incorporating three forms of inter-pill relationships, including co-occurrence likelihood, relative size, and visual semantic correlation. The idea is that first, easy pills (those that do not have a remarkable resemblance to other pills) are differentiated based on their visual appearance. The visual features of those easy samples will then serve as context vectors to assist with making decisions regarding the hard ones (those which appear similar under the given conditions). While this approach handles the actual environmental effects and tablet-tablet size relations well, other features (e.g. colour, prints, score) are semantically not considered. Additionally, a large amount of prescription information (for model training) is required as in the previous (*Nguyen et al., 2022*) method.

In contrast to all these approaches, my proposed solution utilises all possible semantic information about individual pills; it avoids the use of specific models for the imprinted text or prescription data, while it uses only generic models (to process the information leaflet of pills) and can handle few-shot classification in a self-explanatory way.

Based on the above observations, metric learning has advantages over other approaches. In (*Rádli et al., 2023c*), I have shown that OCR can be omitted from the streams, and in (*Rádli et al., 2024b*), I have first shown that pills' textual information can be utilised in visual metric learning.

4.2 | Datasets

Several image datasets, such as those in (*Yaniv et al., 2016*), (*Ling et al., 2020*), (*VAIPE, 2008*), are available for pill recognition. In my experiments, I have chosen two datasets: one is the **CURE** image database (*Ling et al., 2020*) and the other is a custom dataset, entitled **OGYEIv2** (*Rádli et al., 2024b*), (*Rádli et al., 2023a*) that I created with the help of my colleagues.

The purpose behind choosing the CURE dataset is twofold: it is well-documented and accessible and has been widely used in research papers. The dataset contains images of single pills with different backgrounds and varying lighting conditions. I have chosen this dataset for comparison since results of one of the best metric-based methods (*Ling et al., 2020*) are available for it. CURE is divided into reference and consumer quality (query) images. It is important to note that images are also separated by their side (top or bottom of a pill). Reference images are synthetically derived and generated from consumer images by replacing the backgrounds with homogeneous colours. I have found some minor flaws in CURE: some pills are listed twice or more in different classes (with different photographs). CURE lacks pixel-wise segmentation annotations and textual descriptions of pills.

The creation of OGYEIv2 can be described the following way:

- The pill images were captured by an ELP 8.3 MPx USB camera (Sony IMX415 sensor up to 3840×2160 pixels of native 4K resolution) using a 8 mm, F1.8 optic. We applied camera calibration to eliminate optical distortions; after undistortion, the image resolution became 3246×2019 pixels.
- The same homogeneous grey background was used for each drug picture.
- Half of the images were taken under an upper LED ring lamp with a diffuser attached. In this configuration, pills gave a pleasant overall appearance without noticeable shadows (but some engravings almost disappeared; see the second row of Figure 4.1).
- Another half of the images were recorded using a side-mounted LED strip light, making the engraved surface patterns sharply visible.

While images in OGYEIv2 have similar backgrounds (easily achievable in daily conditions), the two different light settings make sign and score-based recognition difficult (similar to real-life consumer applications). The main characteristics of the two datasets are shown in

Figure 4.1: Images from the two datasets employed in the study. The first row shows image pairs from the CURE dataset, with the first images in each pair being a reference-quality image and the second image being a consumer-grade image. The second row presents image pairs from the OGYEIv2 dataset, where the first image in each pair was captured using an upper lighting, while the second image was captured using a side lighting. The unwanted effects of the different lightings is clearly observable.

Table 4.1.

Table 4.1: Comparison of the main properties of the CURE and OGYEIv2 datasets.

	CURE	**OGYEIv2**
Number of pill classes	196	112
Number of images	8973	4480
Image resolution	800×800 2448×2448	3246×2019
Instance per class	40-50	40
Backgrounds	6	1
Segmentation labels	No	Yes
Free-text description	No	Yes

According to my knowledge, OGYEIv2 is the only publicly available pill dataset that contains not only images but also free-text visual appearance descriptions taken from the original leaflets of the medicines. These leaflets, besides others, contain a general description of the visual appearance of the pills. To show a short example of the textual descriptions, I give the original sentence (in Hungarian) from the leaflet of the medicine named Advil Ultra Forte and its English translation:

- Hungarian: *Színtelen, tiszta folyadékot tartalmazó, átlátszó, ovális lágy kapszulák fekete "400" felirattal.*
- The English translation: *Colourless, containing clear liquid, transparent, oval soft capsules, imprinted with black "400" lettering*

Since the leaflet content is strictly structured, I used NLP to gather the relevant morphological information (2 or 3 sentences per leaflet).

To compare the AutoML platforms, YOLO11, and my proposed network, two additional test datasets were created in a controlled laboratory environment. Images were taken of 30 commonly dispensed medications. The first dataset, referred to as the **laboratory test set**, contains 120 images with a single pill per image. The second dataset, called the **exhaustive test set**, contains 100 images with multiple pills (at least 3 and at most 8 pills were present on an image). The medications were placed in a white dosette box, and a controllable top-mounted light source was used to ensure uniform light distribution. Images were captured using a methodology similar to that of OGYEIv2. Further details about these datasets can be found in (*Ashraf et al., 2026*).

Similarly, the training datasets used for the AutoML platforms and YOLO11 are described in detail in (*Ashraf et al., 2026*) and (*Ashraf et al., 2024*). These training datasets were collected by the pharmacist colleagues of the University of Pécs and are collectively referred to as the **PTE Pill Dataset**.

On the other hand, my proposed network was trained on a currently non-public pill dataset (we call it **PTE-PE Pill Dataset**) containing 100 medication classes, with 400 images captured for each class. This dataset was made in collaboration with pharmacist colleagues of the University of Pécs. Among these, 76 classes do not overlap with the medications appearing in the laboratory and exhaustive test sets, and the training of the proposed network – in the key of a one-shot learning procedure – was restricted to these 76 classes.

4.3 | AutoML methods in pill recognition

In parallel with the evolution of code-based models providing flexibility and customization but requiring strong programming skills and longer development time, low-code or code-free automated machine learning (AutoML) platforms have emerged, revealing new possibilities

for relatively simple development and testing of specialized object detection models utilizing easy-to-use cloud-based platforms.

AutoML refers to a set of techniques and tools that automate the end-to-end process of applying machine learning to real-world problems, including data preprocessing, model selection, hyperparameter tuning, and evaluation.

In a traditional deep learning ecosystem, model development involves several manual and iterative processes, such as data collection and pre-processing, feature engineering, model selection, hyperparameter tuning, training, and validation. This approach requires a high level of expertise and intensive experimentation to fine-tune the model's performance. Cloud-based, no-code AutoML platforms have emerged as suitable alternatives to traditional, often code-intensive model development approaches. These platforms automate most of the workflows involved in managing machine learning pipelines, thereby lowering the barrier to entry and making model development accessible even to those without programming skills or data science expertise (*Ayyalasomayajula et al., 2019*), (*Zöller et al., 2021*).

In addition to enhancing accessibility, AutoML platforms accelerate the model development lifecycle by automating repetitive and time-consuming tasks. This enables rapid prototyping and experimentation while reducing complexity by handling many of the underlying tasks associated with computational infrastructure, model training environments, and deployment pipelines (*Ayyalasomayajula et al., 2019*). As a result, users can focus more on dataset preparation, annotation, and domain-specific knowledge rather than the technical aspects of machine learning implementation.

These characteristics make AutoML platforms suitable for special applications such as pill recognition in pharmaceutical environments. Pharmacists and other employees in this field hold extensive knowledge about medications but typically lack advanced, or even bare machine learning or programming skills. AutoML platforms allow these domain experts to participate directly in the development of recognition systems by focusing on tasks such as collecting representative pill images and labeling important visual features, including shape, color, size, and imprints. This collaborative approach can lead to models that better reflect real-world pharmaceutical conditions and practical use cases (*Karmaker et al., 2021*).

Furthermore, pill recognition systems developed through AutoML can support healthcare professionals in several practical scenarios. For example, such systems may assist pharmacists in verifying medications during dispensing, help doctors confirm prescribed drugs during

consultations, or support healthcare workers in identifying unknown pills brought in by patients (*Ashraf et al., 2024*). In emergency settings or cases involving medication misuse or accidental ingestion, rapid identification of pills can be crucial. Therefore, automated recognition systems can function as decision-support tools that improve patient safety and help to reduce medication errors (*Heo et al., 2023*).

On the other hand, AutoML also has disadvantages. While these platforms lower the barrier to developing machine learning models by reducing the required technical knowledge, they often require significant computational resources, which may result in high cloud service costs, especially when training models iteratively on large datasets. In addition, users typically have limited control over the selection of base models, underlying training algorithms, and hyperparameter configurations. Many AutoML platforms also lack transparency regarding model architecture and training processes, which may raise concerns about interpretability and reproducibility. Privacy considerations may also arise when sensitive data must be uploaded to cloud-based systems (*Sun et al., 2023*).

Therefore, the choice of the optimal development approach or platform depends on multiple factors, including the required level of customization, available expertise, computational resources, and data privacy requirements. In many practical scenarios — particularly those involving domain experts without extensive machine learning backgrounds - AutoML platforms represent a balanced solution that enables the development of effective object detection models while maintaining manageable development complexity.

In (*Ashraf et al., 2026*), I evaluated AutoML solutions offered by Amazon Web Services (AWS), Google Cloud, and Microsoft Azure, three of the most popular cloudbased providers for object detection alongside coded models trained with a traditional workflow using YOLO11. The test revealed that no single platform consistently outperforms others across all test scenarios, highlighting the importance of matching platform selection to specific deployment conditions and available training resource.

However, these solutions struggle when new pills are added to an already existing dataset, often requiring full retraining. This limitation is critical in pharmaceutical settings, where new medications can appear frequently. To address this - as mentioned in the intorduction of this chapter - metric learning offers a promising approach, enabling models to recognize new pills based on similarity rather than retraining on every new class, making them perfect candidates for tasks such as pill recognition.

The comparison of my proposed pill recognition network and the aforementioned solutions can be found in Section 4.6.

4.4 | The proposed pill recognition network

In this section, I introduce my proposed two-phase multistream metric model, referred to in this study as Multi-Stream Fusion Network (MSFN) and Dynamic Margin Multi-Stream Fusion Network (DM-MSFN). The model, aimed to estimate the distance among pills, follows the multi-stream design and includes multi-head attention layers. The main innovation lies in generating and integrating high-level textual and visual cues into metrics learning to improve pill recognition. This is achieved by altering the margin of the triplet loss function, enabling a more granular metrics learning process, surpassing the capabilities of traditional triplet loss models. The overview of the proposed MSFN structure is given in Figure 4.2.

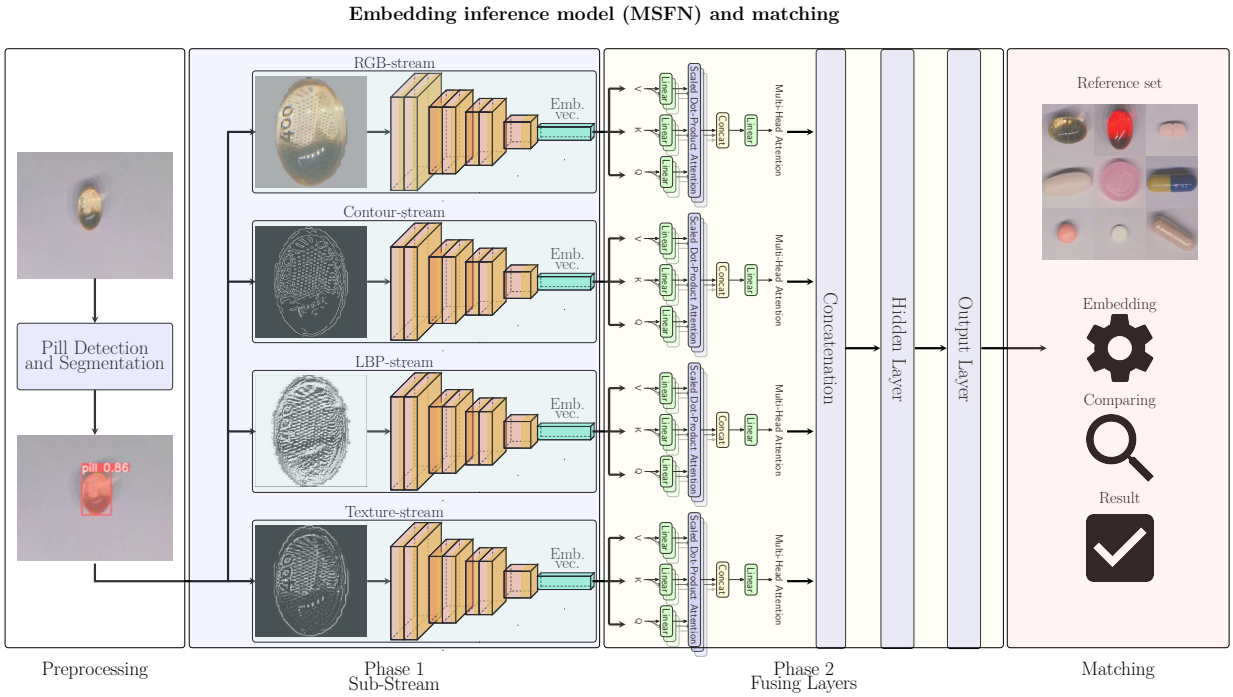

Figure 4.2: Overview of the proposed Multi-Stream Fusion Network.

4.4.1 | Preprocessing: detection and segmentation of pills

Three errors can occur when detecting and recognising multiple or single pills from an image. False detection of other objects, the pill is found but misclassified, or even worse, the pill is not detected. When single-stage detectors (such as YOLO or SSD) are considered, the

whole training process (detection, localisation, recognition) is carried out in a single step. However, metric learning only deals with the recognition of already detected objects. That is why the pills are detected and localised in a preprocessing step. The model has to contend with various factors: noisy backgrounds, challenging lighting, perspective distortions, and scaling from a distance. These problems significantly affect the accuracy of the segmentation results, thereby, the overall process of pill recognition (*Ling et al., 2020*). Pill backgrounds provide very little or no useful information and may even degrade the quality of the training process, so they are considered noise. Although it is crucial to separate the pill regions from the background, which allows the tablet recognition network to be trained and infer from tablet images without the background details, the primary aim of this methodology is not the segmentation itself. Similarly to (*Ling et al., 2020*), (*Nguyen et al., 2022*), (*Heo et al., 2023*), I focus only on the recognition of already segmented pill images and the background is replaced with a constant grey value like in (*Ling et al., 2020*). For segmentation purposes, I have chosen the YOLOv8 (*Jocher et al., 2023*) network, which has comparable instance segmentation capabilities (see the results in Table 4.2).

For the training of YOLOv8, I used reference images, while for the testing, 20% of consumer images were selected from CURE, similarly to (*Ling et al., 2020*). Both in the augmentation and the testing process, images were pixel-wise annotated. The following augmentation operations were performed:

- white balance changes to slightly alter the colour of the images,
- Gaussian smoothing to simulate out-of-focus,
- brightness adjustments: turning the image either brighter or darker,
- geometric transformations: rotation, shifting, and scaling,
- changing the background of pills using the Describable Textures Dataset (DTD) (*Cimpoi et al., 2014*).

These steps are similar to the augmentation applied in (*Ling et al., 2020*). During the generation and testing of the OGYEIv2 dataset, I found that ambient light colour can significantly affect recognition performance. Thus, white balance changes were introduced additionally.

The final number of images after the expansion was 105,661, which were further divided into training and validation sets. At the same time, testing was performed on the 20% of the

consumer images, as mentioned above. In (*Ling et al., 2020*), authors reported 0.94 as the highest segmentation IoU value of their W^2-network, while I could achieve a slightly higher value of 0.956 (see Table 4.2, which also includes the results with vanilla U-Net (*Ronneberger et al., 2015*)). My metric learning models use the pill images this trained YOLOv8 model detected.

Table 4.2: Pill segmentation results on the 20% of CURE consumer images.

	U-Net (35M)	**W^2-Net** (2M)	**YOLOv8m-seg** (27.3M)
IoU	0.90	0.94	0.956

4.4.2 | Multi-stream metric learning

To guide the DNNs in finding the appropriate conceptually designed visual features, I applied a multi-stream network architecture comprising four sub-streams of very similar structures. Each sub-stream was subsequently integrated with multi-head attention modules (*Vaswani et al., 2017*), followed by concatenation and fusion to generate the final output vector. To reach the best performance, I trained my network in two consecutive phases: first, the sub-streams, then the fusing block, including the attention layers. The architecture of MSFN is shown in Figure 4.2.

4.4.2.1 | Feature streams

The idea behind using various sub-streams is to encourage the extraction and utilisation of diverse image features that may be advantageous in different situations and for different tablet classes. My approach implements the following sub-streams:

1. Contour-stream: beginning with a grayscale segmented image I_{grey}, a contour image I_{con} is generated through a series of steps. First, Gaussian smoothing with a kernel size of 7×7 is applied to I_{grey}. Subsequently, the Canny edge detector is employed.

2. LBP-stream: to avoid the usage of language-specific streams (like the OCR stream in (*Ling et al., 2020*)) and to have a homogeneous sub-stream architecture, I use LBP (*Ojala et al., 1994*) descriptors. LBP, known for its effectiveness in computer vision tasks like OCR, was computed from grayscale images I_{grey} to yield the LBP image I_{lbp}.

3. RGB-stream: as for RGB images I_{rgb}, colour input images were passed to the embedding deep neural network.

4. Texture-stream: texture images I_{tex} were generated by initially applying a Gaussian filter to the input grayscale image I_{grey}, followed by subtraction of the resulting image I_{res} from the original one.

Each sub-stream received a segmented image as an input. The output of the data streams consisted of 128 neurons, except for the RGB stream, which comprised 256 neurons.

To train the stream networks, Siamese neural networks (SNNs) were utilised with three inputs, namely anchor, positive, and negative images (denoted by I_a, I_p, and I_n, respectively). In contrast to (*Ling et al., 2020*), where the authors created a lightweight CNN network with only 9 million parameters, I opted for the EfficientNetV2-S (small-sized) network (*Tan et al., 2021b*), exploiting its optimised architecture. Figure 4.3 shows the training mechanism applied for sub-streams.

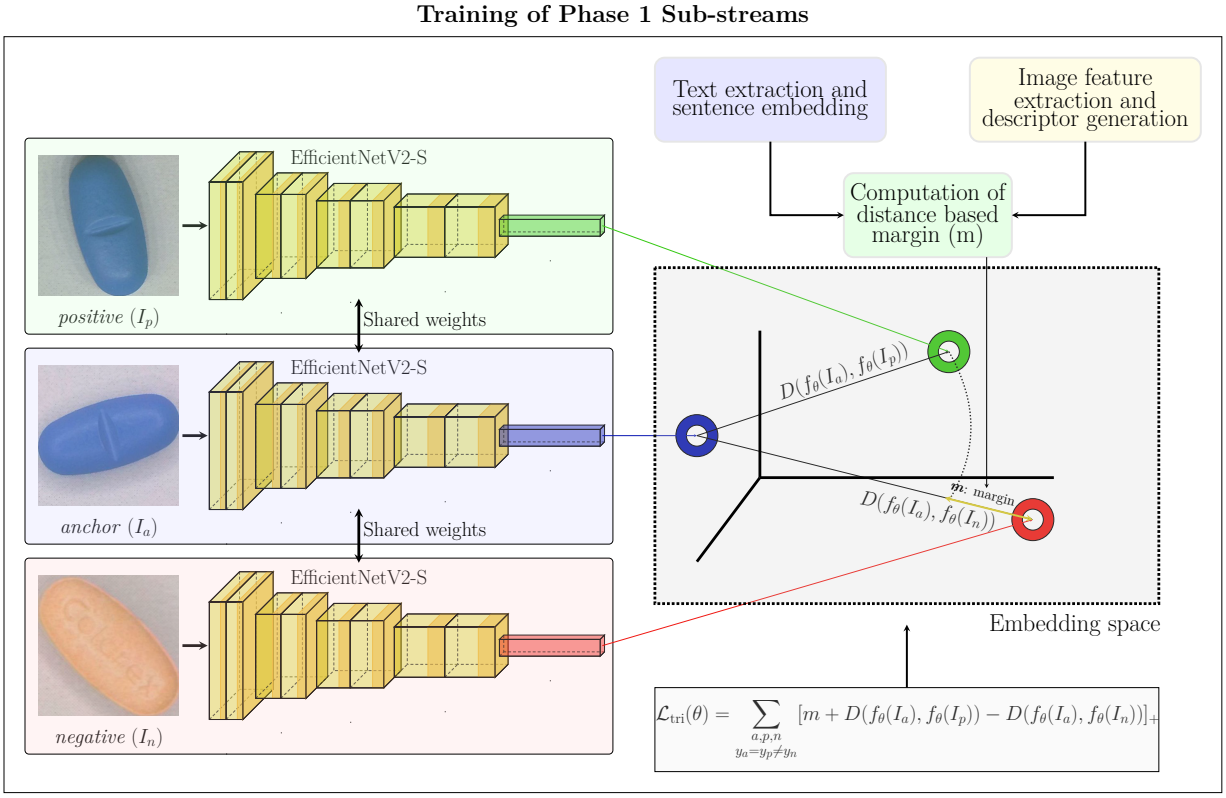

Figure 4.3: Illustration of the training process of a sub-stream. The explanation of distance function D and triplet loss $\mathcal{L}_{tri}(\theta)$ is in Section 4.4.4.

All four sub-streams were trained independently for metric embedding, using a triplet

mining strategy known as batch-all. This strategy considers all possible sample triplets within the current mini-batch during training.

During the initial phase, samples deemed challenging (called hard samples) were collected and preserved for later use in the second phase (when training the fusing layers) - this is called the batch-hard strategy. These hard samples are categorised into two groups: hard positive and hard negative. Hard positive patterns belong to the same class but exhibit feature representations likely to be tightly located in feature space, making it challenging for the model to discriminate effectively. Conversely, hard negative samples belong to different classes yet are close to each other in the feature space, which poses a challenge as the model may misclassify them as belonging to the same class.

4.4.2.2 | Fusion of streams

In the second training phase, I implemented and integrated multi-head attention modules (*Vaswani et al., 2017*) after each stream. The resulting output vectors were concatenated to aggregate the information from these modules. Subsequently, a hidden layer and an output layer were combined to generate the final embedding, resulting in a 640-dimensional vector. It is worth mentioning that at this stage, the previously trained stream networks were frozen in order to preserve their existing weight parameters. The fusion network was trained solely with hard samples with the same approach as the individual streams.

4.4.3 | Adding high-level cues

High-level cues describe features related to high-level semantic meaning that humans can easily describe and understand. Since pills are designed with the intention that people can distinguish them, I am to explicitly use this information to enhance the baseline MSFN metric model. The following semantic features were utilised as high-level visual and textual cues: colour, embossed, engraved or painted marks and texts, score, shape, and free text descriptions from leaflets.

4.4.3.1 | High-level textual cues based on word embedding

To obtain textual semantic information, I used information from the leaflets of the pills as described in Subsection 4.2. Extracted sentences were tokenized, paying close attention to high-level features such as pharmaceutical form, colour, shape, edge, imprint, embossing

or engraving. In this study, to generate high-level semantic embedding of pills, I used the `hu_core_news_lg` network, which is a CNN-based, powerful and pre-trained language model that can be found in the Python package called HuSpaCy (*Orosz et al., 2023*). The `hu_core_news_lg` model is trained on the broad Hungarian Webcorpus 2.0, comprising over 9 billion words from newspapers. Integrated into spaCy's language model pipeline, it provides tokenization, part-of-speech tagging, lemmatization, and dependency parsing. The designation "lg" in its name signifies a large model with 200,000 distinct word vectors and 300-dimensional feature vectors (called florets). To be able to plot the word embeddings, I reduced their dimensionality to 2D by employing t-Distributed Stochastic Neighbor Embedding (t-SNE) (*Van der Maaten et al., 2008*), results can be seen in Figure 4.4.

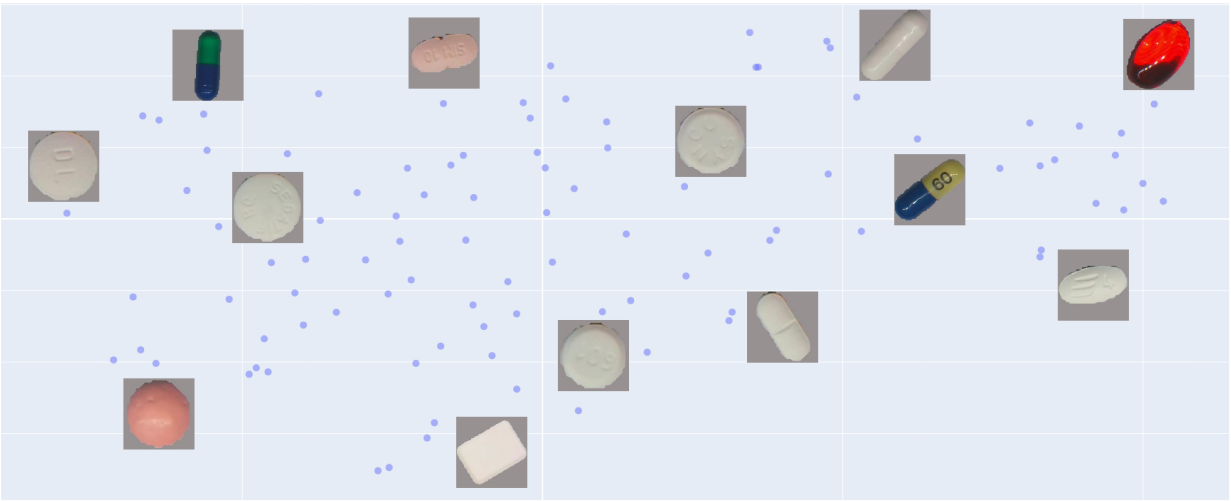

Figure 4.4: Visualisation of pill samples, from the OGYEIv2 dataset, in 2D metric space based on word embeddings using high-level textual cues.

4.4.3.2 | Extracting and embedding high-level visual cues

When no textual information is available, we can still produce high-level visual cues from good-quality pill images (such as reference images of the CURE dataset) or any labelling information that might be available for the classes. Now, I describe the steps to generate the visual descriptors in the following order: colour, markings, score, and shape.

To generate the colour feature vectors, I selected two 30×30 pixel regions on each pill, ensuring they were favourably spaced apart (for the sake of capsules often with two different colours), and sampled colours from these regions. Subsequently, the RGB values obtained from these samples were averaged and converted to the CIELAB colour space

(*Standard et al., 2007*). This perceptual colour space is used because of its relevance to colour-related visual tasks. This process yielded a 12-dimensional vector for the OGYEI and CURE two-sided datasets and a 6-dimensional vector for the CURE one-sided dataset. While using the two colour values for pills for comparisons is not straightforward, I expect the subsequent embedding network to handle complex cases to an acceptable extent.

The markings on the tablets can be categorised into one of the three classes: embossed or engraved, imprinted, and empty. I encoded these classes into $S(i) \in \mathbb{R}^3$ with the scheme shown in Table 4.3. The idea is to have a slightly more sophisticated and expressive coding pattern than one-hot encoding, used as the input of the subsequent embedding network.

Table 4.3: Encoding pill surface signs.

	$S(1)$	$S(2)$	$S(3)$
Embossed/engraved pill	0	$\frac{1}{2}$	1
Pill with printed text	$\frac{1}{2}$	0	1
Plain pill	1	1	0

This process resulted in 6-dimensional vectors for the OGYEIv2 and CURE two-sided datasets and 3-dimensional vectors for the CURE one-sided dataset.

The same approach was applied to the score of the pills. Score defines how many parts a tablet can be broken into (i.e. non-breakable, halvable, quarterable). The corresponding encoding scheme for the score (breakability) descriptor vector $B(i) \in \mathbb{R}^3$ is given in Table 4.4.

Table 4.4: Encoding scores (breakability).

	$B(1)$	$B(2)$	$B(3)$
Not breakable pill	0	1	1
Pill breakable to half	1	0	$\frac{1}{2}$
Pill breakable to quarter	1	$\frac{1}{2}$	0

Shape descriptors for the pills, to approximate and characterise the distinctive shapes of the pills, were generated using elliptic Fourier descriptors (EFD) (*Kuhl et al., 1982*). This step required pre-processing of the images, which involved Gaussian blurring and Canny edge detection. It was followed by a morphological opening, after which the most prominent

contour, based on area, was identified and used for further analysis. For each pill image, the EFD was calculated with a degree of approximation set to 10, resulting in a vector of 10×4 numbers:

- a_n: Amplitude of the cosine component in the x-direction for the n_{th} harmonic.
- b_n: Amplitude of the sine component in the x-direction for the n_{th} harmonic.
- c_n: Amplitude of the cosine component in the y-direction for the n_{th} harmonic.
- d_n: Amplitude of the sine component in the y-direction for the n_{th} harmonic.

I applied normalisation to achieve rotation and size invariance, which also means that the first three values of the coefficients could be omitted from further calculations (since normalised values were equal).

To validate the accuracy of the generated Fourier descriptors, I manually categorised pills into eight groups (as in Figure 4.5). I calculated the average of the Fourier descriptors within each, and finally, I computed the Euclidean distance between these groups as illustrated in Figure 4.6.

Figure 4.5: Illustration of shape types of pills from the CURE dataset.

The capsule's shape seems similar to an oval and a doubleround. The corresponding numbers in Figure 4.6 are 0.14 and 0.16, respectively. On the other hand, a capsule differs from the round, pentagon, and triangle shapes, where the distances are 0.52, 0.53, and 0.56. Take another example: while the distance between a pentagon and round shapes is 0.12, the distance between a pentagon and a doubleround is 0.41, which is close to the intuitive judgement of the observer.

In order to combine the descriptors of the above properties in a compact form, I designed and implemented a fully connected embedding network with a descending architecture of 4 layers, where the number of neurons was 100, 80, 60, and 10 in succession. The input to the network consisted of the concatenated descriptive vectors defined above, while contrastive loss (*Chopra et al., 2005*) was chosen to achieve metric embedding. The training ran for ten

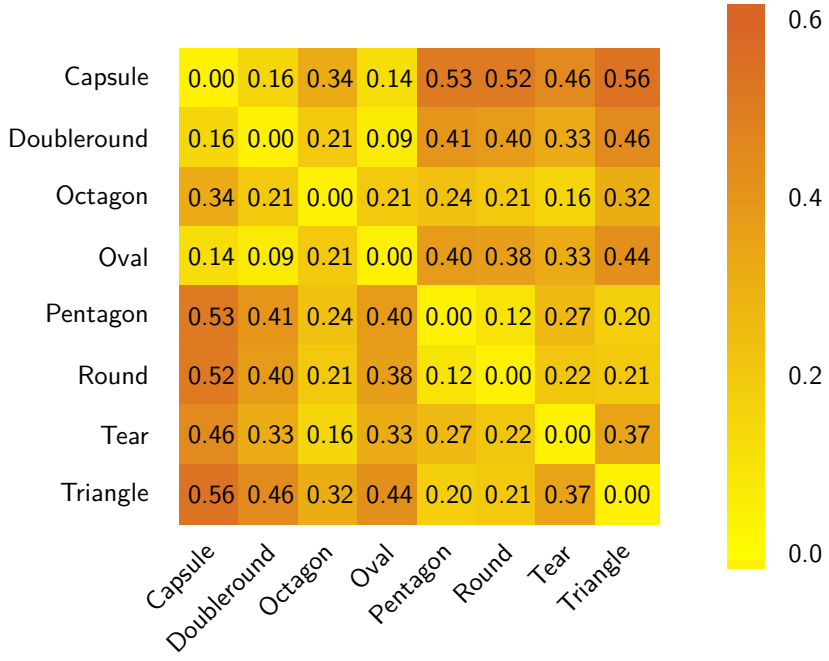

Figure 4.6: Pairwise Euclidean distances between shape groups of pills of the CURE dataset based on contour elliptic Fourier descriptors.

epochs, the batch size was 16, and the Adam optimiser was chosen with a learning rate of 5×10^{-4} and a weight decay of 1×10^{-3}. To visualise the embeddings of pills of the OGYEIv2 dataset, the 2D representation, generated with t-SNE, see Figure 4.7.

The embedding vectors, based either on free text, labels, or image data, will be used to train the multi-stream metric model. Embedding information will be used in the loss function discussed in the next section.

4.4.4 | Triplet loss function of the multi-stream model

We have already seen the illustration of the sub-stream training in Figure 4.2. I use three inputs to learn the metric space representation of images. Each triplet has an anchor sample (I_a), a sample from the same class as the anchor (positive sample, I_p), and one from a different class (negative sample, I_n). The anchor and the positive sample must originate from the same class but should not be identical images. The goal is to learn the mapping function (from the image domain to the E dimensional metric space):

$$f_\theta(I) : \mathbb{I} \rightarrow \mathbb{R}^E \tag{4.1}$$

by optimising the model parameters (θ) to minimise the distance between the feature

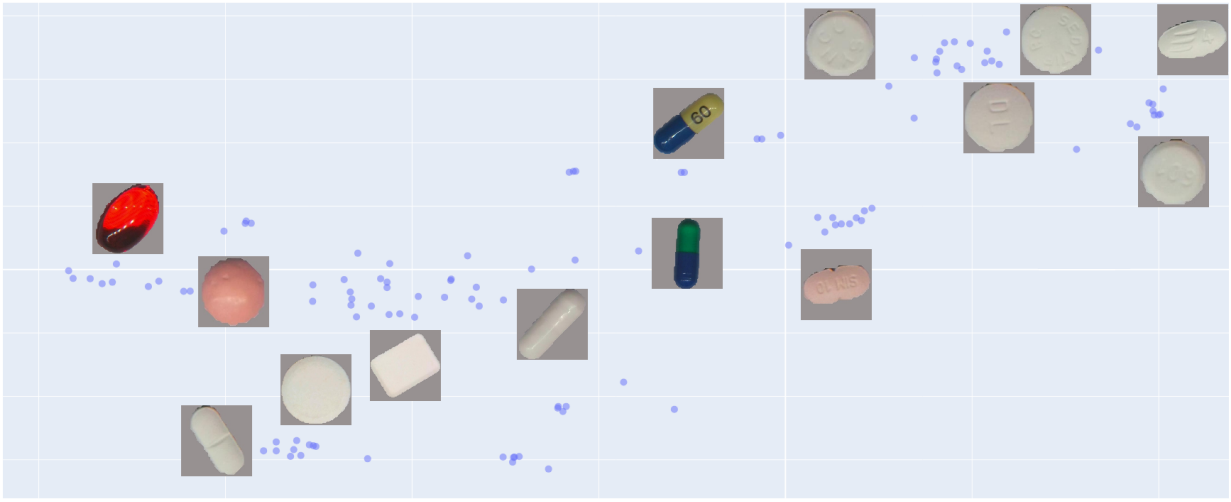

Figure 4.7: Visualisation of pill samples, from the OGYEIv2 dataset, in 2D metrics space based on high-level visual cues.

space representation of the anchor and the positive sample (I_a, I_p), and maximising the distance between the feature space representation of the anchor and the negative sample (I_a, I_n) simultaneously. This can be achieved by the following triplet loss function:

$$\mathcal{L}_{tri}(\theta) = \sum_{\substack{a,p,n \\ y_a=y_p\neq y_n}} [m + D(f_\theta(I_a), f_\theta(I_p)) - D(f_\theta(I_a), f_\theta(I_n))]_+ \tag{4.2}$$

where m represents the margin, which ensures a certain separation between positive and negative samples, and y_i denotes the pill class label for the i_{th} sample. The function $D(f_\theta(I_i), f_\theta(I_j)) : \mathbb{R}^E \times \mathbb{R}^E \to \mathbb{R}$ indicates the metric function that calculates the Euclidean distances between the two images I_i and I_j in the embedding space. Finally, $[\cdot]_+$ denotes the hinge function, $[x]_+ = \max(0, x)$, ensuring only triplets that violate the margin contribute to the loss.

In triplet metric learning models, m is chosen to be constant (*Ling et al., 2020*), (*Rádli et al., 2023c*); I proposed to dynamically set its value in paper (*Rádli et al., 2024b*) and (*Rádli et al., 2024a*). My proposal is more suitable and intuitive than the so called Log-ratio loss (*Kim et al., 2019*), where the margin also changes dynamically but does not assume positive and negative samples, which is a rather strict category in the case of pill recognition. The proposed dynamic margin triplet loss function (DMTL) uses a margin continuously set according to the conceptual difference of the actual input samples. While in (*Rádli et al., 2024b*), the value of m was based on the distances of the word embeddings, now I am extending the concept to visual data as described in Subsection 4.4.3.2. The idea is

that since we have high-level cues about the visual appearance of classes, we can use this information to position the objects in the feature space more gradually if we set m higher to less similar objects and lower for more similar ones:

$$m(a, n) = \alpha \cdot d_i^{Norm} \tag{4.3}$$

where $\alpha = 0.2$ and the normalised distance d_i^{Norm} is determined by the distance between the two samples a and n based on the high-level cue embeddings:

$$d_i^{Norm} = 1 + \frac{(u-1) \cdot (d_i - d_{min})}{d_{max} - d_{min}}. \tag{4.4}$$

Here d_i represents the Euclidean distance of samples, $d_{\max}$ stands for the maximum distance, while $d_{\min}$ denotes the minimum distance within a row of the distance matrix (formed by distance values of classes), with u as the upper limit.

4.5 | Experiments and results

It is important to note that in this research, I conduct one- and two-sided tests for the CURE dataset, while for OGYEIv2, the tests are only two-sided. (The two-sided test considers both sides of a pill to belong to the same category, whereas the one-sided test treats the two sides as belonging to two separate classes, doubling the number of classes.)

To increase the statistical robustness of the evaluation, I employed 5-fold cross-validation. Throughout the assessment process, I followed the conventional methodology used for metric learning models: the reference and consumer images underwent the metric embedding procedure, then the generated vectors were compared to each other using the Euclidean distance (see the top-right part of Figure 4.2) to find the best matching. Top-1 and Top-5 accuracies were calculated, averaged, and then converted to percentages for all five folds.

4.5.1 | Use-cases

Two use cases were examined considering the selection of the reference set:

1. Traditionally, the reference set includes all drug classes in the dataset. I refer to this as the *Complete Reference Set*, also referred to as *gallery set* in other publications. In this case, I selected one fold from the consumer classes for queries during the inference

process. However, all the reference classes are present in the candidate set (thus, the model can miss-classify a query with any of the classes - trained or not trained).

2. In the second type of reference set, I include only such classes that are present in the actual query fold, ensuring that the model compares identical consumer and reference classes. Naturally, this approach performs better, reflected in both Top-1 and Top-5 accuracy. I refer to this smaller reference set as *Actual Fold Reference Set*, and it resembles use-cases when, during pill recognition, only those classes are considered as candidates which are also in the query set (naturally with different sample images).

4.5.2 | Hyperparameter settings

Training the data streams also involved finding the best possible hyperparameter settings. Nowadays, it has become common practice to use Ray Tune (*Liaw et al., 2018*) - or any other hyperparameter tuning library - to find the proper parameters for the given task. However, implementing this in my case would have been way too cumbersome, and tuning the networks for the best parameters would have consumed too much time. Therefore, the hyperparameters were determined empirically, and the results were still acceptable.

After the segmentation, all images were resized to 224 $\times$ 224, and the pixel values were normalised to the 0 - 1 range. For the OGYEIv2 and the CURE datasets, the parameters were almost the same: in the case of the CURE dataset, I employed the Adam optimiser, the learning rate was set to 1×10^{-3} for all four streams; the batch size was 32; and the margin of the triplet loss function was chosen as 0.5. Learning rate scheduling was applied at every five epochs, where gamma was chosen to be 1/3. The weight decay was 1×10^{-5}. In the OGYEIv2 dataset, only the learning rate was set differently to 1×10^{-4}. As for the dynamic margin triplet loss (DMTL) model, I applied a weight decay regularisation with a coefficient of 1×10^{-4} and changed the learning rate to 1×10^{-4} for both datasets - everything else remained unchanged. In all cases, each model was trained for a total of 25 epochs, and only the best weight file was saved.

For the fusion phase, I also tried to apply the best possible hyper-parameters: batch size was kept at 32; the learning rate was changed to 2×10^{-4} for better stability; the weight decay was initialised to 1×10^{-8}. In this phase, I again implemented the learning rate scheduling, which involved adjusting the initial learning rate at every two epochs using a gamma value of 0.1. The network was trained over ten epochs. As with the stream-phase, only the best

weight file was saved.

I substituted the final linear classifier layer of the EfficientNetV2 models with a customised linear layer tailored to my task specification. After this adjustment, the number of parameters in the EfficientNetV2-S model comprised approximately 20.4 million for the contour, LBP, and texture streams and about 20.5 million for the RGB stream. Thus, the entire Fusion network encompassed 82.5 million parameters.

4.5.3 | Multi-stream processing without high-level cues

In this series of tests, the traditional triplet loss function was applied with static margin ($\alpha = 0.2$ as found best in (*Ling et al., 2020*)). I compare my MSFN model against the one published in (*Ling et al., 2020*), (*Pastor et al., 2023*), trained on the CURE dataset, in Table 4.5 and Table 4.6. As for the OGYEIv2 dataset, I compare the performance to the measurements of (*Rádli et al., 2024b*), data are in Table 4.7. More details of these comparisons are in Section 4.7.

Table 4.5: Average performance over 5 folds on the CURE dataset (*one-sided*) using static margin.

	(*Ling et al., 2020*)	(*Rádli et al., 2024a*)	
	Top-1 accuracy	**Top-1 accuracy**	**Top-5 accuracy**
Act. Fold Ref. Set	N.A	87.5	98.52
Comp. Ref. Set	68.30	69.96	96.90

Table 4.6: Average performance over 5 folds on the CURE dataset (*two-sided*) using static margin.

	(*Ling et al., 2020*)	(*Rádli et al., 2024a*)	
	Top-1 accuracy	**Top-1 accuracy**	**Top-5 accuracy**
Act. Fold Ref. Set	N.A	88.63	98.88
Comp. Ref. Set	84.6	85.53	97.61

4.5.4 | Multi-stream processing with textual cues

In this test, I applied the DMTL with textual cues (defined in Subsection 4.4.3.1). As mentioned above, this test could only be performed on the OGYEIv2 dataset (as no textual

Table 4.7: Average performance over 5 folds on the OGYEIv2 dataset using static margin. First two columns contain results from (*Rádli et al., 2024a*), while the last two are from (*Rádli et al., 2024b*) for comparison.

	(*Rádli et al., 2024a*)		(*Rádli et al., 2024b*)	
	Top-1 accuracy	**Top-5 accuracy**	**Top-1 accuracy**	**Top-5 accuracy**
Act. Fold Ref. Set	99.08	100	91.72	98.125
Comp. Ref. Set	96.12	100	N.A.	N.A.

information was included in the CURE database). Average results of the five folds are in Table 4.8, including the previous results of (*Rádli et al., 2024b*).

Table 4.8: Average performance over 5 folds on the OGYEIv2 dataset using dynamic margin (DM-MSFN model) with textual cues. First two columns contain results from (*Rádli et al., 2024a*), while the last two are from (*Rádli et al., 2024b*) for comparison.

	(*Rádli et al., 2024a*)		(*Rádli et al., 2024b*)	
	Top-1 accuracy	**Top-5 accuracy**	**Top-1 accuracy**	**Top-5 accuracy**
Act. Fold Ref. Set	99.23	100	93.56	99.37
Comp. Ref. Set	96.85	100	N.A.	N.A.

4.5.5 | Multi-stream processing with visual cues

In this final test series, DMTL was again applied, this time with visual cues (as defined in Subsection 4.4.3.2). All three datasets were utilised (e.g. one- and two-sided tests for CURE and two-sided tests for OGYEIv2). Results of all these tests are from (*Rádli et al., 2024a*) and are in Table 4.9.

Table 4.9: Average performance over 5 folds on the datasets using dynamic margin (DM-MSFN model) with visual cues.

	OGYEIv2		**CURE one-sided**		**CURE two-sided**	
	Top-1 accuracy	**Top-5 accuracy**	**Top-1 accuracy**	**Top-5 accuracy**	**Top-1 accuracy**	**Top-5 accuracy**
Act. Fold Ref. Set	99.14	100	89.02	98.67	89.20	99.54
Comp. Ref. Set	96.83	100	71.19	97.28	86.20	97.95

4.6 | Testing one-shot learning performance

As proved in (*Ashraf et al., 2026*), the investigated AutoML approaches and the YOLO11 model performed below expectations on the laboratory-controlled single-pill and multi-pill test sets. To address this limitation, I made an experiment to evaluate whether my proposed metric learning network can generalise in scenarios where traditional classification-based methods fall short.

In this experiment, my network was trained on the PTE-PE pill dataset, that originally contains 100 pill classes. Of these 100 classes, 76 are not represented in either the laboratory or the exhaustive test sets. Therefor training was restricted to these 76 classes to ensure strict separation between training and test categories and to prevent the model from relying on memorized class-specific representations. The training procedures (especially datasets) for both the AutoML platforms and YOLO11 differ from those used in my network; details are provided in (*Ashraf et al., 2026*).

The evaluation followed a one-shot learning scenario, in which only a single reference image was available for each test class. This setup demonstrates a realistic situation where only limited labeled data exist for newly appeared categories and provides a thorough test of the model's ability to generalize to unseen classes.

The results – as can be seen in Table 4.10 – highlight the advantages of the proposed metric learning approach, especially in situations where traditional detection or classification models show limited performance and only few reference data are available.

4.7 | Discussion

Under one-sided settings, the Top-1 accuracy reported by (*Ling et al., 2020*) was 68.3% for the publicly available CURE dataset. The proposed baseline (static) MSFN model has outperformed this accuracy by 1.66%. In the two-sided settings, the numerical advantage of the baseline MSFN model is only 0.93%, which is somewhat reasonable since all numbers are higher (and closer to 100%) in two-sided settings. It is also worth noting that the proposed MSFN model is uniform and does not require a language-specific OCR sub-stream, which significantly contributed to the accuracy of the model of (*Ling et al., 2020*).

Using high-level visual cues (DM-MSFN), I could further improve the accuracy with 1.23%

Table 4.10: Comparison of conventional detection models and the proposed metric learning network

Laboratory test dataset			Exhaustive test dataset		
Platform	**Top-1 accuracy**	**Top-5 accuracy**	**Platform**	**Top-1 accuracy**	**Top-5 accuracy**
YOLO11	40	54.74	YOLO11	41.1	67.84
Google Vertex AI	67.5	89.16	Google Vertex AI	52.79	73.87
Microsoft Azure Custom Vision	51.67	65	Microsoft Azure Custom Vision	33.61	57.55
Amazon Rekognition	43.33	74.16	Amazon Rekognition	41.97	54.48
MSFN proposed network	**90.83**	**100**	MSFN proposed network	**71.31**	**99.16**

and 0.67%, respectively, on the CURE dataset. Thus, the overall improvement is 2.89% and 1.6%.

During the assessments on the OGYEIv2 dataset, I could compare the contribution of the textual and visual high-level cues. Considering the Top-1 accuracy value of the static model with 96.12% (a high value itself), the improvement is 0.71% with the visual and 0.73% with purely textual cues. The 99.23% Top-1 accuracy of the DM-MSFN model on the Actual Fold Reference Set means that almost perfect classification is possible if images are good quality and candidates are only from those classes known during training. I also present my earlier results from (*Rádli et al., 2024b*) in Tables 4.7 and 4.8. While there is a moderate improvement in Top-5 accuracy, Top-1 accuracy shows a significant improvement. The reason for this is the pixel-wise segmentation of pills and substituting backgrounds with constant colour in the training process. Furthermore, I did not use the batch-hard strategy in (*Rádli et al., 2024b*).

While in (*Zhang et al., 2023*) (mentioned in Section 4.1), the primary purpose was to make a new method for incremental learning, we can still partly compare their results to ours using the CURE dataset. Considering their setup with 131 classes, including 40 with only five shots, they obtained 84% Top-1 accuracy. In comparison, in my setup with 196 classes, including 39 with only two shots, I reached 86.2% (given in the 5^{th} column of Table 4.9).

In Table D.2, I divided the OGYEIv2 dataset into two subsets based on the direction

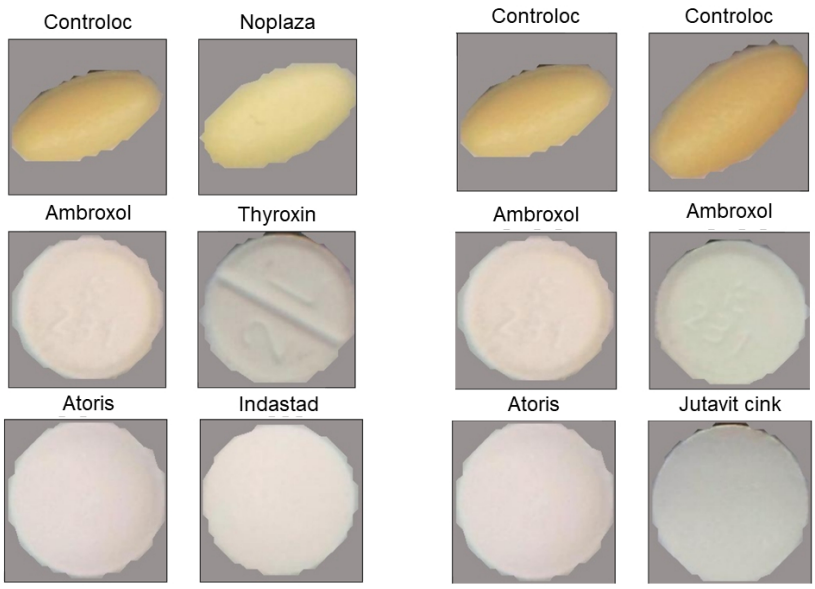

Figure 4.8: In the image pairs on the left, the images of the same types of pills are classified without any high-level cues. Conversely, in the image pairs on the right, the classification is performed using high-level cues resulting in fewer mistakes (only in the last row).

of the light illuminating the pills. It is important to note that these results are not fully comparable with those in Table D.1, as only half of the dataset was used in each training and evaluation (images with upper light and images with side light). First, it is clear that upper light images show higher Top-1 performance compared to side light ones for all streams except for the contour stream. Two example pills well illustrate this in the second line of Figure 4.9: important parts (especially scores, engravings, and embossing) are disappearing from upper light images, reasoning the performance drop of the contour stream in such cases. The contour stream could be omitted since its features resemble the texture stream's (see Figure 4.9 again). Unfortunately, by the comparison of Table 4.5, Table 4.6, Table 4.7, and Table D.3, we see that leaving out the contour stream results in circa 1.5% loss in accuracy.

In Figure 4.8, I present some complex examples of pill classification from the OGYEIv2 dataset. The image pairs on the left illustrate classifications performed without high-level cues, whereas the image pairs on the right were produced with the DM-MSFN model. In the first two rows, only the static margin variant model (where no high-level cues were utilised) misclassified the pills, whereas the dynamic margin variant correctly classified them. In the third row, both variants failed to classify the query pill accurately.

The results in Table 4.10 clearly demonstrate that the proposed metric learning network confidently outperforms traditional AutoML platforms and YOLO11, specifically

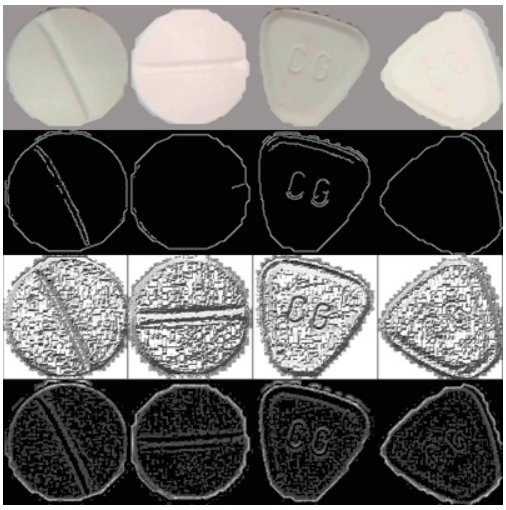

Figure 4.9: Illustration of different lightings on sub-streams by two pills (one pill in the first two columns and the other in the third and fourth columns). The first row is RGB images; the second row is contours; the third row is LBP; the fourth row is texture.

in the one-shot learning scenario. This indicates that metric learning is more effective at generalizing to unseen pill categories, even when only a single reference image is available, addressing the limitations of traditional classification-based methods in low-data or novel-class situations. These results suggest that the MSFN could be especially useful in real-world applications where new pill types frequently appear, and labeled data is limited.

4.8 | Summary

In this chapter, I introduced a new metric model and training mechanism for pill recognition. Metric learning methods are the most adequate due to the possible few-shot application scenarios. While large backbone networks are well-proved for different image recognition tasks, there are two ways that we can improve the classification accuracy of models built upon them:

1. First, my uniform, multi-stream approach is designed to efficiently capture medical pills' visual design intentions efficiently.
2. Second, the metric learning model can be improved by injecting high-level (semantic) information about the presumed appearance of classes into the loss function. This way, the discriminating capability of the metric model increases.

It is noteworthy that besides well-proven image processing features (such as Fourier shape and colour descriptors, encoding of scores, and surface sign), even free-text descriptions of

the pills in the leaflets, processed with automatic NLP methods, can be utilised to improve the accuracy of the metric model.

Since the proposed dynamic triplet loss functions can be used in many areas of metric learning, my approach could be applied in other fields of recognition (e.g. face recognition (*Schroff et al., 2015*), human pose estimation (*Kim et al., 2019*), or industrial object recognition) where discriminating features can be numerically quantified. I have shown, with numerous tests, the high accuracy of my proposal and its readiness for different applications for pill verification.

In a one-shot learning experiment, I also showed that my proposed method outperformed AutoML and object detector networks. Crucially, it was trained and tested on segregated classes, whereas the baselines were not, highlighting its superior generalization to unseen categories.

Chapter 5

Conclusion

In this dissertation, I tackled three computer vision problems, each with significant practical applications in healthcare, visual recognition and classification.

In **Chapter 2**, I compared and modified an existing SSIM-AE by increasing the number of convolutional layers of the network and turning the base model into a denoising autoencoder. The proposed SSIM-DAEe outperformed other variants in the CPU experiments without the growth of the latent space.
Furthermore, I investigated the integration of STNs into vanilla AEs. I discovered, that a pre-trained STN, to normalise the orientation of input patterns, can improve the reconstruction and defect detection, even if the input patterns produce multiple minima in the loss ($\mathbf{J}(\mathbf{x}, \omega)$) when applying the orientation normalising filter mechanism. To underlie the above statements I tested the models for three different datasets. My proposed normalisation approach could outperform the base AE method in almost all cases and metrics.

In **Chapter 3** I investigated the possibility of using randomised structures to build deep networks. These networks are generated much faster and still show better accuracy than backpropagation-trained fully connected networks. The proposed method iteratively builds a deep feed-forward model using randomised layers, pseudo-inverse optimization, pruning, and weight replacements, eliminating the need for backpropagation. This approach, based on biological analogies, is validated with more than a dozen benchmark classification datasets, demonstrating superior accuracy in most cases compared to other random or traditionally trained networks, and notably faster training compared to popular gradient-based optimizers. The technique can also suit convolutional networks and is widely usable in many application

areas.

Finally in **Chapter 4** I tackled the problem of pill recognition, with a multi-stream, two-phase metric embedding neural model. I modified the previous winner model of pill recognition challenge, by omitting the OCR sub-stream and replacing it with an LBP sub-stream. Furthermore, in the first phase, the backbone network was substituted with an EfficientNetV2 Small network, and in the second phase, I added multi-head attention modules to the network. To enhance the metric learning procedure, I introduce dynamic margin setting into the loss function. Moreover, I have shown that besides the visual features of drug samples, even free text of drug leaflets (processed with a natural language model) can be used to set the value of the margin in the triplet loss and thus increase the recognition accuracy of testing. Thus, besides using the conventional metric learning approach, the given discriminating features can be explicitly injected into the metric model using the NLP of the free text of pill leaflets or descriptors of images of selected pills.

I first evaluated existing approaches, including three major AutoML platforms (Google Vertex AI, Microsoft Azure Custom Vision, and Amazon Rekognition Custom Labels) and YOLO11. While Google Vertex AI achieved the highest accuracy among the AutoML solutions, all platforms exhibited notable performance fluctuations on clinical and real-world datasets. To demonstrate the effectiveness of my proposed method, I evaluated it on two laboratory-controlled datasets - created by myself - under a fully segregated one-shot learning setup, where training and testing classes did not overlap, and only a single reference image was provided per class during evaluation.

Furthermore I analysed the performance on two datasets and reported a 1.6% (two-sided) and 2.89% (one-sided) increase in Top-1 accuracy on the CURE dataset compared to existing best results. The inference time on CPU and GPU makes the proposed model suitable for different kinds of applications in medical pill verification; moreover, the approach applies to other areas of object recognition where few-shot problems arise. The proposed high-level feature injection method (into a low-level metric learning model) can also be exploited in other cases, where class features can be well described with textual or visual cues.

5.1 | New Scientific Results

Thesis group I *Enhancing autoencoder architectures to improve visual defect detection abilities*

1.1 If an autoencoder (AE) can reproduce (anomaly free) images with higher fidelity, it does not necessarily imply that it is better in anomaly detection.

I investigated the performance of different SSIM AEs for defect detection on repetitive and semi-repetitive structures across three datasets. In certain instances, despite achieving higher Structural Similarity Index (SSIM) values and lower Mean Squared Error (MSE) values, a model may exhibit poorer overall anomaly detection performance as measured by the Receiver Operating Characteristic Area Under the Curve (ROC AUC) compared to its counterpart. This observation suggests that while SSIM and MSE metrics indicate superior image fidelity and reconstruction accuracy, they do not consistently correlate with improved anomaly detection as reflected in ROC AUC.

1.2 I proposed a framework to show that Spatial Transformer Networks (STNs) can be trained to normalise the orientation of the input images. The combination of AEs with STNs can improve the anomaly detection efficiency of AEs.

During my investigation, I observed that simultaneous training of the convolutional layers of the AEs alongside the weights of STNs did not yield satisfactory reconstructions by the decoder. Instead, my approach focused on training the STN specifically to normalise the orientation of input images (as shown in Figure 2.8). To evaluate the effectiveness of this strategy, I assessed its performance across three classes of input patterns using reconstruction error and standard anomaly detection metrics.

The corresponding articles are [RR4] and [RR5].

Thesis group II *Fast and scalable deep randomized neural networks*

2.1 I proposed a framework where deep randomised structures are built in consecutive phases. The yielded network's training is significantly faster than backpropagation, while its classification is more accurate in most test cases with the same number of neurons.

I developed a framework that allows adding an arbitrary number of hidden layers to a

single-layer feed-forward neural network. In each phase, new neurons are added using the following three strategies:

(a) direct reuse of previously trained output neurons;

(b) perturbation of previously learned output neurons via additive Gaussian noise;

(c) random sampling with orthogonal weights.

This approach allows efficient exploration of the parameter space while preserving useful representations learned in earlier phases. Important to note that in this framework the number of neurons decreases exponentially from layer to layer.

2.2 Built upon the previous framework, I proposed a scalable method to prune the model's least contributing neurons and replace them with the most contributing neurons of the auxiliary network(s) to further improve the previous results.

My proposed iterative refinement method enhances the network's overall classification performance by preserving beneficial random connections while systematically identifying and replacing suboptimal ones. The approach leverages auxiliary networks for evaluation, selecting the most effective connections and integrating them into the original network. The method is fully scalable, allowing for the creation of an arbitrary number of auxiliary networks and extension to additional layers.

The base and enhanced versions of the method were compared against the two most widely used gradient descent-based optimisers, Adam and SGD, and a method utilising autoencoders and the FISTA optimiser across 13 datasets and showed competitive results. A visual comparison of a fully connected neural network (with the Adam optimizer) and the proposed DEV-DRNN AUX 2 can be seen in Figure 3.5.

The corresponding articles are [RR3] and [RR6].

Thesis group III *Metric-based pill recognition with the help of textual and visual cues and multi-head attention*

3.1 I proposed a multi-stream two-phase neural network by employing multi-head attention.

I created a multi-stream, two-phase neural network that integrates an EfficientNetV2-Small network as the backbone in the first phase and introduces

multi-head attention modules in the second phase, as shown in Figure 4.2. Employing CNNs and multi-head attention modules can improve the pill recognition abilities of already existing frameworks. Unlike other approaches, this method does not require semantic networks such as OCR, thus making it more efficient and independent. Evaluations were carried out on two datasets, where the proposed network achieved superior results compared to the concurrent network.

3.2 The model's accuracy can be enhanced by injecting high-level cues, gained from free-text or visual appearance, into the training of multi-stream metric models.

Expanding upon the previous architecture, I demonstrated two methods for improving the model's classification accuracy. I generated discriminative cues: one from natural language text resources and another from high-quality reference images using visual descriptors. By integrating these high-level cues, the model is better able to classify visual differences between samples, improving its overall discriminative performance.

3.3 I compared AutoML platforms and YOLO11 with my proposed method and demonstrated its superior performance under a one-shot learning setting on controlled pill recognition datasets. While the evaluation was limited to this setting, the results suggest that the proposed method offer advantages more broadly.

I trained and evaluated three widely used AutoML platforms—Google Vertex AI, Microsoft Azure Custom Vision, and Amazon Rekognition Custom Labels—alongside the YOLO11 object detector on custom pill datasets without segregating classes during training and testing. Among the AutoML platforms, Google Vertex AI achieved the highest testing accuracy; however, all platforms exhibited substantial performance variability under clinical testing conditions. To further assess robustness and generalization, I evaluated the proposed multi-stream, two-phase neural network on two controlled test sets with fully segregated training and testing classes, using only a single reference image per class at inference time. Even under this constrained one-shot learning scenario, the model maintained superior performance compared to the alternative solutions.

The corresponding articles are [RR1], [RR2], [RR7], [RR8] and [RR9].

5.2 | The author's publications

[RR1] Rádli, R., Vörösházi, Z., & Czúni, L. (2024). Metric-based pill recognition with the help of textual and visual cues. *IET Image Processing*, 18(14), 4623–4638.

[RR2] Ashraf, AR., Rádli, R., Vörösházi, Z., & Fittler, A. (2026). Code-Based Versus AutoML Methods for Pill Recognition in Clinical Settings: Comparative Performance Study. *JMIR Medical Informatics*, 14 (1), e79160

[RR3] Rádli, R., & Czúni, L. (2026). Faster and Accurate: Developmental Deep Randomized Networks. *Neurocomputing*, Submitted to review.

Full papers in conference proceedings

[RR4] Rádli, R., & Czúni, L. (2021). About the Application of Autoencoders for Visual Defect Detection. In *29. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Václav Skala - UNION Agency, pp. 181-188.

[RR5] Rádli, R., & Czúni, L. (2022). Improving the Efficiency of Autoencoders for Visual Defect Detection with Orientation Normalization. In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - Volume 4: VISAPP*, SciTePress, pp. 651-658.

[RR6] Rádli, R., & Czúni, L. (2023). Deep Randomized Networks for Fast Learning. In *International Conference on Learning and Intelligent Optimization*, Springer International Publishing, pp. 121-134.

[RR7] Rádli, R., Vörösházi, Z., & Czúni, L. (2023). Multi-stream pill recognition with attention. In *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*

(IDAACS), IEEE, Vol. 1, pp. 942-946.

[RR8] Rádli, R., Vörösházi, Z., & Czúni, L. (2023). Pill Metrics Learning with Multihead Attention. In *Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KDIR*, SciTePress, pp. 132-140.

[RR9] Rádli, R., Vörösházi, Z., & Czúni, L. (2024). Word and Image Embeddings in Pill Recognition. In *In Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: VISAPP*, SciTePress, pp. 729-736.

Appendices

Appendix of Chapter 1

Here, I will explain the metrics, not elaborated in **Chapter 1**.

A.1 | Metrics

Metrics in machine learning are crucial for evaluating and comparing the performance of models (*Bishop et al., 2024*). They provide quantitative measures to assess how well a model makes predictions and how useful those predictions are for the intended task.

A **confusion matrix** is a table used to describe the performance of a classification model. It provides a more detailed breakdown of correct and incorrect classifications for each class. The matrix is typically organized as follows:

Table A.1: Confusion Matrix.

		Predicted	
		Yes	No
Actual	Yes	True Positive	False Negative
	No	False Positive	True Negative

Where:

- **True Positives (TP)** are instances where the classifier correctly predicts the positive class. In other words, the actual class is positive, and the model also predicts it as positive.
- **False Positives (FP)** are instances where the classifier incorrectly predicts the positive class. The actual class is negative, but the model predicts it as positive.

- **True Negatives (TN)** are instances where the classifier correctly predicts the negative class. The actual class is negative, and the model also predicts it as negative.
- **False Negatives (FN)** are instances where the classifier incorrectly predicts the negative class. The actual class is positive, but the model predicts it as negative.

Accuracy is the ratio of correctly predicted instances (both true positives and true negatives) to the total number of instances.

$$Accuracy = \frac{TN + TP}{TP + FP + TN + FN} \tag{A.1}$$

Precision and **recall** are two important metrics derived from the confusion matrix, especially in the context of imbalanced datasets. Precision is the ratio of true positive predictions to the total predicted positives, while recall is the ratio of true positive predictions to the total actual positives.

$$Precision = \frac{TP}{TP + FP} \tag{A.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{A.3}$$

The **F1-score** is a metric used to evaluate the accuracy of a classification model, particularly when dealing with imbalanced datasets. It balances the precision and recall of the model, providing a single metric that considers both the false positives and false negatives. The F1-score is the harmonic mean of precision and recall, which ensures that both metrics are given equal importance. Unlike the arithmetic mean, the harmonic mean tends to be closer to the lower of the two values, making it a better measure in cases where there's a significant imbalance between precision and recall.

$$F1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{A.4}$$

In some cases it is quite useful to plot the **receiver operating characteristic** or ROC curve (*Fawcett, 2006*). This is a graph of true positive rate versus false positive rate. Sometimes it is useful to have a single number that characterises the whole ROC curve. One approach is to measure the area under the curve (AUC). A value of 0.5 for the AUC represents random guessing whereas a value of 1.0 represents a perfect classifier.

A **precision-recall curve** (PRC) is a graphical representation of the trade-off between precision and recall for different threshold settings of a classifier. The curve is plotted with precision on the y-axis and recall on the x-axis. This curve is especially useful when dealing with imbalanced datasets as it gives a more informative picture of an algorithm's performance compared to an ROC curve in these scenarios. The area under the precision-recall curve (AUC-PR) can be used as a summary statistic to compare different models. A higher area indicates a better model performance.

Intersection over Union (IoU) is a metric used primarily in the context of object detection and image segmentation tasks. It measures the overlap between the predicted bounding box and the ground truth bounding box. IoU is defined as:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}, \tag{A.5}$$

where the area of overlap is the area common to both the predicted and ground truth bounding boxes, and the area of union is the total area covered by both boxes. IoU values range from 0 to 1, with higher values indicating better performance. An IoU threshold (e.g., 0.5) is often set to determine whether a detection is a true positive. For better understanding, IoU is illustrated in Figure A.1.

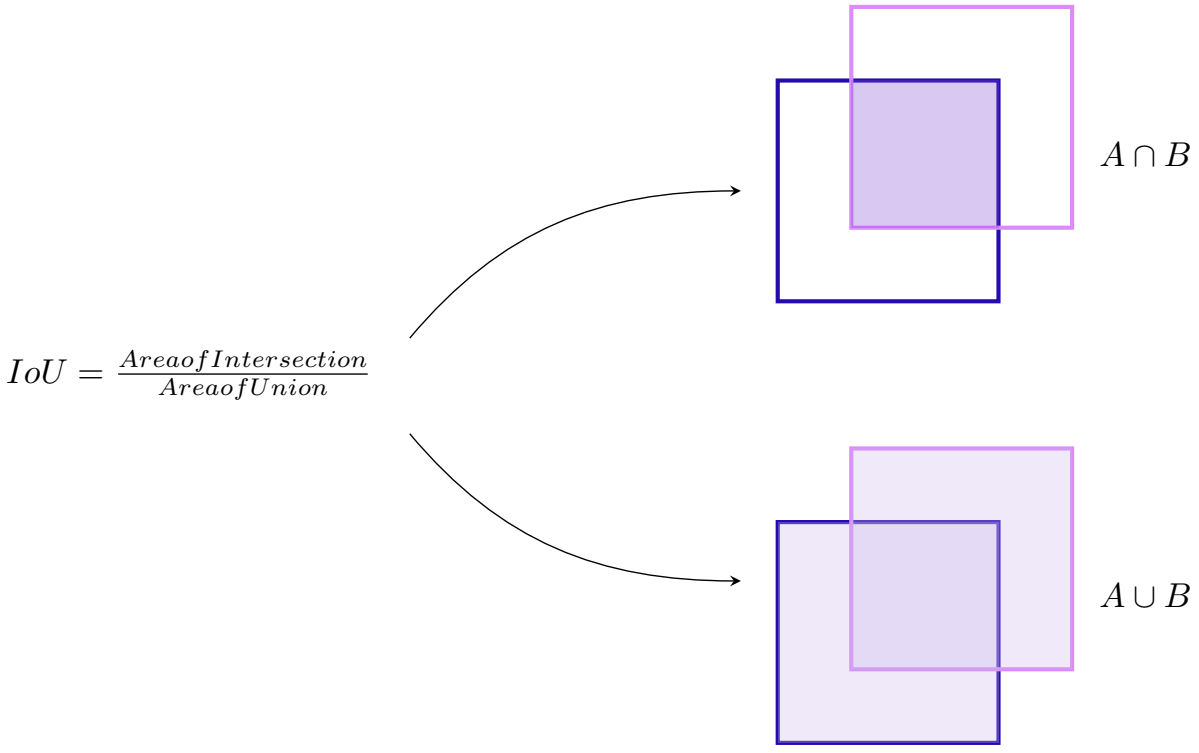

Figure A.1: Intersection over Union.

Appendix of Chapter 2

B.1 | AEs vs. GANs

AEs and Generative Adversarial Networks (GANs) (*Goodfellow et al., 2016*), (*Bishop et al., 2024*) are both widely used for unsupervised representation learning, yet they differ fundamentally in their approach to data generation. While AEs - as it is described in Chap 2 - compress input data into a lower-dimensional latent space and reconstruct it through a decoder, often leading to blurry outputs due to their reliance on minimizing reconstruction error, GANs leverage an adversarial framework where a generator and discriminator compete, typically producing sharper and more realistic samples.

GANs have their roots in a game theory scenario. Essentially, the so-called generator network has to compete with an opponent. The generator network directly generates $\mathbf{x} = g(\mathbf{z}; \theta^{(g)})$ samples, while its adversary, the discriminator network, tries to differentiate between samples taken from the training data and samples taken from the generator. The discriminator outputs the probability value given by $d(\mathbf{x}; \theta^{(d)})$, which represents the probability that $\mathbf{x}$ is a real training example rather than a false sample taken from the model.

Distributions are particularly challenging for real-world applications such as image generation, and as a result the introduction of deep learning has drastically improved the performance of generative models. Although GANs can yield high-quality results, they are not easy to train successfully due to adversarial learning. Moreover, unlike standard error function minimization, there is no measure of progress, as the target can move up and down during training.

Below are some key areas where both AEs and GANs are applied:

1. **Anomaly detection** – AEs reconstruct normal data well but struggle with anomalies, so are useful for detecting outliers. GANs, specially with adversal training, can also be employed to detect anomalies by learning the distribution of normal patterns.

2. **Image denoising, inpainting** – Both AEs (especially denoising AEs) and GANs (e.g. Context Encoders) are used to eliminate noise or fill in missing image regions. GANs generally generate more realistic textures, while AEs preserve structural consistency.

3. **Data Augmentation** – AEs can generate variations of input data, but often provide

fuzzy results. GANs, on the other hand, are widely used to produce high quality synthetic data.

B.2 | Decoder architecture of the investigated AEs

This section contains details regarding the decoder architecture of the investigated autoencoders in Section 2.4.

Table B.1: Architecture of the decoder of SSIM-AE (*Bergmann et al., 2018*).

Layer	**Output Size**	**Kernel**	**Stride**	**Padding**
ConvTranspose1	8×8×32	8×8	1	1
Conv1	8×8×64	3×3	1	1
Conv2	8×8×128	3×3	1	1
ConvTranspose2	16×16×64	4×4	2	1
Conv3	16×16×64	3×3	1	1
ConvTranspose3	32×32×32	4×4	2	1
Conv4	32×32×32	3×3	1	1
ConvTranspose4	64×64×32	4×4	2	1
ConvTranspose5	128×128×c	4×4	2	1

Table B.2: Architecture of the decoder of SSIM-AEe.

Layer	Output Size	Kernel	Stride	Padding
ConvTranspose1	8×256×256	8×8	1	0
Conv1	8×256×256	3×3	1	1
Conv2	8×256×256	3×3	1	1
ConvTranspose2	16×16×128	3×3	2	1
Conv3	16×16×128	3×3	1	1
Conv4	16×16×128	3×3	1	1
ConvTranspose3	32×32×64	3×3	1	1
Conv5	32×32×64	3×3	2	1
Conv6	32×32×64	3×3	1	1
ConvTranspose4	64×64×32	3×3	2	1
Conv7	64×64×32	3×3	1	1
Conv8	64×64×32	3×3	1	1
ConvTranspose5	128×128×c	4×4	2	1

B.3 | Additional reconstruction accuracy and defect detection results

Abbreviations in the upcoming tables are the following:

- AE = plain autoencoder
- S = STN;
- rnd. in. S w. = randomly initialised STN weights;
- pre-tr. frz. S w. = pre-trained frozen STN weights;
- pre-tr. n. frz. S w. = pre-trained not frozen STN weights.

Table B.3: Reconstruction quality of images of the class Texture 1. Abbreviations are given in the text.

	Texture 1			
	AE	**S+AE pre-tr. frz. S w.**	**S+AE rnd. in. S w.**	**S+AE pre-tr. n. frz. S w.**
MSE	82.4781	77.1195	83.0611	**74.8105**
SSIM	0.8453	0.8790	0.8411	**0.8883**

Table B.4: Reconstruction quality of images of the class Screw. Abbreviations are given in the text.

	Screw			
	AE	**S+AE pre-tr. frz. S w.**	**S+AE rnd. in. S w.**	**S+AE pre-tr. n. frz. S w.**
MSE	36.2071	**34.2324**	37.7939	35.2554
SSIM	0.8729	**0.9075**	0.8707	0.8893

Table B.5: Defect detection using SSIM and MSE metrics for the Texture 1 class. Abbreviations are given in the text.

		Texture 1			
		AE	**S+AE pre-tr. frz. S w.**	**S+AE rnd. in. S w.**	**S+AE pre-tr. n. frz. S w.**
SSIM	**ROC**	0.8622	**0.8752**	0.8591	0.8712
	PRC	0.3996	0.3993	0.3882	**0.4008**
	IoU	**0.1228**	0.1228	0.1228	0.1227
MSE	**ROC**	0.6514	**0.6694**	0.6635	0.6672
	PRC	0.2007	**0.2229**	0.2093	0.2215
	IoU	0.0744	0.0796	0.0768	**0.0803**

Table B.6: Defect detection using SSIM and MSE metrics for the Screw class. Abbreviations are given in the text.

		Screw			
		AE	**S+AE pre-tr. frz. S w.**	**S+AE rnd. in. S w.**	**S+AE pre-tr. n. frz. S w.**
SSIM	**ROC**	0,7586	**0.7911**	0.7829	0.7902
	PRC	0.4191	**0.4234**	0.4162	0.4219
	IoU	0.0088	**0.0093**	0.0088	0.0092
MSE	**ROC**	0.5179	**0.5541**	0.4833	0.4751
	PRC	0.3136	**0.3228**	0.2886	0.2823
	IoU	0.0081	**0.0093**	0.0055	0.0048

Appendix of Chapter 3

In this section I present my motivations related to biological analogies, along with related but deterministic methods, not presented in Chapter 3. Furthermore F1-score results of the methods are also among this section. Finally, some interesting limitations of the method and its possible solutions are also elaborated here.

C.1 | Biological motivations

Biological neural networks are sculpted by evolution and constraints, and many artificial neural networks are designed to solve particular tasks. The former is typically a long process, often with subtle changes, and the latter affects or can affect all elements of the model at once, causing faster changes in the design and further away from continuous convergence. Network topologies of the two systems also show significant differences, apart from some analogies. For example, in addition to neural architecture search (*Ren et al., 2021*), there are attempts to mimic biological systems' network topology. In (*Goulas et al., 2021*), the authors investigate strategies for building recurrent neural networks (RNNs) that represent the network topology of the brains of different species. Their bio-instantiated models are evaluated in terms of prediction performance and speed of training.

My goal is not to create structures similar to the brain or other specific neural systems but to use fully connected parts as the basic building blocks in a developmental process that uses random and algebraically computed weights similar to ELMs (*Huang et al., 2006*) and other random networks (*Schmidt et al., 1992*). In the developmental process, we start with a single hidden layer (fully connected, feed-forward) network, which can give a rough solution to the problem; then, by replacing some neurons and adding new layers, a solution closer to the optimum can be reached. These steps are illustrated in Figure 3.1 and Figure 3.3 and discussed in detail in Section 3.2.

Constructing my models resembles the plasticity of brain functions and the emerging nature of biological systems.

The main points of analogies with biological systems are the following:

- The sensory cortex in the human nervous system has relatively conserved input systems, where the primary sensory information processing is more fixed: it is characterised by functionally more stable but subtly modifiable connections. Some

neural connections remain stable over long periods, especially those related to essential sensory and motor functions. However, even these may show fine-tuning and adaptation, especially in the case of long-term learning or injury. Connections show more plasticity during learning, in the event of environmental change or injury, and during development (*Citri et al., 2008*), (*Gerrow et al., 2010*).

- One view (f.e. in (*Mandler, 2000*)) considers it important to distinguish between the child's ability to discriminate more subtly and later the organisation of information based on deeper conceptual knowledge. This means that the detection and processing of lower-level features, which are more detailed but less deep, is developed first, and the development of more hierarchical features, which analyse the larger context, can then take place. This is also true for problem-solving: my iteratively growing network will answer the problem initially, but the solution will be less complex and less based on deep connections, which are later formed in a hierarchy.

- An infant's brain contains many more neural connections than an adult's; the nervous system retains heavily used synapses and removes less important ones. During childhood synaptic pruning, redundant connections are removed, and relevant connections are strengthened (*Mahajan et al., 2012*). In (*Millán et al., 2018*), two kinds of mechanisms are identified. First: birth and removal of synapses (structural plasticity). New synapses are formed randomly or activity-driven, and redundant or less-used synapses are eliminated in a process called synaptic pruning. This is a structural change where some synapses physically disappear. Second: strengthening and weakening of synapses (functional plasticity). Existing synapses change in an activity-dependent manner. Intensely used synapses become more vigorous, while less active ones become weaker but still exist.

- Finally, evolution did not work by creating complex structures that worked right away, but by creating primitive ones that met environmental criteria at a basic level and then evolving these structures further in complexity (*Nilsson et al., 1994*). That is, even the shallow models should be able to answer the tasks; later, deeper models are to refine the solutions.

Table C.1: Biological motivation of my approach.

Phenomenon	**Biological Observation**	**Proposed Method**
Fixed and tuned connections	Primary sensory information processing is mostly fixed, but strong plasticity exists for adaptation.	Alternating fixed random and tuned weights. Input layers use mostly fixed random weights.
Hierarchy	Younger children have a more fine-grained but contingent discrimination ability, while information organisation based on deeper conceptual knowledge tends to be more common in older age.	Initial stages generate shallow networks; later, hierarchy is increased to discover more complex relations.
Pruning	During childhood synaptic pruning less important connections are removed while relevant connections are strengthened.	Pruning removes less efficient random connections; new neurons replace weaker ones.
Full functionality of shallow stages	Evolution first creates primitive structures that meet environmental criteria at a basic level, then refines them for greater complexity.	Shallow stages attempt to solve tasks, while deeper models refine the solution.

These concepts are summarised in Table C.1. While my approach can be described as an incremental developmental network in structure and size, the reader should not confuse it with the *incremental evolution* concept (*Tomko et al., 2010*). In that approach, the end goal task must be broken down into more manageable sub-tasks that act as intermediate steps towards the final, more complicated goal. It is worth mentioning the approach of Gao and co-authors in (*Gao et al., 2020*), which applies neither random weights nor structure

variations but still uses developmental, growing networks. The definition of sub-tasks is common with the incremental evolution concept. Instead of random weights, in (*Gao et al., 2020*), they define new partitioning tasks for the neurons of hidden layers and solve the input weights for those sub-tasks. The hidden layer outputs are again connected to the required outputs with optimal weights computed with an algebraic method. If the quality of the results is not satisfactory (and the maximum number of layers is not reached), further hidden layers can be added with further sub-tasks. Indeed, during the implementation of the approach, one must have a concept of the problem to define proper sub-tasks (e.g., clustering with a classical method). In contrast, I apply random weights, which can be seen as randomised priors without any specific design.

In (*Togelius, 2004*), the term *modularised evolution* is used for such evolving systems, where either several networks are present from the start or only one network is evolved and then duplicated during the optimisation process, but in both cases, the fitness function is the same. In all these cases, a genetic algorithm is used to optimise the data. In the concept of *differential evolution* (*Storn et al., 1997*), during the mutation process, the genes of the members of the populations change based on the difference of the genes of members. In my concept, the model grows by adding new members (neurons, layers) and improves by borrowing new neurons from similar model instances. This way, no large population is needed, but some parts of the model can be refreshed. Some still remain random.

C.2 | A brief look at some related but deterministic methods

Earlier, I mentioned in Section 3.1.1 that even gradient approaches can be considered stochastic depending on the nature of the training data feed. I briefly highlight some deterministic alternatives to the random approaches discussed above. These selected ones build hierarchical models differently but resemble the strategies applied in random methods discussed above.

- *Kernel methods*: Kernel learning methods result in good performance without tuning main parameters (such as the number of hidden neurons or weight parameters) as shown f.e. in (*Huang et al., 2011*) and (*Wong et al., 2016*). While the former applies a shallow and random network, the latter is about a kernel version of ML-ELM. Besides, the applied kernels can introduce strong non-linearity, and weights heavily rely on the distribution of input data and thus can have serious limitations in the case of big data

applications.

- *Direct partitioning*: I briefly mention the method of Gao et al. (*Gao et al., 2020*) where, instead of random weights, they define new partitioning tasks (sub-tasks) for the neurons of hidden layers and solve the input weights for those subtasks. The hidden layer outputs are again connected to the required outputs with optimal weights computed with an algebraic method. Thus, random weights are not explicitly used, but the relation of sub-task/final task implicitly can induce randomness of connections.
- *Difference-based weight generation*: The weight and bias generation method in (*Bolager et al., 2023*) does not mean random weights but a random sampling of data records from the training set. Weights of DNN layers are the function of differences between these randomly sampled points; thus, weights are larger in those dimensions where data show a wider distribution. Random selection also means a kind of implicit randomness in weights.

These approaches show some practicality, but they are also important theoretically in understanding why and how randomness can play an important role in biological or artificial neural systems. In the next part of this chapter, I turn to a developmental approach that is also observable in natural biological systems.

C.3 | Further bottom-up approaches

While H-ELM and its variants (e.g. H-LR21-ELM) have generally good accuracy so far (in the discussed context), there are several questions not answered yet: for example, how many ELM-AE blocks are necessary to achieve the best results, how regularisation constant C (see later Eq. 3.9) is determined.

The multilayer randomised solution for representational learning introduced in (*Yang et al., 2015*) was also named ML-ELM, but it applies a different idea than the method in (*Cambria et al., 2013*) with the same name. One of the purposes of this approach (and the target of the tests) was efficient dimension reduction. Here, instead of using single hidden neurons as in the basic ELM feature mapping, this method employs sub-network nodes that consist of multiple general hidden neurons. It is worth noting that a node in the basic ELM feature mapping can be considered as a special case of the sub-network node. The number of general nodes and output dimension is independent, but the number of hidden neurons

in each general node should equal the dimension of outputs.

I found three general approaches which can be used (even combined) to enhance the general ideas of ELMs, H-ELMs, or ML-ELMs:

- *Adding additional layers to the autoencoder*: One approach to improve H-ELMs or ML-ELMs is to put another non-linear layer in the ELM-AE, so it becomes hierarchical and can do a better non-linear transformation for data representation. This additional layer's input and output weights are computed via low-rank matrix factorisation in (*Ouyang, 2021*), which also helps structure important data and minimise overfitting.

- *Random shift and kernelisation*: In the realm of stacked generalisation philosophy, a different solution is defined in (*Yu et al., 2015*). The model called DrELM utilises ELM as a base building block and incorporates random shift and kernelisation as stacking elements. Specifically, in each layer, DrELM integrates a random projection of the predictions obtained by ELM into the original feature and then applies kernel functions to generate the resultant feature. In other words, the next stacked block can consider the original input and the previous prediction through a kernel function transformation.

- *Ensembles*: As for many classifiers, ensemble learning is also effective for ELMs. Examples of this are in (*Cao et al., 2012*) (voting-based ELM, V-ELM), in (*Cvetković et al., 2018*) (hierarchical ensemble of ensembles, H-ELM-E), or in (*Ksieniewicz et al., 2018*), where a trained combiner is used to integrate components for hyperspectral image classification. In (*Cai et al., 2018*), a hierarchical ensemble of ELMs (HE-ELM) is proposed where two strategies are followed to encourage the diversity of component ELMs: the sparse connection to component ELMs and feature bagging. In contrast to the previous shallow ensembles, it applies multiple-layer feature representation to achieve representation learning. Unfortunately, as the number of ELMs increases, complexity worsens (unfortunately not discussed in (*Cai et al., 2018*)).

Finally, I mention the work of Han and his co-authors (*Han et al., 2014*), who also proposed a hierarchical structure using the basic ideas of ELMs. The model is based on two initial parts (part 1 and part 2): both have a hidden layer connected with inputs; moreover, the output of part 1 is connected to the hidden layer of part 2. The inputs of the two parts overlap partially. This is somewhat similar to my proposed structure (see Figure 3.3) but in addition to the different structure and the generation of new connections, the lack of

pruning and weight replacements shows less development ability. Unfortunately, this type of models was applied to only one problem, namely the activated sludge wastewater treatment processes for predicting the water qualities.

C.4 | Experimental results

Table C.2: Testing F1-score of the different methods as the average of ten experiments. Bold numbers denote the highest values.

Dataset	Adam	SGD	H-ELM	DEV-DRNN	DEV-DRNN AUX 1	DEV-DRNN AUX 2
Connect4	0.6198	**0.6271**	0.5533	0.5515	0.5838	0.5855
Isolete	0.9149	0.9334	0.9402	**0.9495**	0.9419	0.9459
Letter	0.9643	**0.9702**	0.9153	0.9590	0.9666	0.9524
MNIST	0.9705	0.9715	**0.9780**	0.9726	0.9728	0.9750
Fashion-MNIST	0.8787	0.8907	0.8746	0.8927	0.8941	**0.8948**
Musk2	0.8634	0.9012	**0.9322**	0.9203	0.9284	0.9304
Optdigits	0.9795	0.9696	0.9918	0.9907	0.9945	**0.9945**
PageBlocks	0.7708	0.7139	0.7343	0.7405	0.7641	**0.7719**
Satimages	**0.8679**	0.8636	0.8513	0.8557	0.8597	0.8632
Segment	**0.9793**	0.9643	0.9729	0.9678	0.9700	0.9728
Shuttle	0.5137	0.4115	0.4346	0.6181	0.6457	**0.7038**
Spambase	0.9297	**0.9391**	0.9140	0.9270	0.9306	0.9301
USPS	0.9378	0.9541	**0.9720**	0.9623	0.9638	0.9652

Table C.3: Number of first positions along the evaluated metrics. Bold values are for the network with the most first places. (The sum in column Test accuracy is above 13 due to ranking equality.)

Method	Train accuracy	Test accuracy	Train F1-score	Test F1-score	Training time
Adam	0	2	0	2	0
SGD	3	3	2	3	0
H-ELM	0	3	0	3	**11**
DEV-DRNN	1	1	1	1	2
DEV-DRNN AUX 1	**6**	**4**	**8**	0	0
DEV-DRNN AUX 2	3	2	2	**4**	0

Table C.4: Number of last ranking positions along the evaluated metrics for the 13 datasets. Bold values are for the network with the most last places.

Method	Train accuracy	Test accuracy	Train F1-score	Test F1-score	Training time
Adam	4	**4**	4	**4**	3
SGD	4	**4**	4	**4**	**10**
H-ELM	**5**	**4**	**5**	**4**	0
DEV-DRNN	0	1	0	1	0
DEV-DRNN AUX 1	0	0	0	0	0
DEV-DRNN AUX 2	0	0	0	0	0

C.5 | Limitations and possible solutions

If we examine the training times in Table 3.5, it is clear that as the number of features and training instances (see Table 3.1) increases, so does the time required to train the network. It is the consequence of the complexity $O(NL^2)$ of the pseudo matrix inversion since a more significant number of features implies using larger numbers of neurons in the hidden layer. This phenomenon is clearly visible in the measured data of Figure C.1.

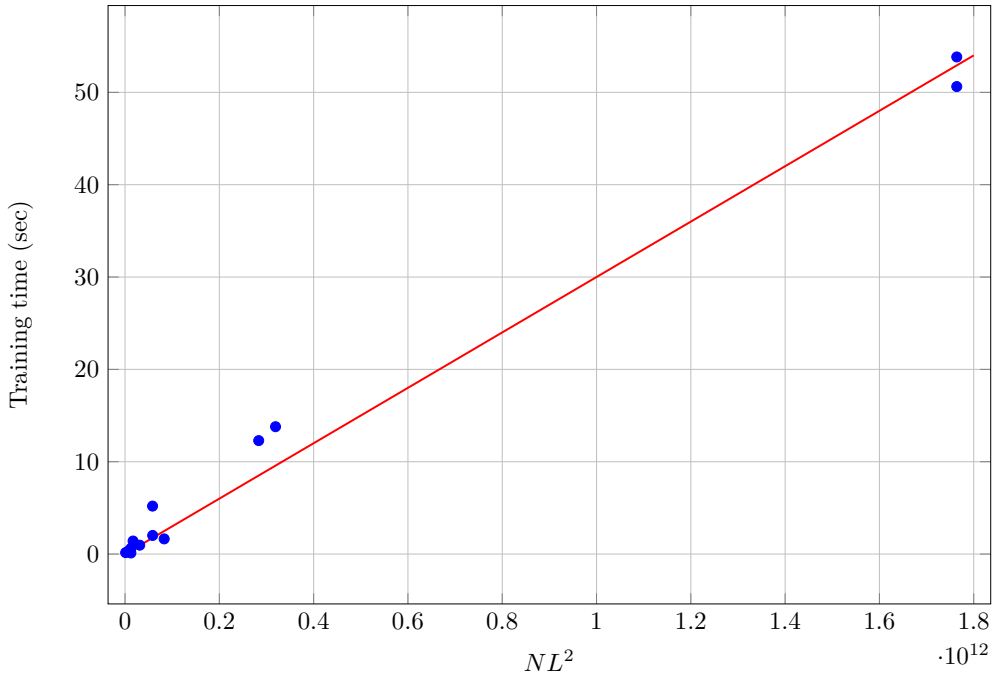

Figure C.1: Scatter plot and linear regression of NL^2 vs. training time of the DEV-DRNN approach on all 13 datasets.

The efficiency of fully connected networks may raise concern in case of low-level image features due to the high correlation of spatial information in images. Thus, it is no surprise that we have to pay by time for the dense connections to sample the whole image. To overcome these limitations, there are different simple workarounds for such problems, for example applying dimension reduction on input features or convolution if spatial coherence is expected.

While it's clear that many neurons are required in the random layers, the limitation is not only from the DEV-DRNN approach itself but also the simple structure of the fully connected networks used in my experiments. Moreover, using PCA as preprocessing, I also have the opportunity for further comparisons with the fast method of Gao et al. in (*Gao et al., 2020*), already briefly discussed in Section C.2, which also uses PCA for dimension reduction. Three classification datasets were chosen in my experiments: MNIST, Fashion-MNIST, and CIFAR-10. The training and testing split is now chosen as in (*Gao et al., 2020*) (see Table C.5 for details).

Table C.5: Main properties of the additional datasets. Under Lx the number of neurons of the actual layer is given.

Dataset	Training samples	Test samples	Features	Classes	Image	L1	L2	L3
MNIST	60,000	10,000	784	10	Yes	4,000	1,500	500
Fashion-MNIST	60,000	10,000	784	10	Yes	4,000	1,500	500
CIFAR10	50,000	10,000	1024	10	Yes	4,000	1,500	500
Adult	29,304	3,257	13	2	No	800	350	200
PageBlocks	4,925	548	10	5	No	400	200	100
Wall	4,910	546	24	4	No	600	400	300
Waveform	4,500	500	21	3	No	700	400	100

C.5.1 | Dimensionality reduction with PCA

The high dimensionality of data can easily lead to overfitting and inefficient learning. Thus, the model may fail to generalise on the training data. By reducing the dimensionality, the dataset is narrowed down to its most elemental and important or new features, compressing information and making it easier for the model to learn meaningful representations. Additionally, with lower dimensions, faster training is achievable. To investigate the effect of the well-known PCA method, I applied dimensionality reduction on three image datasets, the same way as it was in (*Gao et al., 2020*) (retaining 51, 187, and 92 dimensions for MNIST, Fashion-MNIST, and CIFAR-10, respectively by PCA) and tested the DEV-DRNN AUX 1 network on these altered datasets.

Furthermore, I compared my method without PCA and the partitioning approach of (*Gao et al., 2020*) on four non-image datasets specified in Table C.5. I ensured the dataset distributions remained consistent as in (*Gao et al., 2020*). Results can be found in Table C.6. My approach demonstrated significantly faster training times in all cases, and superior testing accuracy in 4 out of the 7 datasets. Although the difference in training times is significant, it should be treated with some caution as the performance of the two computing platforms could not be assumed to be identical. Another significant advantage of my developmental method is the unsupervised random selection of weights instead of finding proper partitioning tasks in the different iterations as described in (*Gao et al., 2020*).

Table C.6: Accuracy and training times of the method in (*Gao et al., 2020*) and my proposed solution on image and non-image datasets. Best values are highlighted in bold.

	Framework in (*Gao et al., 2020*)		**DEV-DRNN AUX 1**		**DEV-DRNN AUX 1 + PCA**	
Dataset	Testing accuracy	Training time	Testing accuracy	Training time	Testing accuracy	Training time
MNIST	97.79	190.55	97.58	173.10	**98.20**	**53.19**
Fashion-MNIST	**90.08**	229.92	89.48	175.43	89.06	**53.65**
CIFAR10 54.79	213.14	44.23	172.43	49.62	**47.25**	
Adult	**84.80**	9.14	82.94	**3.75**	NA	NA
PageBlocks	**97.63**	5.30	97.17	**4.48**	NA	NA
Wall	89.19	4.86	**90.11**	**0.93**	NA	NA
Waveform	88.00	3.76	**88.20**	**0.38**	NA	NA

C.5.2 | Modifying the method with convolution

In order to improve the effectiveness of my model on the previously mentioned three image based datasets, I introduced the following modification: the initial fully connected α weight matrix was replaced by a single convolutional layer followed by a pooling and flatten layer. The process of calculating β weights stayed the same. Furthermore I focus on identifying and replacing the least important convolutional filters in the main network with the most important filters from an auxiliary network, thereby improving the network's performance. The rest of the network stayed intact (including the number of neurons the second and third layers), and followed the previously introduced build-up strategy.

For the MNIST dataset I set the number of filters in the convolution layer to 24, kernel size to 7, for the pooling layer, I applied kernel size of 2. This resulted 3456 neurons in the first hidden layer. In case of Fashion-MNIST, I increased the number of filters up to 70, keeping the kernel size at 7, while increasing the pooling kernel size to 3. The number of neurons in the first layer was 4480. Finally, for CIFAR10 I found the best number of filters was 16, while kernel size is 3, number of neurons in the first layer was 4096. Results can be seen in Table C.7, where I compare my solution with (*Gao et al., 2020*).

Table C.7: Comparison of the method (*Gao et al., 2020*) and the modified versions of my proposed solution on image based datasets.

	DEV-DRNN AUX 1		**DEV-DRNN AUX 1 Conv**		**Framework in (*Gao et al., 2020*)**	
	Testing acc.	Training time	Testing acc.	Training time	Testing acc.	Training time
MNIST	97.58	173.10	**97.91**	136.49	97.79	190.55
Fashion-MNIST	89.48	175.43	90.02	245.98	**90.08**	229.92
CIFAR10	44.23	172.43	**55.33**	168.43	**54.79**	213.14

Appendix of Chapter 4

Finally, in this section I present the ablation studies not shown in Chapter 4.

D.1 | Ablation studies

In-depth comparisons were made with ablative models for both the static and dynamic margin models. The performance of individual streams was compared, and the results are summarised in Table D.1.

The results clearly show that the RGB stream is the most dominant because it contains all raw details and has the highest number of output nodes of all streams. However, this RGB stream alone is not sufficient, and as the results show, when fused with the other three streams, the results are continuously improved. This improvement of the static model is 5.78% for OGYEIv2, 2.15% for CURE one-sided, and 6.28% for CURE two-sided tests.

Table D.1: Individual stream processing with static margin.

		OGYEIv2		**CURE one-sided**		**CURE two-sided**	
	Ablative models	**Top-1 acc.**	**Top-5 acc.**	**Top-1 acc.**	**Top-5 acc.**	**Top-1 acc.**	**Top-5 acc.**
Individual streams	Contour	55.83	89.42	38.04	74.39	44.67	82.48
	LBP	80.96	99.62	53.98	86.47	64.85	93.62
	RGB	90.34	99.88	66.15	95.41	79.27	96.92
	Texture	78.35	98.65	47.83	83.12	56.25	88.68

Table D.2: Individual stream processing with static margin on the seperate OGYEIv2 dataset.

		OGYEIv2 side light		**OGYEIV2 upper light**	
	Ablative models	**Top-1 accuracy**	**Top-5 accuracy**	**Top-1 accuracy**	**Top-5 accuracy**
Individual streams	Contour	49.15	88.81	45.76	88.52
	LBP	60.04	94.66	61.74	94.85
	RGB	76.27	97.98	80.37	96.75
	Texture	61.05	92.98	65.48	94.00

I also ran tests where I omitted specific streams and trained the network accordingly. Results, only given for the static margin variant and on the OGYEIv2 and the CURE one-sided datasets, are summarised in Table D.3. Both tables support my decision to include

the LBP stream: after the RGB stream, LBP showed the highest accuracy (see Table D.1), and its removal caused the highest loss (see Table D.3).

Table D.3: Impact of domain related features with static margin.

		OGYEIv2		**CURE one-sided**	
	Ablated models	**Top-1 accuracy**	**Top-5 accuracy**	**Top-1 accuracy**	**Top-5 accuracy**
Impact of domain related features	w/o contour	94.69	99.84	68.42	95.77
	w/o LBP	93.14	99.70	66.57	94.91
	w/o texture	93.89	99.89	67.83	95.51

I also evaluated the inference time of the individual modules within the proposed MSFN. The reported results represent the mean inference time per single image. Evaluations involved CPU (AMD Ryzen 5 5600X) and GPU (Nvidia RTX5000) devices. Results can be seen in Table D.4, showing the model is applicable on a few pills, even with a CPU system, and applicable on a larger scale with GPU.

Table D.4: Inference time of certain modules in the proposed MSFN.

	Inference time [ms]	
	CPU	*GPU*
Segmentation	148.30	12.50
Stream - Contour	42.51	18.51
Stream - LBP	51.01	20.00
Stream - RGB	54.52	21.00
Stream - Texture	43.01	19.00
Fusion	7.25	2.00
Comparison	1.00	1.00
Total	**347.60**	**94.02**

Bibliography

[1] A Aizerman. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and Remote Control* 25 (1964), pp. 821–837.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. "A convergence theory for deep learning via over-parameterization". In: *International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. PMLR. 2019, pp. 242–252.

[3] Amir Reza Ashraf, Richárd Rádli, Zsolt Vörösházi, and András Fittler. "Code-Based Versus AutoML Methods for Pill Recognition in Clinical Settings: Comparative Performance Study". In: *JMIR Medical Informatics* 14.1 (2026), e79160.

[4] Amir Reza Ashraf, Anna Somogyi-Végh, Sára Merczel, Nóra Gyimesi, and András Fittler. "Leveraging code-free deep learning for pill recognition in clinical settings: A multicenter, real-world study of performance across multiple platforms". In: *Artificial Intelligence in Medicine* 150 (2024), p. 102844.

[5] Arthur Asuncion, David Newman, et al. *UCI Machine Learning Repository*. 2007.

[6] MMT Ayyalasomayajula, SK Chintala, and S Ayyalasomayajula. "A cost-effective analysis of machine learning workloads in public clouds: Is automl always worth using". In: *International Journal of Computer Science Trends and Technology (IJCST)* 7.5 (2019), pp. 107–115.

[7] Amir Beck and Marc Teboulle. "A fast iterative shrinkage-thresholding algorithm for linear inverse problems". In: *SIAM Journal on Imaging Sciences* 2.1 (2009), pp. 183–202.

[8] Laura Beggel, Michael Pfeiffer, and Bernd Bischl. "Robust anomaly detection in images using adversarial autoencoders". In: *Machine Learning and Knowledge*

Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I. Springer. 2020, pp. 206–222.

[9] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. "MVTec AD–A comprehensive real-world dataset for unsupervised anomaly detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9592–9600.

[10] Paul Bergmann, Sindy Löwe, Michael Fauser, David Sattlegger, and Carsten Steger. "Improving unsupervised defect segmentation by applying structural similarity to autoencoders". In: *arXiv preprint arXiv:1807.02011* (2018).

[11] Rene Bidart and Alexander Wong. "Affine Variational Autoencoders: An Efficient Approach for Improving Generalization and Robustness to Distribution Shift". In: *arXiv preprint arXiv:1905.05300* (2019).

[12] Christopher M Bishop and Hugh Bishop. *Deep learning: Foundations and concepts*. Vol. 1. Springer, 2024.

[13] Erik L Bolager, Iryna Burak, Chinmay Datar, Qing Sun, and Felix Dietrich. "Sampling weights of deep neural networks". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 63075–63116.

[14] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. Ed. by David Haussler. 1992, pp. 144–152.

[15] Michal Busta, Lukas Neumann, and Jiri Matas. "Deep textspotter: An end-to-end trainable scene text localization and recognition framework". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2204–2212.

[16] Yaoming Cai, Xiaobo Liu, Yongshan Zhang, and Zhihua Cai. "Hierarchical ensemble of extreme learning machine". In: *Pattern Recognition Letters* 116 (2018), pp. 101–106.

[17] Erik Cambria, Guang-Bin Huang, Liyanaarachchi Lekamalage Chamara Kasun, Hongming Zhou, Chi Man Vong, Jiarun Lin, Jianping Yin, Zhiping Cai, Qiang Liu, Kuan Li, et al. "Extreme learning machines [Trends & Controversies]". In: *IEEE Intelligent Systems* 28.6 (2013), pp. 30–59.

[18] Jiuwen Cao, Zhiping Lin, and Guang-Bin Huang. "Composite function wavelet neural networks with differential evolution and extreme learning machine". In: *Neural Processing Letters* 33 (2011), pp. 251–265.

[19] Jiuwen Cao, Zhiping Lin, Guang-Bin Huang, and Nan Liu. "Voting based extreme learning machine". In: *Information Sciences* 185.1 (2012), pp. 66–77.

[20] Weipeng Cao, Xizhao Wang, Zhong Ming, and Jinzhu Gao. "A review on neural networks with random weights". In: *Neurocomputing* 275 (2018), pp. 278–287.

[21] Dan Chianucci and Andreas Savakis. "Unsupervised change detection using spatial transformer networks". In: *2016 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)*. IEEE. 2016, pp. 1–5.

[22] Lenaic Chizat, Edouard Oyallon, and Francis Bach. "On lazy training in differentiable programming". In: *Advances in Neural Information Processing Systems* 32 (2019).

[23] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1251–1258.

[24] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 539–546.

[25] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. "Describing textures in the wild". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3606–3613.

[26] Ami Citri and Robert C Malenka. "Synaptic plasticity: multiple forms, functions, and mechanisms". In: *Neuropsychopharmacology* 33.1 (2008), pp. 18–41.

[27] Linda R Cronenwett, J Lyle Bootman, Julie Wolcott, Philip Aspden, et al. *Preventing medication errors*. National Academies Press, 2007.

[28] Stevica Cvetković, Miloš B Stojanović, and Saša V Nikolić. "Hierarchical ELM ensembles for visual descriptor fusion". In: *Information Fusion* 41 (2018), pp. 16–24.

[29] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in Neural Information Processing Systems* 27 (2014).

[30] Pengcheng Ding, Shouhong Wan, Peiquan Jin, and Chang Zou. "A rotation invariance spatial transformation network for remote sensing image retrieval". In: *Twelfth International Conference on Digital Image Processing (ICDIP 2020)*. Vol. 11519. International Society for Optics and Photonics. 2020, 115191P.

[31] Huong-Giang Doan, Hong-Quan Luong, Thi-Oanh Ha, and Thi Thanh Thuy Pham. "An Efficient Strategy for Catastrophic Forgetting Reduction in Incremental Learning". In: *Electronics* 12.10 (2023), p. 2265.

[32] AD Dongare, RR Kharde, Amit D Kachare, et al. "Introduction to artificial neural network". In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.1 (2012), pp. 189–194.

[33] Alexey Dosovitskiy and Thomas Brox. "Generating images with perceptual similarity metrics based on deep networks". In: *Advances in Neural Information Processing Systems* 29 (2016).

[34] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016).

[35] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874.

[36] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[37] Claudio Gallicchio and Simone Scardapane. "Deep randomized neural networks". In: *Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL2019)*. Ed. by Luca Oneto, Nicolò Navarin, Alessandro Sperduti, and Davide Anguita. Springer. 2020, pp. 43–68.

[38] Zhentao Gao, Yuanyuan Chen, and Zhang Yi. "A novel method to compute the weights of neural networks". In: *Neurocomputing* 407 (2020), pp. 409–427.

[39] Erol Gelenbe, Zhi-Hong Mao, and Yan-Da Li. “Function approximation with spiked random networks”. In: *IEEE Transactions on Neural Networks* 10.1 (1999), pp. 3–9.

[40] Kimberly Gerrow and Antoine Triller. “Synaptic stability and plasticity in a floating world”. In: *Current Opinion in Neurobiology* 20.5 (2010), pp. 631–639.

[41] Raja Giryes, Guillermo Sapiro, and Alex M. Bronstein. “Deep Neural Networks with Random Gaussian Weights: A Universal Classification Strategy?” In: *IEEE Transactions on Signal Processing* 64.13 (2016), pp. 3444–3457.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[43] Alexandros Goulas, Fabrizio Damicelli, and Claus C Hilgetag. “Bio-instantiated recurrent neural networks: Integrating neurobiology-based network topology in artificial networks”. In: *Neural Networks* 142 (2021), pp. 608–618.

[44] Aman Gupta, Rohan Ramanath, Jun Shi, and S Sathiya Keerthi. “ADAM vs. SGD: Closing the generalization gap on image classification”. In: *OPT2021: 13th Annual Workshop on Optimization for Machine Learning*. Ed. by Aman Gupta, Rohan Ramanath, Jun Shi, and S Sathiya Keerthi. 2021.

[45] Hong-Gui Han, Li-Dan Wang, and Jun-Fei Qiao. “Hierarchical extreme learning machine for feedforward neural network”. In: *Neurocomputing* 128 (2014), pp. 128–135.

[46] Stephen Hanson and Lorien Pratt. “Comparing biases for minimal network construction with back-propagation”. In: *Advances in Neural Information Processing Systems* 1 (1988).

[47] Tasnuva Hassan and Haider Adnan Khan. “Handwritten bangla numeral recognition using local binary pattern”. In: *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*. IEEE. 2015, pp. 1–4.

[48] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. “Masked autoencoders are scalable vision learners”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009.

[49] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. “Mask R-CNN”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2961–2969.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[51] Junyeong Heo, Youjin Kang, SangKeun Lee, Dong-Hwa Jeong, and Kang-Min Kim. “An accurate deep learning–based system for automatic pill identification: Model development and validation”. In: *Journal of Medical Internet Research* 25 (2023), e41043.

[52] Huang. *HELM code*. `https://www.extreme-learning-machines.org`. [Online; accessed 1-July-2024]. 2013.

[53] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang. “Extreme learning machine for regression and multiclass classification”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.2 (2011), pp. 513–529.

[54] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. “Extreme learning machine: theory and applications”. In: *Neurocomputing* 70.1-3 (2006), pp. 489–501.

[55] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. “Spatial transformer networks”. In: *Advances in Neural Information Processing Systems* 28 (2015), pp. 2017–2025.

[56] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. [Online; accessed 1-July-2024]. 2023. URL: `https://github.com/ultralytics/ultralytics`.

[57] Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. “Automl to date and beyond: Challenges and opportunities”. In: *ACM Computing Surveys (CSUR)* 54.8 (2021), pp. 1–36.

[58] Sungyeon Kim, Minkyo Seo, Ivan Laptev, Minsu Cho, and Suha Kwak. “Deep metric learning beyond binary supervision”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2288–2297.

[59] Diederik P Kingma. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[60] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems* 25 (2012).

[62] Paweł Ksieniewicz, Bartosz Krawczyk, and Michał Woźniak. “Ensemble of Extreme Learning Machines with trained classifier combination and statistical features for hyperspectral data”. In: *Neurocomputing* 271 (2018), pp. 28–37.

[63] Frank P Kuhl and Charles R Giardina. “Elliptic Fourier features of a closed contour”. In: *Computer Graphics and Image Processing* 18.3 (1982), pp. 236–258.

[64] Hyuk-Ju Kwon, Hwi-Gang Kim, and Sung-Hak Lee. “Pill detection model for medicine inspection based on deep learning”. In: *Chemosensors* 10.1 (2021), p. 4.

[65] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[66] Matthew CH Lee, Ozan Oktay, Andreas Schuh, Michiel Schaap, and Ben Glocker. “Image-and-spatial transformer networks for structure-guided image registration”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 337–345.

[67] Claude Lemaréchal. “Cauchy and the gradient method”. In: *Doc Math Extra* 251.254 (2012), p. 10.

[68] Gen Li, Shikun Xu, Xiang Liu, Lei Li, and Changhu Wang. “Jersey number recognition with semi-supervised spatial transformer network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1783–1790.

[69] Rui Li, Xiaodan Wang, Yafei Song, and Lei Lei. “Hierarchical extreme learning machine with L21-norm loss and regularization”. In: *International Journal of Machine Learning and Cybernetics* 12.5 (2021), pp. 1297–1310.

[70] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. "Tune: A Research Platform for Distributed Model Selection and Training". In: *arXiv preprint arXiv:1807.05118* (2018).

[71] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2980–2988.

[72] Suiyi Ling, Andreas Pastor, and Jing ... Li. "Few-shot pill recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9789–9798.

[73] Huaping Liu, Fengxue Li, Xinying Xu, and Fuchun Sun. "Multi-modal local receptive field extreme learning machine for object recognition". In: *Neurocomputing* 277 (2018), pp. 4–11.

[74] Lixia Liu, Honggang Zhang, Aiping Feng, Xinxin Wan, and Jun Guo. "Simplified local binary pattern descriptor for character recognition of vehicle license plate". In: *2010 Seventh International Conference on Computer Graphics, Imaging and Visualization*. IEEE. 2010, pp. 157–161.

[75] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "SSD: Single shot multibox detector". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer. 2016, pp. 21–37.

[76] Yatin Mahajan and Genevieve McArthur. "Maturation of visual evoked potentials across adolescence". In: *Brain and Development* 34.8 (2012), pp. 655–666.

[77] Jean M Mandler. "Perceptual and conceptual processes in infancy". In: *Journal of Cognition and Development* 1.1 (2000), pp. 3–36.

[78] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.

[79] Yoan Miche, Antti Sorjamaa, Patrick Bas, Olli Simula, Christian Jutten, and Amaury Lendasse. "OP-ELM: optimally pruned extreme learning machine". In: *IEEE Transactions on Neural Networks* 21.1 (2009), pp. 158–162.

[80] Ana P Millán, JJ Torres, Samuel Johnson, and Joaquin Marro. "Concurrence of form and function in developing networks and its role in synaptic pruning". In: *Nature Communications* 9.1 (2018), p. 2236.

[81] Eliakim H Moore. "On the reciprocal of the general algebraic matrix". In: *Bulletin of the American Mathematical Society* 26 (1920), pp. 294–295.

[82] Amr M Nagy and László Czúni. "Classification and fast few-shot learning of steel surface defects with randomized network". In: *Applied Sciences* 12.8 (2022), p. 3967.

[83] Vladimir Nasteski. "An overview of the supervised machine learning methods". In: *Horizons. B* 4.51-62 (2017), p. 56.

[84] Deanna Needell, Aaron A Nelson, Rayan Saab, Palina Salanevich, and Olov Schavemaker. "Random vector functional link networks for function approximation on manifolds". In: *Frontiers in Applied Mathematics and Statistics* 10 (2024), p. 1284706.

[85] Ariel Neufeld and Philipp Schmocker. "Universal approximation property of random neural networks". In: *arXiv preprint arXiv:2312.08410* (2023).

[86] Anh Duy Nguyen, Thuy Dung Nguyen, Huy Hieu Pham, Thanh Hung Nguyen, and Phi Le Nguyen. "Image-based contextual pill recognition with medical knowledge graph assistance". In: *Asian Conference on Intelligent Information and Database Systems*. Springer. 2022, pp. 354–369.

[87] Anh Duy Nguyen, Huy Hieu Pham, Huynh Thanh Trung, Quoc Viet Hung Nguyen, Thao Nguyen Truong, and Phi Le Nguyen. "High accurate and explainable multi-pill detection framework with graph neural network-assisted multimodal data fusion". In: *PLoS ONE* 18.9 (2023), e0291865.

[88] Dan-E Nilsson and Susanne Pelger. "A pessimistic estimate of the time required for an eye to evolve". In: *Proceedings of the Royal Society of London. Series B: Biological Sciences* 256.1345 (1994), pp. 53–58.

[89] Timo Ojala, Matti Pietikainen, and David Harwood. "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions". In: *Proceedings of 12th International Conference on Pattern Recognition*. Vol. 1. IEEE. 1994, pp. 582–585.

[90] György Orosz, Gergő Szabó, Péter Berkecz, Zsolt Szántó, and Richárd" Farkas. "Advancing Hungarian Text Processing with HuSpaCy: Efficient and Accurate NLP Pipelines". In: *Text, Speech, and Dialogue*. Ed. by Kamil Ekštein, František Pártl, and Miloslav Konopík. Cham: Springer Nature Switzerland, 2023, pp. 58–69. ISBN: 978-3-031-40498-6.

[91] Yang-Yen Ou, An-Chao Tsai, Jhing-Fa Wang, and Jiun Lin. "Automatic drug pills detection based on convolution neural network". In: *2018 International Conference on Orange Technologies (ICOT)*. IEEE. 2018, pp. 1–4.

[92] Tinghui Ouyang. "Feature learning for stacked ELM via low-rank matrix factorization". In: *Neurocomputing* 448 (2021), pp. 82–93.

[93] Yoh-Han Pao, Gwang-Hoon Park, and Dejan J Sobajic. "Learning and generalization characteristics of the random vector functional-link net". In: *Neurocomputing* 6.2 (1994), pp. 163–180.

[94] Dongwon Park and Se Young Chun. "Classification based grasp detection using spatial transformer network". In: *arXiv preprint arXiv:1803.01356* (2018).

[95] Andreas Pastor, Suiyi Ling, Jieum Kim, and Patrick Le Callet. "Chapter 15 - Case study: few-shot pill recognition: How to train an AI model to recognize a new category of pill from only a few samples like humans?" In: *Meta Learning With Medical Imaging and Health Informatics Applications*. Ed. by Hien Van Nguyen, Ronald Summers, and Rama Chellappa. The MICCAI Society book Series. Academic Press, 2023, pp. 275–299. ISBN: 978-0-323-99851-2. DOI: https://doi.org/10.1016/B978-0-32-399851-2.00024-7. URL: https://www.sciencedirect.com/science/article/pii/B9780323998512000247.

[96] Benedikt Pfülb, Alexander Gepperth, Saad Abdullah, and André Kilian. "Catastrophic forgetting: still a problem for DNNs". In: *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*. Springer. 2018, pp. 487–497.

[97] Richárd Rádli, József Bene, and Zsolt Vörösházi. *OGYEIv2*. [Online; accessed 1-July-2024]. 2023. DOI: 10.34740/KAGGLE/DSV/8417535. URL: https://www.kaggle.com/dsv/8417535.

[98] Richárd Rádli and László Czúni. “Deep Randomized Networks for Fast Learning”. In: *International Conference on Learning and Intelligent Optimization*. Ed. by Meinolf Sellmann and Tierney Kevin. Springer. 2023, pp. 121–134.

[99] Richárd Rádli, Zsolt Vörösházi, and László Czúni. “Multi-Stream Pill Recognition with Attention”. In: *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Vol. 1. IEEE. 2023, pp. 942–946.

[100] Richárd Rádli, Zsolt Vörösházi, and László Czúni. “Metric-based pill recognition with the help of textual and visual cues”. In: *IET Image Processing* 18.14 (2024), pp. 4623–4638.

[101] Richárd Rádli, Zsolt Vörösházi, and László Czúni. “Word and Image Embeddings in Pill Recognition”. In: *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2024)*. Vol. 3. 2024, pp. 729–736.

[102] Ali Rahimi and Benjamin Recht. “Random features for large-scale kernel machines”. In: *Advances in Neural Information Processing Systems* 20 (2007).

[103] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. “What’s hidden in a randomly weighted neural network?” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Ed. by Terry Boult, Gerard Medioni, Ramin Zabih, Eric Mortensen, and Margaux Masson-Forsythe. 2020, pp. 11893–11902.

[104] Joseph Redmon and Ali Farhadi. “YOLOv3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[105] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. “A comprehensive survey of neural architecture search: Challenges and solutions”. In: *ACM Computing Surveys (CSUR)* 54.4 (2021), pp. 1–34.

[106] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *The Annals of Mathematical Statistics* (1951), pp. 400–407.

[107] Hai-Jun Rong, Yew-Soon Ong, Ah-Hwee Tan, and Zexuan Zhu. “A fast pruned-extreme learning machine for classification problem”. In: *Neurocomputing* 72.1-3 (2008), pp. 359–366.

[108] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015): 18th International Conference, Munich, Germany, October 5-9*. Vol. Part III 18. Springer. 2015, pp. 234–241.

[109] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958). Ed. by Frank Rosenblatt, p. 386.

[110] Amir Rosenfeld and John K Tsotsos. "Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing". In: *2019 16th Conference on Computer and Robot Vision (CRV)*. Ed. by Liam Paull and Michael Brown. IEEE. 2019, pp. 9–16.

[111] Mayu Sakurada and Takehisa Yairi. "Anomaly detection using autoencoders with nonlinear dimensionality reduction". In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. 2014, pp. 4–11.

[112] Wouter F Schmidt, Martin A Kraaijveld, Robert PW Duin, et al. "Feed forward neural networks with random weights". In: *International Conference on Pattern Recognition*. IEEE Computer Society Press. 1992, pp. 1–1.

[113] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 815–823.

[114] Nina Shvetsova, Bart Bakker, Irina Fedulova, Heinrich Schulz, and Dmitry V Dylov. "Anomaly detection in medical imaging with deep perceptual autoencoders". In: *IEEE Access* 9 (2021), pp. 118571–118583.

[115] Jaswinder Singh and Rajdeep Banerjee. "A study on single and multi-layer perceptron neural network". In: *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE. 2019, pp. 35–40.

[116] Daniel Soukup and Thomas Pinetz. "Reliably decoding autoencoders' latent spaces for one-class learning image inspection scenarios". In: *Proceedings of the OAGM Workshop*. 2018, pp. 90–93.

[117] CI Standard et al. "Colorimetry-part 4: CIE 1976 L* a* b* colour space". In: *International Standard* (2007), pp. 2019–06.

[118] Rainer Storn and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of Global Optimization* 11 (1997), pp. 341–359.

[119] Ponnuthurai N Suganthan and Rakesh Katuwal. "On the origins of randomization-based feedforward neural networks". In: *Applied Soft Computing* 105 (2021), p. 107239.

[120] Yuan Sun, Qiurong Song, Xinning Gui, Fenglong Ma, and Ting Wang. "AutoML in the wild: Obstacles, workarounds, and expectations". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 2023, pp. 1–15.

[121] Lu Tan, Tianran Huangfu, Liyao Wu, and Wenying Chen. "Comparison of RetinaNet, SSD, and YOLOv3 for real-time pill identification". In: *BMC Medical Informatics and Decision Making* 21 (2021), pp. 1–11.

[122] Mingxing Tan and Quoc Le. "EfficientNetv2: Smaller models and faster training". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10096–10106.

[123] Jiexiong Tang, Chenwei Deng, and Guang-Bin Huang. "Extreme learning machine for multilayer perceptron". In: *IEEE Transactions on Neural Networks and Learning Systems* 27.4 (2015), pp. 809–821.

[124] Yingjie Tian, Yuqi Zhang, and Haibin Zhang. "Recent advances in stochastic gradient descent in deep learning". In: *Mathematics* 11.3 (2023), p. 682.

[125] B Tiryaki and EA Oral. "Traffic sign classification using Spatial Transformer Networks (STN) based Convolutional Neural Networks (CNN)". In: *Int. J. Transp. Dev. Integr* 9.4 (2025), pp. 840–850.

[126] Migel D Tissera and Mark D McDonnell. "Deep extreme learning machines: supervised autoencoding architecture for classification". In: *Neurocomputing* 174 (2016), pp. 42–49.

[127] Julian Togelius. "Evolution of a subsumption architecture neurocontroller". In: *Journal of Intelligent & Fuzzy Systems* 15.1 (2004), pp. 15–20.

[128] Nicholas Tomko and Inman Harvey. "Do not disturb: Recommendations for incremental evolution". In: *Proceedings of ALIFE XII, the 12th International Conference on the Synthesis and Simulation of Living Systems*. Ed. by Nicholas Tomko and Inman Harvey. 2010.

[129] Nina Tuluptceva, Bart Bakker, Irina Fedulova, and Anton Konushin. "Perceptual image anomaly detection". In: *Asian Conference on Pattern Recognition*. Springer. 2019, pp. 164–178.

[130] VAIPE. *VAIPE-Pill: A Large-scale, Annotated Benchmark Dataset for Visual Pill Identification*. `https://vaipe.org/`. [Online; accessed 1-July-2024]. 2008.

[131] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.11 (2008).

[132] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in Neural Information Processing Systems* 30 (2017).

[133] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. 2008, pp. 1096–1103.

[134] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." In: *Journal of Machine Learning Research* 11.12 (2010).

[135] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.

[136] Chi Man Wong, Chi Man Vong, Pak Kin Wong, and Jiuwen Cao. "Kernel-based multilayer extreme learning machines for representation learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.3 (2016), pp. 757–762.

[137] Yimin Yang and QM Jonathan Wu. "Multilayer extreme learning machine with subnetwork nodes for representation learning". In: *IEEE Transactions on Cybernetics* 46.11 (2015), pp. 2570–2583.

[138] Ziv Yaniv, Jessica Faruque, Sally Howe, Kathel Dunn, David Sharlip, Andrew Bond, Pablo Perillan, Olivier Bodenreider, Michael J Ackerman, and Terry S Yoo. "The National Library of Medicine pill image recognition challenge: An initial report". In: *2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE. 2016, pp. 1–9. DOI: 10.1109/AIPR.2016.8010584.

[139] Wenchao Yu, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. "Learning deep representations via extreme learning machines". In: *Neurocomputing* 149 (2015), pp. 308–315.

[140] Yash Vivek Zambre, Ekdev Rajkitkul, Akshatha Mohan, and Joshua Peeples. "Spatial Transformer Network YOLO Model for Agricultural Object Detection". In: *2024 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, Dec. 2024, pp. 115–121. DOI: 10.1109/icmla61862.2024.00022. URL: http://dx.doi.org/10.1109/ICMLA61862.2024.00022.

[141] Xiao Zeng, Kai Cao, and Mi Zhang. "MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images". In: *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. 2017, pp. 56–67.

[142] Jinghua Zhang, Li Liu, Kai Gao, and Dewen Hu. "A Forward and Backward Compatible Framework for Few-shot Class-incremental Pill Recognition". In: *arXiv preprint arXiv:2304.11959* (2023).

[143] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. "Deconstructing lottery tickets: Zeros, signs, and the supermask". In: *Advances in Neural Information Processing Systems* 32 (2019).

[144] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Chu Hong Hoi, et al. "Towards theoretically understanding why SGD generalizes better than ADAM in deep learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21285–21296.

[145] Qin-Yu Zhu, A Kai Qin, Ponnuthurai N Suganthan, and Guang-Bin Huang. "Evolutionary extreme learning machine". In: *Pattern Recognition* 38.10 (2005), pp. 1759–1763.

[146] Marc-André Zöller and Marco F Huber. “Benchmark and survey of automated machine learning frameworks”. In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 409–472.