Using deep neural networks to understand the temporal structure of human memory

Auteur : Navez, Lucie
Promoteur(s) : Louppe, Gilles; D'Argembeau, Arnaud
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/17788

Using deep neural networks to understand the temporal structure of human memory

---

Author:
Lucie Navez

Supervisors:
Prof. Gilles Louppe
Prof. Arnaud D'Argembeau

Master's thesis completed in order to obtain the degree of
Master of Science in Computer Science and Engineering
by NAVEZ Lucie

Universiy of Liège
School of Engineering and Computer Science

University of Liège
Liège, Belgium

Academic year 2022-2023

Author:
Lucie Navez

Supervisors:
Prof. Gilles Louppe
Prof. Arnaud D'Argembeau

Institute:
University of Liège, Belgium

CONTENTS

ABSTRACT

This thesis explores the temporal structure of human memory through the use of deep neural networks. The research aims to extend the study conducted by Roseboom *et al.* [1] in "Activity in perceptual classification networks as a basis for human subjective time perception", which investigates the process of time perception based on non-temporal factors, by using a vision classification deep neural network in order to mimic human vision. Building upon this work, the present study focuses on predicting recalling times of episodic memories using a different dataset and further explores the influence of various neural network architectures as well as identifying critical layers and other factors in the prediction process.

In addition to the extension of the work of Roseboom *et al.* [1] to a new task, this study also extends the research scope by investigating the influence of different neural network architectures and identifying the layers that play crucial roles in accurate recall time predictions. Through a deep analysis, it becomes obvious that the temporal dynamics of time perception in humans can be effectively transposed to the specific task of episodic memories recall. Furthermore, we discover that certain types of neural network architectures, particularly more recent ones, such as change detection networks, exhibit superior performance in capturing and modeling the temporal dynamics of memory when compared to traditional feedforward neural networks.

This research provides an analysis of using deep neural networks to comprehend the temporal structure of human memory. The findings have significant implications for the fields of cognitive science and neural networks. Future research in this area can exploit these insights to further advance our understanding of the temporal aspects of memory and contribute to the progress of knowledge in the field.

STATEMENT OF AUTHORSHIP

I, Lucie Navez, born July 18, 2000, hereby confirm that this master's thesis entitled *Using deep learning to understand the temporal structure of the human memory* is my original work and has not been submitted for any other degree or qualification. I further confirm that I have worked under the supervision of Pr. Gilles Louppe and Pr. Arnaud D'Argembeau, who both provided guidance and support throughout the project.

I declare that I have contributed to the conception, design, execution, and interpretation of the research presented in this project. All sources used in the research have been properly acknowledged and referenced. Except where referenced, this thesis does not contain any material previously published elsewhere.

I acknowledge that certain portions of my original text have been reformulated with the assistance of the ChatGPT language model for the purpose of enhancing clarity and readability.

Lucie Navez
Academic year 2022-2023

ACKNOWLEDGMENTS

I am grateful for the support of numerous individuals who made the achievement of this work possible. During my time at ULiège, I had the privilege of meeting individuals who left a mark on my academic journey, including family, friends, and professors. To all of you, I address a heartfelt thank you for making this experience unforgettable.

I would like to take this opportunity to express my sincere gratitude to my supervisor, **Prof. Gilles Louppe**, for his guidance, support, and availability throughout the elaboration of this thesis. His clarity of thought, willingness to provide regular feedback, and trust in my abilities were invaluable.

I would also like to thank my co-supervisor, **Prof. Arnaud D'Argembeau**, for his knowledge and expertise in areas that I needed assistance with, and for providing invaluable feedback and support throughout the project, as well as constructive criticism, and insightful discussions all along.

Furthermore, I am grateful to all of the academic staff who contributed to my growth and learning during my five years of studies, including professors, assistants, and student instructors.

Finally, I would like to extend my thanks to my loved ones for their constant support, both emotional and technical, and for their patience during difficult moments. Without them, I would not have been able to overcome the challenges I faced along the way.

1 INTRODUCTION

The field of artificial intelligence has been a rapidly growing area of research for several decades, and it has seen enormous advancements in recent years with the emergence of deep learning techniques. Deep neural networks have been shown to be highly effective in a wide range of applications, such as computer vision, natural language processing (NLP), and speech recognition, among others.

However, as deep learning models continue to improve, researchers are more and more interested in exploring their potential applications beyond traditional computer science fields. In particular, there has been growing interest in applying deep learning techniques to better understand the mechanisms underlying human cognition.

One promising area of research is the study of human memory. Memory is a fundamental aspect of cognition, and understanding its mechanisms could have important implications for a wide range of fields, including neuroscience, psychology, and artificial intelligence.

The following sections provide an in-depth analysis of the structure of this thesis and its objectives, as well as an introduction to the paper around which it gravitates.

1.1 DEEP NEURAL NETWORKS IN THE CONTEXT OF HUMAN MEMORY

Deep Neural Networks (DNNs) have been receiving significant attention in recent years for their ability to model complex data and perform tasks that were previously considered impossible for machines[2–6]. In particular, the application of DNNs is believed to be a tool that could significantly improve the comprehension of human brain (more specifically human memory), which has long been considered a highly enigmatic subject[7–10]. DNNs have thus emerged as a promising perspective in the pursuit of a better understanding of this fundamental cognitive process. Several studies have investigated the human time-perception structure using DNNs[1, 11, 12].
A notable example is the work by Roseboom *et al.* [1] which showed that the activity in perceptual classification networks can account for variations in subjective time perception. This study[1] showed that deep convolutional neural networks can be used to model the human time perception of various scenes. Specifically, the team worked on building a pipeline that computes a subjective time measure based on convolutional neural network activations and changes occurring in a scene provided as input of this network. The results obtained from this first step are

used as a training set for a machine-learning regression model that predicts the duration of the scene similarly to how humans would evaluate it.
The objectives defined by the authors of this paper are numerous:

1. Provide a new hypothesis of how human time perception is accomplished, based on non-temporal processes.
2. Use a deep neural network trained on a visual classification task to generate stimuli for the time perception task.
3. Test the hypothesis that the temporal structure of neural activity in deep networks corresponds to the subjective experience of time in humans.
4. Contribute to the general understanding of the relationship between neural activity and subjective experiences, with implications for fields such as psychology and neuroscience.

1.2 THESIS RESEARCH PROBLEM STATEMENT

The research problem addressed in this thesis is the understanding of the temporal structure of human memory using deep neural networks. While the use of DNNs has shown promising results in various applications, including computer vision[2, 5, 13] and natural language processing (NLP)[14, 15], their use in modeling human memory is relatively new and unexplored.

This thesis aims to extend the work of Roseboom *et al.* [1] on a dataset designed to test human memory, more specifically rememoration times. In their work, the authors postulate that the quantity of change occurring in stimuli is predictive of the duration of videos as perceived by humans. In this work, we want to see if this postulate also applies to the recalling time of episodic memories by checking if the quantity of change is also predictive of the time necessary to replay a memory mentally. Additionally, this thesis aims to explore the impact of using different deep neural network architectures for predicting these recall times and to identify which layers of the network are most important for making accurate predictions.

1.2.1 *Objectives*

Specifically, the thesis aims to achieve the following objectives:

1. To extend the study conducted by Roseboom *et al.* [1], but using a different dataset designed for rememoration tasks.
2. To investigate the influence of different kinds of neural network architectures on the accuracy of predicting the recall duration of episodic memories.
3. To identify which layers of the deep neural networks are the most important in order to replicate the recalling process of episodic memories occurring in human memory.
4. To identify if the quantity of change is predictive of episodic memories recalling time in general.

1.2.2 *Significance of the thesis*

Achieving the objectives depicted in 1.2.1 will contribute to the understanding of how deep neural networks can be used to model and potentially replicate the processes underlying human memory. It will also provide insights into the strengths and limitations of different neural network architectures in predicting the temporal structure of human memory, which could have implications for developing more effective artificial intelligence systems. The findings could also highlight the potential of deep neural networks in modeling complex cognitive processes. The results of this research can inform future studies on the neural basis of human memory and provide a basis for the development of new computational models that can simulate and predict human memory performance.

2

LITERATURE REVIEW

The study of human memory is a hot topic in the field of neuroscience and cognitive psychology[16, 17]. Over the years, researchers have explored different ways to understand how human memory works and how it can be modeled using computational methods[18–20]. In this section, we first explore a model explaining the phenomenon of time compression in human memory[21]. Additionally, we explore computational methods that aim to model biological processes using artificial intelligence, and are currently considered as promising approaches to simulate human brain behavior[22–26].

2.1 TIME COMPRESSION IN HUMAN MEMORY

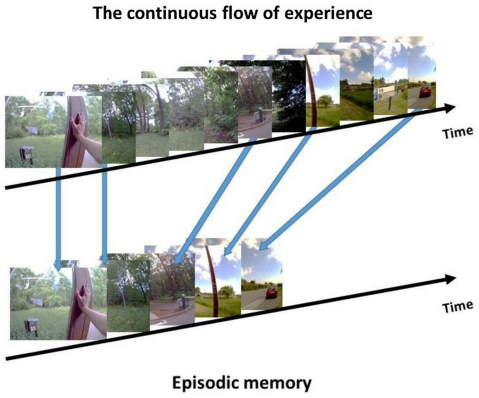


Figure 2.1: Temporal compression of events. The continuous flow of experience is stored and encoded in memory as a chronological sequence of some parts of the episode, that leaves temporal gaps in the unfolding of events, making the memory shorter in time than the original event. Source: Slices of the past: how events are temporally compressed in episodic memory.[21]

This work aims to better understand the temporal compression of events in episodic memory by trying to model this behavior using artificial neural networks.
This field of study is not very well known. Thus, the number of theories and papers that cover this is limited. However, we will discuss here a few theories as explained in related works[27–32], and reassembled by D'Argembeau *et al.* [21]. This section is based on this paper and the knowledge it gathers.

Human memory cannot store every event that occurred in the daily life of a human-being. However, as humans, we are able to recall a few events that happened in the past, with a relative

precision, and sometimes, some omissions. Thus, we can directly guess that memory stores some of these events, and allows us to recall them. A second point concerns the recalling events. When recalling any event stored in memory, the remembering duration is generally shorter than the actual duration of the event as it first occurred. This suggests that the memory does not store the exact event, but stores it with a certain compression rate, which makes it shorter to remember. Questions also arise on the nature of this compression rate.

The remembering of events is also not continuous. This means that when trying to recall previous events, the memory will not be continuous, and some jumps into the episode might be observed, some holes may appear between two parts of the episode, and an individual would not be able to retrieve the content of these holes.

Recent studies proved that this temporal compression of events occurs in memory by simply comparing the recalling time to the actual event duration[29, 32–34]. This was proven for repeated events, for which experience may be acquired from their repetitions, as well as for unique events.

AN EXAMPLE OF EXPERIMENT ON REPEATED EVENTS In 2016, an experiment was led by Bonasia *et al.* [29] on mental navigation in humans. This experiment aimed to verify if the temporal compression rate could be measured, how it was affected, and if the nature of the events had an influence on it. Participants were asked to mentally navigate campus routes that they take on a daily basis. The time that they took to remember these routes was directly comparable to the actual duration of taking these routes. Indeed, it can be very easily measured how long it takes to go from one point to another taking these routes. In the results, they observed that for all participants, and for every route, the time needed to navigate it mentally was shorter than the actual time that would be needed to do so in real conditions, showing that a temporal compression indeed occurs. Also, it was shown that the temporal compression rate was not the same for every participant, neither for every route. It seems that this compression rate varies depending on the route nature: its length, the number of turns, etc.

AN EXAMPLE OF EXPERIMENT ON UNIQUE EVENTS In a study by Jeunehomme and D'Argembeau [32], participants wore small cameras that captured a continuous sequence of pictures while they toured a campus. They were then asked to mentally replay specific events captured in the pictures and indicate the duration of their mental replay (episodic remembering). The results showed that participants consistently compressed their mental replay of events, recalling them at an average rate approximately eight times faster than their actual durations. The compression rates varied across events, which aligns with findings from previous studies.

From the previous works cited here, it can be concluded that episodic remembering indeed has a lower duration time than the real duration of the original event. However, the compression rate with which the episode is stored in memory seems to vary a lot with the event itself and its

content (more specifically, the amount of changes identified by individuals. For instance, changes in locations, entities, and actions).

2.2 BIOLOGICALLY PLAUSIBLE DEEP LEARNING

The use of artificial neural networks to model human mechanisms has been widely explored in recent years[2, 13, 14], but many of these models do not take into account the biological plausibility of their learning rules and architectures. There have been attempts to develop biologically plausible solutions that can be used in artificial neural networks. In this section, we will review promising approaches in this field, focusing on biologically plausible architectures and learning algorithms.

2.2.1 *Biologically plausible architectures*

Biologically plausible architectures are neural networks that are designed to simulate the structure and function of the human brain. Their goal is to better understand how biological systems process information. One approach to achieve this is to gradually increase the complexity of traditional neural network architectures, starting with simpler ones like convolutional neural networks (CNNs). These are inspired by the structure and function of the visual cortex in the human brain. The early layers of a CNN learn to detect simple features such as edges and lines, while the deeper layers learn to detect more complex features such as shapes and objects. This organization allows CNNs to classify images with high accuracy and efficiency.

Then, move on to more complex architectures like recurrent neural networks (RNNs). Recurrent Neural Networks have connections between their neurons that form a directed cycle, allowing them to process sequences of input data such as speech or text. The ability to remember previous inputs and use them to inform future predictions makes RNNs well-suited for tasks such as language modeling, speech recognition, and time series prediction.

Finally, there exist architectures that more closely resemble the organization of the brain itself[22–26]. These allow for a more nuanced understanding of how different neural network architectures perform on different tasks and can provide insights into how the brain processes and stores information.

In recent years, researchers have progressively turned to biologically plausible architectures as a way to develop models of the human brain. One such architecture is the Spiking Neural Network (SNN)[24], which models the behavior of neurons in the brain more accurately than traditional artificial neural networks. SNNs have been used to model a range of cognitive processes, including visual attention, working memory, and decision-making, among others. Other biologically plausible architectures that have been explored include Liquid State Machines (LSMs)[25], which are based on the dynamics of recurrent neural networks, and Echo State

Networks (ESNs)[26], which use a fixed recurrent network as a reservoir to process input signals.

These biologically inspired architectures offer several advantages over traditional artificial neural networks in terms of biological accuracy, and they can capture the temporal dynamics of neural activity and are able to simulate the emergence of complex behaviors from the interactions of many simple components, as in the human brain.

2.2.1.1 *Hierarchical Convolutional Neural Networks (HCNNs)*

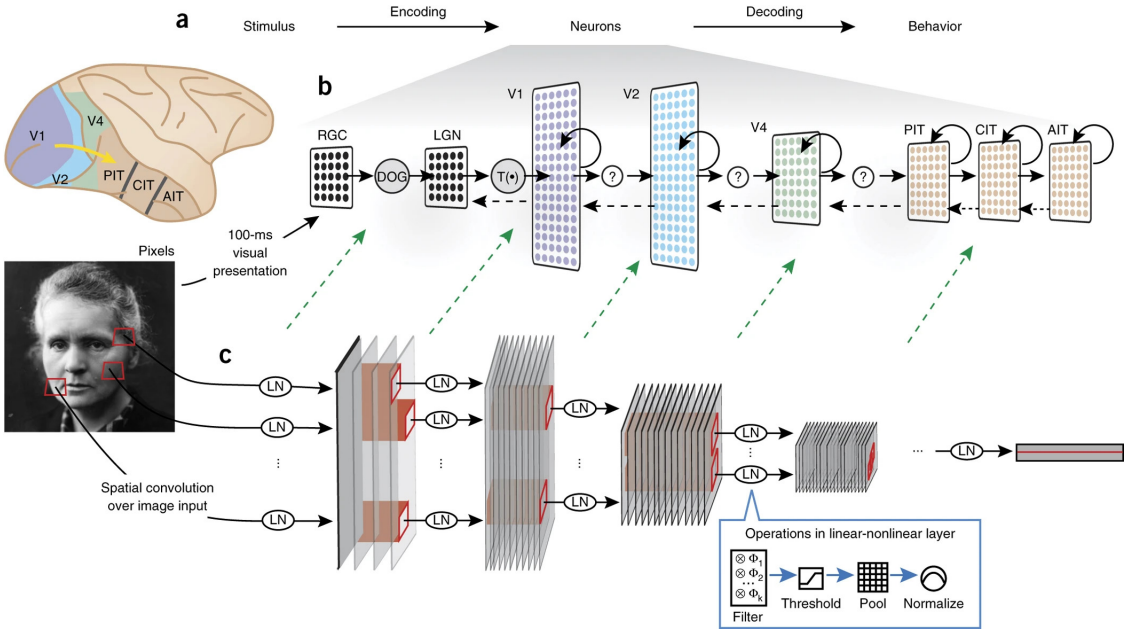


Figure 2.2: (a) Sensory cortex is studied through encoding and decoding: HCNNs model stimulus-to-neural response mapping. (b) The ventral visual pathway includes connected cortical areas. (c) HCNNs are multilayer networks with local operations and complex transformations. They map with observable structures of the ventral visual pathway and accurately predict neural response patterns. Source: Using goal-driven deep learning models to understand sensory cortex[23].

Hierarchical Convolutional Neural Networks (HCNNs) have emerged as a valuable tool for modeling neural responses in higher visual cortical areas[23].

Cortical sensory systems consist of a series of distinct but connected areas. The neural activity is transmitted in cascade in these areas, and each area applies some basic transformations to the information it receives. The succession of these transformations results in complex non-linear transformations. The human brain is thus able to compute a huge amount of non-linear transformations, and thus, identifying the role of each visual cortical area is a challenge.
To achieve this goal, models should be inspired by the hierarchical organization of the visual system. They should take as input a stimulus, and should output a prediction of real neural responses to the same stimulus. Moreover, their components should correspond to actual cortical areas, and the neural responses obtained in each component of the network should be

similar to the real responses observed in the corresponding brain area.

HCNNs are stacks of layers containing simple, repeated, neural circuit patterns (for instance, filtering, activations, pooling, normalization, see Section 3.2). These basic layers are then composed in series. Thus, even if each layer performs simple operations, the composition of such layers produces a complex non-linear transformation of the input, which is the observed behavior in the cortical area of the human brain. Each layer of an HCNN is basically linked to a single cortical area, as can be observed in Figure 2.2. HCNNs can also be stacked into deep networks. They do not denote a single precise architecture, but rather a family of parameterized models. Any HCNN must satisfy the following properties:

- Architectural parameters must be discrete.
- Filter parameters must be continuous.

Thus, finding a single parameters combination following these criteria, and for which the associated HCNN layers are mappable to distinct areas of the cortex would allow to model neural responses in the same fashion as in human brain.

2.2.1.2 *Spiking Neural Networks (SNNs)*

Spiking Neural Networks (SNNs)[24, 35–37] are a newer, more biologically close to a brain architecture type of neural networks. They simulate the behavior of actual neurons better than traditional architectures by using spiking activations rather than continuous-valued activation functions (such as ReLus, sigmoids, etc.). These activations better mimic the electrical spikes observed in neurons.

ELECTRICAL-TRANSMISSION IN NEURONS Action potentials are generated in neurons as a way to transmit signals over long distances within the nervous system. They are a rapid and transitory change in the electrical potential across the neuron's cell membrane.

When a neuron receives a strong enough stimulus, it causes a depolarization of the membrane potential, which occurs when an influx of positive ions enter the neuron through channels. This influx of positive charges gradually reverses the polarity of the membrane potential (it is negatively charged at rest). If the depolarization reaches a certain threshold, it triggers the opening of more channels, resulting in a cascade of positive feedback. The reversal of the membrane potential from negative to positive is complete after this phase. This phase is known as the depolarization phase.

The next phase is the re-polarization phase, in which new channels open, allowing the positive charges that previously entered the neuron to get out, which restores the negative charge inside the cell back to its resting state.

Thus, once the threshold is reached, the action potential will occur with a fixed amplitude and duration, generating a spike, characterizing the transmission of specific signals.

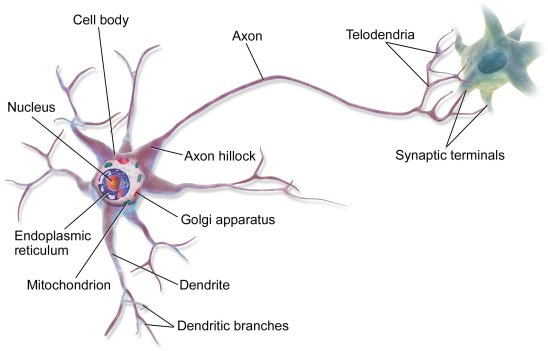


Figure 2.3: A biological neuron and its different parts. Source: Wikipedia[38].

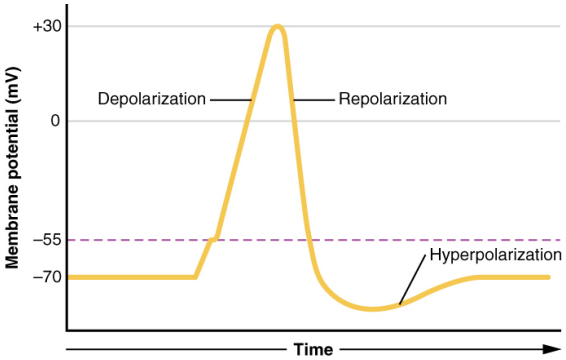


Figure 2.4: The action potential during its different phases. Depolarization corresponds to the influx of positive Na^+ ions, re-polarization corresponds to the efflux of positive K^+ ions. Source: Wikipedia[39].

The generation of multiple spikes at interval of times creates what is called a **spike train**, and indicates how intensely, and when a neuron got activated by an action potential. This **spike train** contains useful information, because many patterns can be generated:

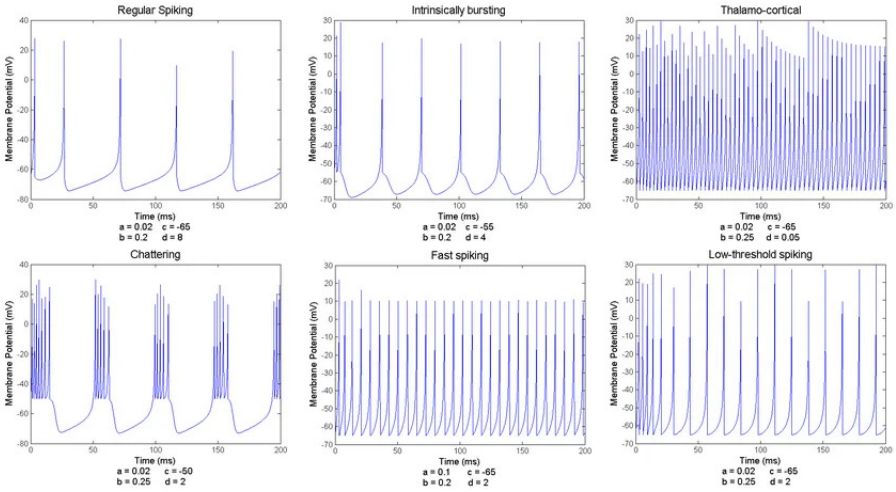


Figure 2.5: Different types of spike trains, generating different spiking patterns. Source: Medium, An introduction to spiking neural networks (part 1), Chiplunkar [37]

PHOTO-RECEPTORS The main task that neural networks are expected to accomplish in this work is vision. Hence, this paragraph provides some insight on how neurons responsible for vision handle this task. These specific neurons are known as photo-receptors. They are directly connected to the visual cortex (the part of the brain that is responsible for vision) through the optic nerve and other neurons located in the eyes.

There are two types of photo-receptors:

- Rods: Highly sensitive to light, responsible for vision in low-light conditions. Their density is higher in the peripheral region of the retina.
- Cones: Less sensitive to light, they are responsible for color vision and visual acuity. They are more concentrated in the central region of the retina (fovea).

Each photo-receptor is receptive to light in a different way. The incidence angle and the wave-length of rays, the intensity, etc. determine the photo-receptor response. This is called the **receptive field**, and this receptive field is responsible for the generation of an action potential, as described in previous paragraph.

SNNs operate based on the timing and frequency of spikes: neurons in an SNN accumulate input signals over time, and generate a spike that is transmitted to connected neurons once a certain threshold is reached. This behavior captures the timing, and temporal patterns of information processing, which is a good thing, because the spike timing and frequency can serve as an encoding of information. This encoding can capture important features and contribute in a positive way to the performance of such networks.

2.2.2 *Biologically plausible learning algorithms*

In the pursuit of biologically plausible architectures, it is important to consider the algorithms used for learning. In machine learning, a commonly used algorithm is backpropagation.

BACKPROPAGATION Traditionally, back-propagation[40] aims to compute the gradient of a loss function with respect to the network's weights and update them to minimize the loss.

Backpropagation involves two phases: forward propagation computes the activations of each neuron layer by layer, while backward propagation calculates the error and updates the weights based on the gradients to minimize the prediction error (See Figure 2.6). Given a dataset of size n, $\left\{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right\}_{k=0}^{n}$ and a loss function $\mathcal{L}$, a neural network of N layers is trained by updating the weights of its weight matrix $\mathbf{W}$ using backpropagation. Given an activation function ϕ and the bias at the *ith* layer $\mathbf{b}_i$, the output of the *ith* layer $\mathbf{y}_i$ of such a neural network is given by:

$$\mathbf{y}_i = \phi(\mathbf{z}_i) \qquad \text{with } \mathbf{z}_i = \mathbf{W}_i \mathbf{y}_{i-1} + \mathbf{b}_i \tag{2.1}$$

Thus, the equation 2.2 shows the synaptic weights update rule with backpropagation:

$$\delta \mathbf{W}_i = -((\mathbf{W}_{i+1}^T \delta \mathbf{z}_{i+1}) \odot \phi'(\mathbf{z}_i))\mathbf{y}_{i-1}^T, \qquad \text{with } \delta \mathbf{z}_{i+1} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_{i+1}} \tag{2.2}$$

However, although this mechanism is efficient, backpropagation is not biologically plausible due to several reasons[22]:

- The weight update in backpropagation is purely linear, while biological neurons exhibit both linear and non-linear behaviors.

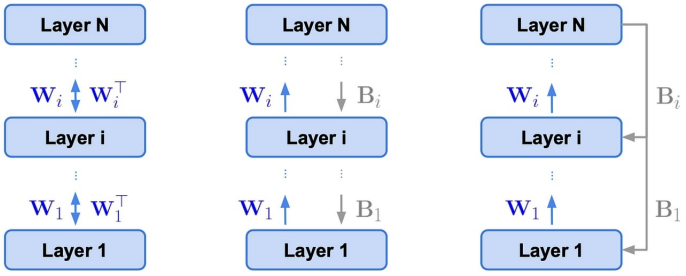


Figure 2.6: Different learning algorithms. (a) Backpropagation, where the blue arrow denotes the forward and backward propagation, utilizing the same weights for both. (b) Feed-back alignment, where the blue arrow denotes the forward propagation, and the grey arrow denotes backward propagations, utilizing different weights. (c) Direct Feedback Alignment (not discussed here). Source: Benchmarking the Accuracy and Robustness of Feedback Alignment Algorithms[41]

- Precise knowledge of the derivatives of non-linearities used in feedforward computation is required during backward propagation, which is not feasible in real brains.
- Backpropagation assumes that weights for both forward and backward propagation are symmetric, whereas synapses in the brain are unidirectional.
- Real neurons communicate through binary spikes (see Section 2.2.1.2), unlike the continuous values in backpropagation.
- Biological systems do not exhibit any precise timing that alternates between forward and backward propagation like backpropagation does.
- The origin of output targets, crucial for learning, remains unclear in biological implementation.

To address these limitations and achieve biologically plausible learning, alternative algorithms have been proposed. One approach is feedback alignment[41].

FEEDBACK ALIGNMENT Feedback alignment methods aim to overcome the weight symmetry issue in backpropagation, making it more biologically plausible. Instead of using symmetric weights for both forward and backward propagation, feedback alignment employs different synaptic weights and randomizes the backward propagation weights[42–45].

Given a dataset of size n, $\{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\}_{k=0}^{n}$ and a loss function $\mathcal{L}$, a neural network of N layers is trained by updating the weights of its weight matrix $\mathbf{W}$ using backpropagation. Given an activation function ϕ and the bias at the ith layer $\mathbf{b}_i$, the output of the ith layer $\mathbf{y}_i$ of such a neural network is denoted by:

$$\mathbf{y}_i = \phi(\mathbf{z}_i) \qquad \text{with } \mathbf{z}_i = \mathbf{W}_i \mathbf{y}_{i-1} + \mathbf{b}_i \tag{2.3}$$

Thus, given a fixed random matrix $\mathbf{B}_i$ and the loss at the last layer $\delta \mathbf{z}_N$, the equation 2.4 shows the synaptic weights update rule with feedback alignment (FA)[41]:

$$\delta \mathbf{W}_i = -((\mathbf{B}_i \delta \mathbf{z}_N) \odot \phi'(\mathbf{z}_i))\mathbf{y}_{i-1}^T, \qquad \text{with } \delta \mathbf{z}_N = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_N} \tag{2.4}$$

Feedback alignment provides an alternative to backpropagation, allowing for more biologically plausible learning in neural networks.

For instance, the BioTorch[41] Python framework incorporates feedback alignment into classical learning algorithms.

3

MACHINE LEARNING

The following chapter presents the theoretical background upon which this research is based.

3.1 TRADITIONAL MACHINE LEARNING

In this work, traditional machine learning refers to the classical methods and algorithms that have been developed and widely used before the advent of deep learning. These methods focus on extracting features from the data and applying statistical techniques to make predictions. Traditional machine learning algorithms include linear regression, logistic regression, decision trees, support vector machines (SVM), k-nearest neighbors (KNN), and random forests, among others.

In traditional machine learning, domain knowledge is used to manually select or design relevant features that capture the underlying patterns in the data. These features are then fed into learning algorithms, which aim to find the best mathematical models to fit the data.

Traditional machine learning algorithms can handle a variety of tasks, including classification, regression, clustering, and dimensionality reduction. These algorithms have been successfully applied in various domains.

In this section, the focus will be to explain two specific types of traditional machine learning algorithms: Support Vector Regression (SVR) and Lasso regression. These algorithms are widely used for regression tasks, where the goal is to predict a continuous numerical value based on input features. While there are various other traditional machine learning algorithms available for regression, we will focus on these two methods as they offer distinct advantages and have been extensively used in this work.

3.1.1 *Suppport Vector Machines*

Support Vector Regression (SVR) is a machine learning algorithm that is particularly useful for solving regression problems. It is an extension of Support Vector Machines (SVM), which are used for classification tasks, so let us introduce SVM first.

3.1.1.1 *SVM*

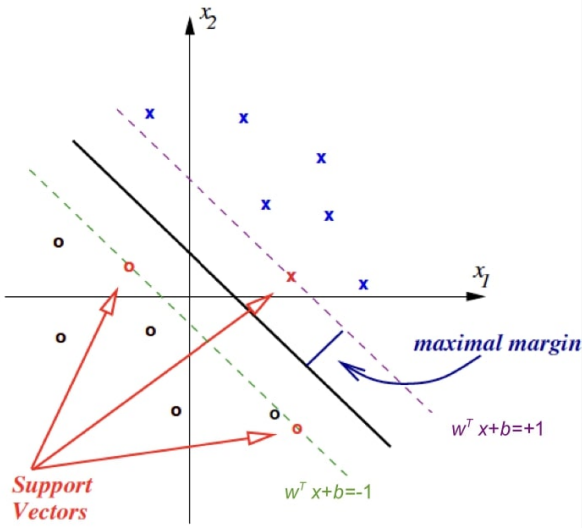


Figure 3.1: Separation between two classes of points with SVM. The dashed lines represent the margins, the C parameter influence their spacing. Points located directly on the margins define the support vectors. Source: Support vector machines and kernel-based methods[46].

Support Vector Machines (SVM)[47] is a popular machine learning algorithm used for classification tasks. SVM works by finding an optimal hyperplane that separates data based on values of the target variable. The main idea behind SVM is to transform the input data into a higher-dimensional feature space using a kernel function. In this transformed space, SVM aims to find a hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points of each class. SVM can handle linearly separable data, as well as non-linear relationships by using different kernel functions, such as linear, polynomial, or radial basis functions. During training, SVM finds the optimal hyperplane to separate the data by solving an optimization problem that minimizes classification errors while maximizing the margin.

Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ of features $\mathbf{x}_i$ and target values y_i, the goal is to find the hyperplane that best separates the points of the dataset $\mathbf{x}_i$ based on the classes represented in y_i (for binary classification, the positive and negative classes), with maximum sized margins. The hyperplane is characterized by the following equation, where $\mathbf{w}$ is the vector normal to the hyperplane:

$$\mathbf{w}^T\mathbf{x} - b = 0$$

Thus, the optimization problem behind SVM is that the distance between the nearest points $\mathbf{x}_i$ of either classes to the hyperplane must be maximized. This is equivalent to maximizing the margin, the distance between two hyperplanes parallel to the original hyperplane. For a binary classification problem, these hyperplanes are described by the equations:

$$\mathbf{w}^T\mathbf{x} - b = 1$$
$$\mathbf{w}^T\mathbf{x} - b = -1$$

These hyperplanes must be designed such as each point of the dataset must lie out of the margin, following this constraint:

$$y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$$

Since the margin is equal to $\frac{2}{\|\mathbf{w}\|}$, the optimization problem is equivalent to:

$$\begin{aligned} min_{\mathbf{w},b} \quad & \|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1, \qquad \forall i \in \{1, \dots n\} \end{aligned}$$

However, in some cases, the dataset is not linearly separable. In that case, slack variables ξ_i are introduced to measure the discrepancy of points that are not correctly classified with the margin (Figure 3.2). The optimization problem becomes:

$$\begin{aligned} min_{\mathbf{w},b,\xi} \quad & \|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 - \xi_i, \qquad \forall i \in \{1, \dots n\} \end{aligned}$$

This optimization problem might not have any solutions. In that case, a kernel is applied in order to map the feature space to a new one, in which the transformed optimization problem has a solution.

Several parameters are used to control the behavior and performance of the model:

- C: The regularization parameter C controls the trade-off between achieving a low training error and a low margin. A smaller value of C allows for a wider margin but may lead to more training errors, while a larger value of C enforces a smaller margin but aims for fewer training errors.
- Kernel: Different types of kernels can be used to transform the input data into a higher-dimensional feature space. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid. The choice of kernel depends on the characteristics of the data and the complexity of the decision boundary required.
- γ: This parameter is specific to the RBF kernel and controls the smoothness of the decision boundary. A smaller γ value makes the decision boundary smoother and can lead to underfitting, while a larger gamma value makes the decision boundary more wiggly and can lead to overfitting.

Once trained, the SVM model can make predictions on new, unseen data by mapping it into the same feature space and predicting their class based on the position of the new data relative to the learnt hyperplane. SVM is effective in a variety of domains and is known for its ability to handle high-dimensional data and handle both linear and non-linear relationships.

3.1.1.2 *SVR*

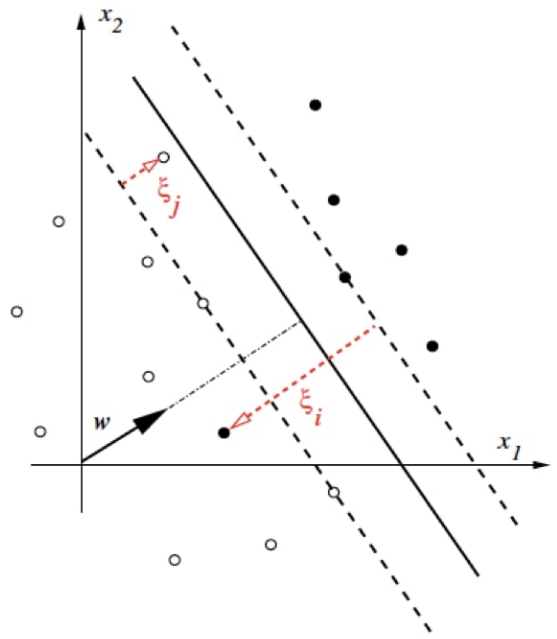


Figure 3.2: Support Vector Regression (SVR) with epsilon-insensitive loss. The plain curve represents the regression function learned by SVR, that fits training data within a specified margin (dashed lines). The slack variables ζ_i represent the deviations of individual data points from the regression function. Loss is computed based on the sum of these deviations. Source: Support vector machines and kernel-based methods[46].

Support Vector Machines (SVM) can be extended to Support Vector Regression (SVR) by adapting the formulation to handle continuous target variables instead of discrete classes. SVR aims to find a hyperplane that best fits the training data and predicts the continuous values of the target variable.

In SVR, the objective is to find a hyperplane that lies within a specified margin called the ϵ-tube around the training data points. The ϵ-tube determines the acceptable deviation or error for the predicted values from the true values. Data points within the ϵ-tube are considered correctly predicted, while those outside the ϵ-tube contribute to the regression loss.

SVR introduces one more parameter compared to SVM: The ϵ parameter defines the margin of tolerance around the predicted values within which no penalty is associated with the errors. It represents the tube around the regression line where errors are ignored.

By adapting the principles of SVM, SVR provides a powerful framework for performing regression tasks. It combines the concepts of loss functions, margins, and kernel functions to find an optimal hyperplane that captures the underlying patterns and predicts continuous target variables accurately.

3.1.2 *Lasso*

LINEAR REGRESSION Linear regression [48] is a popular statistical method used to model the relationship between a dependent variable and one or more variables. It assumes a linear relationship between the variables, where the dependent variable can be expressed as a linear combination of the independent variables.

Given a training set $\mathcal{TS} = (\mathbf{X}, \mathbf{y})$ of dimension $N \times K$ (where N denotes the size of the dataset, and K is the number of elements in the feature vectors $\mathbf{X}_i$ for $i \in \{0, \dots N-1\}$), linear regression approximates the targets $\mathbf{y}$ by the following:

$$\hat{\mathbf{y}} = \mathbf{X}^T\mathbf{w}$$

The optimization problem aims to find the optimal weights $\mathbf{w}$ to fit the training data optimally. This estimation is typically done using the method of least squares, which minimizes the sum of the squared differences between the predicted values $\hat{\mathbf{y}}$ and the actual values of the dependent variable $\mathbf{y}$:

$$\begin{aligned}\mathcal{L}(\mathcal{TS}, \mathbf{w}) &= (\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}}) \\ &= (\mathbf{y} - \mathbf{X}^T\mathbf{w})^T(\mathbf{y} - \mathbf{X}^T\mathbf{w})\end{aligned}$$

Lasso[48] (Least Absolute Shrinkage and Selection Operator), is a regularization technique commonly used in machine learning and statistical modeling. It is particularly useful for feature selection in linear regression models.

The Lasso regression adds a penalty term to the traditional linear regression objective function (see 3.1.2). This penalty term is the L1-norm of the coefficient vector, multiplied by a regularization parameter, usually denoted as α. Thus, given a training set $\mathcal{TS} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is the matrix of independent variables and $\mathbf{y}$ is the vector of observed values of the dependent variable, and a regularization term α that controls the strength of the penalty term, the objective function for Lasso regression is:

$$\mathcal{L}(\mathcal{TS}, \mathbf{w}) = (\mathbf{y} - \mathbf{X}^T\mathbf{w})^T(\mathbf{y} - \mathbf{X}^T\mathbf{w}) + \alpha\|\mathbf{w}\|$$

The optimization problem aims to find the optimal weights $\mathbf{w}$ to fit the training data optimally by minimizing this objective function.

The L1-norm encourages sparsity in the coefficient vector, effectively shrinking some coefficients towards zero. As a result, Lasso can automatically perform feature selection by setting the coefficients of irrelevant features to zero.

The amount of regularization is controlled by the α parameter. A higher value of α leads to stronger regularization and more coefficients being pushed towards zero. Conversely, a lower value of α allows more coefficients to have non-zero values.

By using Lasso regression, we can obtain a model that not only predicts the target variable, but also identifies the most important features for prediction.

3.2 CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNNs)[2, 49, 50] are a type of deep neural network that are particularly effective for computer vision tasks. They are composed of multiple layers that perform feature extraction and classification tasks in a hierarchical manner. The first layer typically extracts low-level features such as edges and corners, while next layers learn increasingly complex features based on the outputs of previous layers. CNNs are characterized by the use of different types of layers:

- Convolutional layers: these layers are based on the convolution operation. A convolution layer uses a **filter** (or **kernel**), which is applied at every position of the input. The resulting output is called the output feature map. Let the input be a 2D matrix with elements $x_{i,j}$, a is the activation function of the layer, β is the bias associated with the layer, and ω_{mn} are the elements of the kernel of dimension $M \times N$, then, the output feature map $h_{i,j}$ is computed as such:

$$h_{ij} = a\left[\beta + \sum_{m=1}^{3}\sum_{n=1}^{3} \omega_{mn} x_{i+m-2,j+n-2}\right]$$

  In this equation, the kernel has dimensions 3×3. The kernel is shifted horizontally and vertically until the whole input has been covered. An output h_{ij} is computed at each position. This operation can be visualized in Figure 3.3 (a) and (b).

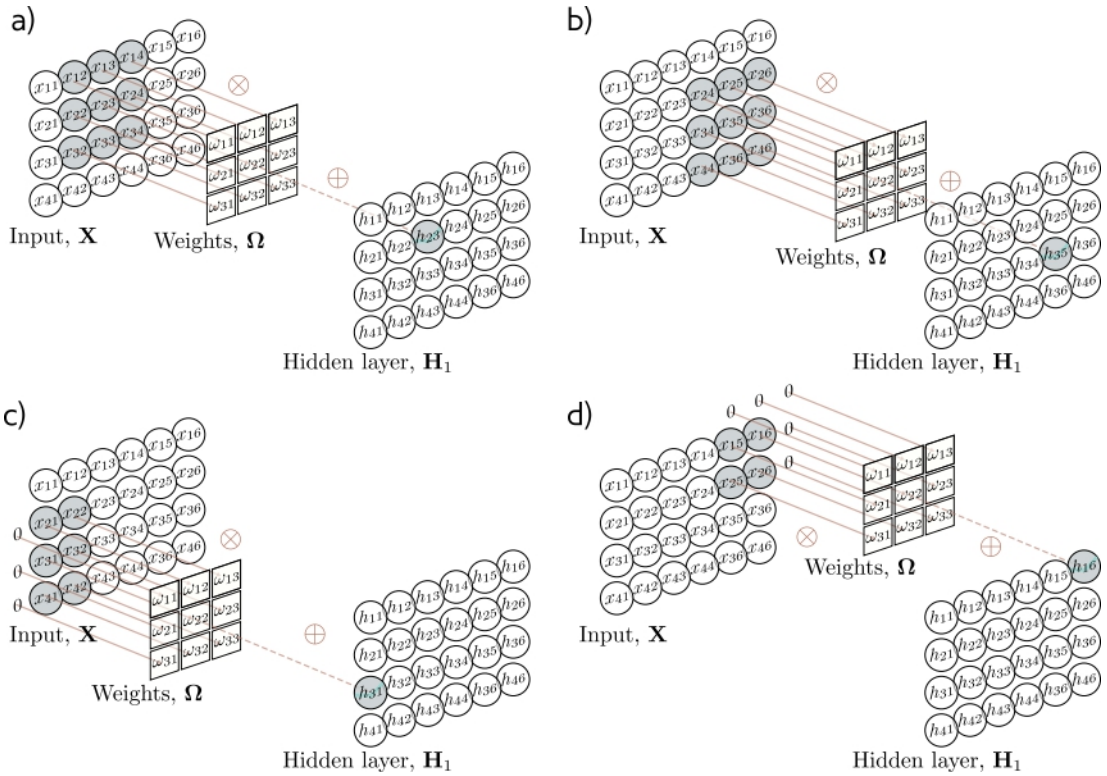


Figure 3.3: A convolutional layer of a convolutional neural network. (a)-(b) The kernel $\mathbf{\Omega}$ (of dimension 3×3) is moved along the axes of the input feature map $\mathbf{X}$, and an operation of convolution between the kernel and the selected area is performed to produce the output $\mathbf{H_1}$. (c)-(d) Using, zero-padding, pixels that do not fall within the image are considered to be zero. Without zero-padding, positions where the the kernel does not fit entirely in the input are skipped. Source: Understanding Deep Learning[49].

The convolution operation is characterized by a set of parameters:

FILTER SIZE The filter size refers to the dimensions of the convolutional kernel that moves across the input data during the convolutional operation. This filter size determines the size of the receptive field (the region of the input data that the filter analyzes at any given point). A larger filter size results in a larger receptive field, which allows the network to capture larger spatial features in the input data. However, this comes at the cost of an increased number of weights.

STRIDE The stride determines the step size of the filter as it moves across the input data. The stride value defines the number of pixels the filter will move at a time, in both the horizontal and vertical directions. Using larger stride values can reduce the size of the output feature maps (see Figure 3.4(a)), and also reduce the amount of computation required during training and inference. However, larger strides may also result in loss of information.

DILATION Dilation is used to increase the receptive field of a neuron, without increasing the number of parameters in the network, interleaving kernel weights with zero-values. With dilation, the filter is dilated by inserting zeroes between its values, so that it spans a larger area of the input. This has the effect of increasing the receptive field of the neurons that use that filter, allowing them to capture larger patterns in the input. Unlike increasing the kernel size, dilation does not add more parameters to the model, making it a more efficient way to increase the network's capacity. The number of zeros that is added to the kernel is called the dilation rate.

ZERO-PADDING Zero-padding is used to preserve the spatial size of the input volume when applying filters or convolutions. The process involves adding layers of zeros to the input data around the edges of the feature map, so that the spatial size remains the same even after applying filters with a certain size (Figure 3.3 (c) and (d)).

- Pooling layers:

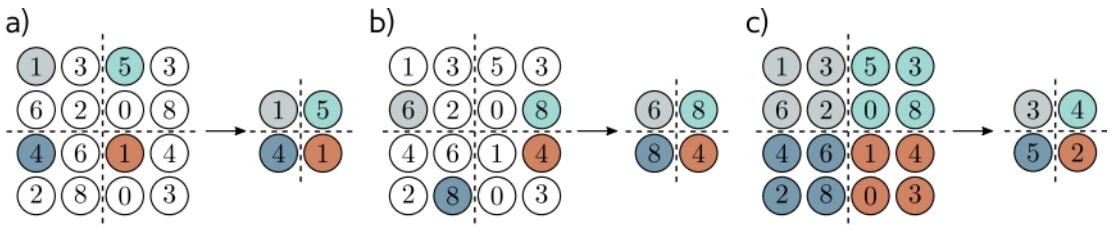


Figure 3.4: Down-sampling of the representation size. (a) A stride > 1 convolution operation down-samples the input. (b) Max-pooling. Each element of the output produced by a max-pooling layer is the maximum value of $K \times K$ block (here $K = 2$). (c) Average-pooling. Each element of the output produced by an average-pooling layer is computed as the average of values contained in a $K \times K$ block (here $K = 2$). Source: Understanding Deep Learning[49].

Pooling layers in CNNs are used to down-sample the feature maps obtained from the convolutional layers. This is done to reduce the spatial size of the input and extract the most important features, while also reducing the number of parameters in the model. The most commonly used pooling operation are:

  - Max pooling, where the maximum value in a local region of the feature map is selected and the rest are discarded.
  - Average pooling, where the average value in the local region is computed instead of the maximum.

  Pooling layers can also help to prevent overfitting by introducing some degree of translation invariance in the model. An example of these operations can be observed in Figure 3.4 (b) and (c).

- Fully connected layers: Fully connected layers in CNNs are layers where every neuron in one layer is connected to every neuron in the next layer (Figure 3.5). These layers are typically placed at the end of the network, after several convolutional and pooling layers. The purpose of fully connected layers is to use the high-level features learned by earlier layers and use them to make a final classification decision. These layers are often followed by a softmax activation function to produce a probability distribution over the possible classes. However, fully connected layers are computationally expensive.

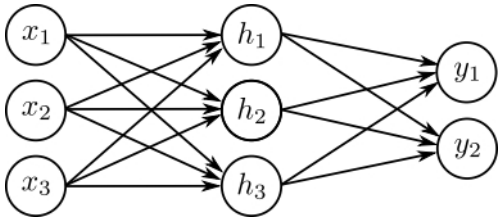


Figure 3.5: This network is composed of an input layer, a hidden layer and an output layer. The hidden layer is said to be fully-connected, because every neuron in this layer is connected to every neuron in the next layer. Source: Understanding Deep Learning[49].

CNNs have achieved state-of-the-art performance in a wide range of computer vision tasks, including image classification, object detection, segmentation, and tracking. They have also been successfully applied to other domains such as natural language processing and audio analysis.

3.2.1 *AlexNet*

AlexNet is a deep convolutional neural network that was proposed in 2012 by Krizhevsky *et al.* [2]. It was one of the first deep learning models to demonstrate high accuracy in the ImageNet Large Scale Visual Recognition Challenge, which is a benchmark dataset for image classification tasks. AlexNet consists of eight layers with ReLu activation function, including five convolutional layers and three fully connected layers. The convolutional layers use filters of size 11×11, 5×5, and 3×3 to extract features from the input image. The fully connected layers then perform classification based on the learnt features. The output of the last layer is transformed using a softmax function, and thus outputs a probability distribution for the input

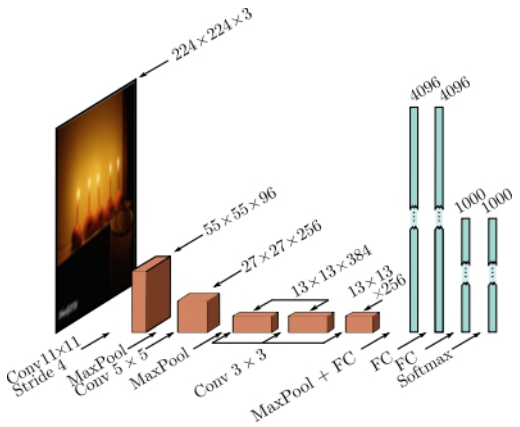


Figure 3.6: Architecture of AlexNet. This image illustrates the layout and connections of the layers in AlexNet. The network consists of five convolutional layers followed by three fully connected layers, and the output layer. Each convolutional layer performs spatial feature extraction through filters and applies ReLU activation functions. Max pooling layers are inserted between the convolutional layers. Source: Understanding Deep Learning[49]

to belong to one of the 1000 classes represented in ImageNet (see Figure 3.6).

AlexNet introduced several new techniques, such as the use of Rectified Linear Units (ReLU) for activation functions and dropout regularization to prevent overfitting. Its success played a significant role in popularizing the use of deep learning for computer vision tasks.

3.2.2 *Resnet*

ResNet (Residual Network) is a familiy of deep convolutional neural network architecture developed in 2015 by He *et al.* [51]. It is based on the concept of residual blocks. In ResNets, each residual block contains (see Figure 3.7):

- A batch normalization operation
- A ReLU activation function
- A convolutional layer

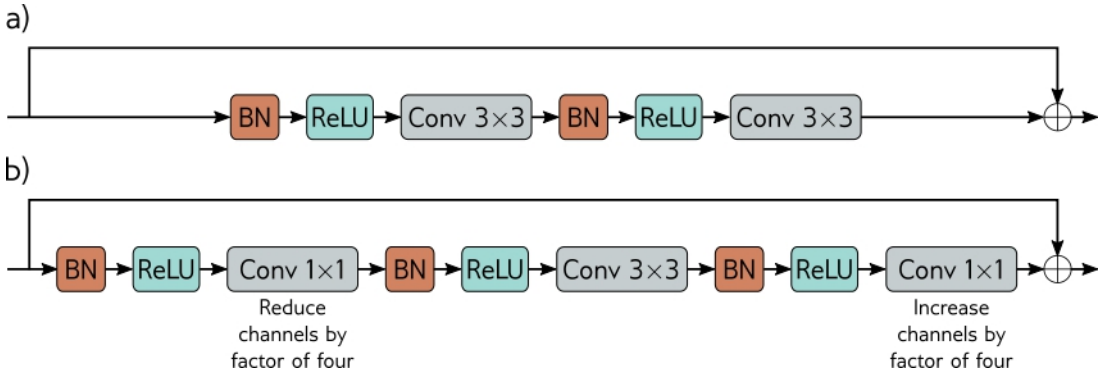


Figure 3.7: Residual block of ResNets. BN stands for Batch Normalization. (a) BN is followed by a ReLU activation function, followed by a 3×3 convolutional layer. The sequence is repeated once before being added back to the input. (b) For very deep neural networks, bottleneck residual blocks are a variation of (a) where the sequence is repeated three times, where the first and second use a 1×1 kernel for the convolution, and a 3×3 kernel for the convolution in the middle. This allows to reduce the number of channels, perform convolution, and re-increase the number of channels, while using less parameters. Source: Understanding Deep Learning[49].

Residual connections between layers, allow the network to skip over certain layers and learn a residual function that is easier to optimize.

RESNET-200 ResNet-200 is a variant of the Residual Neural Network (ResNet) architecture, which is known for its exceptional performance in deep learning tasks, particularly image recognition. ResNet-200 extends the original ResNet by significantly increasing its depth to 200 layers (Figure 3.8). This increased depth allows ResNet-200 to capture more intricate and abstract features from images, enabling it to achieve even higher accuracy in challenging visual recognition tasks. By using residual connections, ResNet-200 mitigates the vanishing gradient problem and enhances gradient flow during training. With its extensive depth and skip connections, ResNet-200 demonstrates the potential of deep neural networks for achieving state-of-the-art results in image classification and other computer vision tasks.

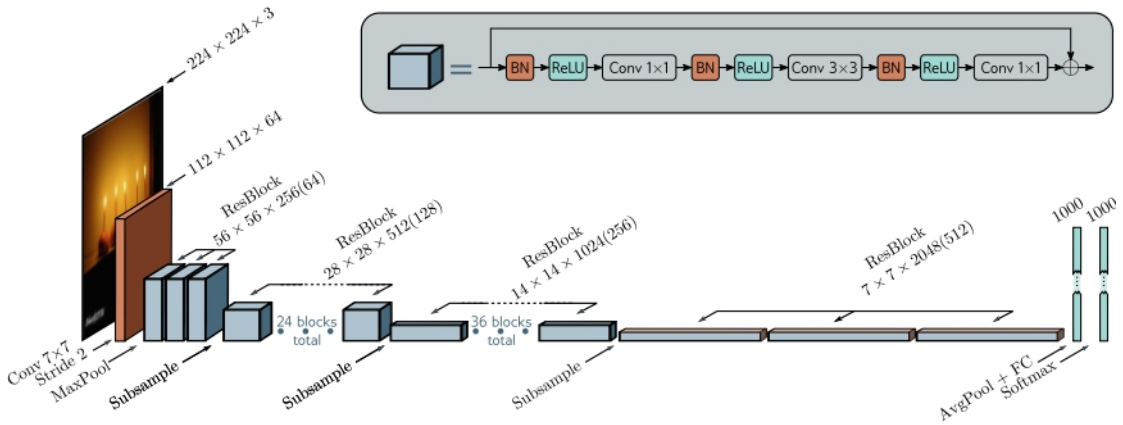


Figure 3.8: ResNet-200 Architecture. The ResNet-200 architecture consists of 200 layers and uses residual connections. Each building block contains multiple convolutional layers with skip connections, allowing for the learning of residual mappings. Downsampling is achieved through a combination of max pooling and convolutional layers with stride 2. The network is organized into stages, gradually increasing the number of building blocks to capture complex features. Fully connected layers are employed in the final stage for classification. Source: Understanding Deep Learning[49]

RESNET-18 ResNet-18 is a smaller version of the original ResNet architecture, with only 18 layers, making it faster to train and less prone to overfitting, while still achieving state-of-the-art performance on a variety of image classification tasks. The details of the architecture are precised in Figure 3.9, but the overall architecture is similar to the one observed in Figure 3.8.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.9: Different architectures of ResNets. Building blocks depicted in Figure 3.7 are shown in brackets, with the numbers of blocks stacked. Source: Deep Residual Learning for Image Recognition[51].

3.2.3 *EfficientNet*

EfficientNet[52] is a family of convolutional neural networks that were designed to achieve state-of-the-art accuracy with a relatively small number of parameters. These networks were developed by scaling up the model width, depth, and resolution in a principled manner, while also incorporating a new compound scaling method that uniformly scales all three dimensions.

This approach enables EfficientNet models to achieve high accuracy with significantly fewer parameters than other state-of-the-art models, making them attractive for applications where computational resources are limited. In addition, EfficientNet has demonstrated strong performance across a wide range of image recognition tasks, including object detection and semantic segmentation.

COMPOUND SCALING The compound scaling factor is the approach used to scale the network's depth, width, and resolution simultaneously. It is a single parameter that controls the overall size of the model.

The scaling factor is denoted as ϕ, where $\phi > 1$ indicates scaling up the model and $\phi < 1$ indicates scaling down the model.

By varying the value of ϕ, different variants of EfficientNet can be obtained, such as EfficientNet B0, B1, B2, and so on. As the value of ϕ increases, the network becomes deeper, wider, and has a higher input resolution. This scaling strategy allows for a trade-off between model size and accuracy.

The compound scaling factor is designed to ensure that the model scales up efficiently while maintaining a good balance between model capacity and computational resources. It enables EfficientNet to achieve state-of-the-art performance on various computer vision tasks across different scales and resource constraints.

MBCONV BLOCKS MBConv (Mobile Inverted Residual Bottleneck Convolution)[53] blocks are a fundamental component of EfficientNet models. They are designed to efficiently capture complex patterns and increase model capacity while minimizing computational costs.

An MBConv block consists of several steps (Figure 3.10):

- Expansion and Squeeze: The MBConv block expands the input channels before applying the depthwise separable convolution. This expansion is achieved by using a 1x1 convolutional layer with a nonlinear activation function.
- Depthwise Separable Convolution: It starts with a depthwise convolution, which applies a separate convolutional filter to each input channel, followed by a pointwise convolution that combines the output channels of the depthwise convolution. This approach reduces the number of parameters and computational complexity compared to traditional convolutions.
- Bottleneck Structure: The bottleneck structure reduces the number of input channels before the depth-wise convolution, and then expands the channels back after the depth-wise convolution. This bottleneck structure further reduces the computational cost while maintaining representational power.

- Skip Connection: A skip connection is added that bypasses the MBConv block. This allows gradients to flow directly from the output to the input, facilitating the training process and enabling the network to learn more effectively.

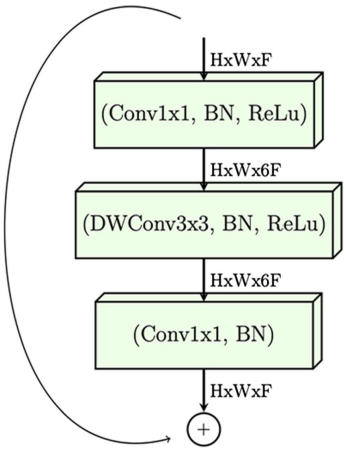


Figure 3.10: The Mobile Inverted Residual Bottleneck Convolution (MBConv) block, a key component of EfficientNet models, employs depth-wise separable convolutions, expansion and squeeze operations, bottleneck structures, and skip connections

These blocks enable the network to achieve high accuracy with fewer parameters and computational resources, making EfficientNet models well-suited for various applications with limited computational constraints.

EFFICIENTNET-B0 EfficientNet-B0 is the baseline model of the EfficientNet family of convolutional neural networks. It is designed to achieve a good balance between model size and accuracy. The "B0" in the name refers to the compound scaling factor used to control the network's depth, width, and resolution.

EfficientNet-B0 consists of a stack of convolutional layers. It is a simple stack of MBConv blocks.

The baseline EfficientNet-B0 has relatively fewer parameters compared to larger variants in the family, making it computationally efficient and suitable for resource-constrained environments. Despite its compact size, EfficientNet-B0 has demonstrated competitive performance on various computer vision tasks.

EFFICIENTNET-B4 EfficientNet-B4 is an advanced variant of the EfficientNet architecture that offers higher performance and larger model size compared to EfficientNet-B0. Similar to other models in the EfficientNet family, EfficientNet-B4 follows the principle of compound scaling.

In terms of architecture, EfficientNet-B4 follows the same sequence of operations and blocks as EfficientNet-B0, in a scaled-up version.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 3.11: EfficientNet-B0. EfficientNet-B0 starts with a convolutional layer ($i = 1$) followed by a series of MBConv blocks ($i \in [2, 8]$). The blocks are stacked in a hierarchical manner, with downsampling. The final part of the network consists of a global average pooling layer that aggregates spatial information and a fully connected layer for the final prediction task ($i = 9$). Source: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks[52].

EfficientNet-B4 benefits from its larger size and increased model capacity, allowing it to handle more complex tasks and capture finer-grained features in the input data.

EFFICIENTNET-B7 EfficientNet B7 is the top-performing variant of the EfficientNet architecture, offering the highest accuracy and model size among the EfficientNet models.

EfficientNet-B7 follows a similar architecture as other models in the EfficientNet family but with increased depth, width, and resolution. It consists of a stack of MBConv blocks.
By scaling the expansion factor, EfficientNet-B7 achieves a higher level of performance while maintaining model efficiency.

EfficientNet-B7 represents the state-of-the-art in the EfficientNet series, providing exceptional accuracy and performance for various computer vision applications. Its larger size makes it well-suited for tasks that demand higher-level feature extraction and intricate pattern recognition.

3.3 CHANGE DETECTION NETWORKS

Change detection networks[54–57] are a type of neural network architectures designed to identify and localize changes between two or more input images or data instances. These networks are particularly useful in applications such as video surveillance, and anomaly detection. The main goal of change detection networks is to learn representations that capture the characteristics of both the unchanged and changed regions in the input data. By comparing the representations of the input data before and after the change, these networks can effectively detect and highlight the areas that present alterations. Change detection networks usually use convolutional neural network (CNN) layers to extract features from the input data and utilize various techniques such as siamese architectures[56] or differencing operations to emphasize the changes. The output of these networks is a binary mask or heat map indicating the presence and location of the detected changes. These networks have demonstrated promising performance in

detecting and tracking changes in various domains.

In the context of this work, change detection networks are highly relevant as they could play a role in identifying and localizing changes in the input videos. In the following, we will discuss and explain one architecture of such networks[54], that will be used as a new way to evaluate the quantity of change in next sections.

3.3.1 *Dynamic Receptive Temporal Attention Network*

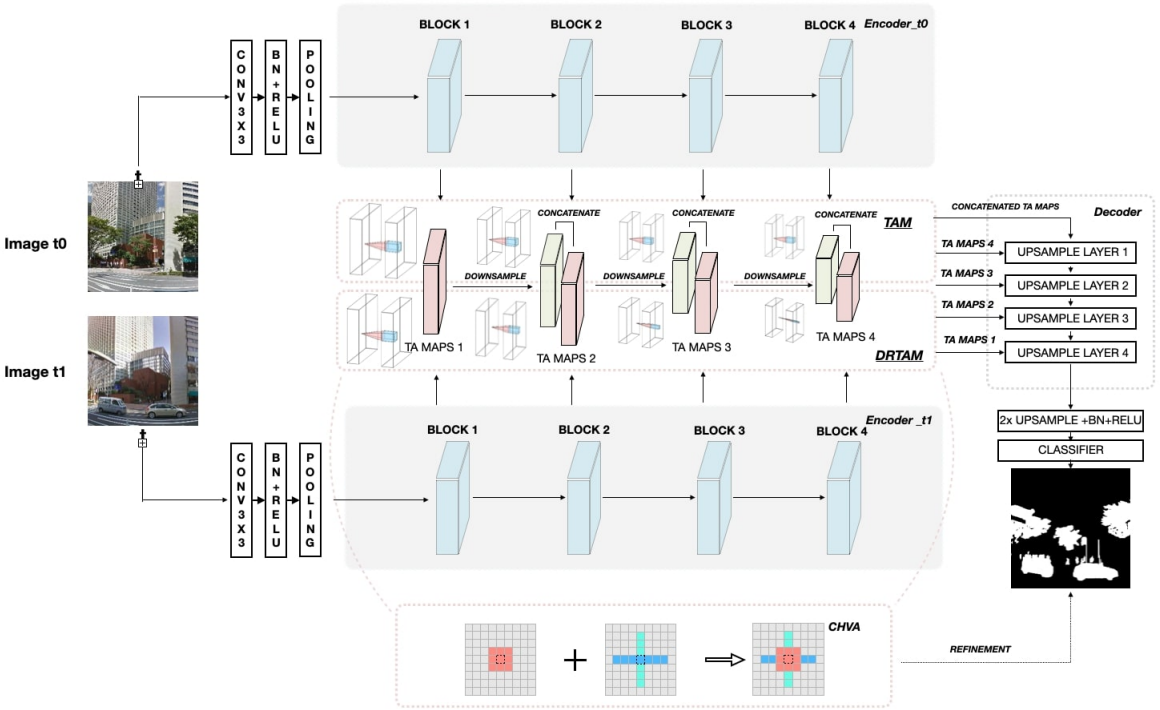


Figure 3.12: TANet architecture. The architecture is based on three components: an encoder, a decoder and an attention module (TAM). Source: DR-TANet: Dynamic Receptive Temporal Attention Network for Street Scene Change Detection[54]

Dynamic Receptive Temporal Attention Network (DR-TANet)[54] introduces attention mechanisms into change detection tasks. This network architecture was originally designed for street change detection. The goal was to identify regions where semantic changes occurred in street-view images captured at different times.

The proposed architecture introduces attention in change detection networks, using Dynamic Receptive Temporal Attention Module (DRTAM) as an improved temporal attention module, and Concurrent Horizontal and Vertical Attention (CHVA).

DR-TANet is directly based on a simpler achitecture: TANet. This network is based on an encoder-decoder architecture, where the encoder is based on ResNet-18. The encoder path is

split in two ways, one for each image, and each of them will perform feature extraction on the two input images. These features are then passed as input to the Temporal Attention Module (TAM), which uses attention mechanisms to find similarities in a fixed dependency scope. The obtained attention maps are then fed to the decoder component, which will use up-sampling to retrieve the final change mask. See Figure 3.12.

DR-TANet is based upon the described architecture, but uses Dynamic Receptive Temporal Attention Module (DRTAM) instead of TAM to compute attention maps, and also uses Concurrent Horizontal and Vertical Attention (CHVA) to perform refinements.

3.3.1.1 *Temporal Attention Module (TAM)*

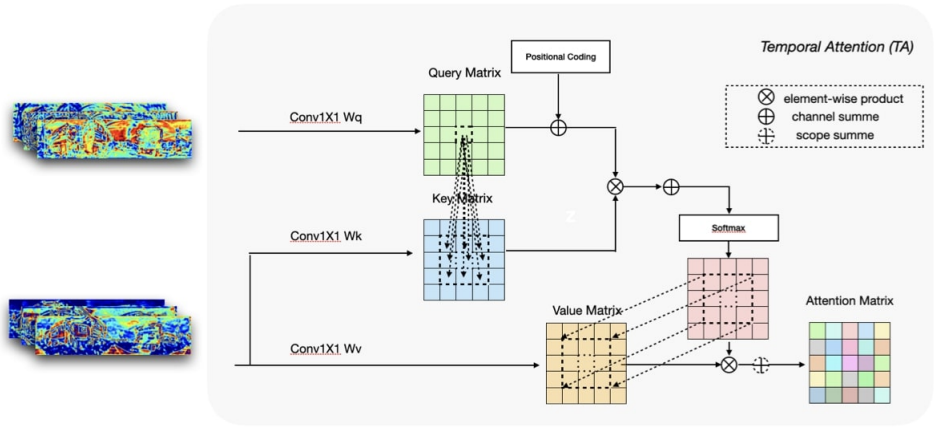


Figure 3.13: Temporal Attention (TA). Source: DR-TANet: Dynamic Receptive Temporal Attention Network for Street Scene Change Detection[54]

Temporal Attention (TA) (Figure 3.13) is inspired by the self-attention mechanism in transformers and its application in vision models. The TA module enhances the model's ability to capture temporal dependencies and identify changes between consecutive frames.

During the calculation procedure of TA, the feature maps of the previous frame (t_0) are used to generate a query matrix. This query matrix is then operated with a key matrix to compute the covariance between each pixel in t_0 and the corresponding pixels in the dependency scope from the current frame (t_1). The resulting covariance matrix, after a softmax operation, provides a measure of similarity between pixels. This similarity matrix is then combined with the value matrix, derived from the feature maps of t_1, to predict the regions where changes have occurred.

To optimize computational efficiency and representativeness, a fixed dependency-scope size is applied. This means that each pixel in t_0 is considered to depend only on a specific scope of pixels in t_1, centered around the same position. The size of the fixed dependency scope can be adjusted.

Furthermore, multi-head temporal attention is employed to divide the feature maps into multiple groups in the channel dimension to capture relations between channels.

Similar to the positional encoding module in transformer models, Positional Coding (PC) is added to the TA module. This PC technique accounts for the relative positional information between pixels. By considering both the dependency and similarity of two temporal channels and the proximity of local positions, the TA module effectively models the temporal dependencies and spatial proximity in the scene change detection task.

The Temporal Attention Module (TAM) is built upon the Temporal Attention (TA) mechanism. TAM consists of four layers, where each layer takes feature maps from two temporal channels as input and generates an attention map using the TA mechanism. As the feature maps have different sizes, the previously computed attention maps are downsampled and concatenated with the subsequent attention maps. To preserve information during the upsampling process, each attention map is also passed through skip connections and inserted into the decoder.

In summary, TAM operates on multiple layers to capture temporal dependencies and generate attention maps. The downsampling and concatenation steps ensure the integration of attention maps across different scales, while the skip connections preserve important information during the upsampling process. This multi-layer architecture enhances the ability of the model to capture and represent complex temporal relationships in the scene change detection task.

3.3.1.2 *Dynamic Receptive Temporal Attention Module (DRTAM)*

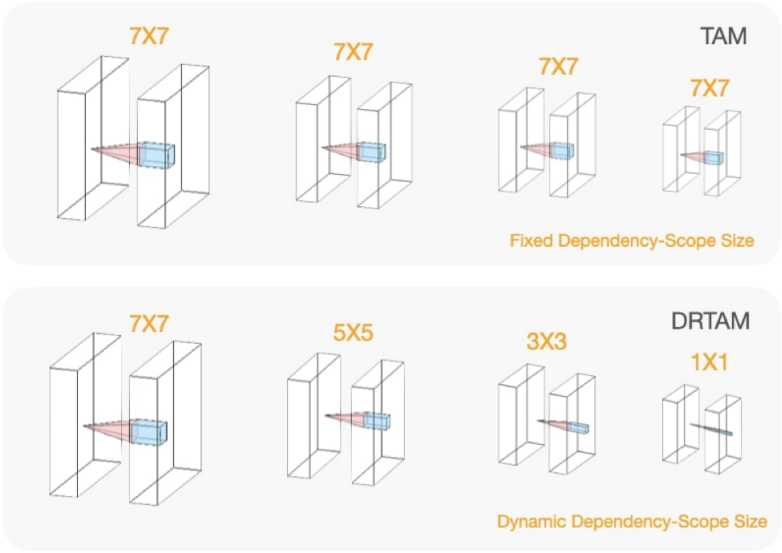


Figure 3.14: TAM and DRTAM. This figure illustrates the fixed-dependency-scope size of TAM, and the dynampic dependency-scope size of DRTAM. Source: DR-TANet: Dynamic Receptive Temporal Attention Network for Street Scene Change Detection[54]

The Temporal Attention Module (TAM) in the previously described approach initially employs a fixed dependency-scope size throughout the module. However, in vision models, during the

down-sampling path of the encoder, the feature maps progressively capture low-level entities such as color and edges, followed by a focus on high-level entities such as texture, shapes, and objects.

Therefore, the Dynamic Receptive Temporal Attention Module (DRTAM) aims to better utilize the computational capacity. In DRTAM, the dependency-scope size varies across different temporal attention layers. Specifically, in DR-TANet, the initial layer uses a (7×7) dependency-scope size to capture low-level features, while subsequent layers employ (5×5), (3×3), and (1×1) sizes. This dynamic adjustment allows for effective utilization of computational resources while ensuring the collection of sufficient neighborhood similarity and dependency, which are crucial for accurate change detection.

3.3.1.3 *Concurrent Horizontal and Vertical Attention (CHVA)*

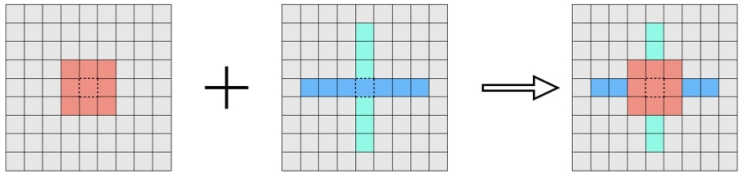


Figure 3.15: Concurrent Horizontal and Vertical Attention (CHVA). Source: DR-TANet: Dynamic Receptive Temporal Attention Network for Street Scene Change Detection[54]

Typically, convolutional neural networks (CNNs) define a square neighborhood around the central pixel as the associated region for convolution. In DR-TANet, the authors adopt a similar configuration by considering the shape of the dependency scope in the Temporal Attention (TA) module. However, the task of change detection has its own specific requirements. Similar to dense semantic segmentation, it involves pixel-wise detection, where certain objects like tree branches or road lamps may extend horizontally or vertically. They introduce the Concurrent Horizontal and Vertical Attention (CHVA) mechanism to gather more information along both directions for refining the detection of strip-like changes. The length of the horizontal and vertical attention windows is adjusted to $(2K + 1)$, where K represents the size of the temporal attention dependency scope. The CHVA module is illustrated in Figure 3.15.

4

METHODOLOGY

This chapter presents the detailed process and techniques used to achieve the research objectives. The chapter is divided into two main sections: the first section discusses the pipeline adopted from the work of Roseboom *et al.* [1], which forms the foundation for the research. This section gets in details into the accumulation mechanism, the dataset used, how the training is performed on accumulators, and how predictions of values are made. The second section is dedicated to the variations implemented in this work, including other accumulation mechanisms such as the calibration of parameters, the use of other CNNs as feature detector networks, and the DR-TANet as a base for another change detection method. Additionally, the section covers the dataset, including participants, data collection method, and dataset generation method. Finally, this section presents other regression schemes implemented in this work, such as the LassoCV model for feature selection and the GridSearchCV model for single and bi-layer results.

4.1 METHODS FROM ROSEBOOM *et al.* [1]

Roseboom *et al.* [1] present a paper that investigates the hypothesis that accumulators capturing salient changes in videos by computing the difference in successive activations across layers of a classification network, can be used to predict human-like estimates of video duration in seconds.

4.1.1 *Dataset*

The dataset used in this master thesis differs from the one utilized in the original paper. In this section, we will outline the composition of the dataset in the original paper, while the dataset used for this thesis will be described in Section 4.2.1.

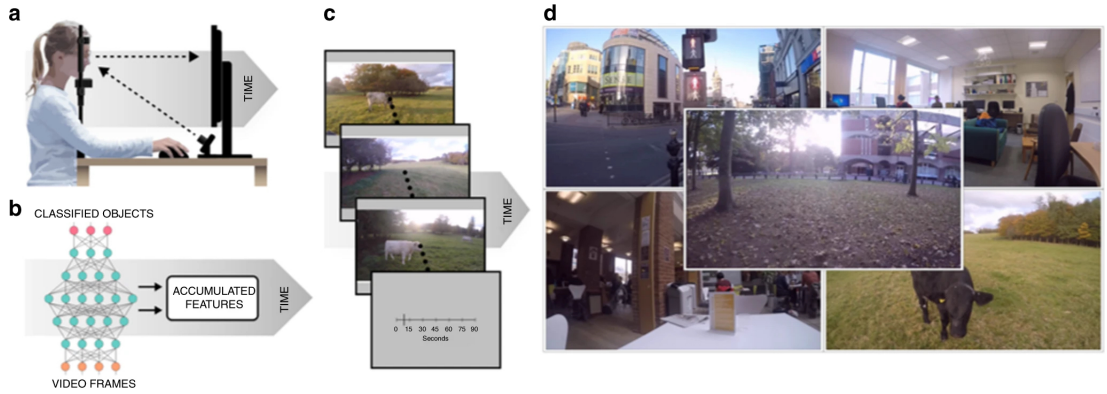


Figure 4.1: Illustration of the experimental setup and procedure. (a) Human participants viewed videos of natural scenes and indicated the perceived duration while their gaze direction was tracked according to the represented setup. (b) Overview of the high-level architecture of the model employed for simulations. (c) Frames from a video used as a stimulus for human participants and input for simulated experiments. Human participants used a visual analogue scale to report the duration of the videos in seconds. (d) Screenshots from videos used as stimuli for the human experiment and as input for the model experiments. Source: Activity in perceptual classification networks as a basis for human subjective time perception[1].

4.1.1.1 *Video stimuli*

The stimuli videos presented to participants in the original paper consisted of various scenes of different durations. These scenes included walking around a city, in an office, in a cafe, in the countryside, and around a leafy campus, as depicted in Figure 4.1(d).

All videos were recorded in the City of Brighton, UK, using a GoPro camera.

4.1.1.2 *Human experiment*

To create the dataset used by Roseboom *et al.* [1], a human experiment was conducted following the process outlined below. 55 adult participants were recruited from the University of Sussex, and were remunerated for their participation. The participants viewed a selection of videos depicting natural scenes while their gaze direction was tracked. They were then asked to report the perceived duration of each video in seconds. The setup of this process is illustrated in Figure 4.1(a).

Some videos were presented to a set of participants. The videos used as stimuli included different kind of scenes, and were of different durations. For instance, they included walking around a city, in an office, in a cafe, in the countryside, and around a leafy campus scenes, as depicted in Figure 4.1(d). The videos were displayed on a screen positioned in front of the participants, with their head fixed at a distance of 57cm from the screen using a rest (Figure 4.1).

Each participant completed approximately 80 trials, organized into blocks of 20 trials with short breaks in between, during a 1-hour session. Some participants only completed 20 trials

due to certain constraints.

Participants initiated the presentation of a stimulus by pressing the mouse buttons. At the end of each stimulus, a visual scale displaying durations between 0 and 90 seconds was shown, and participants were instructed to report their perceived duration of the corresponding video.

Importantly, participants were explicitly instructed not to attempt any form of counting to determine the length of the videos.

4.1.2 *Pipeline*

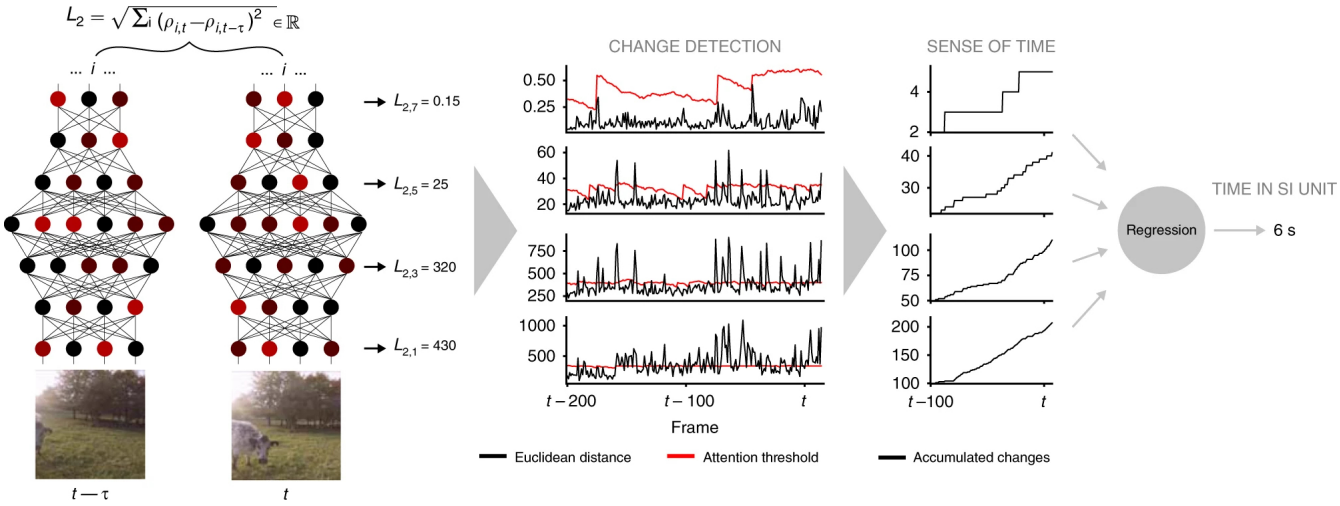


Figure 4.2: Pipeline used in the paper of Roseboom *et al.* [1] There are four main components. In the first step, activations of different layers of AlexNet are used between consecutive frames and the distance between them is computed for each layer. In the next step, these distances are compared to a varying threshold. Depending on the result of the comparison, accumulators for each layer are increased, or not. In the final stage of the pipeline, the accumulators are fed to a regression model that will predict human-like duration estimates for the video. Source: Activity in perceptual classification networks as a basis for human subjective time perception[1].

The pipeline proposed by Roseboom *et al.* [1] consists in four parts:

1. Image classification deep neural network
2. Threshold mechanism
3. Set of accumulators
4. Regression scheme

In the following, the different parts will be discussed in more detail.

4.1.2.1 *Image classification deep neural network*

The accumulation mechanism used by Roseboom *et al.* [1] is based on the use of convolutional neural networks. The network used is a classification network for computer vision, AlexNet.

By the time, AlexNet was considered a reference and state-of-the-art in the computer vision domain, and it was thus a very well-motivated choice to use it. The authors use a pre-trained version of AlexNet on ImageNet[2], a dataset of high-resolution images, on its version of LSVRC-2010, which aims at classifying images into 1000 classes. The weights of the pre-trained network were imported using Caffe[58], a deep learning framework that is less used nowadays. The input videos were fed to the model, frame-by-frame (with a frequency of 30Hz). The activations for each frame and for a set of layers were then retrieved, and used in the next steps.

4.1.2.2 *Threshold mechanism*

Based on the activations retrieved in the previous step, the Euclidean distance between the activations of two successive frames x_{t-1} and x_t was computed for each layer. This measure serves as the distance, or difference between two successive frames, and gives a first idea of the quantity of change observed between these two moments. Thus, when the frames are similar (meaning, that no significant change occurs), the distance is low, and when the frames are more different (meaning, that some more significant change occurred), the distance is high. Given $\phi^k(x_t)$ the activation in layer k of input frame x_t at time step t, the Euclidean distance between two successive frames is given by:

$$d_t^k = \left\| \phi(x_t)^k - \phi(x_{t-1})^k \right\|$$

A threshold is compared to the distance d_t^k obtained. This threshold is fixed a priori for each layer, and evolves during the experiment, following a decaying law, with stochasticity involved. Let T_t^k denote the threshold value of the kth layer at time step t. Given T_{max}^k and T_{min}^k, the maximum and minimum threshold values (respectively) of the kth layer at time step t, τ^k the decay constant for the kth layer at time t, D the number of time steps since the last time the threshold value was reset and α, a dividing constant that adjusts the variance of the noise $\mathcal{N}\left(0, \frac{T_{max}^k - T_{min}^k}{\alpha}\right)$ added to the threshold, the next threshold value T_{t+1}^k is computed as:

$$T_{t+1}^k = T_t^k - \left(\frac{T_{max}^k - T_{min}^k}{\tau^k}\right) e^{-\left(\frac{D}{\tau^k}\right)} + \mathcal{N}\left(0, \frac{T_{max}^k - T_{min}^k}{\alpha}\right)$$

This threshold is computed at each time step of the experiment for each layer considered.

At each time step, the distance between consecutive frames computed at the previous step is compared to this threshold. If it is bigger than the threshold, accumulators are modified, and if not, the threshold is reset to its maximum value.

The chosen values for the parameters that must be fixed prior to the experiment are reported in Table A.6. However, the authors do not provide any information regarding the process or criteria used to determine these values.

Table 4.1: Threshold mechanism prior parameters

Layer	T_{max}	T_{min}	τ	α
`conv2`	340	100	100	50
`pool5`	400	100	100	50
`fc7`	35	5	100	50
`output`	0.55	0.15	100	50

4.1.2.3 *Set of accumulators*

A set of accumulators is initialized at the beginning of each experiment, for each video stimulus. Initially, the accumulators are set to 0 for each layer. As previously mentioned, when the distance between the activations of two consecutive frames is bigger than the threshold, the accumulators are modified. Indeed, the value corresponding to the layer is incremented by one unit, indicating that a significant change has been detected in the corresponding layer.
The threshold value is also reset to its maximum value when a change is detected.

The final value of the accumulator recorded for each layer is a representation (in an arbitrary unit) of the quantity of change detected in this layer, in the input video.

4.1.2.4 *Regression scheme*

Since the estimates obtained in the previous steps are an arbitrary measure of change detection, they must be converted to something that is known to humans.
One of the fundamental beliefs that this paper aims to confirm, is the fact that the quantity of change can predict the estimated duration of the videos. Thus, the values obtained for each of the accumulators could be predictive of human-like duration predictions.

A simple regression method is thus used in order to convert these subjective time units into seconds. To do so, the authors use an epsilon support vector regression, implemented using Scikit-learn[59], and 10-fold cross-validation.
The exact parameters for this experiment are reported in appendix A in Table A.4.

4.2 METHODS IMPLEMENTED IN THIS WORK

As previously explained, the goal of this work is not the same as the one explored by Roseboom *et al.* [1]. In their work, they postulate that the quantity of change may be predictive of the duration of videos as perceived by humans.
In this work, we want to see if this postulate also applies to the recalling time of episodic memories. That is, we want to check if the quantity of change is predictive of the time necessary to replay a memory mentally, in humans.

Hence, the dataset employed in this study differs from the one utilized by Roseboom *et al.* [1]. Additionally, while some aspects of the methods remain unchanged, certain modifications have been introduced as part of this research. The following sections will dig into these variations and provide detailed technical explanations.

4.2.1 *Dataset*

As the objective of this master's thesis differs from the study conducted by Roseboom *et al.* [1], the dataset requirements also varied. For this research, the dataset utilized was obtained from [21] and customized to align with the specific demands of this study. The subsequent section outlines the detailed procedure employed to generate the dataset.

4.2.1.1 *Participants*

Participant involvement is essential for this study, as they will be tasked with recalling video stimuli presented to them. To ensure statistical power and detect significant within-subject differences, and considering the possibility of issues during the experiments and with the participants, a total of 40 participants were initially planned for recruitment.
However, three participants had to be excluded from the study. One participant was excluded due to a significant flu-like condition during the experiment, while two participants were excluded due to computer failures during testing.
Consequently, the final sample comprised 37 participants, with an average age of 22.37 years, including 18 women. All participants had normal or corrected vision and were native French speakers with a mean education level, assessed by the number of successful years of study from the beginning of elementary school, of 14.89. None of the participants were undergoing drug treatment that could potentially affect concentration, had recent hospitalization, or had a diagnosed neurological psychiatric disorder.

4.2.1.2 *Data collection method*

VIDEOS A set of stimuli was constructed, incorporating variations in duration and density of event boundaries (EBs, perceptual or cognitive shifts that occur when transitioning from one event or experience to another). The stimuli consist in four types of videos:

- Short videos (30sec) with a low density of EBs (S/L)
- Short videos (30sec) with a high density of EBs (S/H)
- Long videos (60sec) with a low density of EBs (L/L)
- Long videos (60sec) with a high density of EBs (L/H)

To create these videos, sequences were extracted from eight already established films commonly used in event segmentation studies[60–64].
Each film depicted an individual engaged in a daily life activity (See Figure 4.3). Following a procedure inspired by Bangert *et al.* [60], the video stimuli were edited in four steps:

Figure 4.3: Snapshots of the films used as the basis for creating the stimuli. These include a woman washing her car, a woman preparing breakfast , a man gardening, a man photocopying the pages of a book, a man sweeping a coin, a lady setting up a tent, a man changing the tire of his car, and a man preparing a living room for a birthday party. Each film was recorded using a digital camera, at a fixed head-height perspective without any edits. Source:[60–64]

1. Norm creation: Initially, a segmentation norm was created for each movie by utilizing segmentation data obtained from previous studies[61–63]. Participants in these studies were instructed to press a button whenever they perceived a meaningful event boundary while watching the movies. The movies were divided into one-second bins, and the number of button presses within each bin was calculated as an index of EBs[63].

2. Segments creation: For each movie, the number of EBs was computed for each possible 30-second and 60-second window. The segments of each duration were then ranked based on the number of button presses associated with them.

3. Segments selection: The selection process involved choosing four segments, one from each category (S/L, S/H, L/L, L/H), within each film (there were 8 raw films). This selection aimed to minimize the influence of different "raw films" on the variables of interest. Once a segment was chosen, the 20-second interval preceding and following it was removed to avoid overlap between stimuli. The segments were selected in the following order:

   a) The 30-second interval with the most button presses (30BP+)

   b) The 30-second interval with the least button presses (30BP-)

   c) The 60-second segment with the most button presses (60BP+)

   d) The 60-second interval with the least button presses (60BP-)

   This process resulted in a set of 32 stimuli (eight short with a high density of EBs (S/H), eight short with a low density of EBs (S/L), eight long with a high density of EBs (L/H), eight long with a low density of EBs (L/L)). Additionally, a second set of 32 stimuli was created following the same selection procedure, but with the order of selecting long and short segments reversed. Each segment was labeled as "SF" (Short First) or "LF" (Long First) to indicate its source.

4. Adjustments: To ensure that the density of EBs did not vary as a function of duration, a correction was made. The number of participants used to create the segmentation norms varied across the eight raw films, making a direct comparison of button presses per minute (BP/min) associated with segments from different films not feasible. To address this, the BP value of each segment was divided by the number of participants involved in constructing the segmentation norm for the respective film. This manipulation resulted in an index called BPP (Button Press / Participant). The density of EBs was estimated by computing a "BPP / Minute" index for each segment. Comparisons revealed that, on average, the BPP/min was higher for the 30-second segments compared to the 60-second segments. To rectify this, certain segments were manually replaced to achieve a similar average BPP/min between corresponding short and long segments. The impact of duration (30 vs. 60) and EBs density (High vs. Low) on BPP was assessed, revealing no significant effect of duration on BPP.

   To address the issue of potential time overlaps between SF and LF segments from the same films, two sets of stimuli were created. Participants were divided into two groups, where one group was tested with Set A and the other with Set B (Table 4.2).

Table 4.2: Composition of sets A and B.

Set A		**Set B**	
SF	**LF**	**SF**	**LF**
Breakfast	Photocopier	Photocopier	Breakfast
Broom	Party	Party	Broom
Tent	Gardening	Gardening	Tent
Tire	Washing car	Washing car	Tire

EXPERIMENTS WITH PARTICIPANTS Participants were engaged in a task where they watched the videos, depicting everyday activities, while their pupil diameters and eye movements were recorded (however, this aspect was not taken into consideration in this thesis). After each video, participants were instructed to mentally replay the content and verbally describe their mental replay. The duration of the mental replays and the density of recalled experience units were measured.

Each participant was presented with a set of 32 videos (either set A or set B). They were asked to mentally replay each video in a detailed manner and then verbally describe what they recalled. To ensure proper understanding of the task instructions, a training phase was conducted before the main task, involving similar trials with different videos.

The task was implemented using Tobii Pro Lab software, and the stimuli were displayed on a 27" LCD screen positioned 65 cm away from the participants. To ensure accurate eye tracking

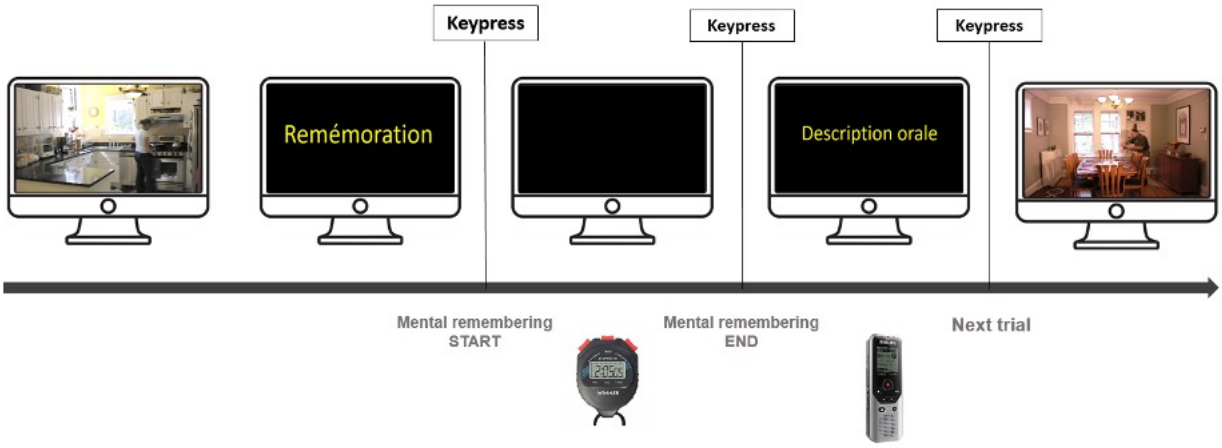


Figure 4.4: Sequence of events in a trial of the memory task. Participants were first instructed to watch a video. After the video ended, a black background with the word "Remember" in yellow was presented on the screen. Participants initiated their mental replay of the video by pressing the space bar, causing the screen to turn black. They pressed the space bar again when they completed their mental replay. The words "Oral description" then appeared, prompting participants to verbally describe the content of their mental replay. The verbal reports were recorded. Finally, participants pressed the space bar to proceed to the next trial.

data, participants were instructed to rest their heads on a chin rest and underwent calibration at the beginning of the experiment. This setup can be observed in Figure 4.4.

After viewing each video, participants initiated and concluded their mental replays by pressing the space bar. This allowed the measurement of the time taken for mental recall. A Temporal Compression Rate (TCR) index was calculated for each stimulus, representing the ratio of the video duration to the time taken for mental replay.

4.2.2 *Pipeline*

Th pipeline used to run experiments in this work is directly adapted from the pipeline described by Roseboom *et al.* [1] A few mechanisms were also added to check additional hypotheses proposed in this work.

4.2.2.1 *Image classification deep neural network*

One modification made to the pipeline mentioned earlier involves the use of other, newer deep neural networks for image classification to calculate accumulators.
Given the age of the original publication, it was indeed worthwhile to investigate whether more recent and larger Convolutional Neural Network architectures could yield improved outcomes compared to AlexNet[2]. However, it's important to note that all the alternative models were still pre-trained on the same dataset as described in the original paper. In this study, the pre-trained models were loaded using the PyTorch[65] deep learning framework.

ALEXNET In this study, we replicate the identical experiment conducted by Roseboom *et al.* [1] to assess the transferability of their task to our specific case. However, we extend the experiment by incorporating the utilization of AlexNet within various pipelines, which will be elaborated on in next sections.
In our experiments, specific layers are chosen from AlexNet. The associations between these selected layers and the corresponding names used to denote them can be found in Section A.1.1.

RESNET-18 The architecture used in this work has been previously described in section 3.2.2, and can be observed in Figure A.1. The exact composition of layers and blocks is depicted in Figure 3.9. This architecture aims to perform image classification. In general, ResNet-18 is expected to perform better than AlexNet in terms of accuracy and overall performance, because ResNet-18 uses a deeper architecture, and skip connections.
However, like any model, it also has its limitations and potential areas for improvement, and its performance is dependent on the specific task.
The prospect of obtaining better results for the task at hand in this thesis motivated the use of ResNet-18.

In our experiments, specific layers are chosen from ResNet-18. The associations between these selected layers and the corresponding names used to denote them can be found in Section A.1.2.

EFFICIENTNET To address one of the questions posed by this study regarding the impact of network size on the generation of human-like outcomes, multiple versions of EfficientNet have been employed. Utilizing various versions of EfficientNet could shed light on this question.
Furthermore, it is expected that EfficientNet will surpass ResNet-18 in terms of image classification performance. Therefore, this hypothesis will also be examined in our findings. The specific architectures utilized in this implementation are EfficientNet-B0, B4, and B7, and their detailed descriptions can be found in Section 3.2.3.

In our experiments, specific layers are chosen from EfficientNets. The associations between these selected layers and the corresponding names used to denote them can be found in Section A.1.3.

DR-TANET AS CHANGE DETECTION METHOD This approach deviates slightly from the previous ones employed. Previously, classification deep neural networks served as the foundation for our accumulation mechanism, aiming to quantify the amount of change in a complete video. However, considering alternative methods becomes necessary for this mechanism. Notably, since the publication of Roseboom *et al.* [1], a novel type of network specifically designed for change detection has emerged. DR-TANet[54] is an example of such a network, capable of detecting changes between two frames by generating a change detection map as output. For further details on the implementation of this model, refer to Section 3.3.

Hence, exploring the use of the output from this model presents another track to investigate whether the quantity of change can predict the duration of video recall. To accumulate the amount of change using this method, the frames were fed to DR-TANet, and the change quantity between frames was computed as the ratio of pixels where a change occurred. These ratios were then summed at each time step to generate a final value representing the changes that took place throughout the entire video.

4.2.2.2 *Threshold mechanism*

In this thesis, the threshold mechanism closely follows the one described in the original work of Roseboom *et al.* [1] However, some minor adjustments have been implemented regarding the predefined values of the parameters T_{max}, T_{min}, τ, and α for each layer.

PARAMETERS CALIBRATION The study conducted by Roseboom *et al.* [1] does not provide specific details on how they determined the values of T_{max}, T_{min}, τ, and α for the layers of AlexNet that they considered (as shown in Table A.6). Since this thesis incorporates additional layers of AlexNet and other networks, it was necessary to define values of parameters for these layers as well. Several methods for determining these values are explained below. However, it should be noted that these techniques may not be optimal and could be further refined in future research. Another potential approach would be to calibrate the parameter values based on a change detection analysis, by using DR-TANet, for instance.

- Manual calibration: One approach for calibrating the parameters of the layers is manual adjustment based on the results obtained from an initial run. By analyzing the graphs depicting the L2-norm between two frames at each time step (see an example in Figure 5.4), T_{max} and T_{min} can be adjusted to these results. However, it is important to note that this method lacks theoretical justification, is prone to high imprecision, and does not really allow to calibrate τ and α in a similar fashion.
- Statistical calibration: In the final version of this work, a statistical analysis is employed to determine the threshold values obtained from an initial run of the pipeline on the video dataset. This approach can be seen as a more quantitative counterpart to the manual calibration method discussed earlier. Specifically, in this method, T_{max} and T_{min} are defined as the 75th percentile and the 25th percentile, respectively, of the observed threshold values across all videos in the dataset. The average of these percentile values is used to establish the final thresholds. The same remark stands: this method may not be optimal, and does not take into account the calibration of parameters τ and α, for which values are automatically set to 100 and 50, respectively, as in the original paper[1].

4.2.2.3 *Set of accumulators*

The mechanisms relative to the sets of accumulators are the same as they were in Roseboom *et al.* [1]. The reader can find in Appendix A.1.1, A.1.2 and A.1.3 the layers for which accumulators are computed for each network architecture.

4.2.2.4 *Regression schemes*

In this section, we will discuss the additional regression schemes investigated in this thesis, alongside the regression scheme implemented by Roseboom *et al.* [1]. Before diving into the details of these, we will first explain how the dataset for these schemes was constructed.

Each participant is exposed to a set of distinct videos, some of which may also be shown to other patients. As a result, certain samples may include the same video but have different target values for recalling times. Consequently, the regression dataset is constructed by extracting the final values of each accumulator as features and generating a sample for every combination of these values with all possible predictions made by patients. This dataset is exemplified in Table 4.3.

Table 4.3: Example of training set for regression schemes. x_i for $i \in \{1, \ldots, n\}$ represent the final accumulator values for layers $i = 1, \ldots, n$ and constitute the features X. y_true is the human target value for recalling time in seconds.

x_1	x_2	...	x_n	$y\_true(s)$
1	0	...	59	19.499
16	0	...	59	26.335
15	0	...	59	65.978
15	0	...	59	45.143
8	0	...	30	11.455
1	0	...	59	9.069

To avoid bias caused by the presence of the same videos in both testing and training phases, the dataset is divided into a training set and a testing set, determined by the videos from which the samples are generated. The testing set consists of 127 samples from 37 different patients and is created using a seeded random selection of 8 videos from the initial video dataset, which contains a total of 64 videos. These selected videos will remain consistent for testing purposes and include:

- Breakfast.LP.30.B.256
- Binder.LP.60.H.150
- Party.CP.60.B.304
- Tire.CP.60.B.130.
- tire.LP.30.H.141
- se.LP.60.B.270
- Breakfast.CP.60.H.10
- Breakfast.LP.60.H.77

On the other hand, the training set comprises 962 samples from 37 different patients and is based on the remaining 56 videos.

LASSOCV MODEL FOR FEATURE SELECTION In this regression scheme, all layers of the network are considered, and accumulators are computed for each layer. The chosen regression method is Lasso cross-validation, which is implemented using Scikit-learn[59]. The theoretical aspects of this method are further discussed in Section 3.1.2. By employing LassoCV, the most influential features that accurately predict values can be identified, making it suitable for our task. It allows us to determine which layers contribute the most to the accurate prediction of human-like values.

To ensure proper cross-validation, a GroupKFold strategy is employed based on the videos. This strategy ensures that each video appears in only one fold at a time, minimizing any bias associated with specific videos.

GRIDSEARCHCV MODEL FOR INDIVIDUAL AND PAIRWISE RESULTS The purpose of this regression scheme was to apply SVR (Section 3.1.1.2) to each individual layer and pairwise combinations of layers within the network. This approach aimed to evaluate the performance of each layer in isolation and in combination, assessing their respective contributions to generating human-like estimates of recall time duration. The SVR parameters were optimized using a grid search and cross-validation to achieve the best possible results.

The grid search was conducted on the following parameter grid for SVR:

```
grid = {
            "C": [1, 1e1, 1e2, 1e3, 1e4, 1e5],
            "epsilon": [0.1],
            "kernel": ["rbf"]
        }
```

Similar to the implementation of LassoCV, a GroupKFold[59] was also employed to ensure appropriate cross-validation. This approach addresses any bias related to specific videos by ensuring that each video appears in only one fold at a time.

5

RESULTS

5.1 EXPERIMENTS

The previous sections introduced various methods, and based on these, several experiments were conducted to compare and discuss multiple aspects. The objectives of these experiments were as follows:

- Assess the transferability of the task proposed by Roseboom *et al.* [1] to our specific case study, investigating whether the quantity of change is predictive of human recall time of episodic memories.
- Analyze the impact of different layers on generating human-like recall times, examining the contributions of individual layers.
- Compare the performance of different models in predicting human-like recall times and assess the influence of model size on these predictions.
- Determine if the position of a layer within the network has any influence on the quality of the predictions.

These objectives guided the design of the experiments and formed the basis for the following analysis and discussion.

5.1.1 *Baselines*

To facilitate the comparison of results obtained from our various experiments, it is essential to establish baseline measures. These baselines serve as reference points to determine whether the experiments yield improved or inferior outcomes. To assess this, two distinct baselines were implemented.

5.1.1.1 *Dummy regressor baseline*

As a baseline, a simple dummy regressor was employed, utilizing the implementation provided by Scikit-learn[59]. This regressor is trained to predict the average of all the ground truth values (i.e., the recalling durations), without considering any input variables.

5.1.1.2 *Real-duration baseline*

The real-duration baseline serves as another reference point for comparison. In this approach, the predicting model is provided with the actual duration of the videos as the target, aiming to assess the performance when using these values.

5.1.2 *Change detection using DR-TANet*

This experiment is based off the description given in Paragraph 4.2.2.1. The idea is to use the quantity of change predicted by DR-TANet instead of the accumulators to constitute the training set to input the regression model with. This can be done in two ways:

- Using only the final value obtained for the accumulated quantity of change.
- Using the whole history of the accumulated quantity of change.

In the following, these experiments will be used as another (different) kind of baselines to compare our results to.

5.1.3 *Experiments and Trials*

In the following, the different experiments and trials led will be described. The names used to distinguish the different experiments in the following sub-sections are all reported in Table 5.1.

Table 5.1: Naming conventions for the different conducted experiments. Vanilla stands for the exact replication of the regression scheme used by Roseboom *et al.* [1], on a selection of layers, detailed in Section 4.1.2.4. LASSO stands for the LassoCV regression scheme detailed in Section 4.2.2.4, and Naive stands for the single and bi-layer regression scheme detailed in Section 4.2.2.4.

Name	Model	Vanilla	LASSO	Naive
exp1	AlexNet	✓	✗	✗
exp2	AlexNet	✗	✓	✗
exp3	AlexNet	✗	✗	✓
exp4	ResNet-18	✗	✓	✗
exp5	ResNet-18	✗	✗	✓
exp6	EfficientNet-B0	✗	✓	✗
exp7	EfficientNet-B0	✗	✗	✓
exp8	EfficientNet-B4	✗	✓	✗
exp9	EfficientNet-B4	✗	✗	✓
exp10	EfficientNet-B7	✗	✓	✗
exp11	EfficientNet-B7	✗	✗	✓

Note that to each naive experiment corresponds a number of trials. Indeed, the naive method implements a grid search on each single or pairwise layers combinations. This number is ($C_{11}^2 + C_{11}^1 = 66$ for AlexNet, $C_7^2 + C_7^1 = 28$ for ResNet-18, and $C_{11}^2 + C_{11}^1 = 66$ for EfficientNets).

5.2 METRICS

In the upcoming section, the performance of various experiments will be compared using a range of commonly employed metrics that evaluate the effectiveness of machine learning models. The following metrics were utilized to assess and compare the performance of the experiments:

- Mean Squared Error (MSE): This metric measures the average squared difference between the predicted values and the actual values. Lower values indicate better performance.
- Mean Absolute Error (MAE): MAE calculates the average absolute difference between the predicted values and the actual values. It provides an indication of the average magnitude of errors without considering their direction.
- R2-score: The R2-score, also known as the coefficient of determination, assesses the proportion of the variance in the target variable that is explained by the model. In theory, the R2-score is expected to be in $[0, 1]$, where a value of 1 indicates a perfect fit and 0 suggests that the model's predictions are no better than randomly guessing. However, in certain implementations such as Scikit-learn[59], the R2-score can occasionally take on negative values.

  Indeed, Scikit-learn's implementation of R2-score compares the model's performance to a baseline model that predicts the mean of the target variable. If the model performs worse than this baseline, the R2-score can become negative, indicating that the model is performing poorly or is even worse than the baseline.
- Maximum Error: This metric determines the maximum difference between the predicted values and the actual values. It highlights the largest individual error in the predictions.
- Explained Variance Score: The explained variance score measures the proportion of the variance in the target variable that is accounted for by the model. It focuses solely on the explained variance, excluding any potential bias.

To assess the relative performance of different experiments, an effort was made to compute the associated uncertainties for each experiment. This was achieved by extracting the metrics obtained during any cross-validation performed on the experiment. These uncertainties are then depicted overlaid on the corresponding graphs in the following figures. The primary type of uncertainty in this task is the aleatoric uncertainty, which refers to the uncertainty present in the data. For example, the target values used in the regression schemes may vary significantly between two patients watching the same video, as their personal memory durations are likely to differ. Note that in LassoCV experiments, only the Mean Squared Error (MSE) can be retrieved, unlike classical cross-validations which allow for the retrieval of various metrics.

5.3 RESULTS PRESENTATION

This section presents the results achieved across various dimensions, driven by the primary questions addressed by this study.

5.3.1 *Analyzing the threshold mechanism and accumulators*

This section presents the results obtained for the threshold and accumulation mechanisms and examines the reliability of the obtained curves in relation to the videos for each network type and their number of segmentation points/EBs.

A few videos were used in order to observe the behavior of the different networks activations on different layers, and see if the network is able to capture some of the changes occurring in it. Four videos were randomly selected in order to compare and analyze against the patterns observed in the graphs:

- Binder.CP.30.B.226: This video features a man standing in an office, holding documents in a binder to scan them on a printer.
- Party.CP.60.B.304: In this video, a man prepares a party by blowing up balloons and arranging them on a table.
- tire.LP.60.B.51: The video shows a man sitting near the tires of his car and repairing them.
- Sweeping.LP.30.H.5: This video features room in which a man enters. The man approaches a bucket, and starts sweeping the floor with dynamism.

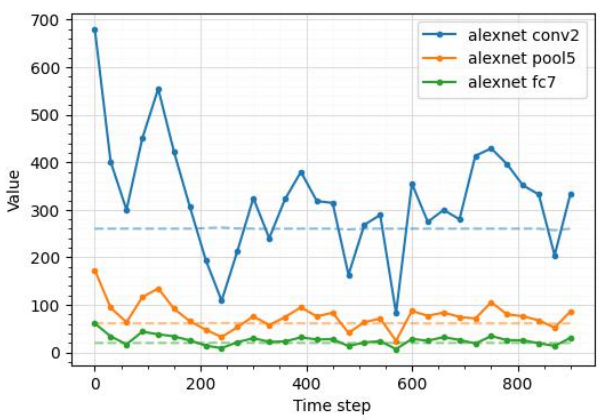
Note that the final video has been purposely selected from the set of videos with more EBs, to compare the results to the other three videos, which belong to the set with fewer event boundaries.

Figure 5.1 presents the results obtained for a selection of layers in AlexNet for the video Binder.CP.30.B.226. The curves obtained for each layer show a similar overall pattern, but as we go deeper into the network, they tend to become scaled-down versions of the initial curve with varying intensities. However, the peaking points remain the same. Similar results can be observed for all network architectures, across different layers, and for all videos.

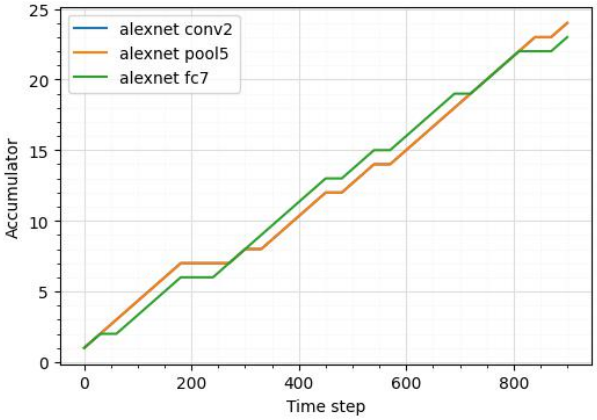
Next, we compare the behaviors of different networks in capturing changes in the selected videos.

Different layers in different networks exhibit similar overall change detection curves, and these curves correspond to events occurring in the videos. Let's consider the analysis of the selected videos:

- Binder.CP.30.B.226: This video shows a man in an office who initially faces away from the camera and then turns around holding a binder. The man opens the binder, scans a document, and enters some numbers on a small keyboard while waiting for the document to be scanned. Notably, the most significant changes in the video occur when the man turns around and opens the printer. These changes are effectively recognized by the networks, as indicated by the corresponding peaks in the graphs. Additionally, the networks seem to detect the vibrant green flashes emitted by the printer. See Figure 5.2 and Figure 5.6 for a detailed parallel between the graph and the video.

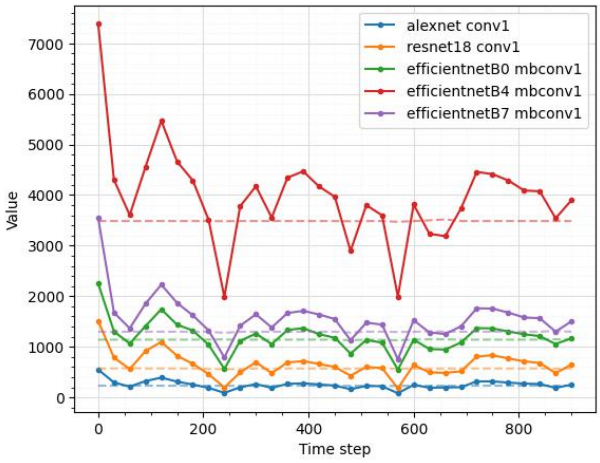


(a) Change detection graphs. The plain curves represent the L2-norms computed between successive layer activations of the network at each time step. The dashed curves in matching colors represent the thresholds computed at each time step, and to which the L2-norm is compared to to for the accumulation mechanism.

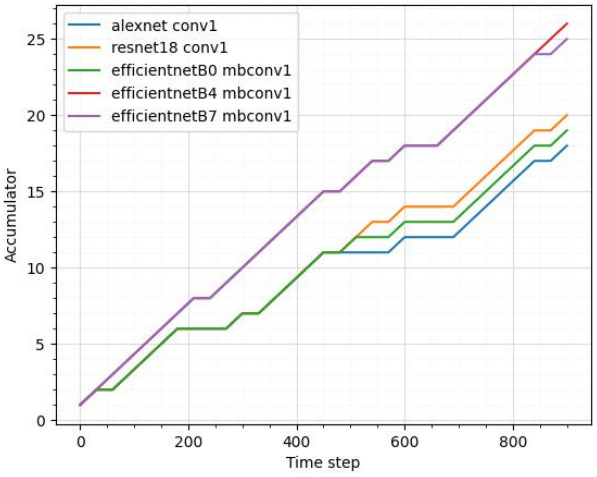


(b) Sense of time graphs. The curves represent the accumulated values present in the layers of network at each time step.

Figure 5.1: Change detection graphs (reproduced from Figure 4.2) and corresponding sense of time graphs for layers conv2, pool5 and fc7 in AlexNet for input video Binder.CP.30.B.226.

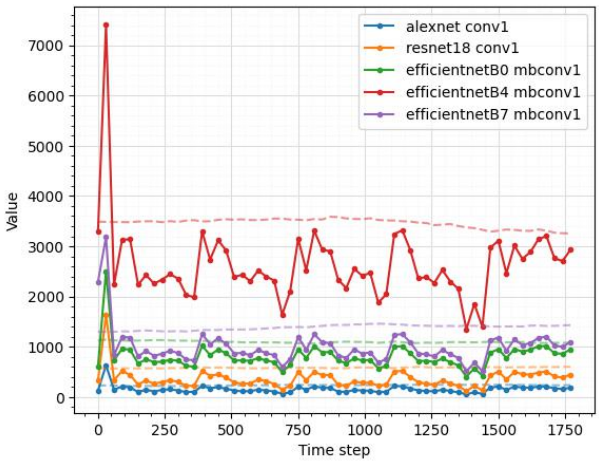


(a) Change detection graphs. The plain curves represent the L2-norms computed between successive layer activations of the network at each time step, for the first layer of each network. The dashed curves in matching colors represent the thresholds computed at each time step, and to which the L2-norm is compared for the accumulation mechanism.

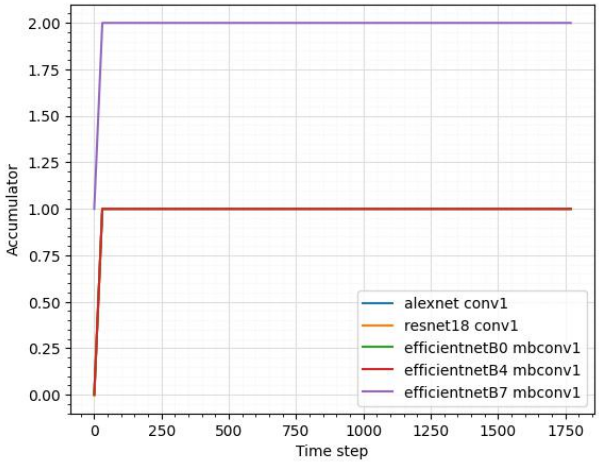


(b) Sense of time graphs. The curves represent the accumulated values present in the layers of network at each time step.

Figure 5.2: Change detection graphs and their corresponding sense of time graph for different layers in ResNet-18, EfficientNet-B0, EfficientNet-B4 and EfficientNet-B7 for input video Binder.CP.30.B.226.

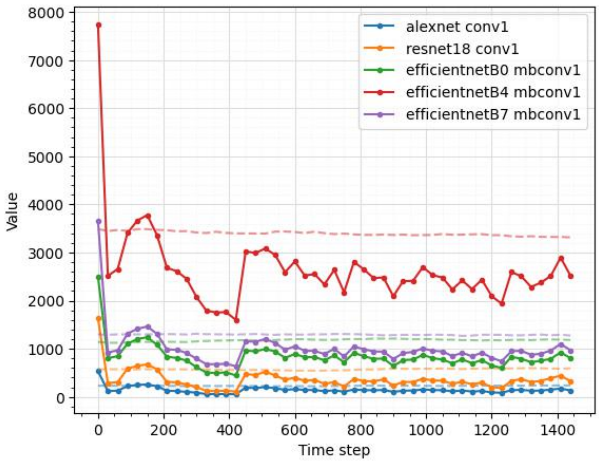


(a) Change detection graphs. The plain curves represent the L2-norms computed between successive layer activations of the network at each time step, for the first layer of each network. The dashed curves in matching colors represent the thresholds computed at each time step, and to which the L2-norm is compared for the accumulation mechanism.

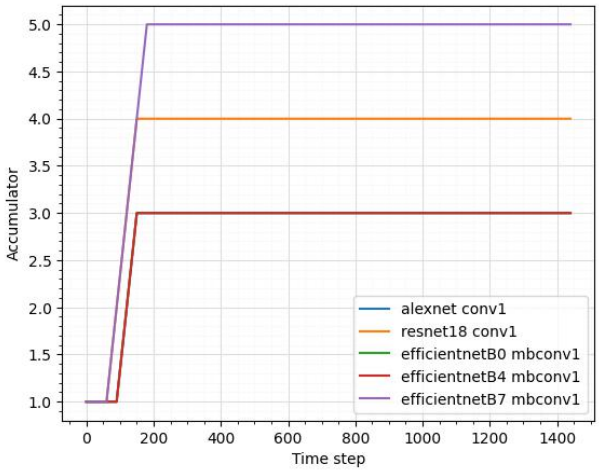


(b) Sense of time graphs. The curves represent the accumulated values present in the layers of network at each time step.

Figure 5.3: Change detection graphs and their corresponding sense of time graph for different layers in ResNet-18, EfficientNet-B0, EfficientNet-B4 and EfficientNet-B7 for input video Party.CP.60.B.304.

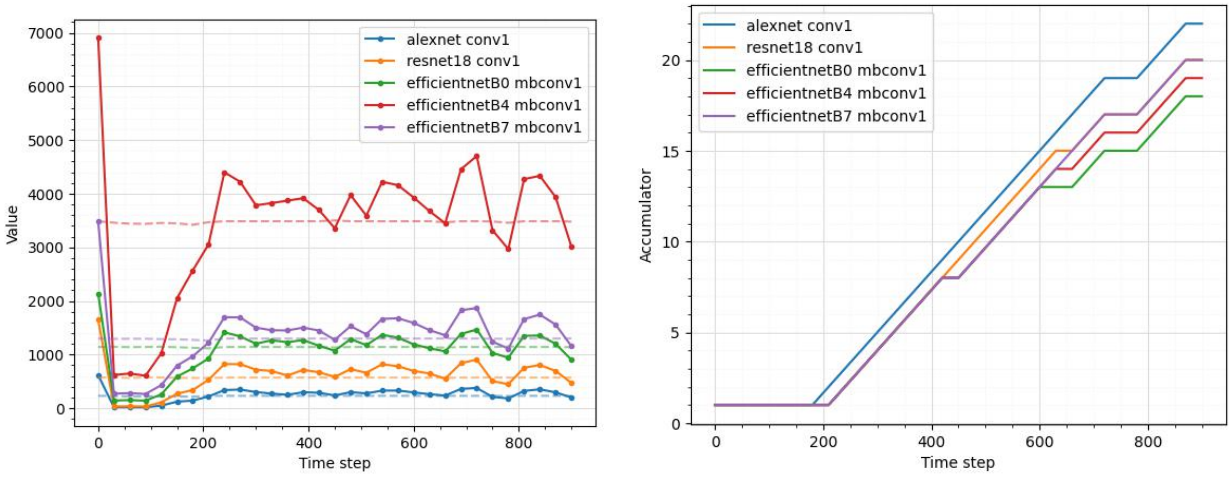


(a) Change detection graphs. The plain curves represent the L2-norms computed between successive layer activations of the network at each time step, for the first layer of each network. The dashed curves in matching colors represent the thresholds computed at each time step, and to which the L2-norm is compared for the accumulation mechanism.



(b) Sense of time graphs. The curves represent the accumulated values present in the layers of network at each time step.

Figure 5.4: Change detection graphs and their corresponding sense of time graph for different layers in ResNet-18, EfficientNet-B0, EfficientNet-B4 and EfficientNet-B7 for input video tire.LP.60.B.51.



(a) Change detection graphs. The plain curves represent the L2-norms computed between successive layer activations of the network at each time step, for the first layer of each network. The dashed curves in matching colors represent the thresholds computed at each time step, and to which the L2-norm is compared for the accumulation mechanism.

(b) Sense of time graphs. The curves represent the accumulated values present in the layers of network at each time step.

Figure 5.5: Change detection graphs and their corresponding sense of time graph for different layers in ResNet-18, EfficientNet-B0, EfficientNet-B4 and EfficientNet-B7 for input video Sweeping.LP.30.H.5.

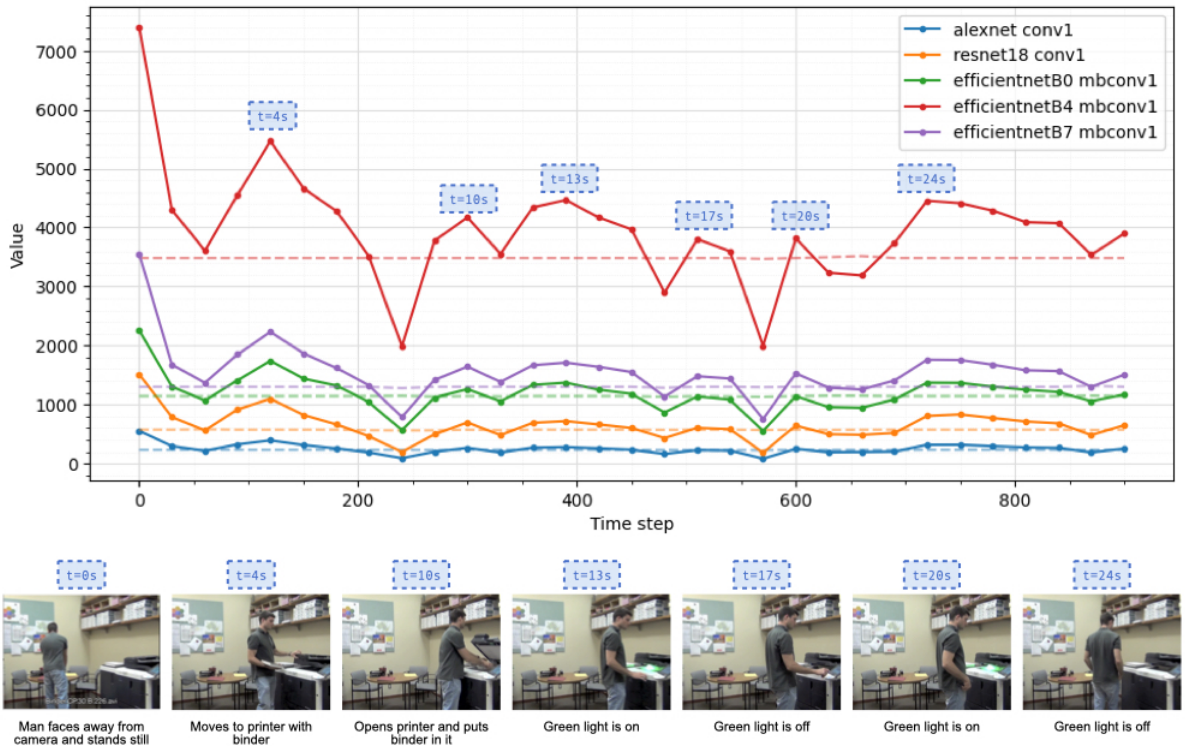


Figure 5.6: Visual comparison between the change detection graph and key events in video Binder.CP.30.B.226. Significant changes are identified as peaks in the graph, indicated by blue boxes. At the bottom, snapshots of the video are shown at the corresponding times indicated in the boxes, accompanied by brief captions explaining the changes that occurred.

- Party.CP.60.B.304: In this video, the man repeatedly blows up a balloon, leans over a table, and places the balloon on it. This repetitive pattern aligns with the periodic-like sequence of peaks observed in Figure 5.3. Towards the end of the video, the man spends more time rearranging objects on the table, which is reflected in the graph showing a constant movement during that period.

- tire.LP.60.B.51: This video is relatively static, with minor movement observed at the beginning as the man changes his position near the car. He briefly kneels, partially stands up, and then kneels again a bit further. Afterward, there is minimal activity until around the halfway point when the man starts rotating his arm continuously until the end of the video. This rotational movement is depicted in Figure 5.4 as a wiggly plateau, which corresponds to the motion of the man's arm and hand.

- Sweeping.LP.30.H.5: At the very beginning of the video, no change occurs until the arrival of the man, which is clearly visible in Figure 5.5, around the 100^{th} time step. The first peak is observed as the man picks up a broom that was leaning against a wall until then. The man also places the dustpan that was attached to the broom on a chair. This move is detected around the 450^{th} time step. The man then starts sweeping the floor, and moves around in the room in constant movement. We observe that the amount of changes detected by the models for this video seem to be significantly higher as in the two previous videos, were almost no changes were occurring.

The networks exhibit their ability to capture significant changes and movements in the input videos, as evident from the change detection graphs.
For the sense of time graphs, two cases can be observed:

- For videos coming from the low EBs videos set (Binder.CP.30.B.226, Party.CP.60.B.304 and tire.LP.60.B.51), minimal changes are detected, leading to plateaus in the graphs, as illustrated in Figure 5.5b and Figure 5.3b. This phenomenon occurs because the corresponding change detection graphs rarely surpass the threshold value of the L2-norm, resulting in few increments of the accumulators.

- For the video coming from the high EBs videos set (Sweeping.LP.30.H.5), we observe the opposite. Way more changes are detected by all the different architectures, as illustrated in Figure 5.5. This suggests the analysis conducted in Figure 5.7.

The content of Figure 5.7 confirms that the number of EBs in a video is indeed correlated to the quantity of changes detected by the method presented in this work. Videos with a higher number of EBs (H) indeed get a higher number of detected changes using our method, and videos with a lower number of EBS (B) obtain a lower number of detected changes using our method.

Also, the accumulated values in different layers for a same network (Figure 5.1) do not seem to vary as much as in Roseboom *et al.* [1]. The issue stems from the calibration of the T_{min} and T_{max} values, causing the threshold to be consistently set too high for the L2-norms to exceed.

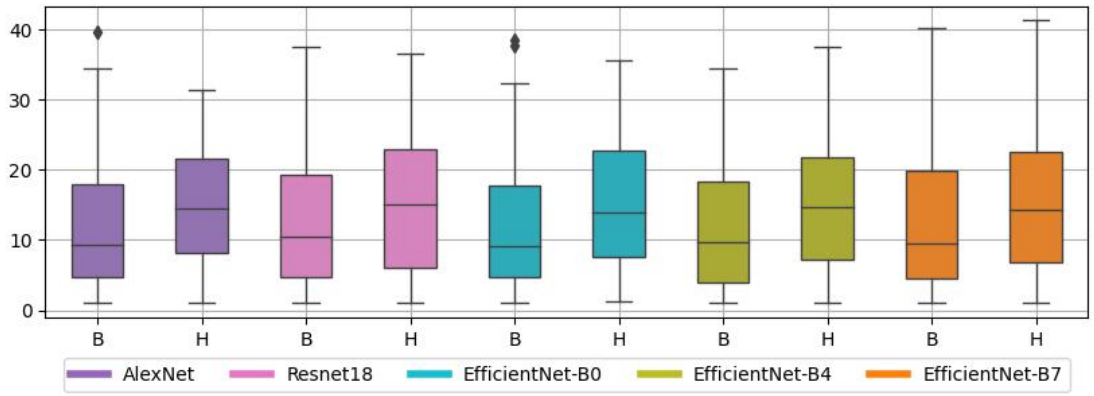


Figure 5.7: Average number of detected changes across all layers of each network architectures for all videos contained in both sets B and H. B denotes the set containing videos with fewer event boundaries (EBs), and H denotes the set containing videos with more EBs.

5.3.2 *Analyzing the Impact of network architecture*

The easiest way to analyze the performances yielded by different network architectures is to compare their performance on the LASSO experiment. The obtained graph can be observed in Figure 5.8.

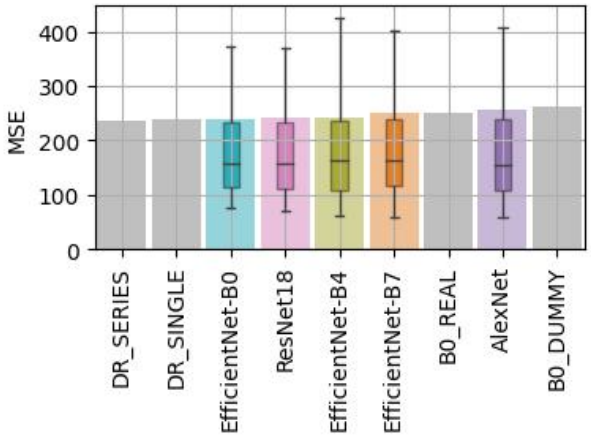
Based on the observed performance metrics in Figure 5.8, we can draw the following conclusions regarding the performance of the network architectures for our problem:

- MSE: The baseline DR-TANet (series) and DR-TANet (final values) demonstrate superior performance, indicating their effectiveness in minimizing the squared differences between predicted and actual values. EfficientNet-B0 and ResNet-18 also perform reasonably well, while EfficientNet-B4, EfficientNet-B7, AlexNet, and the baseline models (real values and dummy) show higher MSE values.

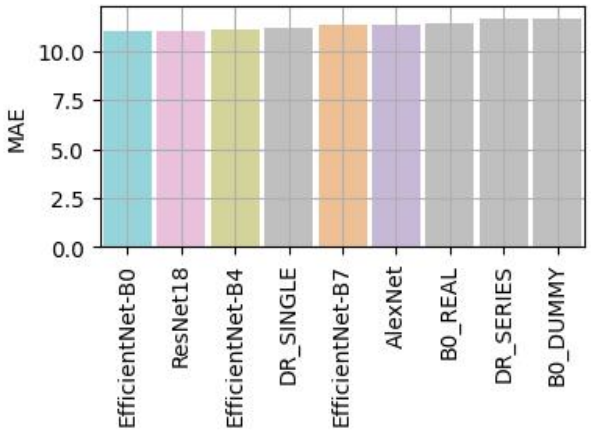
- MAE: EfficientNet-B0 shows the lowest MAE, indicating its ability to provide accurate predictions with minimal absolute errors. ResNet-18, EfficientNet-B4 and DR-TANet (final values), also perform well in terms of MAE. EfficientNet-B7 and AlexNet, as well as the baseline (real values), DR-TANet (series), and the baseline (dummy) exhibit higher MAE values.

- R2-score: EfficientNet-B4, ResNet-18, and EfficientNet-B0 demonstrate positive R2-scores, indicating their ability to explain a significant portion of the variance in the data. On the other hand, the dummy baseline, AlexNet, EfficientNet-B0, the real values baseline, and EfficientNet-B7 exhibit negative R2-scores, suggesting that they perform relatively poorly in capturing the variance.

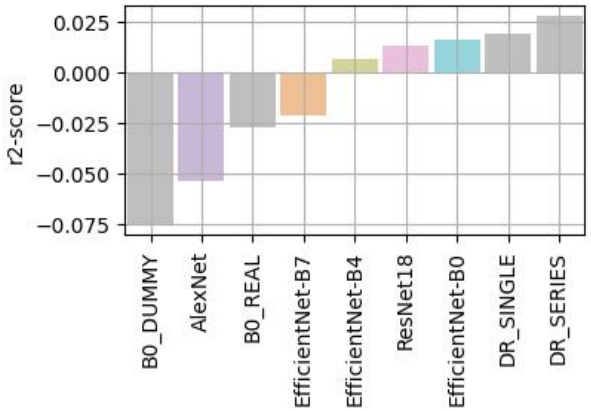
- Max Error: DR-TANet (series) exhibits the lowest maximum error, indicating its ability to provide predictions with minimal deviation from the actual values. All the other baselines also demonstrate similar results. Among the remaining, actual network architectures, EfficientNet-B0 and ResNet-19 perform the best, following the baselines.



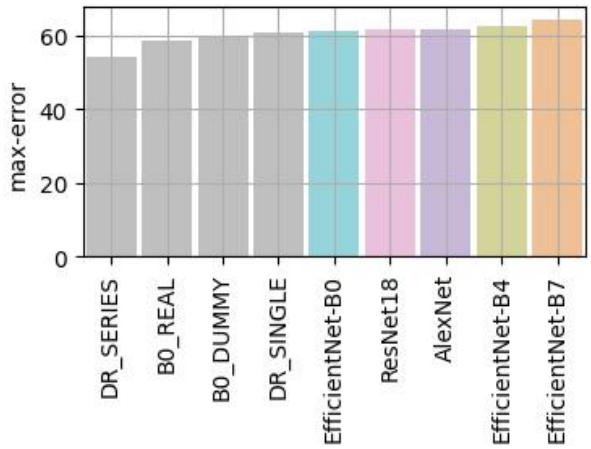
(a) MSE



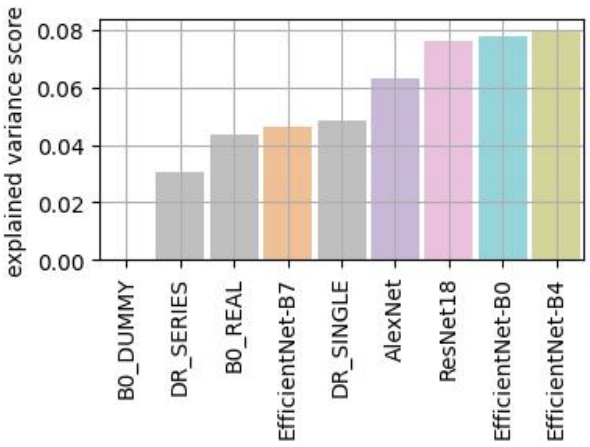
(b) MAE



(c) R2-score



(d) Maximum error



(e) Explained variance score

Figure 5.8: Metrics obtained for each network architecture on the LASSO experiment (exp2, exp4, exp6, exp8, exp10). The grey bars represent the 4 baselines: DR-TANet using only the final value (DR_SINGLE), DR-TANet using the complete history of accumulated values (DR_SERIES), the baseline trained on the videos' real duration (B0_REAL), and the dummy baseline (B0_DUMMY). The results are ranked in ascending order.

- Explained Variance Score: Baseline (dummy) demonstrates the lowest (null) explained variance score, indicating its inability to account for a significant proportion of the variance in the data. EfficientNet-B0 and EfficientNet-B4 show relatively high explained variance scores.

In summary, the DR-TANet (series) and DR-TANet (final values) models generally exhibit superior performance across multiple metrics, followed by EfficientNet-B0 and ResNet-18. However, the performance of network architectures can vary depending on the specific metric evaluated.

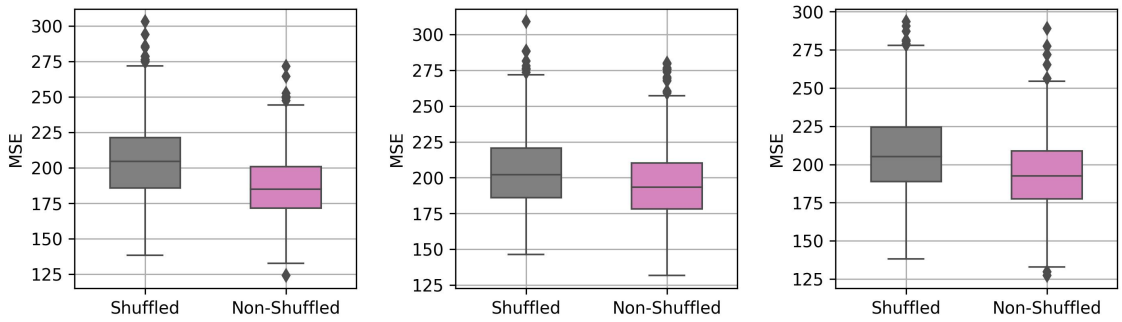
These observations highlight the importance of selecting an appropriate model for the given task. The negative R2-scores indicate models that fail to capture meaningful patterns in the data and perform worse than a simple baseline. DR-TANets, EfficientNet-B0 and ResNet-18 consistently emerge as the top-performing architecture, achieving the highest R2-scores and the lowest MSE, indicating their suitability for the task at hand. In contrast, AlexNet performs poorly across all metrics, suggesting limitations in capturing the underlying relationships in the data.

However, one might wonder if the results obtained for DR-TANet (final), ResNet-18 and EfficientNet-B0 are statistically significant. That is, if the networks identified as the best are really performing better than a random model. To evaluate this, we conduct a statistical significance test. For each selected network architecture, two samples sets are generated:

- Shuffled sampled: The y-values of the dataset are permuted randomly. The dataset is then split into a training and testing set. The Lasso model achieving the best performance on the network architecture is fit on the training set and predictions are made on the testing set. We compute and store the MSE. This process is repeated n times.
  This sample represent a baseline/dummy model.

- Non-Shuffled: The y-values of the dataset are not permuted. The dataset is then split into a training and testing set. The Lasso model achieving the best performance on the network architecture is fit on the training set and predictions are made on the testing set. We compute and store the MSE. This process is repeated n times.
  This sample represent the model that we wish to evaluate.

The statistical test aims to evaluate if these two samples are distributed in the same way. The null hypothesis postulates that the two distributions are identical, and that our model does not perform better than a random model. The alternative hypothesis states that the two distributions are different. By conducting a Kolmogorov-Smirnov test on samples of size $n = 1000$, for each network architecture, we obtain that the null hypothesis can be rejected for DR-TANet (final), ResNet-18 and EfficientNet-B0.
We can then draw conclusions on their performance by observing the results depicted in Figure 5.9, in which we indeed observe that the non-shuffled training data distribution achieves lower MSE metrics for each network architecture, and thus, the difference of performance of these networks compared to a random model is statistically significant.



(a) DR-TANet (final) (b) ResNet-18 (c) EfficientNet-B0

Figure 5.9: Distributions of MSE values for shuffled (random model) and non-shuffled training data for each network architecture.

5.3.3 *Analyzing the Impact of network size*

To investigate the influence of network size, we focus on the lasso experiments using EfficientNet architectures. The corresponding graph is the same as previously, presented in Figure 5.8, on which we will mostly focus on the three instances of EfficientNets: EfficientNet-B0, B4 and B7.

EfficientNet-B0 consistently demonstrates competitive performance across various metrics. It achieves lower Mean Squared Error (MSE) and Mean Absolute Error (MAE) values compared to other architectures such as EfficientNet-B4, and EfficientNet-B7. EfficientNet-B0 also shows positive R2-scores, indicating its ability to explain a significant portion of the variance in the data.
Furthermore, EfficientNet-B0 exhibits lower maximum error and higher explained variance score compared to other architectures (except for explained variance score, where EfficientNet-B4 achieves the best score. EfficientNet-B0 is still ranked second, before EfficientNet-B7).

These observations suggest that the network size has an influence on the performance of the EfficientNet models. EfficientNet-B0, being the smallest variant, performs well across multiple metrics, indicating its effectiveness in capturing patterns and providing accurate predictions for our problem. On the other hand, larger variants such as EfficientNet-B4 and EfficientNet-B7 do not consistently outperform EfficientNet-B0, suggesting that increasing the network size does not necessarily lead to improved performance for our specific task.

In summary, for our problem, EfficientNet-B0 emerges as a strong candidate among the evaluated network architectures. Its compact size, combined with competitive performance across multiple metrics, makes it a suitable choice for capturing meaningful patterns and achieving accurate predictions.

Experiment	Layers
exp2 - AlexNet	[conv1, pool1, pool2, pool5, fc6, output]
exp4 - ResNet-18	[layer3, avgpool]
exp6 - EfficientNet-B0	[mbconv5, conv2]
exp8 - EfficientNet-B4	[conv1, mbconv1, mbconv2, mbconv3, mbconv4, conv2]
exp10 - EfficientNet-B7	[conv1, mbconv1, mbconv2, mbconv5, mbconv6, mbconv7, conv2, pool]

Table 5.2: Layers returned as the most relevant feature combination by Lasso feature selection performed on different experiments.

5.3.4 *Analyzing the Impact of single and pairwise layers*

Experiments using LASSO and naive regression schemes help assess the influence of single and pairwise layers. The following analysis will be conducted in two separate phases:

1. First phase: This phase will compare the results obtained through LASSO experiments for all networks and determine which layers played the most significant role in predicting the most accurate values. This is done using the results of LASSO, which directly indicates which features are the most important by putting a weight of 0 on all other features.

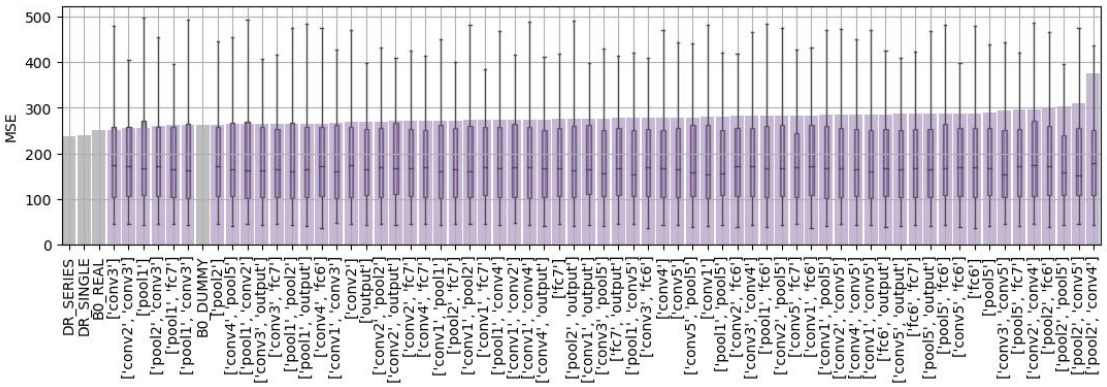
2. Second phase: This phase aims to see which role each layer plays individually, and in pairs into predicting the most correct value, using the naive method. The accumulator of a single layer are used as feature in the training data (or by pair). The parameters of the regression method are decided through a grid search. This phase aims to verify the coherence between these results, and the ones obtained in the first phase using Lasso.

The results of Lasso experiments can be observed in Figure 5.8. For each network architecture, the experiment returns the layers depicted in Table 5.2 as the most relevant features.

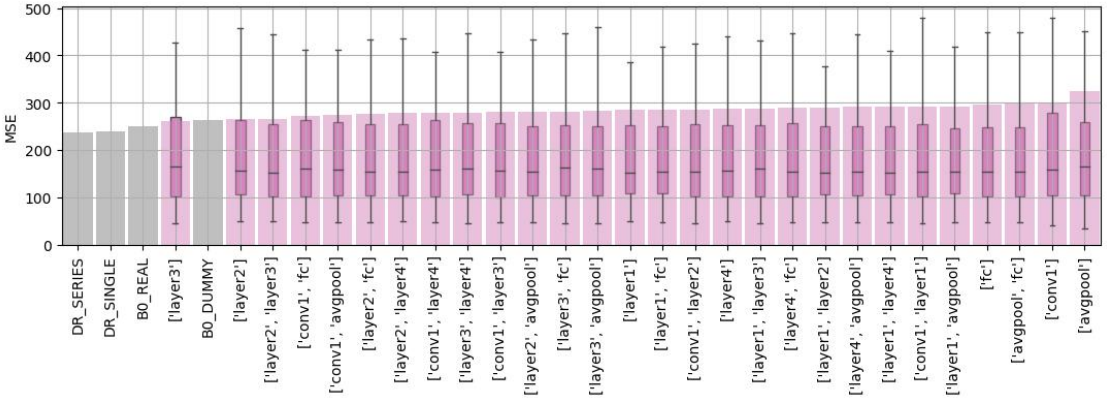
Now, onto the second phase, evaluating each single pair performance, as well as each pairwise layer performance, depicted in Figure 5.10. The figures for other metrics can be found in Section A.3.

From Figure 5.10, several observations can be made regarding the performance of different network accumulation mechanisms on the Mean Squared Error (MSE) metric. The baselines, particularly those using DR-TANet as the accumulation mechanism, generally exhibit the best overall performance. However, when analyzing the results produced by each network accumulation mechanism individually, the following observations can be made:

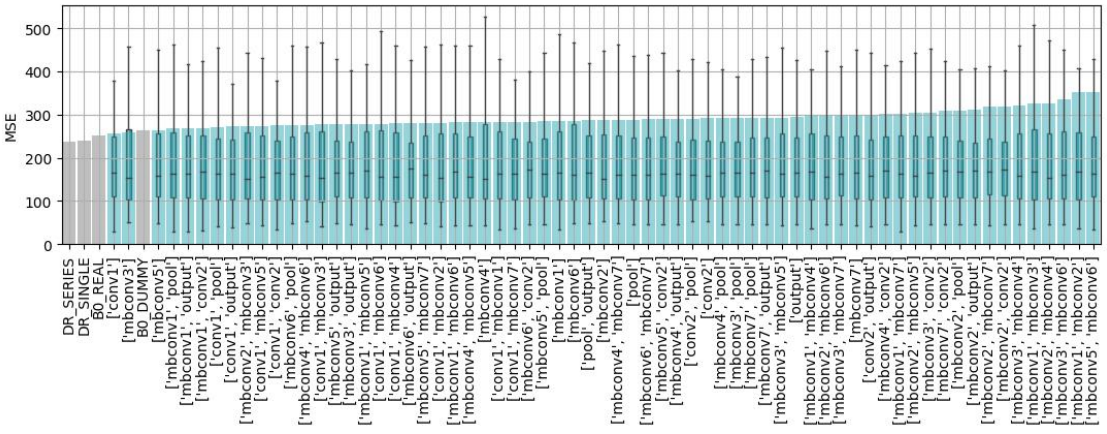
- AlexNet: Among the single layers, `conv3` appears to capture the most information, achieving the best MSE score. This layer is also commonly found in other well-performing combinations such as `[conv2, conv3]` and `[pool2, conv3]`. Surprisingly, the ranking of single layers does not align with the best features combination determined through Lasso feature selection (Table 5.2), as `conv3` is not included in that combination. Additionally, `pool1` demonstrates promising results in this architecture, achieving the second-best MSE score among all single layers.



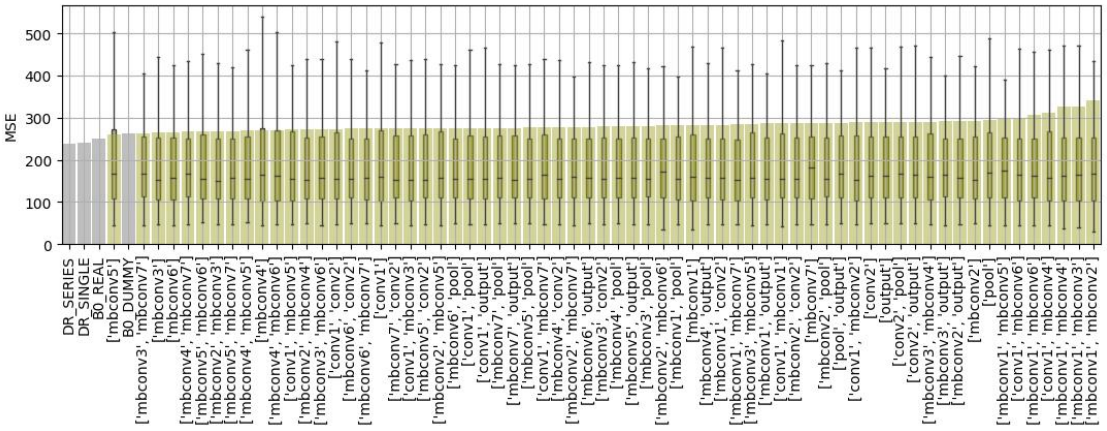
(a) AlexNet - Experiment 3



(b) ResNet-18 - Experiment 5

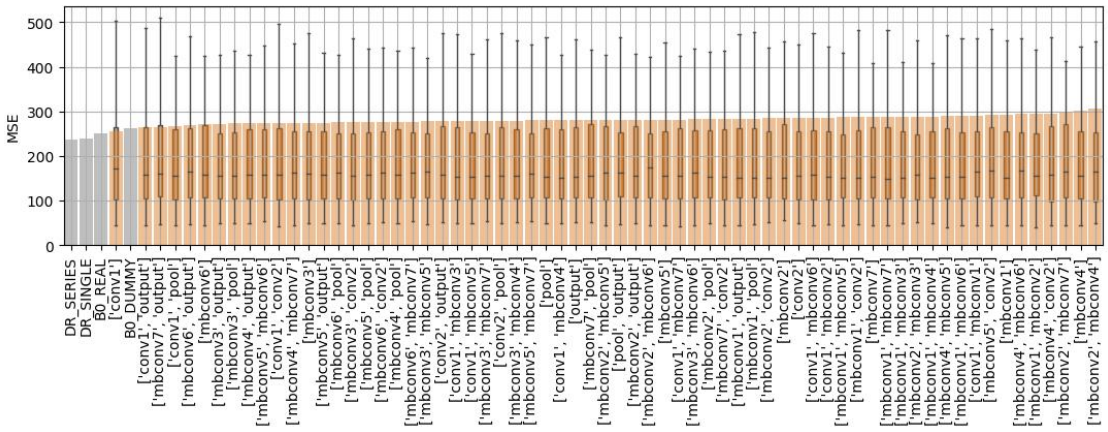


(c) EfficientNet-B0 - Experiment 7



(d) EfficientNet-B4 - Experiment 9

Figure 5.10: Comparison of the performance (in MSE) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.



(e) EfficientNet-B7 - Experiment 11

Figure 5.10: Comparison of the performance achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.

- ResNet-18: The top-performing single layers in terms of capturing information are `layer3` and `layer2`, achieving the best and second-best MSE scores, respectively. These layers are also gathered in combination `[layer2, layer3]`, which also yields good results. This ranking partially aligns with the best features combination from Lasso feature selection (Table 5.2), as `layer3` is included in that combination. However, the presence of `avgpool` in the Lasso combination is surprising, as it performs poorly in the naive experiment.

- EfficientNet-B0: The top-performing single layers are `conv1` and `mbconv3`, achieving the best and second-best MSE scores, respectively. `mbconv5` also demonstrates good performance. This ranking partially aligns with the best features combination from Lasso feature selection (Table 5.2), as `mbconv5` is included in that combination. However, the inclusion of `conv2` in the Lasso combination is surprising, as it performs considerably worse in terms of MSE in the naive experiment. Models that combine specific combinations of **mbconv** layers together tend to yield the least favorable performances.

- EfficientNet-B4: In this architecture, `mbconv5` is the top-performing layer, while `conv1` does not perform as well compared to the previous instance of EfficientNet. The second and third best-performing layers are `mbconv3` and `mbconv6`. The feature selection conducted by Lasso experiments is not highly restrictive, as it includes a combination of six layers out of eleven. Interestingly, the Lasso-selected combination includes only `mbconv3` from the layers considered best in the naive experiment.

- EfficientNet-B7: In this architecture, `conv1` once again emerges as the top-ranking feature, as in EfficientNet-B0, followed by `mbconv6` and `mbconv7`. However, the closeness of their performance is not as pronounced as in other experiments. The feature selection carried out by Lasso is even more restrictive, incorporating a combination of eight layers out of eleven. Importantly, all of the layers identified through the naive experiments are included in this Lasso-selected combination.

Analyzing the impact of single and pairwise layers on the performance of different network architectures provides valuable insights into their effectiveness for our task. From the observed results, several conclusions can be drawn:

- Single Layer Impact: The choice of single layers significantly affects the performance of the accumulation mechanism, for different network architectures. Different layers demonstrate varying levels of information capture and predictive power. For example, in AlexNet, `conv3` and `pool1` perform well, while in ResNet-18, `layer3` and `layer2` exhibit superior performance. EfficientNet-B0 shows promising results with `conv1` and `mbconv3`, while EfficientNet-B4 and EfficientNet-B7 highlight the importance of `mbconv5` and `conv1` respectively. However, the alignment between the top-performing single layers and the features selected by Lasso is not always consistent, indicating that the best-performing single layers may not always be the most important features for prediction, and that even more powerful combinations can be obtained by using more than one layer.
- Pairwise Layer Impact: Considering pairwise combinations of layers further enhances the predictive performance of the network architectures in some cases. Combinations such as `[conv2, conv3]`, `[pool2, conv3]` in AlexNet, and `[layer2, layer3]` in ResNet-18 exhibit improved performance compared to other individual layers. The pairwise layer impact reinforces the idea that certain layers complement each other, capturing different aspects of the input data.

Overall, the analysis of single and pairwise layers highlights the importance of layer selection in network architectures. While some layers consistently demonstrate superior performance across different architectures, others show variations depending on the specific network design. These findings emphasize the need for careful consideration and experimentation when selecting and combining layers to use for the accumulation mechanism.

It was also observed that Lasso feature selection and the naive experiments can yield different sets of features for several reasons:

- Interactions and Combinations: The naive experiments consider the performance of individual layers or pairwise layer combinations. However, Lasso feature selection evaluates the collective impact of all features in the model. It accounts for interactions and combinations of features, enabling it to capture the joint effects and dependencies between features. This can result in different feature selections that emphasize the collective predictive power rather than the individual performance of specific layers.
- Sensitivity to Data and Model Variations: Lasso feature selection is sensitive to the specific dataset and model configuration. Small variations in the dataset or model architecture can lead to different feature selections. The naive experiments, on the other hand, may provide a more direct assessment of the individual layer performance without being influenced by the specific feature selection algorithm.

5.3.5 *Analyzing the Impact of Layer Positions within Networks*

To investigate the influence of layer positions within the networks, we examine plots that exclusively display the results for individual layers in each network architecture. This section focuses on the placement of single layers in each network, utilizing the naive implementation of experiments 3, 5, 7, 9, and 11. The graphs presented in Figure 5.11 illustrate this analysis.
N.B. Within the following graphs, the layers are arranged in the same order as their respective positions within the network.

In the case of AlexNet, as discussed earlier, the `conv3` and `pool1` layers exhibit the lowest mean squared error (MSE), indicating superior performance in terms of error minimization. Conversely, the `pool5` and `fc6` layers demonstrate higher MSE values, suggesting relatively larger errors for these layers. The internal layers, particularly those closer to the center, tend to display lower MSE values compared to the layers closer to the input and output, implying that the network is more likely to make significant errors at these boundary points.

Similarly, for ResNet-18, the `layer2` and `layer3` layers achieve the smallest MSE, while layers 1 and 4 exhibit higher MSE values. This observation suggests that layers closer to the input and output may produce larger errors, indicating a tendency for greater mistakes in these regions.

Shifting our focus to EfficientNet-B0, we find that the `conv1` layer, located at the entry point of the network, attains the lowest MSE. As we move from the central layers towards the extremities of the network, we observe a symmetrical pattern where the central layer achieves very low MSE, followed by an increase in MSE with position before decreasing slightly again at the network's entry points. This overall pattern resembles an "M" shape.

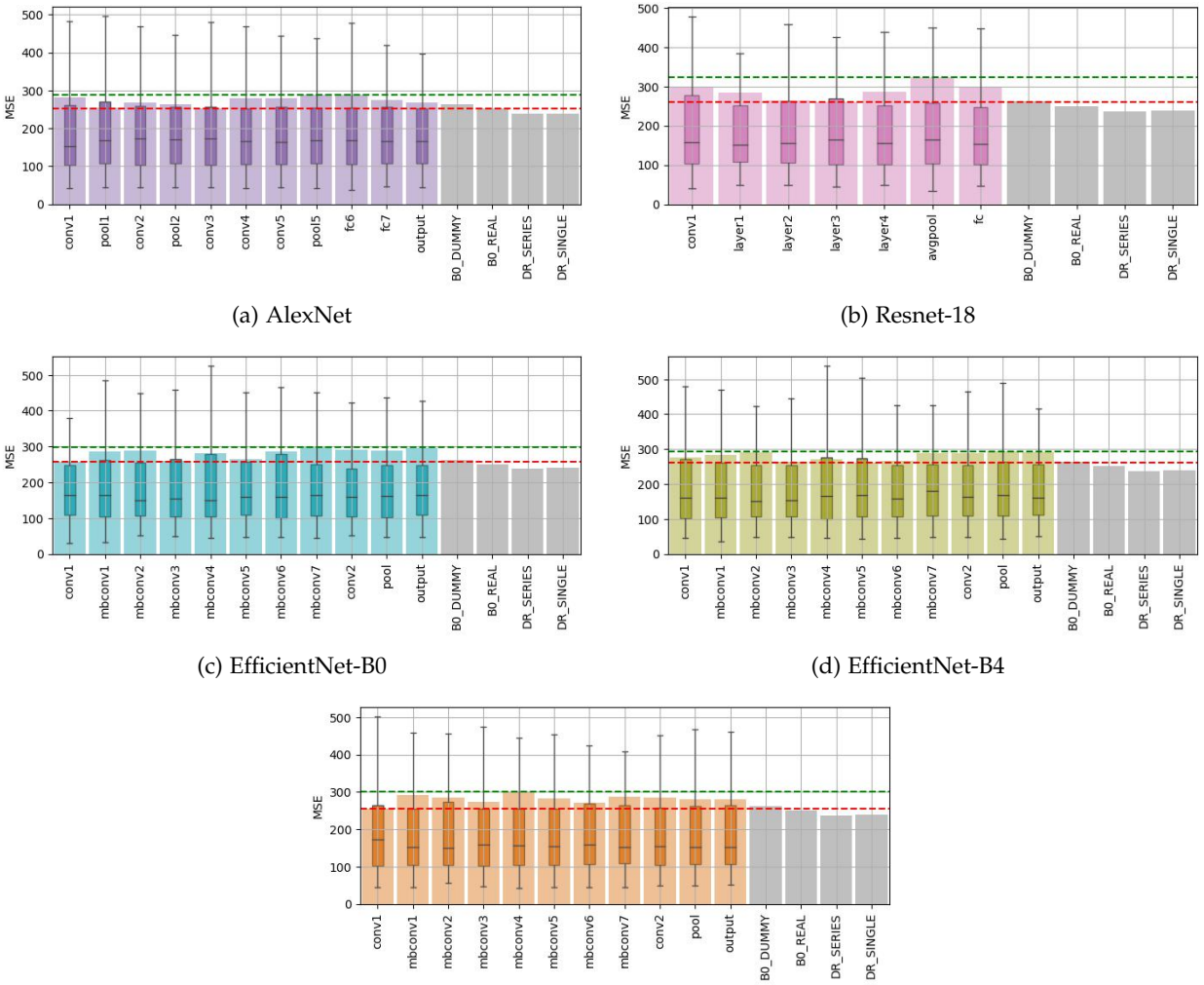
Similarly, in EfficientNet-B4, a similar "M" shape pattern emerges, with the lowest MSE observed in the `mbconv5` and `mbconv6` layers.

Lastly, EfficientNet-B7 exhibits more diverse results, with the `conv1` and `mbconv6` layers achieving the smallest MSE values.

In conclusion, this analysis reveals that MSE values tend to be lower in the internal layers of the networks, while higher errors are observed near the input and output layers. This suggests that network architectures may be more effective at capturing and reducing errors within the intermediate layers.

5.4 INTERPRETATION OF THE RESULTS

In the following, we will summarize the conclusions that can be drawn from every sub-section in Section 5.3.



(a) AlexNet

(b) Resnet-18

(c) EfficientNet-B0

(d) EfficientNet-B4

(e) EfficientNet-B7

Figure 5.11: MSEs for each layer of each network architecture. The grey bars represent the 4 baselines. Green dashed line (–): Maximum value of all candidate layers. Red dashed line (–): Minimum value of all candidate layers (except baselines).

The analysis of the obtained graphs for the replication of the accumulation mechanism from Roseboom *et al.* [1] (Section 5.3.1) demonstrated the capacity of the network to capture salient changes in the input videos. However, it also shed light on the fact that the threshold mechanism was not good in our case, because of a bad calibration of the parameters. Because of this bad calibration, the changes detected by the network are not necessarily detected by the threshold mechanism as a sufficiently significant change.

The analysis of the impact of network architecture (Section 5.3.2) demonstrated that the baseline DR-TANet (series) and DR-TANet (final values) models generally exhibited superior performance across multiple metrics, followed by EfficientNet-B0 and ResNet-18. However, the performance of network architectures varied depending on the specific metric evaluated. The baseline models, as well as AlexNet, exhibited higher MSE, indicating limitations in capturing the underlying relationships in the data. The best architecture being EfficientNet-B0 and ResNet-18, we can conclude that newer and bigger architectures have a tendency to perform

better, however, networks that are too big may not be suitable for this task, for multiple reasons.

Indeed, by examining the impact of network size (Section 5.3.2), we observed that larger network sizes did not consistently lead to better performance. EfficientNet-B0 consistently achieved better performance compared to larger variants such as EfficientNet-B4 and EfficientNet-B7. This suggests that increasing the network size does not necessarily guarantee improved predictive performance and may introduce additional complexity without significant benefits.

The analysis of single and pairwise layers (Section 5.3.4) revealed insights into the predictive role of individual layers within the networks. Different layers in each network architecture showed varying performance in terms of MSE. For example, in AlexNet, `conv3` and `pool1` exhibited the best performance, while in ResNet-18, `layer2` and `layer3` achieved the lowest MSE. These findings indicate that the selection of specific layers within the network architecture for the accumulation mechanism can significantly influence prediction accuracy.

Analyzing the impact of layer positions within networks (Section 5.3.5) further supported the observation that errors tend to be lower in the internal layers of the networks and higher near the input and output layers. This suggests that network architectures are more effective at capturing and minimizing errors within the intermediate layers.

6 CONCLUSION

6.1 SUMMARY OF FINDINGS

The task proposed by Roseboom *et al.* [1] to predict duration of input stimuli based on the salient changes occurring in them can be successfully applied to our specific case study: predict the duration of the human recall time based on the quantity of change.
Our analysis indicates that there is a predictive relationship between the quantity of change and human-like recall times.

By examining the contributions of individual layers, we found that different layers within different network architectures can play a significant role in generating human-like recall times. Certain layers, such as `conv3` in AlexNet and `layer2` and `layer3` in ResNet-18, exhibited better performance in predicting recall times, indicating their importance in capturing relevant information for the task. However, it is a hard task to establish a correspondence between the nature of these layers, and their contribution to the performance of the experiment. Convolutional layers are indeed very suited for vision, and are able to detect features in input. This could explain why it achieves such high performance in AlexNet. However, other convolutional layers are present in AlexNet architecture (and in other networks too), but do not necessarily achieve such high performance, which makes it difficult to draw conclusions based on the sole nature of the layer. Also, layers `layer2` and `layer3` in ResNet-18 are compositions of multiple (convolutional) layers, as depicted in A.1.2. Hence, the same remarks as for AlexNet apply. Ultimately, the position of the layers within the network might be the factor explaining their performance. This point is discussed below.

We compared the performance of different models in predicting human-like recall times and observed variations in their effectiveness. The baseline DR-TANet (series) and DR-TANet (final values) models consistently outperformed other models in terms of minimizing prediction errors. EfficientNet-B0 and ResNet-18 also showed good performance, and we confirmed the statistical significance of these results. Architectures such as EfficientNet-B4, EfficientNet-B7, and AlexNet, presented relatively higher errors.
The influence of model size on predicting human-like recall times was also analyzed. It was found that larger network sizes, such as EfficientNet-B4 and EfficientNet-B7, did not necessarily lead to improved performance. EfficientNet-B0, with a smaller size, consistently achieved better results, suggesting that increasing model size may not always enhance predictive capabilities,

or at least, can improve results, to some extent.

The position of a layer within the network was found to have an influence on the quality of predictions. Internal layers tended to exhibit lower prediction errors, while errors were higher near the input and output layers. This indicates that the selection and positioning of layers within the network architecture are important considerations to design an accurate accumulation mechanism, for achieving accurate predictions.

In conclusion, our analysis supports the transferability of the task proposed by Roseboom *et al.* [1] to our case study, where the quantity of change is predictive of human recall time of episodic memories. Different layers within the network architectures contribute differently to generating human-like recall times. The performance of models varies, with the DR-TANet models and EfficientNet-B0 showing better performance. Model size and layer position also play significant roles in prediction accuracy.

6.2 IMPLICATIONS OF THE STUDY

The implications of this study are twofold: practical implications for understanding human recall times and theoretical implications for network architecture design.

From a practical perspective, the findings suggest that the quantity of change in stimuli can serve as a predictive factor for human recall times of these stimuli. This has potential applications in various domains where understanding and predicting human memory performance is crucial, such as education, user experience design, and cognitive psychology research. By exploiting the relationship between the quantity of change and recall times, practitioners can optimize the design of stimuli and interventions to enhance memory retention and retrieval.

From a theoretical perspective now, the analysis of different layers within network architectures sheds light on their contributions to generating human-like recall times. The identification of specific layers that exhibit better predictive performance provides valuable insights for designing more effective and efficient neural network models. Moreover, the examination of model size and layer position highlights the importance of careful calibration and selection of layers in the network architecture to achieve accurate predictions. These findings contribute to the ongoing research on network design, optimization and biological plausibility, guiding researchers and practitioners in developing more robust and interpretable models.

By combining these practical and theoretical implications, this research contributes to advancing our understanding of human cognition and enhancing the performance of neural network models in various domains.

6.3 LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

While this study provides valuable insights into predicting human recall times based on the quantity of change, there are several limitations that should be acknowledged. These limitations open up opportunities for future research to address these issues and further advance the field.

First, the calibration methods for parameters such as T_{min}, T_{max}, α, and τ could be improved. Fine-tuning these parameters could significantly enhance the performance and accuracy of the models. It has been tested that using other values for these parameters has a huge impact on the accumulation mechanism, and thus, on the quality of the dataset fed to the regression model. Statistical calibration has been selected as the method presented in this work despite its quite poor performance, because it was more theoretically motivated than arbitrary calibration of the parameters, but the obtained results could be improved by using better methods. Future research should explore more sophisticated calibration techniques to optimize these parameters specifically for the task of predicting recall times.

Second, there is a need to better exploit the power of change detection networks. The DR-TANet model demonstrated promising results with minimal effort in comparison to other models. Further investigation and experimentation should be conducted to fully exploit the capabilities of change detection networks and explore their potential for improving recall time predictions. This could involve exploring different architectures and training strategies to maximize their effectiveness.

Third, there is a scope for developing better quantity of change metrics. The metrics used in this study provided a basic measure of change (the L2-norm between activations of layers between consecutive frames for the accumulation mechanisms inspired by Roseboom *et al.* [1], or the pixel-change ratio with DR-TANet), but more sophisticated and comprehensive metrics could be developed to capture the complexity and diversity of changes in stimuli. Future research should focus on developing more refined and accurate metrics that can better quantify the quantity of change in a stimulus.

Another limitation is the challenge posed by the variability of human predictions in the dataset. Different individuals may have diverse estimations of the recalling time for a given stimulus, leading to high uncertainty, and high variability in the dataset. Future research should address this challenge by incorporating methods to account for individual differences and develop models that can better handle the variability in human predictions. This could involve incorporating personalized models or developing ensemble approaches that can account for diverse perspectives. Another solution to this, could be to increase the size of the dataset to overcome these limitations. A larger and more diverse dataset would provide a richer and more representative sample of human recall times, enabling more robust and generalizable predictions.

Furthermore, it is worth noting that considering series of values for each layer, rather than solely relying on the final value present in the accumulator, could be beneficial. The DR-TANet model demonstrated superior results by utilizing series of values, which allowed for capturing the evolution of changes over time. Incorporating this approach more widely could enhance the models' ability to capture and utilize temporal information effectively.

Additionally, future research should explore the use of more biologically plausible architectures instead of simple convolutional neural networks (CNNs) for the accumulation mechanisms. The human brain processes stimuli and recalls information in a complex and dynamic manner, and incorporating architectural elements inspired by neurobiology could potentially improve the predictive capabilities of the models.
Moreover, recurrent neural networks (RNNs) should be considered as an alternative for the regression scheme, as they are particularly well-suited for modeling time-series data. RNNs can capture temporal dependencies and dynamics in the recall process, which may lead to more accurate predictions of recall times. Exploring the use of RNNs in this context could provide valuable insights and improve the overall performance of the models.

In summary, this study has shed light on the predictive nature of the quantity of change in relation to human recall times. However, we acknowledged some the limitations of this work, which present opportunities for future investigations to advance the field. These include the crucial need for better calibration methods for parameters, the exploration of alternative accumulation mechanisms, the development of more refined quantity of change metrics, addressing the challenge of variability in human predictions, increasing the size of the dataset, considering series of values for each layer, and exploring more biologically plausible architectures and the use of recurrent neural networks (RNNs). By addressing these limitations and advancing the field in these directions, we can further improve the accuracy and effectiveness of predicting human recall times based on the quantity of change, leading to a better understanding of human memory processes and potential applications in various domains.

APPENDIX A

A.1 MAPPING BETWEEN LAYER NAMES AND ARCHITECTURES

The names used to denote different layers in this work are mapped to specific blocks or layers in the original architecture. For each network, we detail below the mappings between these names and the position that they refer to in the network.

A.1.1 *AlexNet*

The table provided in A.1 clearly shows the mapping between the layer names used in this work and their corresponding layers in the architecture. This mapping is derived from the layers depicted in Listing 1. For a visual representation of the architecture, please refer to Figure 3.6.

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Listing 1: AlexNet PyTorch Evaluation Model.

Table A.1: Layers mappings for AlexNet

Name	Name from Listing 1	Characteristics
conv1	(features)(0)	11×11 conv. 64
pool1	(features)(2)	/
conv2	(features)(3)	5×5 conv. 192
pool2	(features)(5)	/
conv3	(features)(6)	3×3 conv. 384
conv4	(features)(8)	3×3 conv. 256
conv5	(features)(10)	3×3 conv. 256
pool5	(features)(12)	/
fc6	(classifier)(1)	/
fc7	(classifier)(4)	/
output	(classifier)(6)	/

A.1.2 *ResNet-18*

The table provided in A.2 clearly shows the mapping between the layer names used in this work and their corresponding layers in the architecture. This mapping is derived from the layers depicted in Figure 3.9 and the associated names mentioned in Listing 2. For a visual representation of the architecture, please refer to Figure A.1.

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
```

```
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Listing 2: ResNet-18 PyTorch Evaluation Model.

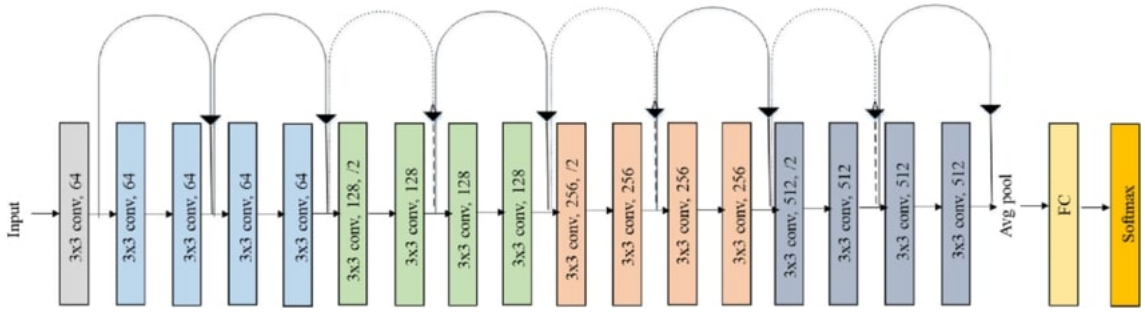


Figure A.1: ResNet-18 Architecture. Source: A deep learning approach for automated diagnosis and multi-class classification of Alzheimer's disease stages using resting-state fMRI and residual neural networks[66].

Table A.2: Layers mappings for ResNet-18.

Name	Name from Listing 2	Name from Figure 3.9	Characteristics
conv1	conv1	conv1	7×7 conv. 64
layer1	layer1	conv2_x	3×3 conv. 64
layer2	layer2	conv3_x	3×3 conv. 128
layer3	layer3	conv4_x	3×3 conv. 256
layer4	layer4	conv5_x	3×3 conv. 512
avgpool	avgpool	average pool	/
fc	fc	fc	/

A.1.3 *EfficientNets*

Since the chosen layers for EfficientNet models (B0, B7, and B7) are identical, only the EfficientNet-B0 model will be described in detail. This decision is made to optimize space, as the basic blocks remain the same in the other two versions of EfficientNet (except for their content, which is scaled). Furthermore, the listings will not be shown here due to their large size.
The mappings are directly retrieved from Table 3.11 in Table A.3.

A.2 PARAMETERS FOR EXPERIMENTS

Table A.7: Parameters for experiment exp3 using naive method

	C	epsilon	kernel	layers
00	1.0	0.1	rbf	[conv3, output]
01	10.0	0.1	rbf	[conv4, conv5]

Continued on next page

Table A.7: Parameters for experiment exp3 using naive method

	C	epsilon	kernel	layers
02	10.0	0.1	rbf	[conv4, pool5]
03	1000.0	0.1	rbf	[conv4, fc6]
04	100.0	0.1	rbf	[conv3]
05	1.0	0.1	rbf	[conv4]
06	10.0	0.1	rbf	[conv3, conv5]
07	10.0	0.1	rbf	[conv3, pool5]
08	1.0	0.1	rbf	[conv1, fc6]
09	1.0	0.1	rbf	[conv1, fc7]
10	10.0	0.1	rbf	[output]
11	1.0	0.1	rbf	[conv1, pool1]
12	1.0	0.1	rbf	[conv1, conv2]
13	1.0	0.1	rbf	[conv1, pool2]
14	1.0	0.1	rbf	[conv1, conv3]
15	1.0	0.1	rbf	[conv1, conv4]
16	1.0	0.1	rbf	[conv1, conv5]
17	1.0	0.1	rbf	[conv1, pool5]
18	1.0	0.1	rbf	[conv1, fc6]
19	1.0	0.1	rbf	[conv1, fc7]
20	1.0	0.1	rbf	[conv1, output]
21	1.0	0.1	rbf	[pool1, conv2]
22	1.0	0.1	rbf	[pool1, pool2]
23	1.0	0.1	rbf	[pool1, conv3]
24	1.0	0.1	rbf	[pool1, conv4]
25	10.0	0.1	rbf	[pool1, conv5]
26	10.0	0.1	rbf	[pool1, pool5]
27	1.0	0.1	rbf	[pool1, fc6]
28	1.0	0.1	rbf	[pool1, fc7]
29	1.0	0.1	rbf	[pool1, output]
30	10.0	0.1	rbf	[conv2, pool2]
31	10.0	0.1	rbf	[conv2, conv3]
32	100.0	0.1	rbf	[conv2, conv4]

Continued on next page

Table A.7: Parameters for experiment exp3 using naive method

	C	epsilon	kernel	layers
33	10.0	0.1	rbf	[conv2, conv5]
34	10.0	0.1	rbf	[conv2, pool5]
35	10.0	0.1	rbf	[conv2, fc6]
36	1.0	0.1	rbf	[conv2, fc7]
37	10.0	0.1	rbf	[conv2, output]
38	10.0	0.1	rbf	[pool2, conv3]
39	100.0	0.1	rbf	[pool2, conv4]
40	10.0	0.1	rbf	[pool2, conv5]
41	10.0	0.1	rbf	[pool2, pool5]
42	10.0	0.1	rbf	[pool2, fc6]
43	1.0	0.1	rbf	[pool2, fc7]
44	1.0	0.1	rbf	[pool2, output]
45	10.0	0.1	rbf	[conv3, conv4]
46	10.0	0.1	rbf	[conv3, conv5]
47	10.0	0.1	rbf	[conv3, pool5]
48	10.0	0.1	rbf	[conv3, fc6]
49	1.0	0.1	rbf	[conv3, fc7]
50	1.0	0.1	rbf	[conv3, output]
51	10.0	0.1	rbf	[conv4, conv5]
52	10.0	0.1	rbf	[conv4, pool5]
53	10.0	0.1	rbf	[conv4, fc6]
54	1.0	0.1	rbf	[conv4, fc7]
55	1.0	0.1	rbf	[conv4, output]
56	10.0	0.1	rbf	[conv5, pool5]
57	10.0	0.1	rbf	[conv5, fc6]
58	10.0	0.1	rbf	[conv5, fc7]
59	1.0	0.1	rbf	[conv5, output]
60	10.0	0.1	rbf	[pool5, fc6]
61	10.0	0.1	rbf	[pool5, fc7]
62	1.0	0.1	rbf	[pool5, output]
63	1.0	0.1	rbf	[fc6, fc7]

Continued on next page

Table A.7: Parameters for experiment exp3 using naive method

	C	epsilon	kernel	layers
64	1.0	0.1	rbf	[fc6, output]
65	1.0	0.1	rbf	[fc7, output]

Table A.8: Parameters for experiment exp5 using naive method

	C	epsilon	kernel	layers
00	1000.0	0.1	rbf	[conv1]
01	1.0	0.1	rbf	[layer1]
02	1000.0	0.1	rbf	[layer2]
03	1000.0	0.1	rbf	[layer3]
04	1.0	0.1	rbf	[layer4]
05	100000.0	0.1	rbf	[avgpool]
06	1.0	0.1	rbf	[fc]
07	1.0	0.1	rbf	[conv1, layer1]
08	1.0	0.1	rbf	[layer2, layer3]
09	1.0	0.1	rbf	[layer2, layer4]
10	1.0	0.1	rbf	[conv1, layer4]
11	1.0	0.1	rbf	[conv1, avgpool]
12	1.0	0.1	rbf	[conv1, fc]
13	10.0	0.1	rbf	[layer1, layer2]
14	10.0	0.1	rbf	[layer1, layer3]
15	1.0	0.1	rbf	[layer1, layer4]
16	1.0	0.1	rbf	[layer1, avgpool]
17	1.0	0.1	rbf	[layer1, fc]
18	1.0	0.1	rbf	[layer2, layer3]
19	1.0	0.1	rbf	[layer2, layer4]
20	1.0	0.1	rbf	[layer2, avgpool]
21	1.0	0.1	rbf	[layer2, fc]
22	10.0	0.1	rbf	[layer3, layer4]
23	10.0	0.1	rbf	[layer3, avgpool]
24	10.0	0.1	rbf	[layer3, fc]
25	1.0	0.1	rbf	[layer4, avgpool]

Continued on next page

Table A.8: Parameters for experiment exp5 using naive method

	C	epsilon	kernel	layers
26	1.0	0.1	rbf	[layer4, fc]
27	1.0	0.1	rbf	[avgpool, fc]

Table A.9: Parameters for experiment exp7 using naive method

	C	epsilon	kernel	layers
00	10000.0	0.1	rbf	[mbconv4, output]
01	1.0	0.1	rbf	[mbconv5, mbconv6]
02	1.0	0.1	rbf	[mbconv5, mbconv7]
03	10.0	0.1	rbf	[mbconv5, conv2]
04	100.0	0.1	rbf	[mbconv4]
05	10.0	0.1	rbf	[mbconv5]
06	100000.0	0.1	rbf	[mbconv4, mbconv6]
07	100.0	0.1	rbf	[mbconv4, mbconv7]
08	1.0	0.1	rbf	[conv1, conv2]
09	1000.0	0.1	rbf	[conv1, pool]
10	10.0	0.1	rbf	[output]
11	1.0	0.1	rbf	[conv1, mbconv1]
12	1.0	0.1	rbf	[conv1, mbconv2]
13	1.0	0.1	rbf	[conv1, mbconv3]
14	1.0	0.1	rbf	[conv1, mbconv4]
15	1.0	0.1	rbf	[conv1, mbconv5]
16	1.0	0.1	rbf	[conv1, mbconv6]
17	1.0	0.1	rbf	[conv1, mbconv7]
18	1.0	0.1	rbf	[conv1, conv2]
19	1.0	0.1	rbf	[conv1, pool]
20	1.0	0.1	rbf	[conv1, output]
21	10.0	0.1	rbf	[mbconv1, mbconv2]
22	10.0	0.1	rbf	[mbconv1, mbconv3]
23	10.0	0.1	rbf	[mbconv1, mbconv4]
24	10.0	0.1	rbf	[mbconv1, mbconv5]
25	10.0	0.1	rbf	[mbconv1, mbconv6]

Continued on next page

Table A.9: Parameters for experiment exp7 using naive method

	C	epsilon	kernel	layers
26	10.0	0.1	rbf	[mbconv1, mbconv7]
27	10.0	0.1	rbf	[mbconv1, conv2]
28	10.0	0.1	rbf	[mbconv1, pool]
29	10.0	0.1	rbf	[mbconv1, output]
30	10.0	0.1	rbf	[mbconv2, mbconv3]
31	100.0	0.1	rbf	[mbconv2, mbconv4]
32	100.0	0.1	rbf	[mbconv2, mbconv5]
33	10.0	0.1	rbf	[mbconv2, mbconv6]
34	10.0	0.1	rbf	[mbconv2, mbconv7]
35	10.0	0.1	rbf	[mbconv2, conv2]
36	10.0	0.1	rbf	[mbconv2, pool]
37	10.0	0.1	rbf	[mbconv2, output]
38	100.0	0.1	rbf	[mbconv3, mbconv4]
39	100.0	0.1	rbf	[mbconv3, mbconv5]
40	100.0	0.1	rbf	[mbconv3, mbconv6]
41	10.0	0.1	rbf	[mbconv3, mbconv7]
42	10.0	0.1	rbf	[mbconv3, conv2]
43	10.0	0.1	rbf	[mbconv3, pool]
44	10.0	0.1	rbf	[mbconv3, output]
45	100.0	0.1	rbf	[mbconv4, mbconv5]
46	1.0	0.1	rbf	[mbconv4, mbconv6]
47	1.0	0.1	rbf	[mbconv4, mbconv7]
48	10.0	0.1	rbf	[mbconv4, conv2]
49	10.0	0.1	rbf	[mbconv4, pool]
50	10.0	0.1	rbf	[mbconv4, output]
51	1000.0	0.1	rbf	[mbconv5, mbconv6]
52	1.0	0.1	rbf	[mbconv5, mbconv7]
53	10.0	0.1	rbf	[mbconv5, conv2]
54	10.0	0.1	rbf	[mbconv5, pool]
55	10.0	0.1	rbf	[mbconv5, output]
56	100.0	0.1	rbf	[mbconv6, mbconv7]

Continued on next page

Table A.9: Parameters for experiment exp7 using naive method

	C	epsilon	kernel	layers
57	1000.0	0.1	rbf	[mbconv6, conv2]
58	100.0	0.1	rbf	[mbconv6, pool]
59	10.0	0.1	rbf	[mbconv6, output]
60	10.0	0.1	rbf	[mbconv7, conv2]
61	10.0	0.1	rbf	[mbconv7, pool]
62	10.0	0.1	rbf	[mbconv7, output]
63	1.0	0.1	rbf	[conv2, pool]
64	10.0	0.1	rbf	[conv2, output]
65	10.0	0.1	rbf	[pool, output]

Table A.10: Parameters for experiment exp9 using naive method

	C	epsilon	kernel	layers
00	1.0	0.1	rbf	[mbconv4, output]
01	1.0	0.1	rbf	[mbconv5, mbconv6]
02	1.0	0.1	rbf	[mbconv5, mbconv7]
03	100.0	0.1	rbf	[mbconv5, conv2]
04	1000.0	0.1	rbf	[mbconv4]
05	100.0	0.1	rbf	[mbconv5]
06	10.0	0.1	rbf	[mbconv4, mbconv6]
07	10.0	0.1	rbf	[mbconv4, mbconv7]
08	1.0	0.1	rbf	[conv1, conv2]
09	100.0	0.1	rbf	[conv1, pool]
10	1.0	0.1	rbf	[output]
11	10.0	0.1	rbf	[conv1, mbconv1]
12	1.0	0.1	rbf	[conv1, mbconv2]
13	1.0	0.1	rbf	[conv1, mbconv3]
14	10.0	0.1	rbf	[conv1, mbconv4]
15	1.0	0.1	rbf	[conv1, mbconv5]
16	10.0	0.1	rbf	[conv1, mbconv6]
17	1.0	0.1	rbf	[conv1, mbconv7]
18	1.0	0.1	rbf	[conv1, conv2]

Continued on next page

Table A.10: Parameters for experiment exp9 using naive method

	C	epsilon	kernel	layers
19	1.0	0.1	rbf	[conv1, pool]
20	1.0	0.1	rbf	[conv1, output]
21	100.0	0.1	rbf	[mbconv1, mbconv2]
22	10.0	0.1	rbf	[mbconv1, mbconv3]
23	10.0	0.1	rbf	[mbconv1, mbconv4]
24	100.0	0.1	rbf	[mbconv1, mbconv5]
25	100.0	0.1	rbf	[mbconv1, mbconv6]
26	1.0	0.1	rbf	[mbconv1, mbconv7]
27	1.0	0.1	rbf	[mbconv1, conv2]
28	1.0	0.1	rbf	[mbconv1, pool]
29	1.0	0.1	rbf	[mbconv1, output]
30	10.0	0.1	rbf	[mbconv2, mbconv3]
31	10.0	0.1	rbf	[mbconv2, mbconv4]
32	10.0	0.1	rbf	[mbconv2, mbconv5]
33	100.0	0.1	rbf	[mbconv2, mbconv6]
34	1.0	0.1	rbf	[mbconv2, mbconv7]
35	1.0	0.1	rbf	[mbconv2, conv2]
36	1.0	0.1	rbf	[mbconv2, pool]
37	1.0	0.1	rbf	[mbconv2, output]
38	100.0	0.1	rbf	[mbconv3, mbconv4]
39	100.0	0.1	rbf	[mbconv3, mbconv5]
40	10.0	0.1	rbf	[mbconv3, mbconv6]
41	10.0	0.1	rbf	[mbconv3, mbconv7]
42	1.0	0.1	rbf	[mbconv3, conv2]
43	1.0	0.1	rbf	[mbconv3, pool]
44	10.0	0.1	rbf	[mbconv3, output]
45	1.0	0.1	rbf	[mbconv4, mbconv5]
46	10.0	0.1	rbf	[mbconv4, mbconv6]
47	10.0	0.1	rbf	[mbconv4, mbconv7]
48	1.0	0.1	rbf	[mbconv4, conv2]
49	1.0	0.1	rbf	[mbconv4, pool]

Continued on next page

Table A.10: Parameters for experiment exp9 using naive method

	C	epsilon	kernel	layers
50	1.0	0.1	rbf	[mbconv4, output]
51	1.0	0.1	rbf	[mbconv5, mbconv6]
52	1.0	0.1	rbf	[mbconv5, mbconv7]
53	1.0	0.1	rbf	[mbconv5, conv2]
54	1.0	0.1	rbf	[mbconv5, pool]
55	1.0	0.1	rbf	[mbconv5, output]
56	1.0	0.1	rbf	[mbconv6, mbconv7]
57	1.0	0.1	rbf	[mbconv6, conv2]
58	1.0	0.1	rbf	[mbconv6, pool]
59	1.0	0.1	rbf	[mbconv6, output]
60	1.0	0.1	rbf	[mbconv7, conv2]
61	1.0	0.1	rbf	[mbconv7, pool]
62	1.0	0.1	rbf	[mbconv7, output]
63	1.0	0.1	rbf	[conv2, pool]
64	1.0	0.1	rbf	[conv2, output]
65	1.0	0.1	rbf	[pool, output]

Table A.11: Parameters for experiment exp11 using naive method

	C	epsilon	kernel	layers
00	100.0	0.1	rbf	[mbconv4, output]
01	1.0	0.1	rbf	[mbconv5, mbconv6]
02	1.0	0.1	rbf	[mbconv5, mbconv7]
03	100.0	0.1	rbf	[mbconv5, conv2]
04	1000.0	0.1	rbf	[mbconv4]
05	100.0	0.1	rbf	[mbconv5]
06	10.0	0.1	rbf	[mbconv4, mbconv6]
07	10.0	0.1	rbf	[mbconv4, mbconv7]
08	1.0	0.1	rbf	[conv1, conv2]
09	1.0	0.1	rbf	[conv1, pool]
10	1.0	0.1	rbf	[output]
11	10.0	0.1	rbf	[conv1, mbconv1]

Continued on next page

Table A.11: Parameters for experiment exp11 using naive method

	C	epsilon	kernel	layers
12	1.0	0.1	rbf	[conv1, mbconv2]
13	1.0	0.1	rbf	[conv1, mbconv3]
14	1.0	0.1	rbf	[conv1, mbconv4]
15	1.0	0.1	rbf	[conv1, mbconv5]
16	10.0	0.1	rbf	[conv1, mbconv6]
17	1.0	0.1	rbf	[conv1, mbconv7]
18	1.0	0.1	rbf	[conv1, conv2]
19	1.0	0.1	rbf	[conv1, pool]
20	1.0	0.1	rbf	[conv1, output]
21	10.0	0.1	rbf	[mbconv1, mbconv2]
22	1.0	0.1	rbf	[mbconv1, mbconv3]
23	1.0	0.1	rbf	[mbconv1, mbconv4]
24	1.0	0.1	rbf	[mbconv1, mbconv5]
25	1.0	0.1	rbf	[mbconv1, mbconv6]
26	1.0	0.1	rbf	[mbconv1, mbconv7]
27	1.0	0.1	rbf	[mbconv1, conv2]
28	1.0	0.1	rbf	[mbconv1, pool]
29	1.0	0.1	rbf	[mbconv1, output]
30	10.0	0.1	rbf	[mbconv2, mbconv3]
31	100.0	0.1	rbf	[mbconv2, mbconv4]
32	100.0	0.1	rbf	[mbconv2, mbconv5]
33	100.0	0.1	rbf	[mbconv2, mbconv6]
34	100.0	0.1	rbf	[mbconv2, mbconv7]
35	1.0	0.1	rbf	[mbconv2, conv2]
36	1.0	0.1	rbf	[mbconv2, pool]
37	1.0	0.1	rbf	[mbconv2, output]
38	1.0	0.1	rbf	[mbconv3, mbconv4]
39	10.0	0.1	rbf	[mbconv3, mbconv5]
40	100.0	0.1	rbf	[mbconv3, mbconv6]
41	1.0	0.1	rbf	[mbconv3, mbconv7]
42	10.0	0.1	rbf	[mbconv3, conv2]

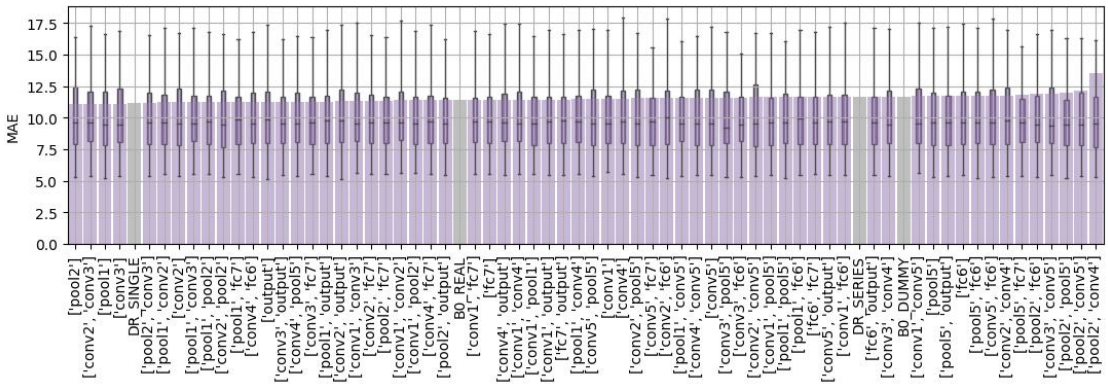
Continued on next page

Table A.11: Parameters for experiment exp11 using naive method

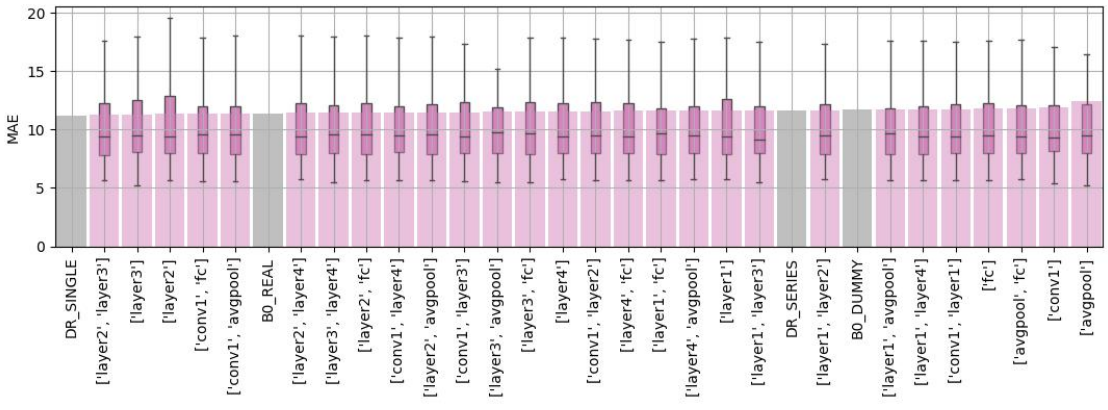
	C	epsilon	kernel	layers
43	1.0	0.1	rbf	[mbconv3, pool]
44	1.0	0.1	rbf	[mbconv3, output]
45	1000.0	0.1	rbf	[mbconv4, mbconv5]
46	100.0	0.1	rbf	[mbconv4, mbconv6]
47	100.0	0.1	rbf	[mbconv4, mbconv7]
48	100.0	0.1	rbf	[mbconv4, conv2]
49	1.0	0.1	rbf	[mbconv4, pool]
50	1.0	0.1	rbf	[mbconv4, output]
51	1.0	0.1	rbf	[mbconv5, mbconv6]
52	1.0	0.1	rbf	[mbconv5, mbconv7]
53	1000.0	0.1	rbf	[mbconv5, conv2]
54	1.0	0.1	rbf	[mbconv5, pool]
55	1.0	0.1	rbf	[mbconv5, output]
56	1.0	0.1	rbf	[mbconv6, mbconv7]
57	1.0	0.1	rbf	[mbconv6, conv2]
58	1.0	0.1	rbf	[mbconv6, pool]
59	10.0	0.1	rbf	[mbconv6, output]
60	10.0	0.1	rbf	[mbconv7, conv2]
61	1.0	0.1	rbf	[mbconv7, pool]
62	10.0	0.1	rbf	[mbconv7, output]
63	1.0	0.1	rbf	[conv2, pool]
64	1.0	0.1	rbf	[conv2, output]
65	10.0	0.1	rbf	[pool, output]

A.3 MORE RESULTS ON ANALYZING THE IMPACT OF SINGLE AND PAIRWISE LAYERS

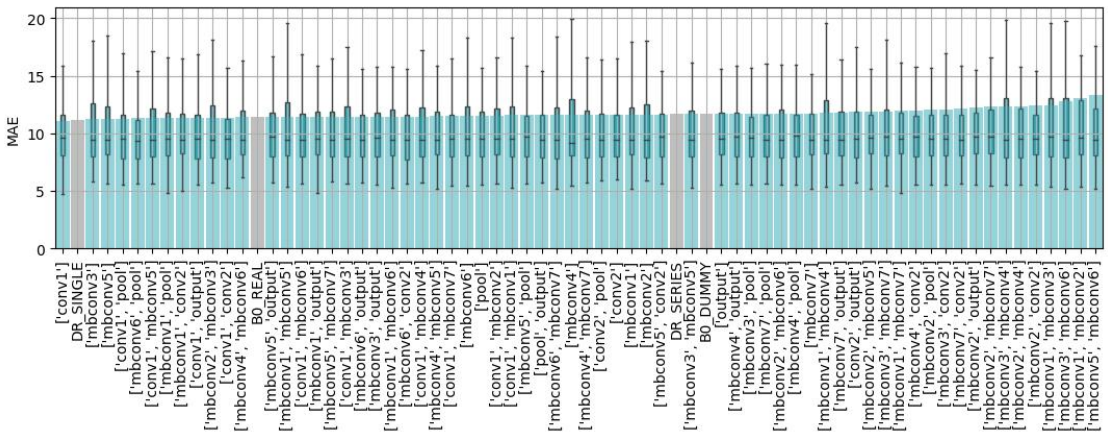
A.4 MORE RESULTS ON ANALYZING THE IMPACT OF LAYER POSITIONS WITHIN NETWORKS



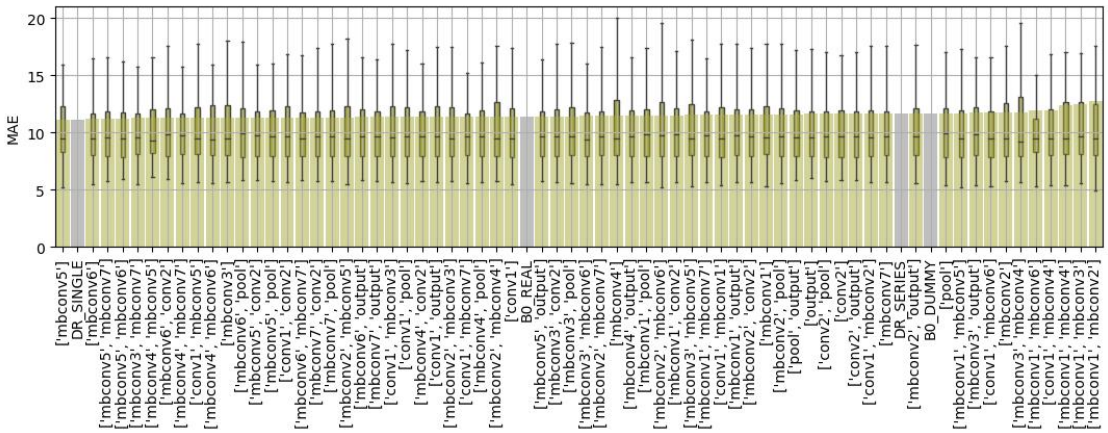
(a) AlexNet



(b) ResNet-18



(c) EfficientNet-B0



(d) EfficientNet-B4

Figure A.2: Comparison of the performance (in MAE) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.

Table A.3: Layers mappings for EfficientNets

Name	**Name from Listing 3.11**
`conv1`	Stage 1 Conv3 × 3
`mbconv1`	Stage 2 MBConv1, k3 × 3
`mbconv2`	Stage 3 MBConv6, k3 × 3
`mbconv3`	Stage 4 MBConv6, k5 × 5
`mbconv4`	Stage 5 MBConv6, k3 × 3
`mbconv5`	Stage 6 MBConv6, k5 × 5
`mbconv6`	Stage 7 MBConv6, k5 × 5
`mbconv7`	Stage 8 MBConv6, 3 × 3
`conv2`	Stage 9 Conv1 × 1
`pool`	Stage 9 Pooling
`output`	Stage 9 FC

Table A.4: Parameters of vanilla experiment conducted by Roseboom et al.[1] in their original work.

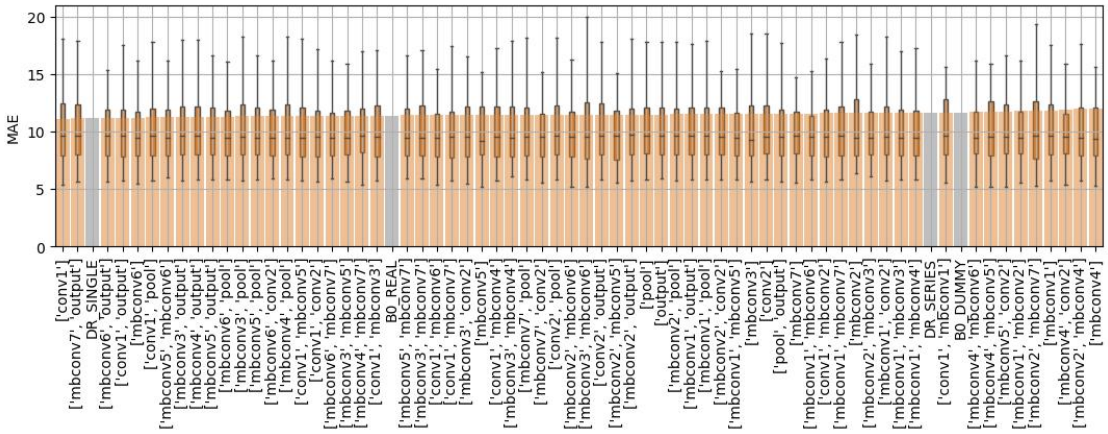
`kernel`	`gamma`	**C**
'rbf'	10^{-4}	10^{-3}

Table A.5: Parameters for LASSO experiments

	alphas	eps	max_iter	n_alphas	tol
exp2	None	0.001	1000	100	0.0001
exp4	None	0.001	1000	100	0.0001
exp6	None	0.001	1000	100	0.0001
exp8	None	0.001	1000	100	0.0001
exp10	None	0.001	1000	100	0.0001

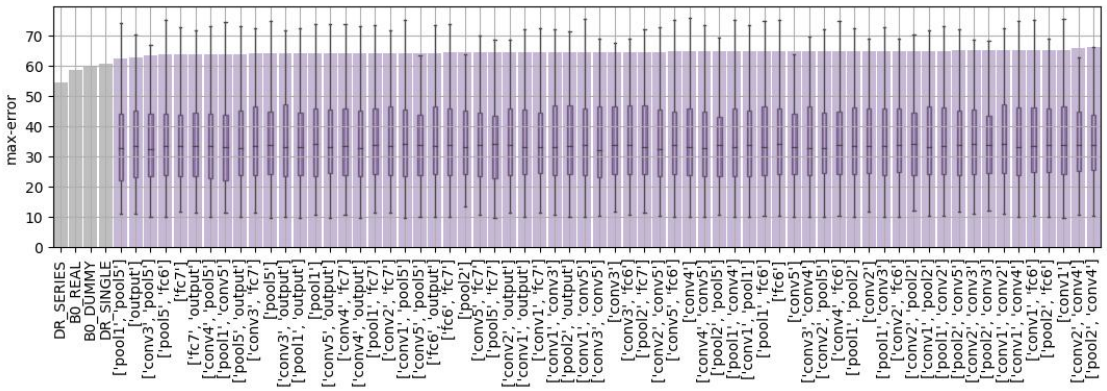
Table A.6: Parameters for the vanilla experiment

	C	gamma	epsilon	kernel
exp1	0.001	0.0001	0.1	rbf

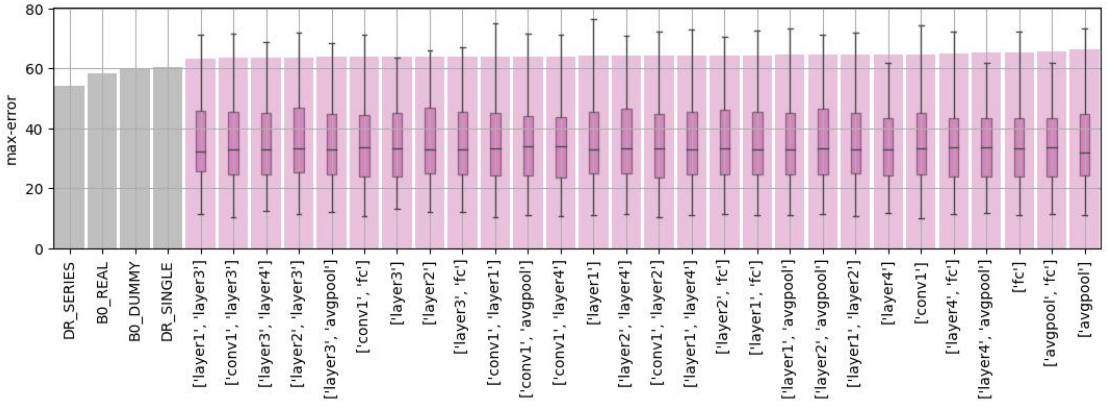


(e) EfficientNet-B7

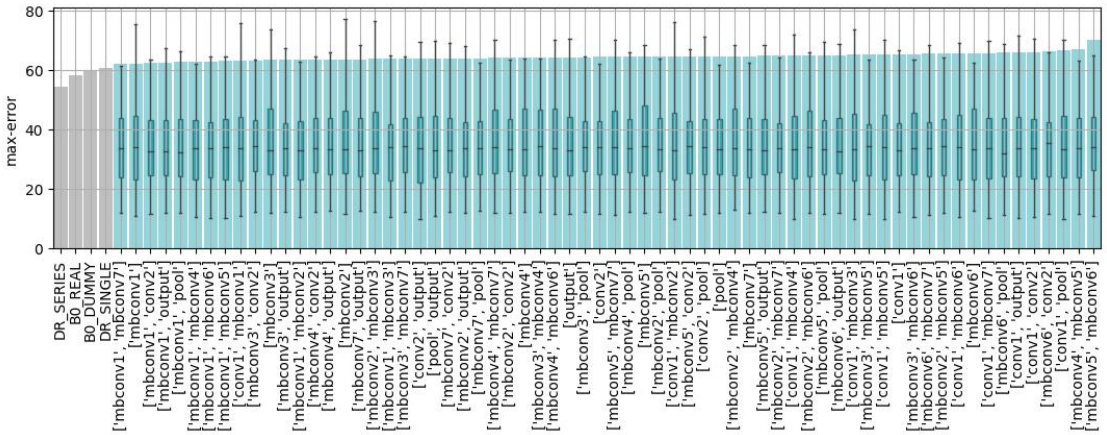
Figure A.2: Comparison of the performance (in MAE) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.



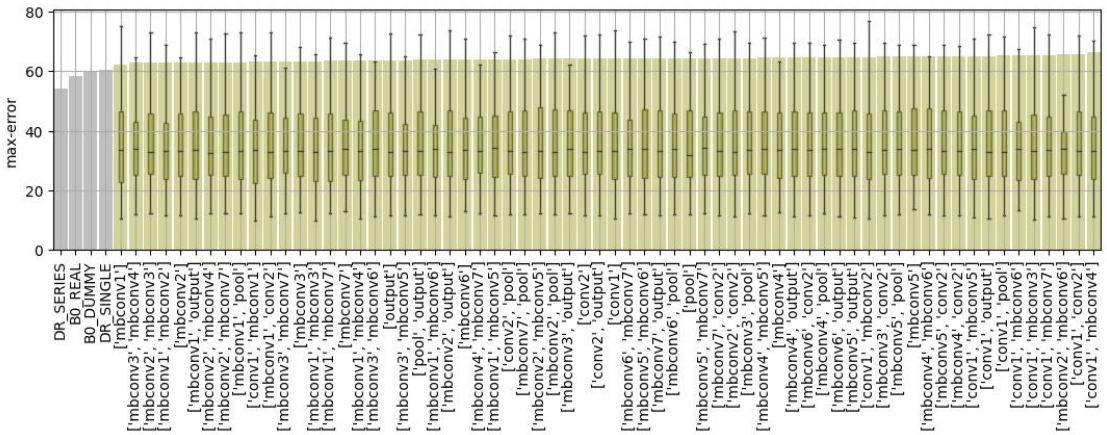
(a) AlexNet



(b) ResNet-18

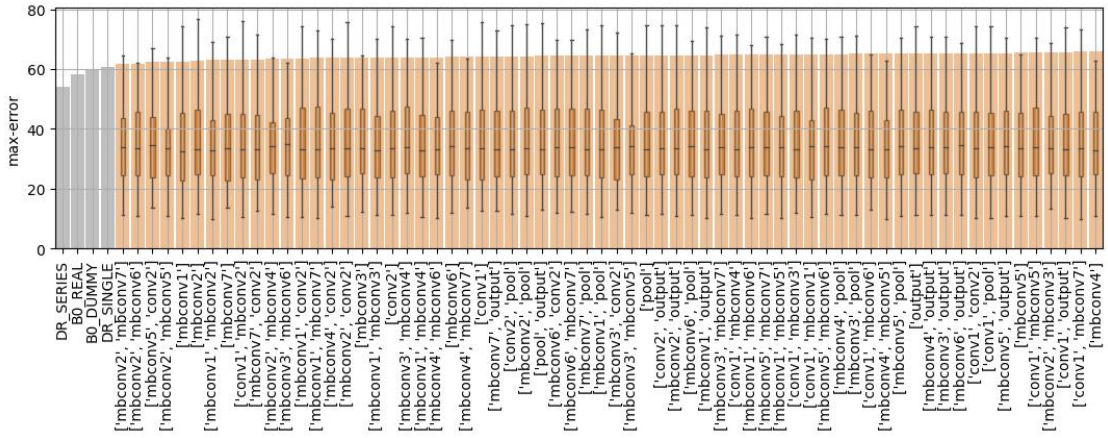


(c) EfficientNet-B0



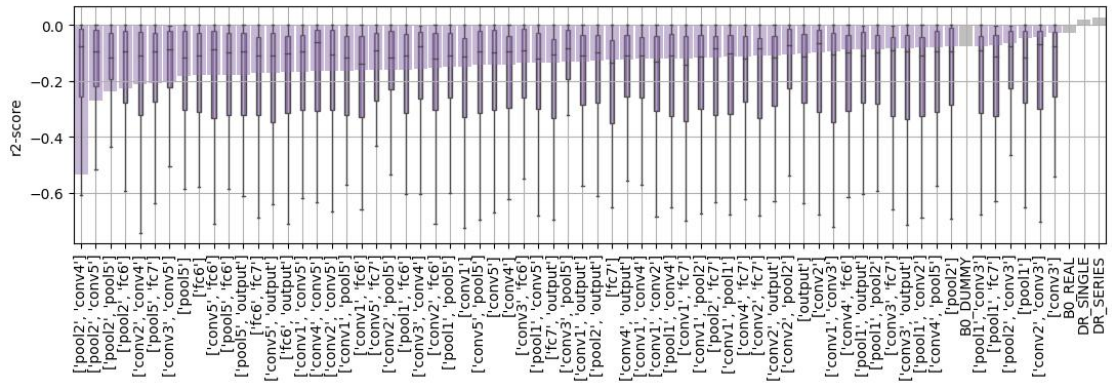
(d) EfficientNet-B4

Figure A.3: Comparison of the performance (in maximum error) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.

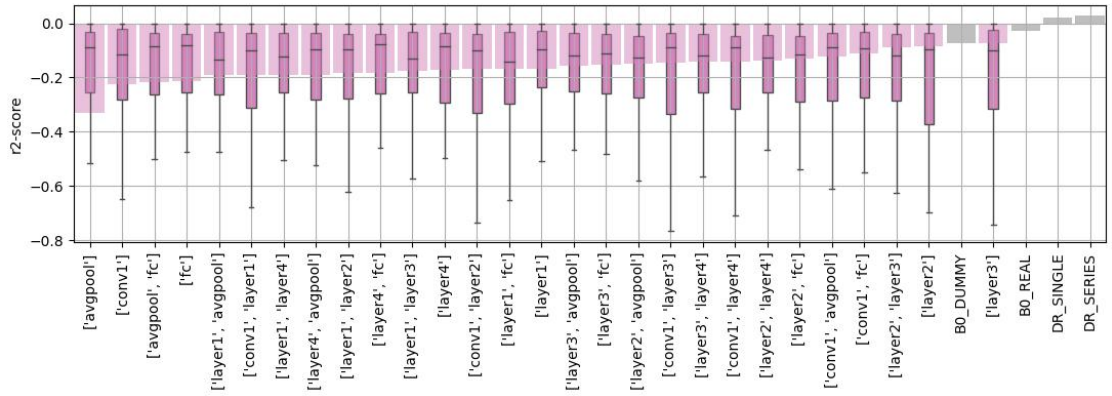


(e) EfficientNet-B7

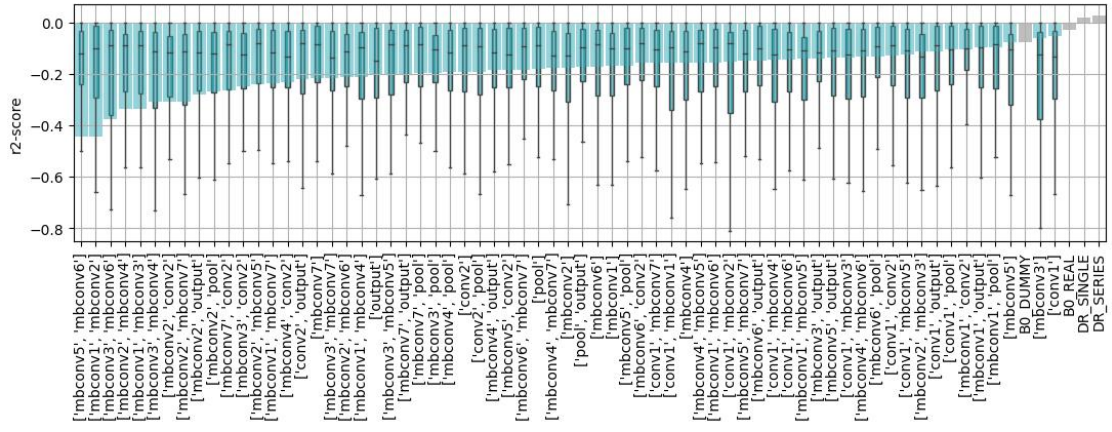
Figure A.3: Comparison of the performance (in maximum error) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.



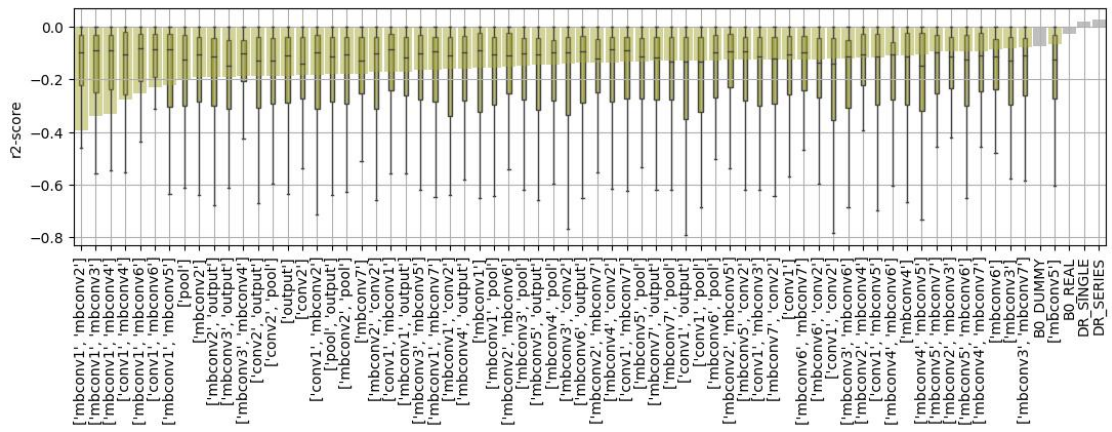
(a) AlexNet



(b) ResNet-18

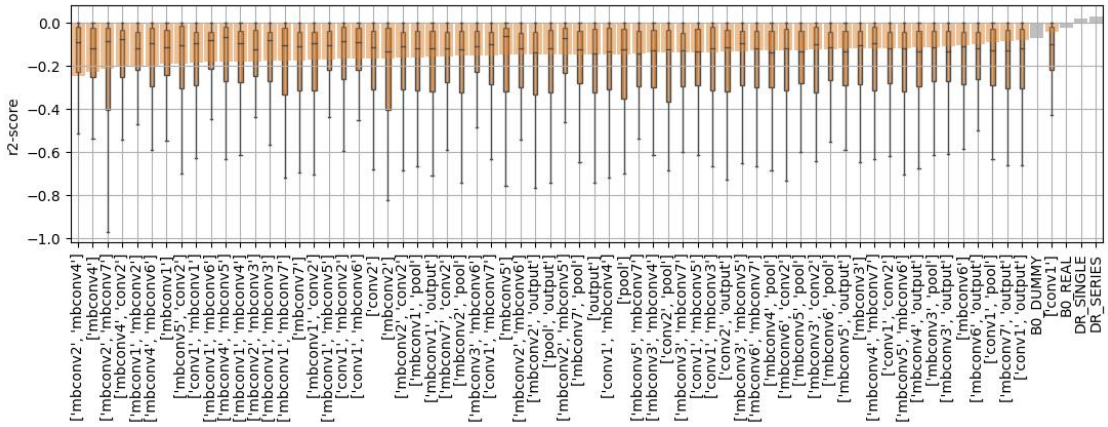


(c) EfficientNet-B0



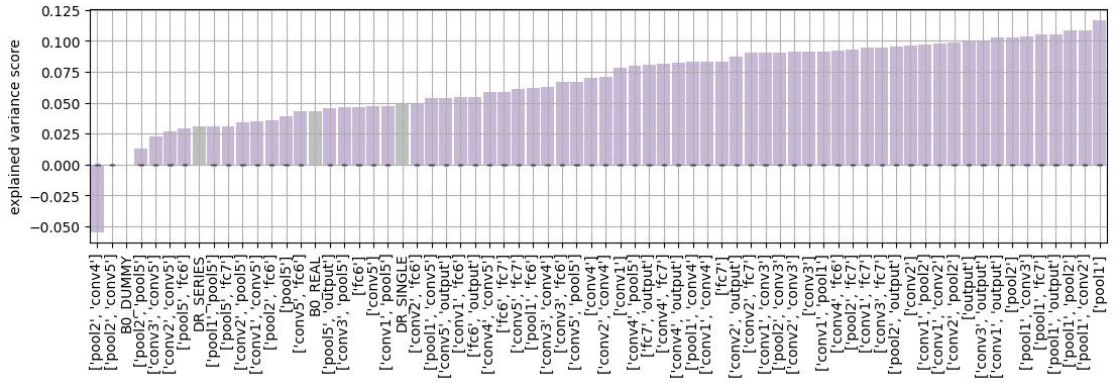
(d) EfficientNet-B4

Figure A.4: Comparison of the performance (in R2-score) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.

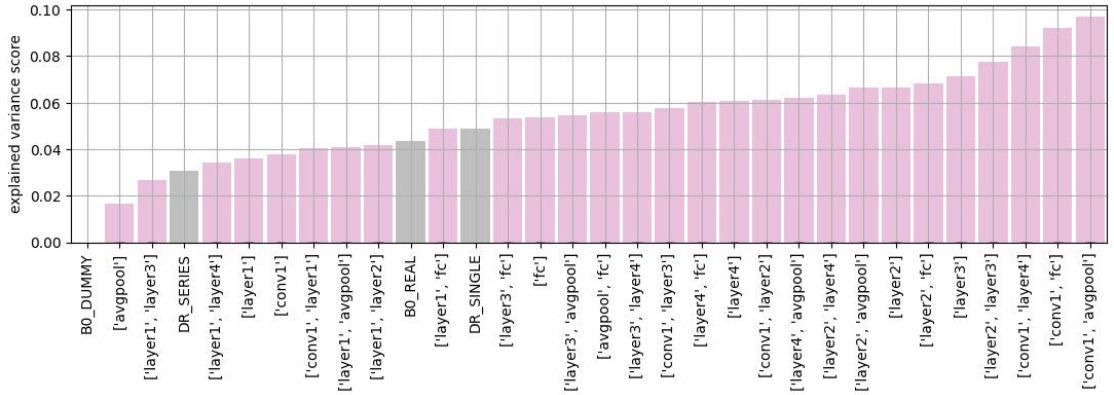


(e) EfficientNet-B7

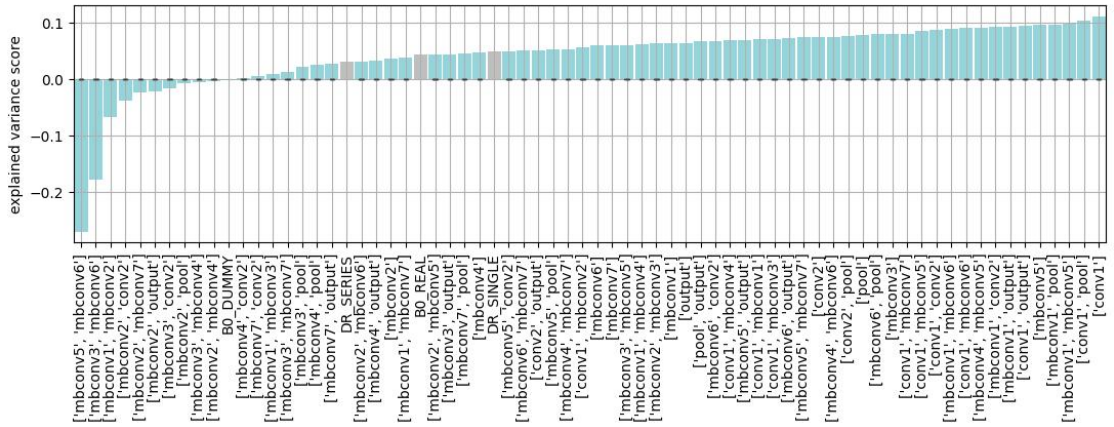
Figure A.4: Comparison of the performance (in R2-score) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.



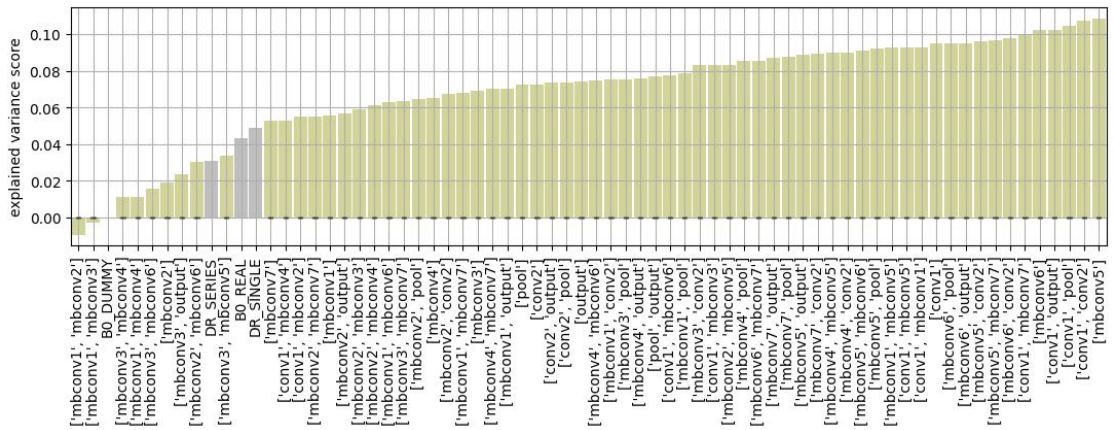
(a) AlexNet



(b) ResNet-18

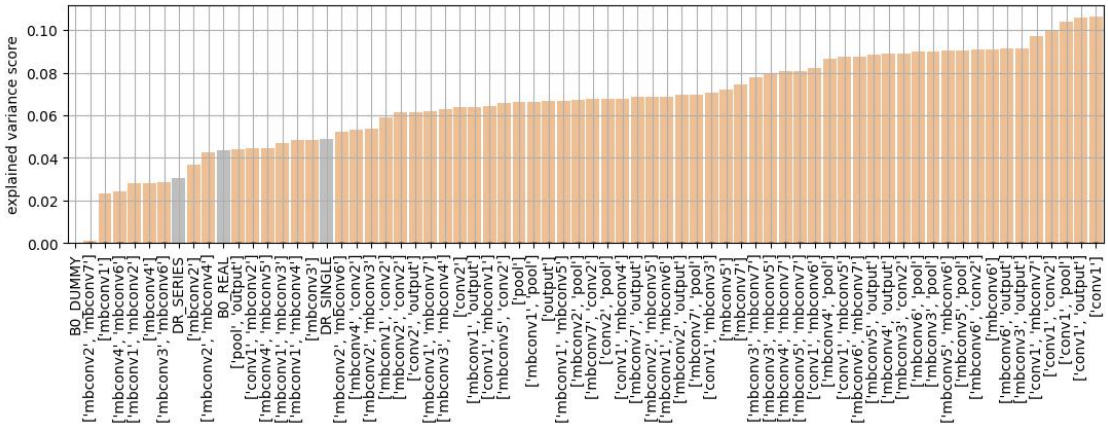


(c) EfficientNet-B0



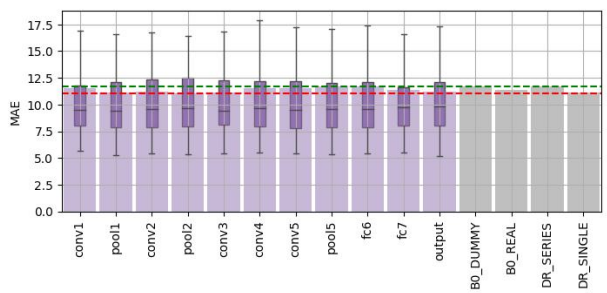
(d) EfficientNet-B4

Figure A.5: Comparison of the performance (in explained variance) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.

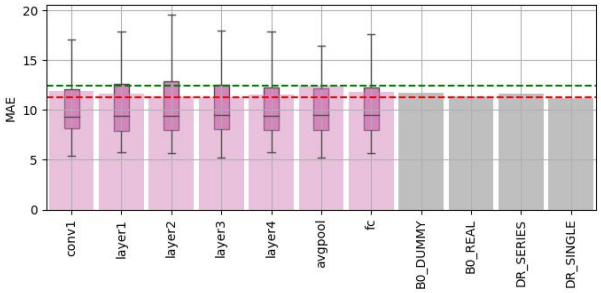


(e) EfficientNet-B7

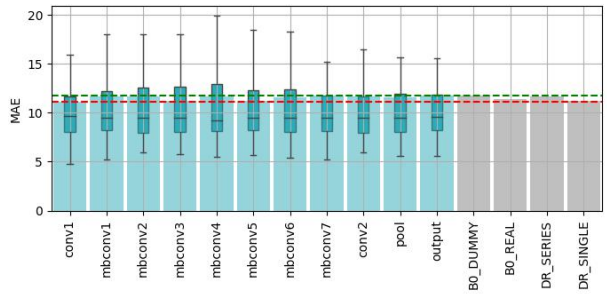
Figure A.5: Comparison of the performance (in explained variance) achieved by single and pairwise layer accumulators as training set using naive regression scheme for all network architectures. Baselines are represented in grey.



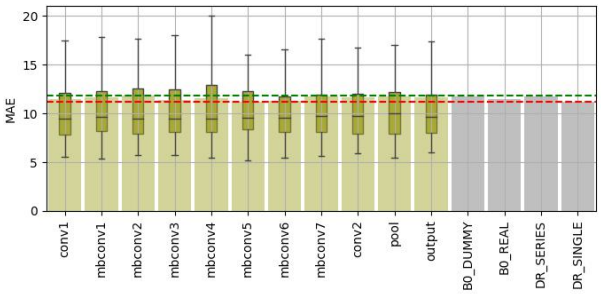
(a) AlexNet



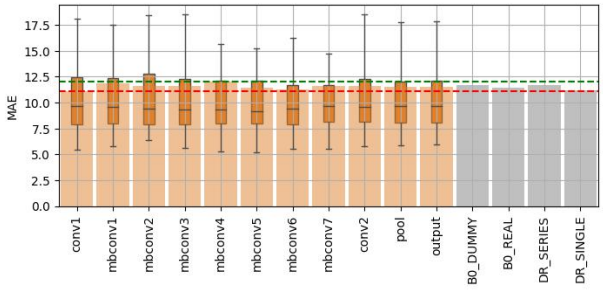
(b) Resnet-18



(c) EfficientNet-B0

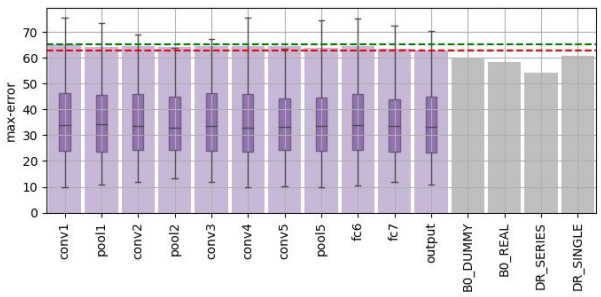


(d) EfficientNet-B4

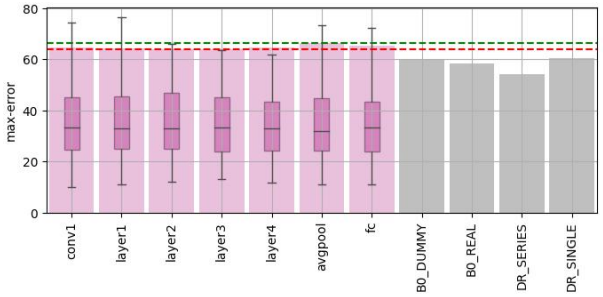


(e) EfficientNet-B7

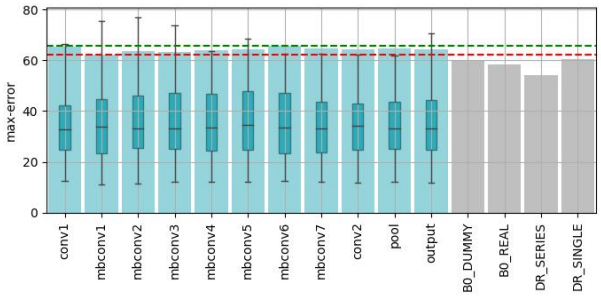
Figure A.6: MAEs for each layer of each network architecture. The grey bars represent the 4 baselines. Green dashed line (–): Maximum value of all candidate layers. Red dashed line (–): Minimum value of all candidate layers (except baselines).



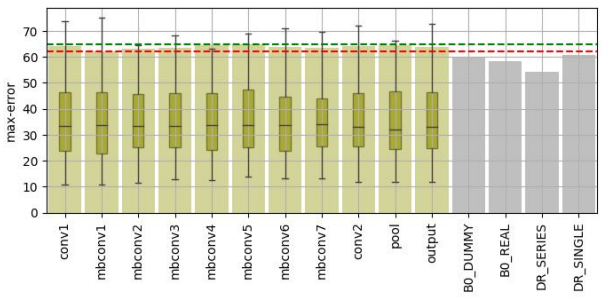
(a) AlexNet



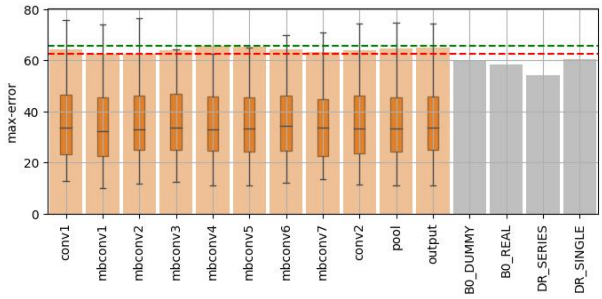
(b) Resnet-18



(c) EfficientNet-B0

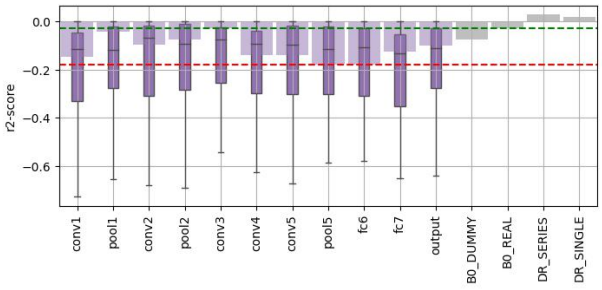


(d) EfficientNet-B4

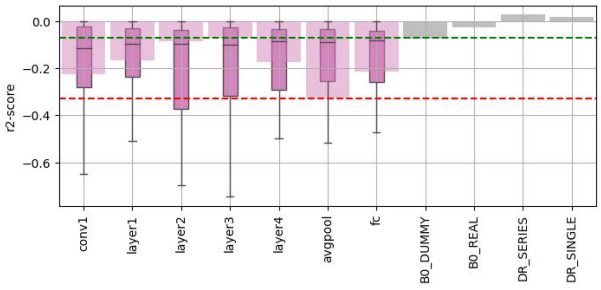


(e) EfficientNet-B7

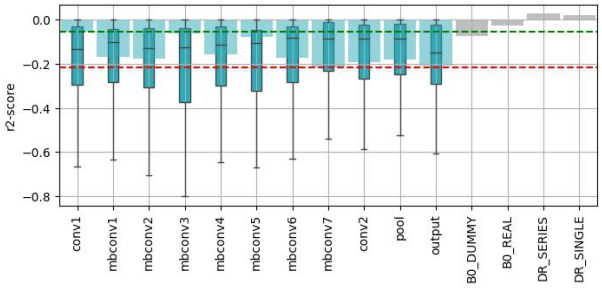
Figure A.7: Maximum errors for each layer of each network architecture. The grey bars represent the 4 baselines. Green dashed line (–): Maximum value of all candidate layers. Red dashed line (–): Minimum value of all candidate layers (except baselines).



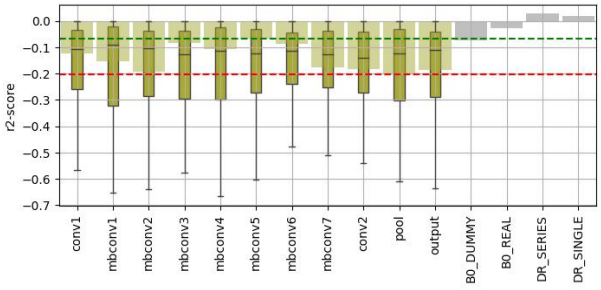
(a) AlexNet



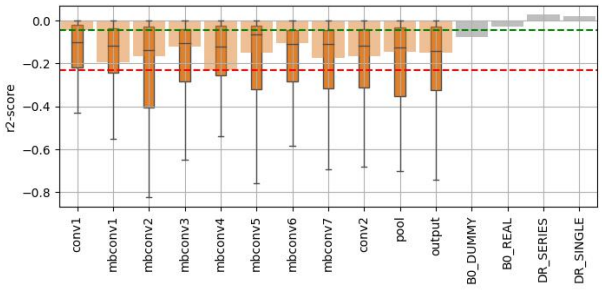
(b) Resnet-18



(c) EfficientNet-B0

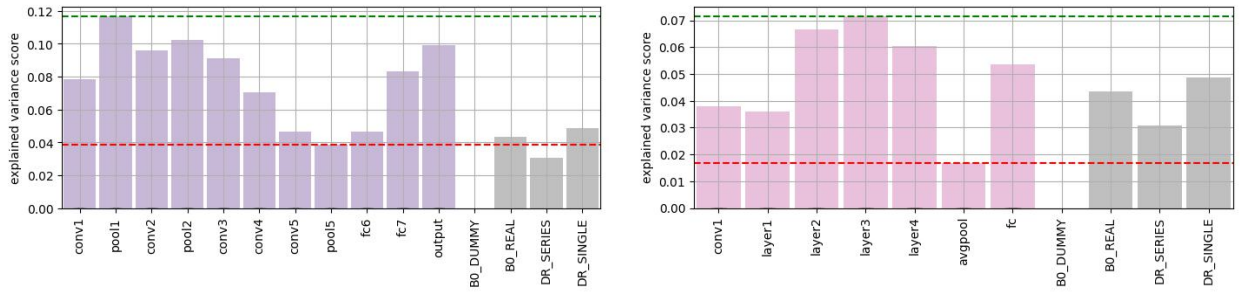


(d) EfficientNet-B4



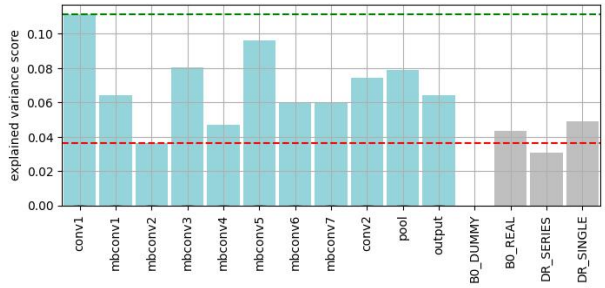
(e) EfficientNet-B7

Figure A.8: R2-scores for each layer of each network architecture. The grey bars represent the 4 baselines. Green dashed line (–): Maximum value of all candidate layers. Red dashed line (–): Minimum value of all candidate layers (except baselines).

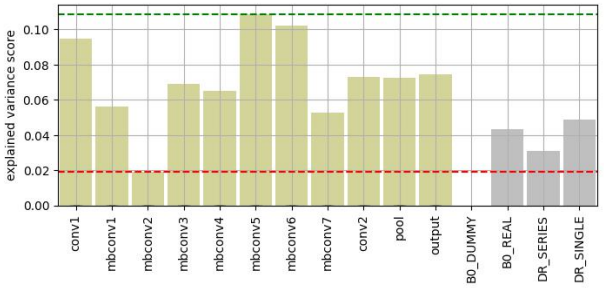


(a) AlexNet

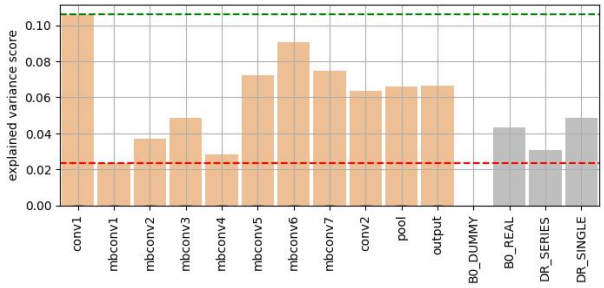
(b) Resnet-18



(c) EfficientNet-B0



(d) EfficientNet-B4



(e) EfficientNet-B7

Figure A.9: Explained variance scores for each layer of each network architecture. The grey bars represent the 4 baselines. Green dashed line (–): Maximum value of all candidate layers. Red dashed line (–): Minimum value of all candidate layers (except baselines).

B

APPENDIX B : SOURCE CODE

Figure B.1: GitHub logo. Home of the code written for this project.

The source code for the implementation of the proposed model and experiments presented in this report is available on GitHub at https://github.com/LuNavUlg/using-deep-neural-networks-to-understand-the-temporal-structure-of-human-memory. The code is written in Python.

BIBLIOGRAPHY

[1] W. Roseboom, Z. Fountas, K. Nikiforou, D. Bhowmik, M. Shanahan and A. K. Seth, 'Activity in perceptual classification networks as a basis for human subjective time perception', *Nature communications*, vol. 10, no. 1, p. 267, 2019.

[2] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks', *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[3] Y. LeCun, Y. Bengio and G. Hinton, 'Deep learning', *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, 'Highly accurate protein structure prediction with alphafold', *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[5] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau and S. Thrun, 'Dermatologist-level classification of skin cancer with deep neural networks', *nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[6] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans *et al.*, 'Photorealistic text-to-image diffusion models with deep language understanding', *Advances in Neural Information Processing Systems*, vol. 35, pp. 36 479–36 494, 2022.

[7] J. McClelland and M. Botvinick, 'Deep learning: Implications for human learning and memory', 2020.

[8] Z. Zhang, 'A computational cognitive model of human memory based on invertible neural networks', 2022.

[9] N. Kriegeskorte, 'Deep neural networks: A new framework for modeling biological vision and brain information processing', *Annual review of vision science*, vol. 1, pp. 417–446, 2015.

[10] A. J. Kell, D. L. Yamins, E. N. Shook, S. V. Norman-Haignere and J. H. McDermott, 'A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy', *Neuron*, vol. 98, no. 3, pp. 630–644, 2018.

[11] Z. Fountas, A. Sylaidi, K. Nikiforou, A. K. Seth, M. Shanahan and W. Roseboom, 'A predictive processing model of episodic memory and time perception', *Neural Computation*, vol. 34, no. 7, pp. 1501–1544, 2022.

[12] Z. Fountas, K. Nikiforou, D. Bhowmik, M. Shanahan, W. Roseboom and A. Seth, 'Clockless biologically-plausible architecture for temporal perception using convolutional neural networks',

[13] C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao, 'Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors', in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7464–7475.

[14] L. Floridi and M. Chiriatti, 'Gpt-3: Its nature, scope, limits, and consequences', *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[15] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805*, 2018.

[16] F. I. Craik and J. M. Jennings, 'Human memory.', 1992.

[17] J. D. Gabrieli, 'Cognitive neuroscience of human memory', *Annual review of psychology*, vol. 49, no. 1, pp. 87–115, 1998.

[18] R. C. Atkinson and R. M. Shiffrin, 'Human memory: A proposed system and its control processes', in *Psychology of learning and motivation*, vol. 2, Elsevier, 1968, pp. 89–195.

[19] M. U. G. Khan, S. Bashir, A. Nasir, A. A. Shah and S. Mehmood, 'Towards computational model of human brain memory', *Journal of Mathematics (ISSN 1016-2526)*, vol. 46, no. 2, pp. 35–45, 2014.

[20] N. Sato and Y. Yamaguchi, 'Simulation of human episodic memory by using a computational model of the hippocampus.', *Advances in Artificial Intelligence (16877470)*, 2010.

[21] A. D'Argembeau, O. Jeunehomme and D. Stawarczyk, 'Slices of the past: How events are temporally compressed in episodic memory', *Memory*, vol. 30, no. 1, pp. 43–48, 2022.

[22] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard and Z. Lin, 'Towards biologically plausible deep learning', *arXiv preprint arXiv:1502.04156*, 2015.

[23] D. L. Yamins and J. J. DiCarlo, 'Using goal-driven deep learning models to understand sensory cortex', *Nature neuroscience*, vol. 19, no. 3, pp. 356–365, 2016.

[24] S. Ghosh-Dastidar and H. Adeli, 'Spiking neural networks', *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[25] W. Maass, 'Liquid state machines: Motivation, theory, and applications', *Computability in context: computation and logic in the real world*, pp. 275–296, 2011.

[26] H. Jaeger, 'Echo state network', *scholarpedia*, vol. 2, no. 9, p. 2330, 2007.

[27] C. Baldassano, J. Chen, A. Zadbood, J. W. Pillow, U. Hasson and K. A. Norman, 'Discovering event structure in continuous narrative perception and memory', *Neuron*, vol. 95, no. 3, pp. 709–721, 2017.

[28] C. J. Bates and R. A. Jacobs, 'Efficient data compression in perception and perceptual memory.', *Psychological review*, vol. 127, no. 5, p. 891, 2020.

[29] K. Bonasia, J. Blommesteyn and M. Moscovitch, 'Memory and navigation: Compression of space varies with route length and turns', *Hippocampus*, vol. 26, no. 1, pp. 9–12, 2016.

[30] M. A. Conway, 'Episodic memories', *Neuropsychologia*, vol. 47, no. 11, pp. 2305–2313, 2009.

[31] M. Faber and S. P. Gennari, 'In search of lost time: Reconstructing the unfolding of events from memory', *Cognition*, vol. 143, pp. 193–202, 2015.

[32] O. Jeunehomme and A. D'Argembeau, 'The time to remember: Temporal compression and duration judgements in memory for real-life events', *Quarterly Journal of Experimental Psychology*, vol. 72, no. 4, pp. 930–942, 2019.

[33] O. Jeunehomme and A. D'Argembeau, 'Event segmentation and the temporal compression of experience in episodic memory', *Psychological research*, vol. 84, pp. 481–490, 2020.

[34] O. Jeunehomme, A. Folville, D. Stawarczyk, M. Van der Linden and A. D'Argembeau, 'Temporal compression in episodic memory for real-life events', *Memory*, vol. 26, no. 6, pp. 759–770, 2018.

[35] D. Marković, *Spiking neural networks. what makes these biologically realistic... | by danijela marković | towards data science*, https://towardsdatascience.com/spiking-neural-networks-558dc4479903, (Accessed on 05/24/2023), Feb. 2020.

[36] J. L. Lobo, *Spiking neural networks in stream learning | towards data science*, https://towardsdatascience.com/spiking-neural-networks-in-stream-learning-scenarios-7bf2112b0c35, (Accessed on 05/24/2023), 2020 January.

[37] C. Chiplunkar, *An introduction to spiking neural networks (part 1) | by chinmay chiplunkar | analytics vidhya | medium*, https://medium.com/analytics-vidhya/an-introduction-to-spiking-neural-networks-part-1-the-neurons-5261dd9358cd, (Accessed on 05/15/2023), Jul. 2019.

[38] *Neuron - wikipedia*, https://en.wikipedia.org/wiki/Neuron, (Accessed on 05/24/2023).

[39] *Depolarization - wikipedia*, https://en.wikipedia.org/wiki/Depolarization, (Accessed on 05/24/2023).

[40] P. G. Louppe, *Lecture 3: Automatic differentiation*, https://glouppe.github.io/info8010-deep-learning/pdf/lec3.pdf, 2022.

[41] A. J. Sanfiz and M. Akrout, *Benchmarking the accuracy and robustness of feedback alignment algorithms*, 2021. arXiv: 2108.13446 [cs.LG].

[42] T. P. Lillicrap, D. Cownden, D. B. Tweed and C. J. Akerman, 'Random feedback weights support learning in deep neural networks', *arXiv preprint arXiv:1411.0247*, 2014.

[43] T. P. Lillicrap, D. Cownden, D. B. Tweed and C. J. Akerman, 'Random synaptic feedback weights support error backpropagation for deep learning', *Nature communications*, vol. 7, no. 1, p. 13 276, 2016.

[44] A. Nøkland, 'Direct feedback alignment provides learning in deep neural networks', *Advances in neural information processing systems*, vol. 29, 2016.

[45] A. Jimenez, *Feedback alignment methods. a biologically-motivated alternative to... | by albert jimenez | towards data science*, https://towardsdatascience.com/feedback-alignment-methods-7e6c41446e36, (Accessed on 05/26/2023), Sep. 2021.

[46] S. Dobilas, *Support vector regression (svr) — one of the most flexible yet robust prediction algorithms*, https://towardsdatascience.com/support-vector-regression-svr-one-of-the-most-flexible-yet-robust-prediction-algorithms-4d25fbdaca60, (Accessed on 05/17/2023), Dec. 2020.

[47] L. W. Pierre Geurts, *Support vector machines and kernel-based methods*, https://people.montefiore.uliege.be/lwh/AIA/IML___SVM_and_kernels.pdf, 2020.

[48] L. W. Pierre Geurts, *Linear regression*, https://people.montefiore.uliege.be/lwh/AIA/IML___Linear_regression.pdf, 2021.

[49] S. J. Prince, *Understanding Deep Learning*. MIT Press, 2023. [Online]. Available: https://udlbook.github.io/udlbook/.

[50] G. Louppe, *Lecture 5: Convolutional networks*, https://glouppe.github.io/info8010-deep-learning/pdf/lec5.pdf, Accessed: 2023–04-29, 2023.

[51] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[52] M. Tan and Q. Le, 'Efficientnet: Rethinking model scaling for convolutional neural networks', in *International conference on machine learning*, PMLR, 2019, pp. 6105–6114.

[53] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, 'Mobilenetv2: Inverted residuals and linear bottlenecks', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[54] S. Chen, K. Yang and R. Stiefelhagen, 'Dr-tanet: Dynamic receptive temporal attention network for street scene change detection', in *2021 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2021, pp. 502–509.

[55] K. Sakurada, M. Shibuya and W. Wang, 'Weakly supervised silhouette-based semantic scene change detection', in *2020 IEEE International conference on robotics and automation (ICRA)*, IEEE, 2020, pp. 6861–6867.

[56] E. Guo, X. Fu, J. Zhu, M. Deng, Y. Liu, Q. Zhu and H. Li, 'Learning to measure change: Fully convolutional siamese metric networks for scene change detection', *arXiv preprint arXiv:1810.09111*, 2018.

[57] A. Varghese, J. Gubbi, A. Ramaswamy and P. Balamuralidhar, 'Changenet: A deep learning architecture for visual change detection', in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[58] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell, 'Caffe: Convolutional architecture for fast feature embedding', *arXiv preprint arXiv:1408.5093*, 2014.

[59] F. Pedregosa *et al.*, 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[60] A. S. Bangert, C. A. Kurby and J. M. Zacks, 'The influence of everyday events on prospective timing "in the moment"', *Psychonomic bulletin & review*, vol. 26, pp. 677–684, 2019.

[61] M. L. Eisenberg and J. M. Zacks, 'Ambient and focal visual processing of naturalistic activity', *Journal of Vision*, vol. 16, no. 2, pp. 5–5, 2016.

[62] C. A. Kurby and J. M. Zacks, 'Age differences in the perception of hierarchical structure in events', *Memory & cognition*, vol. 39, pp. 75–91, 2011.

[63] J. Q. Sargent, J. M. Zacks, D. Z. Hambrick, R. T. Zacks, C. A. Kurby, H. R. Bailey, M. L. Eisenberg and T. M. Beck, 'Event segmentation ability uniquely predicts event memory', *Cognition*, vol. 129, no. 2, pp. 241–255, 2013.

[64] J. M. Zacks, C. A. Kurby, M. L. Eisenberg and N. Haroutunian, 'Prediction error associated with the perceptual segmentation of naturalistic events', *Journal of cognitive neuroscience*, vol. 23, no. 12, pp. 4057–4066, 2011.

[65] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, 'Pytorch: An imperative style, high-performance deep learning library', *Advances in neural information processing systems*, vol. 32, 2019.

[66] F. Ramzan, M. U. G. Khan, A. Rehmat, S. Iqbal, T. Saba, A. Rehman and Z. Mehmood, 'A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using resting-state fmri and residual neural networks', *Journal of medical systems*, vol. 44, pp. 1–16, 2020.