Predicting Retinal Ganglion Cell Responses Based On Visual Features Via Graph Filters

Yasaman Parhizkar

A Thesis Submitted to the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science

Graduate Program in Electrical Engineering and Computer Science

York University
Toronto, Ontario

August 31, 2023

Abstract

This thesis presents a novel graph-based approach to classify video clips with binary labels. Each video clip is described by a feature vector instead of raw pixel values.

At the model's core, a similarity graph is defined where each node is associated with a feature vector and its corresponding label. The weight of an edge connecting two nodes delineates the similarity of the nodes' feature vectors which is computed via the Mahalanobis distance and its metric matrix. The metric matrix is learned using labeled training data. Unknown labels are then estimated using the optimized metric and the similarity graph.

The main advantage of our model is enabling interpretations of how the predictions are made. Nevertheless, the model achieves competitive accuracy with state-of-the-art approaches as well. We apply this model to a retinal coding problem where explainability is essential to gain conceptual insight about the retina.

Contents

List of Algorithms

List of Figures

1 Introduction and Literature Review

This thesis brings three different subjects together: graph signal processing, retinal visual coding, and AI explainability. Graph signal processing combines spectral graph theory and discrete signal processing to analyze signals that arise from an underlying network structure. Retinal visual encoding happens when an image received by the vertebrate eye gets translated into electrical pulses carrying visual information to the brain. Explainability means the ability to explain, in a humanly understandable way, how a mathematical model captures the dynamics of a physical or computational system. More thorough introduction of these subjects is presented in this chapter.

1.1 Graph Signal Processing

1.1.1 Introduction

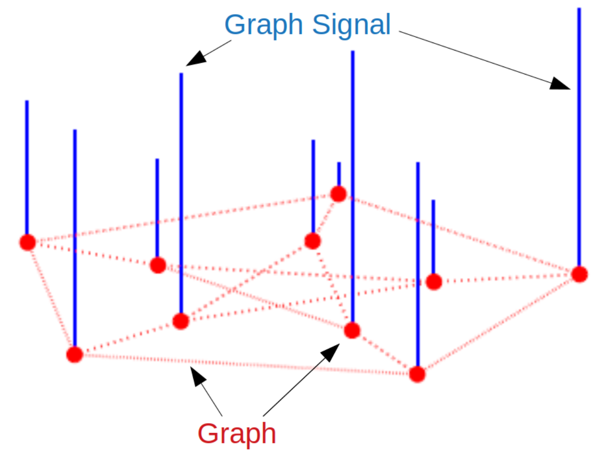
A network is a set of objects (called nodes or vertices) that are connected together. The connections between the nodes are called edges or links. Many signals in the real world appear on a network. These signals are generated by individual components which interact with each other in some way. A few examples of these networks include the Internet, sensor networks, social networks and neural networks of the brain. To illustrate, the Internet is a collection of individual computers which are connected to each other via cable or electromagnetic waves (i.e. links). Each computer can be represented by a node and each link by an edge. Edges can be weighted where the weights show bandwidths of the links.

Signals appear on networks as a set of numbers associated to the nodes of the network. For example, the CPU power of each computer in the Internet can be a signal. Other examples can be waiting times for packets to be processed in each node, or number of packets that are delivered to each node during a specific time interval.

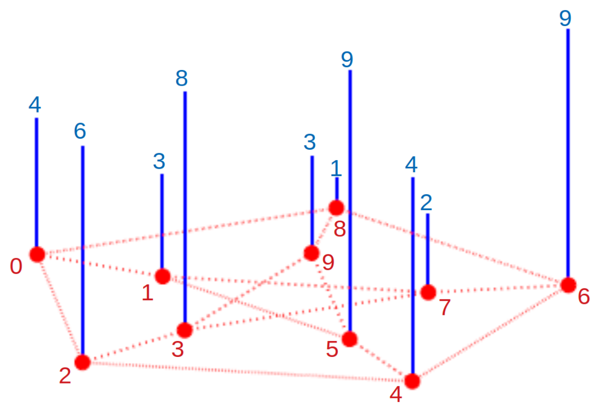
Graphs are mathematical tools to display and analyze networks. Graph signals (i.e. a set of numbers associated to the nodes of the graph) inherit important information from their underlying structure which need to be considered in their analysis. Graph signal processing (GSP) combines graph spectral theory and digital signal processing to address this need. Common GSP tools include filtering, denoising, inpainting and compressing graph signals [4].

In the following, we explain a few basic concepts of GSP such as graph filtering and Fourier transform. First of all, please note the distinction between the graph and the graph signal. The signal refers to the set of numbers that are associated with nodes in a graph, while the graph itself primarily shows the interactions between the nodes and secondarily the nodes themselves. Figure 1a delineates the distinction. This figure shows an undirected graph. Undirected graph signals are more widely studied in the field. However, undirected graphs are not mandatory; a graph signal can generally exist on a directed graph as well.

We use linear algebra to algebraically represent and manipulate graph signals. To do this, first we need to



(a) A graph is a set of nodes and edges. A graph *signal* is a set of samples (i.e. numbers) residing on the nodes of a specific graph.



(b) To algebraically represent a graph signal, we need to index the nodes. Any arbitrary indexing method is acceptable.

Figure 1: A graph signal and its underlying graph

assign a number to all nodes. These numbers identify the nodes in the algebraic representation. Any arbitrary numbering method would work. To illustrate, see Figure 1b; it shows how numbers 0 to 9 are assigned to the nodes of the graph. Now that each node has an identifying index, we can represent the graph signal with a vector as in Equation (1). Black numbers in this equation show graph signal samples and red numbers show the identifying indices.

$$\boldsymbol{x} = \begin{array}{c} [\; 4 \quad 3 \quad 6 \quad 8 \quad 4 \quad 9 \quad 9 \quad 2 \quad 1 \quad 3 \;]^T \\ \;\; 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \end{array} \tag{1}$$

However, this vector is not enough to uniquely define a graph signal. We also need a matrix showing the edges and their weights. This matrix can be the graph's adjacency matrix or the graph Laplacian matrix. Equation (2) notes the adjacency matrix and Equation (3) notes the Laplacian matrix for the graph in Figure 1. Assume all edges in the graph have weight 1.

$$\boldsymbol{W} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

$$\begin{aligned} &\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} = diag(\boldsymbol{W}\mathbf{1}) - \boldsymbol{W} \\ &\boldsymbol{L} = \begin{bmatrix} 3 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ -1 & 3 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ -1 & 0 & 3 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 0 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 3 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 3 & -1 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 3 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & -1 & 3 \end{bmatrix} \end{aligned} \tag{3}$$

Either of the pairs $(\boldsymbol{x}, \boldsymbol{W})$ or $(\boldsymbol{x}, \boldsymbol{L})$ can uniquely define the graph signal. Here we use $(\boldsymbol{x}, \boldsymbol{L})$ for reasons that will be explained by the end of this section.

If $\boldsymbol{L}$ is diagonalizable with real entries, it means we can decompose it as

$$\begin{aligned} &\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T \\ &\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}, \end{aligned} \tag{4}$$

where $\boldsymbol{U} \in \mathbb{R}^N$ has eigenvectors of $\boldsymbol{L}$ as its columns, and $\boldsymbol{\Lambda}$ is a diagonal matrix with the corresponding eigenvalues on its diagonal,

$$\begin{aligned} &\boldsymbol{U} = [\boldsymbol{v_1}, ..., \boldsymbol{v_N}] \\ &\boldsymbol{\Lambda} = diag(\lambda_1, ..., \lambda_N). \end{aligned} \tag{5}$$

Borrowing from spectral graph theory, eigenvectors of the Laplacian matrix can be defined as frequency components while their corresponding eigenvalues are their frequencies. Consequently, matrix $\boldsymbol{U}^T$ in this equation can be defined as the graph Fourier transform (GFT). By this definition, the spectral representation of a graph signal is obtained by

$$\boldsymbol{x_f} = \boldsymbol{U}^T \boldsymbol{x} \, . \tag{6}$$

Conversely, we can get back the nodal representation by multiplying the inverse GFT matrix ($\triangleq \boldsymbol{U}$) by the spectral representation,

$$\boldsymbol{x} = \boldsymbol{U} \, \boldsymbol{x_f} . \tag{7}$$

It is possible to define the graph Fourier transform for defective (non-diagonalizable) matrices as well; however, it would lead to some numerical difficulties which we rather avoid. For this reason, we only consider undirected graphs with real edge weights. These graphs have real and symmetric Laplacian matrices, and any real and symmetric matrix is *guaranteed* to be diagonalizable [5].

Furthermore, assuming that all edges have non-negative weights, we can prove that the graph Laplacian is always positive semi-definite by using the Gershgorin Circle Theorem [6]. A real symmetric matrix is positive semi-definite (PSD) if and only if all of its eigenvalues are non-negative [7]. Thus, if we use the graph Laplacian to define GFT, all graph frequencies (i.e. eigenvalues of $\boldsymbol{L}$) would be non-negative. This property makes the concept of graph frequency more accessible. This is the reason why we chose $(\boldsymbol{x}, \boldsymbol{L})$ to represent our graph signal instead of $(\boldsymbol{x}, \boldsymbol{W})$.

Since we chose $(\boldsymbol{x}, \boldsymbol{L})$ to represent graph signals, a graph filter is defined as any matrix that can be written as a linear combination of powers of the Laplacian matrix. The following equation shows this definition,

$$\boldsymbol{H} = \sum_{k=0}^{N-1} H_k \, \boldsymbol{L}^k \, . \tag{8}$$

Filtering a graph signal means multiplying a filter by the signal,

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} \, . \tag{9}$$

If we write the eigen decomposition of $\boldsymbol{H}$ in Equation (9), it shows the process of taking Fourier transform of our signal, filtering it in the spectral domain and taking inverse Fourier transform of the result. The output is the same regardless of filtering the signal in the nodal or spectral domain

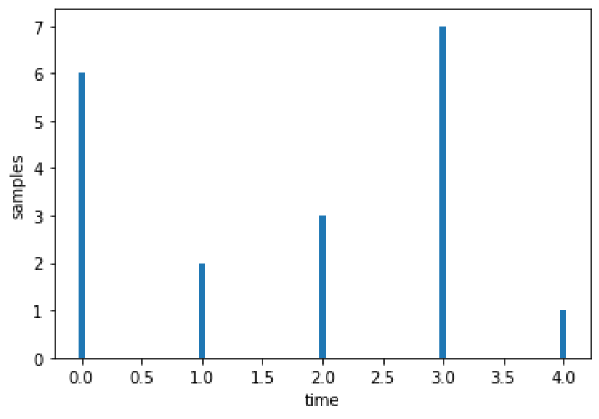
$$\begin{aligned}
\text{(arbitrary filter) } \boldsymbol{H} = \sum_{k=0}^{N-1} H_k\, \boldsymbol{L}^k \,, \\
\boldsymbol{y} &= \boldsymbol{H}\,\boldsymbol{x} \\
&= \boldsymbol{U}\,\boldsymbol{\Lambda_H}\,\boldsymbol{U}^T \boldsymbol{x} \ \text{ (eigen-decomposition)} \\
&= \boldsymbol{U}\,\left(\sum_{k=0}^{N-1} H_k \boldsymbol{\Lambda}^k\right) \boldsymbol{U}^T\,\boldsymbol{x} \\
&= \boldsymbol{U}\,\left(\sum_{k=0}^{N-1} H_k \boldsymbol{\Lambda}^k\right) \boldsymbol{x_f} \ \text{ (Fourier transform)} \\
&= \boldsymbol{U}\, diag\left(\sum_{k=0}^{N-1} H_k \lambda_1^k, \ldots, \sum_{k=0}^{N-1} H_k \lambda_N^k\right) \boldsymbol{x_f} \\
&= \boldsymbol{U}\,\boldsymbol{y_f} \ \text{ (filtering in the spectral domain)} \\
&= \boldsymbol{y} \ \text{ (inverse GFT)}\,.
\end{aligned} \tag{10}$$

In the above equation, $\boldsymbol{x}$ is our input signal which is filtered by $\boldsymbol{H}$ to yield output signal $\boldsymbol{y}$. $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ are the same matrices derived in Equation (4). $\boldsymbol{\Lambda_H}$ is the diagonal representation of filter $\boldsymbol{H}$. $\boldsymbol{\Lambda_H}$ is a polynomial of $\boldsymbol{\Lambda}$.

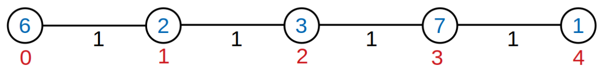
Multiplying $\boldsymbol{U}^T$ by $\boldsymbol{x}$ gives us the spectral representation of the input signal $\boldsymbol{x_f}$. Then, the spectral representation of the filter (i.e. $\boldsymbol{\Lambda_H}$) is multiplied by $\boldsymbol{x_f}$ which is the definition of filtering $\boldsymbol{x_f}$ by $\boldsymbol{\Lambda_H}$. This multiplication is essentially just N point-wise multiplications since $\boldsymbol{\Lambda_H}$ is diagonal. The multiplication's result is our output in spectral domain $\boldsymbol{y_f}$. To convert the output back into the nodal domain, we take inverse GFT of $\boldsymbol{y_f}$ which yields $\boldsymbol{y}$. It is witnessed that the result is the same result obtained by filtering in the nodal domain.

To gain more intuition, consider a line graph (Figure 2). Signals on this graph are equivalent to traditional time signals. The graph Laplacian is equivalent to the second derivative operator with zero-slope boundary condition (Equation (11)). Moreover, GFT is equivalent to the discrete cosine transform (DCT). Equation (12) shows the GFT matrix of this line graph; rows of this matrix are eigenvectors of the graph Laplacian. Figure 3 shows rows of the GFT matrix in Equation (12). It is observed that these frequency components have roughly sinusoidal shapes. If we increase the number of nodes in the graph, the shapes of the frequency components become smoother.

$$\boldsymbol{W} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \Longrightarrow \boldsymbol{L} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \tag{11}$$



(a) A time signal of length N = 5



(b) A graph signal with an underlying line graph of N = 5 nodes.

Figure 2: A graph signal on a line graph can be seen as a time signal.

$$\boldsymbol{U}^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -0.382 & -1.236 & -0.382 & 1 \\ -1 & 1.618 & 0 & -1.618 & 1 \\ -1 & -0.618 & 0 & 0.618 & 1 \\ 1 & -2.618 & 3.236 & -2.618 & 1 \end{bmatrix} \tag{12}$$

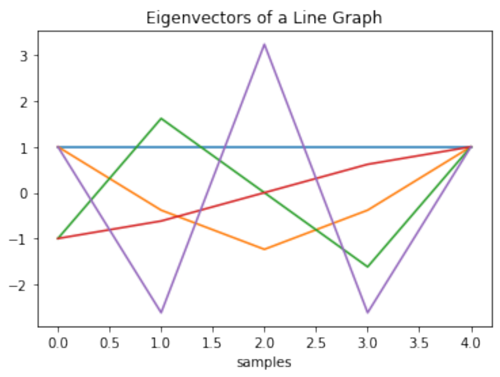


Figure 3: Frequency components of the line graph in Figure 2b

Lastly, we need to talk about the concept of smoothness for graph signals. Smoothness is a general concept which can take on various exact mathematical definitions in different contexts. For example, one can talk about signal smoothness in the vertex domain which would generally mean that samples that are *close* to each other on the graph tend to have similar values. The *closeness* can be defined as the number of hops between two nodes, the shortest path connecting them, etc. Similarly, one can speak of smoothness in the spectral domain, otherwise known as bandlimitedness. In this case, a smooth graph signal's spectral representation does not include a sharp cutoff. Instead, it slowly decays according to a specific formula [8].

For example, Figure 4 shows a piece-wise smooth approximation of an irregular nonsmooth graph signal. This figure is drawn from [8]. The image on the left shows the number of taxi pickups in Manhattan from 6 p.m. to 7 p.m. on January 4, 2015, projected to the nearest intersection. By analyzing this signal, it is observed that intersections in a neighbourhood show similar activity patterns, while the pattern might change between the neighbourhoods; for example, a neighbourhood which is full of office buildings behaves differently than a mostly residential neighbourhood. A piece-wise smooth signal can capture both the large differences between the neighbourhoods and small differences inside a neighbourhood. In this thesis, we use a quantity called Graph Laplacian Regularizer (GLR) to measure smoothness. More explanation about this quantity is given in Section 2.2.3.

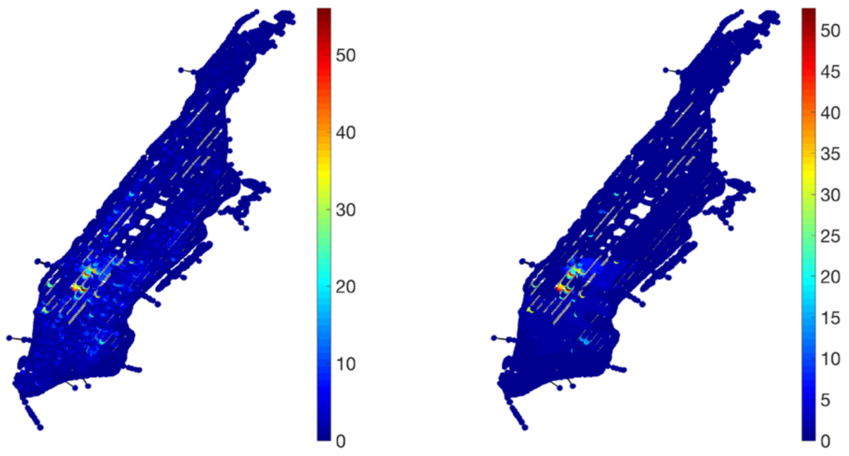


Figure 4: (Left) An irregular nonsmooth graph signal. (Right) Piece-wise smooth approximation of the signal.

1.1.2 Literature Review

We review applications of GSP in two relevant areas to this thesis: image processing and analysis of the brain's structure and function.

A) Analyzing Brain's Structure and Function

Diffusion MRI is a modern brain imaging technique which captures the neuron fiber tracts in the brain's white matter. Another type of MRI called Functional MRI (or fMRI) reveals activity levels of various regions of the brain while performing a specific task.

Correspondingly, two common networks are used in analyzing brain-related signals with GSP. One network represents different regions of the brain with nodes and uses edges to indicate their physical connection through neuron fiber tracts. Higher edge weights in this network suggest higher estimated number of neuron tracts between two regions. The other network's nodes represent brain regions as well. However, here the edges capture functional correlation between regions. Higher edge weights suggest higher probability of simultaneous activity between two regions.

One useful graph signal presents each region's activity measurement (obtained by fMRI) on the nodes of a structural graph. Low-frequency components in such a network show strong correlated activity between physically connected regions. In contrast, high-frequency components show tendency of non-connected regions to activate simultaneously. Analyzing this type of graph signal can reveal interesting brain behaviors. For example, the amount of high frequency energy of a given brain signal is correlated with the performed task's difficulty [9, 10].

GSP has been used in many other neurological applications as well. For example, some have used graph based dimensionality reduction on fMRI images and other brain signals [11, 12]. Another work finds that people's familiarity with a simple motor task affects the frequency content of their brain signals while trying to master said task. This work then proceeds to recognize the most important frequency contents at different familiarity levels [13]. One work analyzes connectivity patterns of brains infected with the Alzheimer's disease [14]. Finally, some works build upon graph spectral transforms to incorporate several biologically known connections in the brain and deliver meaningful brain signal decompositions [15, 16].

B) Image Processing

In image processing applications, the graph's nodes designate pixels of an image while edges capture pixel position or similarity. Samples on nodes contain intensity or color information of the corresponding pixel. Common image processing tools in GSP include filtering, compression, restoration and segmentation.

If the graph is undirected and all edge weights are equal to 1, and assuming the graph structure is regular such as a 1D line graph or a 2D grid, the graph Fourier transform will be equivalent to the DCT transform [17]. However, GFT is more flexible and adapts better to application-specific conditions if the underlying graph is constructed insightfully such as the one suggested in [18]. The underlying graph can be learned from data. For instance, a statistical method is as follows. First, one breaks an image into small 8×8 blocks and reshapes them into 64-dimensional vectors. Then, one computes the covariance matrix of these vectors. The graph Laplacian is approximately equal to the inverse of this covariance matrix. By estimating the graph Laplacian, an underlying graph is inferred whose edges show the conditional independence of the connected nodes [19].

Another graph learning method is similarity-based. This method constructs an $N \times N$ distance matrix containing the distances between all pairs of nodes. Then, considers that distance matrix to be the adjacency matrix of the underlying graph. The main point in this method is defining a measure of distance between nodes. One common definition combines the spatial and intensity differences between nodes as follows,

$$
\begin{aligned}
&\boldsymbol{W} \in \mathbb{R}^{N} , \\
&w_{i,j} = \exp\left(-\frac{d_s(i,j)^2}{2\sigma_s^2}\right)\exp\left(-\frac{d_p(i,j)^2}{2\sigma_p^2}\right) , \\
&d_s(i,j) = \sqrt{(i_1-j_1)^2+(i_2-j_2)^2} , \\
&d_p(i,j) = |x(i_1,i_2)-x(j_1,j_2)| .
\end{aligned} \tag{13}
$$

The two pixels are $i=(i_1,i_2)$ and $j=(j_1,j_2)$. The pixels' intensities are depicted by $x(i_1,i_2)$ and $x(j_1,j_2)$. The spatial distance between the nodes is noted by d_s while their intensity difference is noted by d_p. The (i,j)th entry of the adjacency matrix $\boldsymbol{W}$ incorporates both distances [20].

Note that matrix $\boldsymbol{W}$ in Equation (13) practically contains a bilateral filter's weights. Assume $\boldsymbol{x}$ contains intensity values from a block of an image. A bilateral filter applied to $\boldsymbol{x}$ will output the signal $\hat{\boldsymbol{x}}$ as

$$
\hat{x}(i_1,i_2) = \frac{1}{\sum_{j\in N_i} w_{i,j}} \sum_{j\in N_i} w_{i,j} x_j . \tag{14}
$$

Equation (14) can be rewritten as

$$
\begin{aligned}
&\hat{\boldsymbol{x}} = \boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{x} , \\
&\boldsymbol{D} = diag(\boldsymbol{W}\mathbf{1})
\end{aligned} . \tag{15}
$$

which can be seen as graph filtering by a normalized version of the graph's adjacency matrix.

Other image filters such as non-local means [21] can be viewed from a GSP perspective as well. Non-local graph-based transform (NLGBT) is a graph-based denoising technique which significantly outperforms state-of-the-art denoising techniques such as BM3D for piecewise smooth images like depth images [22].

1.2 Visual Coding in the Retina

1.2.1 Introduction

The eye receives visual information by light and translates it into electrical signals which are carried to the brain for further processing. Curiously, this conversion is not a simple RGB code. Rather, the visual information goes through a sophisticated process evidenced by the complexity of the retina's cell structure. Researchers have formed many hypotheses to clarify this process although much remains unknown. In this section, we explain how the eye encodes visual information.

The retina is a thin layer of neural cells that lines the back of the eyeball of vertebrates and some cephalopods. In vertebrate embryonic development, the retina and the optic nerve originate as outgrowths of the developing brain. Hence, the retina is part of the central nervous system (CNS). The vertebrate retina contains

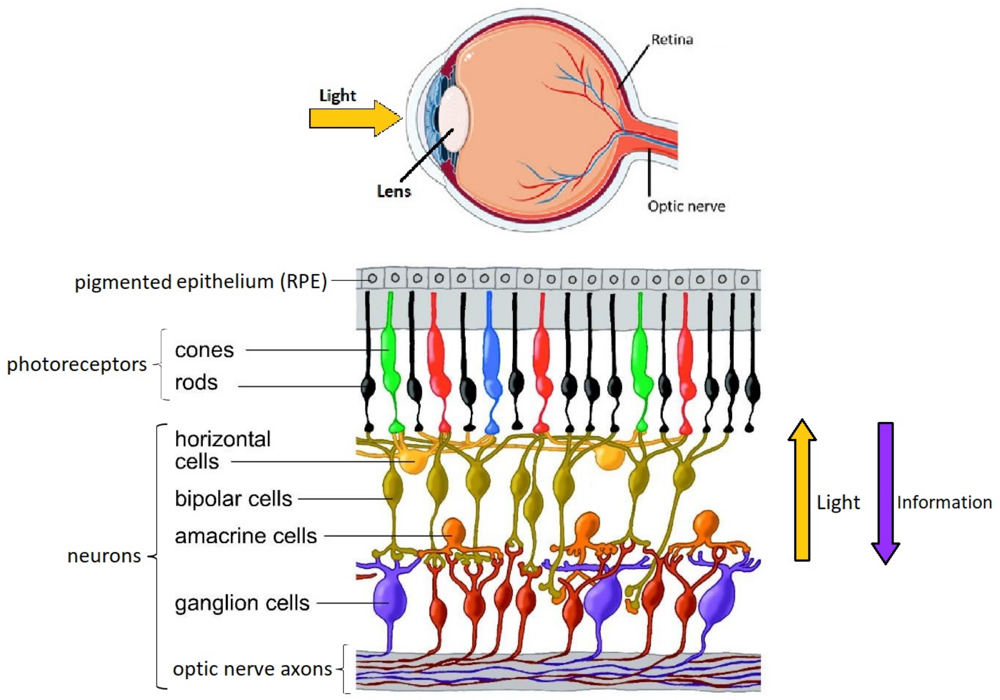


Figure 5: Cross section of the eyeball and the retina. The retina contains several layers of cells. Images are adapted from [1, 2] with minor edits.

photoreceptor cells (rods and cones) that respond to light; the resulting neural signals then undergo complex processing by other neurons of the retina. The last layer of the retina's cells are called ganglion cells. These cells have long axons which form the optic nerve. The optic nerve carries visual information to the brain in the form of neuron action potentials. Several important features of visual perception can be traced to the retinal encoding and processing of light.

Figure 5 shows how light enters the eye through the lens and proceeds to reach the retina. The retina is mostly transparent, so light moves through it until it reaches the pigment epithelium (RPE). RPE cells are pigmented; therefore, they absorb light instead of passing it. Photoreceptor cells (rods and cones) are placed immediately adjacent to RPE cells. They receive the absorbed light and convert it to electrical signals. In humans, there are three different types of cone cells which respond to three different colors: red, green and blue.

Photoreceptors are connected to horizontal and bipolar cells. Bipolar cells transmit electrical signals from photoreceptors to the next stage. They serve as a link between photoreceptors and ganglion cells (RGC). Ganglion cells receive signal from bipolar and Amacrine cells, then transmit the signal upward through their axons. At least 18 different types of ganglion cells are now thought to be present in the primate and human retina, all of them functionally and morphologically distinct [23]. The collection of ganglion cell axons is called the Retinal Nerve Fiber Layer (RNFL), also known as the optic nerve, which carries visual information to the brain.

Each ganglion cell is connected to a set of photoreceptors via a complex circuit consisting of bipolar, horizontal and Amacrine cells. If we move a bright spot in front of the retina, we see that the ganglion cells in a local area around the bright spot respond to the light, while other ganglion cells positioned far from the light source remain silent. Hence, each ganglion cell has a *receptive field* consisting of the photoreceptors that are connected to it via a cellular circuit. For a fixed eye position, we can also think of the receptive field as the region of the visual field in which the stimulus evokes a response in the ganglion cell [3].

To get more acquainted with ganglion cell signaling, let us investigate two types of ganglion cell receptive fields. The receptive field is subdivided into two regions, center and surround. There are two primary types of ganglion cell receptive fields:

1. ON-center/OFF-surround: Flashing a small bright spot in the center subregion increases the cell's response. Flashing a bright annulus in the surround subregion inhibits the cell's response. There is little or no response to a large (full field) spot of light that covers both the center and the surround because excitation in the center cancels the inhibition from the surround. This phenomenon is called lateral inhibition.

2. OFF-center/ON-surround: This type has the opposite arrangement. It gets inhibition from a bright spot in the center, and excitation from an annulus in the surround.

Figure 6 plots a ganglion cell's action potentials through time if a small bright spot is flashed on its receptive field. Icons on the left represent various visual stimuli. Plots next to the icons show the electrical pulses recorded through time at the cell's axon for indicated stimuli. We can see that the ON-center cell fires more frequently when excited by a bright-center-dark-surround stimulus, and less frequently by the inverse stimulus. The OFF-center ganglion cell behaves in opposite of the ON-center cell. This behavior already portrays a sort of derivation computed by the ganglion cells.

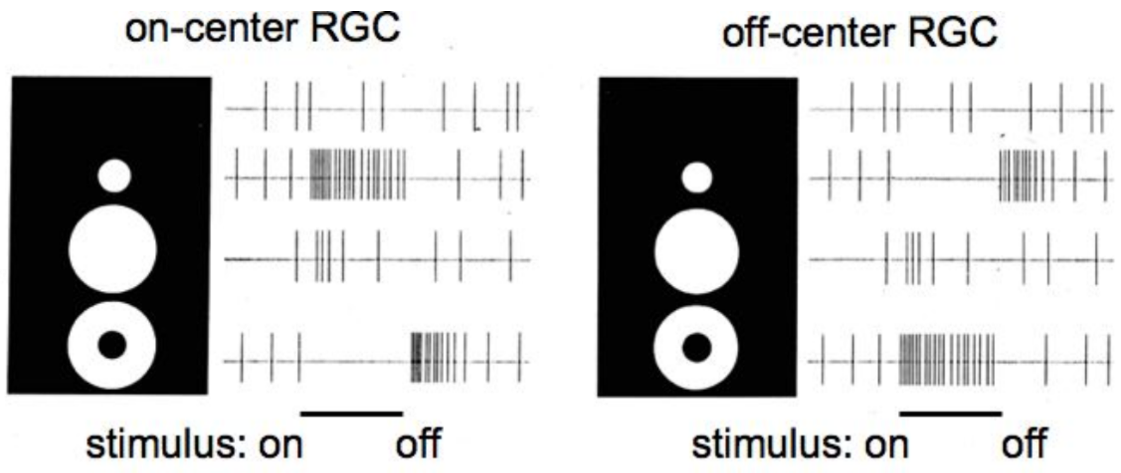


Figure 6: Responses of two primary ganglion cell types to various stimuli. Adapted from [3]

Most textbooks and review articles regard the retina akin to a pre-filter and a channel. The retina's principal function is to convey the visual image to the brain, where complex processing can be applied to it. Nevertheless, the retina enhances the brain's input as well. For instance, a light adaptation mechanism exists in the retina which acts like an automatic gain control, keeping the visual input's illumination in a suitable range. Moreover, the lateral inhibition in the ganglion cell's receptive field can serve to sharpen the image in space and time. Although the picture of the retina as a simple spatiotemporal prefilter is widely adopted, it fails to explain the need for various RGC types and intricate cell connections in the retina [24].

An alternative picture of the retina proposes that each ganglion cell type computes a specific feature about the visual scene. In this picture, each ganglion cell type is connected to the photoreceptors via a dedicated neural circuit which extracts the feature of interest. This neural circuit is implemented by the Amacrine, bipolar and horizontal cells. Therefore, the downstream regions in the brain receive a highly processed set of extracted features, instead of a generic pixel representation of the image. Indeed, there is a well-known example of this kind of processing: the direction-selective ganglion cell. These neurons respond strongly to moving stimuli, such as traveling spots or bars, but they greatly prefer one direction of motion over the others [25, 26, 27].

The prefilter picture may well apply to particular kinds of retinal ganglion cells under certain conditions; but in other cases, it reflects a crude average of the ganglion cell's behavior under stimuli that fail to probe its function properly. To better understand the retinal code, it helps to work with visual stimuli that somehow reflect the actual challenges that the visual system faces in its natural environment. The data used in this thesis is generated by naturalistic stimuli for the same reason. A number of visual features computed by the retina are reviewed in the next section.

1.2.2 Literature Review

Since we discuss general computational abilities of the vertebrate retina, we will focus on visual tasks relevant to all species: detecting light at low intensity, dealing with image motion caused by objects in the scene or the movement of the observer, and adapting to changing visual environments. Because of the generic nature of these tasks, we will freely discuss results obtained from different animal models.

A) Light Detection

The most straightforward task of the visual system is the detection of dim lights. Human observers can sense a flash of light even at very low intensities that lead to only a handful of successful photon absorptions in the retina [28, 29]. Correspondingly, rod photoreceptors display small responses to single photon absorptions, 1 mV in amplitude [30, 31], and retinal ganglion cells can indeed signal these events to the brain [32]. A ganglion cell typically collects inputs from many hundreds of rods [33]. Thus, the computational challenge for the retina lies in separating the small single-photon signal in one or a few rods from the continuous electrical noise that is present in all photoreceptors. How the retina isolates the signal from the ubiquitous noise is beginning to be understood [34, 35].

B) Motion Detection and Discrimination

Beyond mere light detection, the retina can interpret behaviorally significant patterns in the visual input. A dominant feature of the retina's input is motion. Motion can come from two sources. First is the movement of the observer's body, head or eye which causes a global optic flow. This type of global motion is usually interpreted as noise since it does not contain important information about the environment which guides the subject's behavior. The second source is the movement of objects in the scene which causes the behaviorally relevant "signal". As we will see, The retina contributes to sorting this important signal from the morass of distracting noise before it is presented to the brain.

The Y-type ganglion cell, according to one hypothesis, detects texture motion. When a textured image patch moves across the retina, it causes an increase of light intensity at some points and a decrease at others. Y-type ganglion cells are connected to a circuit of bipolar cells seemingly able to detect and integrate such local changes. These neurons fire when the texture moves, and the activity is largely independent of the direction of motion or the spatial layout of the moving pattern. Such ganglion cells have been identified in many species [36, 37, 38, 39, 40, 41].

Saccadic eye movements are unconscious rapid movements of both eyes that shift the center of our gaze even if we consciously fix our gaze on an object. Although these fixational eye movements have a magnitude that should make them visible to us, we are unaware of them. If saccadic eye movements are counteracted, our visual perception fades completely as a result of neural adaptation. Hence, our visual system has a built-in paradox — we must fix our gaze to inspect the minute details of our world, but if we were to fixate perfectly, the entire world would fade from view [42].

As a result of this constant movement, object motion manifests itself on the retina as the difference between the motion trajectory of a local patch and that of the background. Neurons that detect this differential motion have been found in various parts of the visual system [43, 44, 45], and they are particularly well studied in the retina. Here, object motion sensitive (OMS) ganglion cells were discovered that respond selectively to differential motion [46, 47]. These neurons remain silent when all visible objects move rigidly across the retina, but fire vigorously when a local patch on the receptive field center moves with a trajectory different from the background.

Objects moving vertically or horizontally in the visual field lead to translation on the retina. But what about motion in the third dimension of depth? An approaching object would produce an image patch that gradually expands on the retina, with no net displacement. Recently, a ganglion cell type was described that is indeed selective for this stimulus feature [48]. These ganglion cells showed OFF-type responses. They were driven strongly by an expanding dark spot, even if it was accompanied by a global brightening of the scene. Yet they remained silent during lateral motion of a dark spot.

C) Anticipation

There is great survival value in being able to anticipate the future. Indeed, because of delays in our sensory pathways, the brain really experiences the past, and even knowing the present requires a measure of prediction. Here we mention two ways in which the retina contributes to this prediction.

Detecting a moving object and the direction of its motion is generally not sufficient; to catch fleeing prey or to avoid an approaching predator, an animal needs to track the object location precisely. This becomes a challenge because of the cells' natural delay to respond to stimuli. Even in bright daylight conditions, the cone photoreceptor responds with a time delay of several tens of milliseconds [49, 50]. In this amount of time, a well-served tennis ball flies $\sim$ 2m, a distance many times larger than the receiving player's racket. Obviously, undoing this delay is critical for the observer's survival.

It has been shown that the retina extrapolates an object's trajectory by relying on the fact that objects tend to move in a smooth fashion. Surprisingly, this complex computation comes about through the interplay of rather generic features of retinal circuitry. The main contributors are the spatiotemporal receptive field of the ganglion cell and a dynamic gain-control mechanism. Because the receptive fields are extended in space, the object already activates ganglion cells that lie some distance ahead in its motion path. The spatial receptive field alone would predict an equally extended zone of activation behind the object. However, firing of those ganglion cells is suppressed first by the biphasic temporal receptive field, and second by a dynamic gain control mechanism, which itself gets activated by the response to the object [51, 52, 53]. The gain control lets the ganglion cell be sensitive upon first entry of the object into the receptive field, but then shuts down the response. It is an essential component for the anticipatory response to motion stimuli [54] and gives this computation a highly nonlinear flavor.

A different form of anticipation can be observed when the visual system is exposed to a periodic stimulus, such as a regular series of flashes. The activated visual neurons typically become entrained into a periodic response. If the stimulus sequence is interrupted, for example by omitting just one of the flashes, some neurons generate a pulse of activity at the time corresponding to the missing stimulus [55, 56]. This phenomenon, termed the "omitted stimulus response", is quite widespread and has been noted in the brains of many species, including humans [57]. Qualitatively it suggests the build-up of an anticipation for the next stimulus, and the large response reflects surprise at the missing element in the sequence. Unlike in the case of moving objects, where the anticipation involves lateral processing in space, here the information about the stimulus sequence must be propagated forward purely in time. Many works scrutinize the omitted stimulus response, attempting to find its underlying neural mechanism [58, 59, 60].

1.3 Explainability of Artificial Intelligence Models

1.3.1 Introduction

A) Model-based Processing

Model-based processing is the traditional way of inferring constructive information about a system or manipulating it into a desired state. This procedure, which is rooted in the scientific method itself, starts with observation, making a hypothesis about the system, testing said hypothesis by collecting data from controlled experiments and, finally, using results from the experiments to modify and further develop the hypothesis.

This paradigm has been used in many disciplines of engineering and science. The discovery of mathematical relations and laws governing a physical system is an example of model-based processing, from simple kinematic equations to complex theoretical models such as electromagnetism or thermodynamics. These mathematical models of physical systems, in turn, lay the foundation of new technology which manipulates said systems to reach a target [61].

The same procedure can be followed to develop *abstract* systems as well. For example, imagine that we want to create an audio processing system which can extract one voice from a recording of multiple speakers having a discussion. We need to analyze the recording using mathematical tools such as Fourier transform, identify useful variables to distinguish and isolate the desired voice, run multiple tests of our invented method, and repeatedly improve it until we reach an acceptable quality. Finally, we implement the method as our final product. To illustrate, a model-based solution for this problem can be found in [62].

B) Issues with Model-based Processing

Model-based processing is slow and expensive. People who develop models need to be experts in multiple disciplines and spend a tremendous amount of time testing and modifying their model. In addition, the experiments they conduct can be expensive. Finding out all relevant variables and dynamics in the system costs time, expertise and money for running experiments [61]. On the other hand, complex models are hard to work with when designing new technology or predicting the future state of the system; therefore, simpler models are favored. However, these simple models fail to account for some relevant variables and dynamics which may cause unforeseen behavior and system failure [63].

Above that, scientists and engineers tend to build upon previously established models than disrupt the foundations; so model-based hypotheses are usually biased by previously developed models. To boot, data gathered from controlled experiments may not reflect the full complexity of real applications and miss patterns that only appear in the full perspective [64]. These issues fostered a need for a new problem solving approach.

C) Data-driven Processing

In the past few decades, growing capacity and decreasing cost of sensors, computational power and data storage has led to an unprecedented abundance of accessible data [61]. Computers became able to run sophisticated algorithms on huge amounts of data which paved the way of data-driven processing.

Data-driven processing starts by devising an algorithm that can find patterns and correlations in data, then applies it to huge datasets that include as many variations and conditions of the system as feasible. At first glance, this might appear as an automation of the model-based approach; instead of humans meticulously curating a model, a computer tries out numerous possible models in a specific space and chooses the best one based on defined criteria.

Since computers can deal with many more variables than humans, data-driven models can be much more complex than ones devised by humans, taking more factors and conditions into account. This usually leads to better prediction of the system's behavior [65]. The data used in this approach is gathered in real uncontrolled conditions so it reflects more of the natural complexities of the system compared to controlled data used in the model-based approach. Consequently, measurement errors become more tolerable in such a setting [61].

D) Issues with Data-driven Processing

Benefits of data-driven solutions excited many scientists and engineers, although a lot of criticism have been made as well. One criticism points out that, counter to popular belief, data provided by online databases are highly selective. The subjects of these data and the way they are presented are influenced by underlying social, political, economic and technical factors. These unknown biases in data lead to erroneous data-driven models. Furthermore, these subtle biases and their consequent model errors are hard to detect and fix [66].

Other issues arise when too complex models are derived from data that do not rely on previously established knowledge. While these data-driven models yield extremely good performance, they tend to become a black box; in other words, explaining why and how they come to conclusions, in a humanly understandable way, becomes nearly impossible. This unexplainability causes important issues in technology enhancement, robustness, fairness and acceptance by the users.

Adoption of a model or taking action based on a prediction requires user trust in the model or prediction. A model with unintelligible inner dynamics or a prediction based on unknown patterns and trends cannot engender such a trust [67]. For instance, consider computer-based diagnosis of a cancerous tumor. Accepting the risks and costs of treating such a tumor requires a level of trust in the diagnosis that cannot yet be generated without human expert knowledge.

A closely related issue is that of safety. Engineers need to make sure that a data-driven model will work correctly not only on available validation data, but also on real user input in real applications. Understanding how the model will extrapolate to unseen data is crucial for this goal which requires some level of explainability [68].

Furthermore, enforcing legal responsibility on corporations and individuals requires a clear description of each party's function [69]. For instance, if an autonomous car crashes, it is not clear whether it is the driving system's fault or the passenger's. Another legal aspect is the "right to an explanation" which was adopted by the European Union in May 2018. This regulation states that individuals have the right to an explanation of a decision based on automated processing [70].

Technology developers need explainability as much as consumers. Detailed explanation of each part of the system helps developers locate sources of malfunction and uncertainty. They can then fix malfunctions or develop schemes to account for uncertainty. An example of this procedure is robust control theory which expands model-based control theory by considering uncertainty [63]. Explainability is also essential in developing new technology by using experience and feedback from previous ones.

A third merit of explainability is addressing upper-level concerns such as fairness, nondiscrimination and avoiding technical debt. Technical debt means the costs of maintaining a software such as deep learning models as opposed to the costs of their design and implementation [71]. On another note, since data-driven models are usually tuned to uncontrolled data, they are prone to adapting any discriminatory judgement that might exist in the data. To boot, the complex structure of these models makes it hard to detect where and why these biases happen [72]. To reduce technical debt and detect biases, the model needs to be moderately transparent.

1.3.2 Literature Review

Despite the crucial need for explainability, this concept does not yet have a formal mathematical definition, or a metric by which it can be measured. Various works define explainability in many intuitive ways. One popular definition is stated in [73] as "the ability to explain or to present in understandable terms to a human". Numerous works try to establish clear desiderata for an acceptable explanation such as [74, 75].

Despite the absence of clear definitions, a myriad of AI explanation methods have been developed over the years. As seen in Figure 7, these methods can be categorized based on different criteria. Some explanation methods only work for certain algorithms; these are called model-specific. While other methods, which are called model-agnostic, can be used to explain any AI model. Explanation methods may explain the model's result for one specific input (local), or generally explain the model for all inputs (global). Furthermore, explanation methods can be categorized based on their input data types; most common types are tabular and image [76]. Figure 7 is taken from [76].
Four major categories of AI explanation methods based on their purpose are as follows.

1. *Intrinsic or white-box*: These methods aim to create a self-explanatory model or a white box. Examples include linear, decision tree and rule-based models.

2. *Post-hoc*: These methods aim to explain already trained and complex models (black boxes). Examples include gradient-based attribution [77] and Shapley additive explanations (SHAP) [78].

3. *Fairness enhancement*: These methods aim to enhance AI fairness whether that leads to more explainability or not. Examples include disparate impact testing [79] and adversarial debiasing [80].

4. *Sensitivity analysis*: These methods analyze the output's sensitivity to input features. Examples include traditional methods such as [81, 82, 83] and methods of creating adversarial examples such as [84, 85, 86].

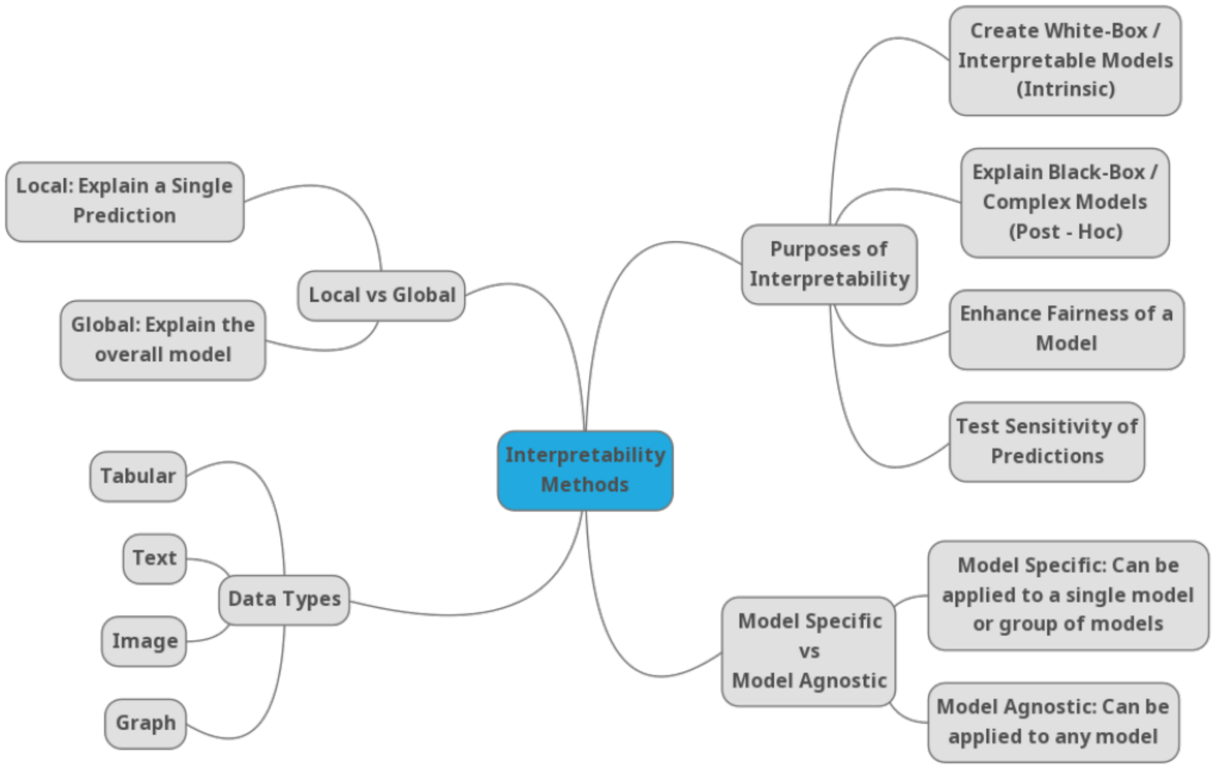

Interpretability Methods
Local vs Global
Local: Explain a Single Prediction
Global: Explain the overall model
Data Types
Tabular
Text
Image
Graph
Purposes of Interpretability
Create White-Box / Interpretable Models (Intrinsic)
Explain Black-Box / Complex Models (Post - Hoc)
Enhance Fairness of a Model
Test Sensitivity of Predictions
Model Specific vs Model Agnostic
Model Specific: Can be applied to a single model or group of models
Model Agnostic: Can be applied to any model

Figure 7: Categories of AI explanation methods.

2 Methodology

2.1 Big Picture

2.1.1 Graph Signal Processing

This work stems from GSP and branches into retinal coding and AI explainability. GSP offers an arsenal of various tools and concepts, one of which is the similarity graph. A similarity graph is an undirected graph where each edge describes the similarity relationship between two connected nodes. The similarity between two nodes can be delineated both by their connectivity (whether an edge exists between them or not) and the edge's weight. The higher the weight, the stronger the similarity. The edge's weight can quantify similarity with more precision than just the connectivity. This similarity-based structure is then used to process the graph signal, which is defined on top of the nodes.

The definition of similarity is application-dependent. One idea is to parameterize this definition and learn it using experimental data. Thus, the similarity graph becomes a machine learning model. One application of such a model is classification. Let each node represent a data point, e.g., an image of an animal (see Figure 8). The graph signal on top of nodes shows the class to which each image belongs. In Figure 8, class labels are shown with letters; however, they can easily be converted to numbers by a simple map: *cat* is 0, *dog* is 1, and *horse* is 2.

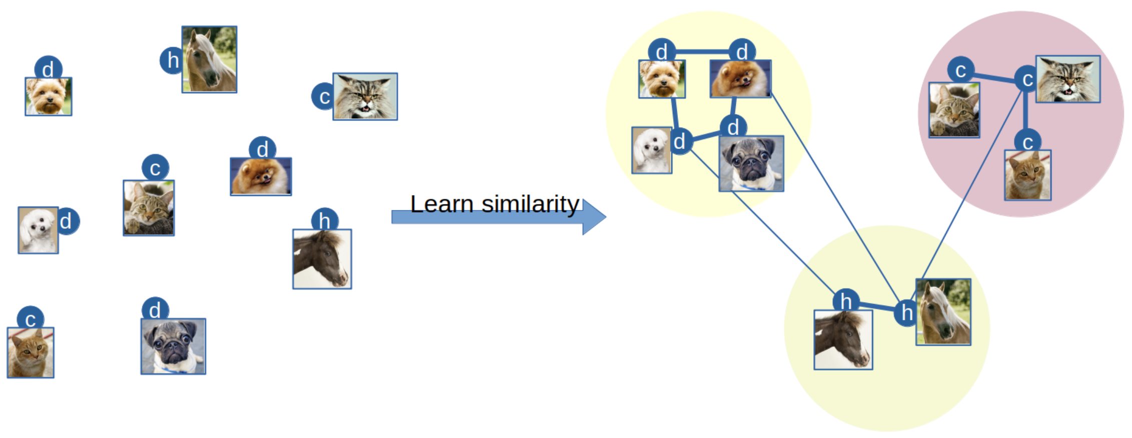


Figure 8: A similarity graph is used to classify images of animals. The definition of similarity is parameterized and optimized on labeled data.

Ideally, the similarity definition would be learned in a way that many edges with high weights would connect images of the same label, while images of different labels are either connected with low-weighted edges or not connected at all. In other words, the similarity should be learned in a way that the given graph signal becomes smooth (or piece-wise smooth) on the resulting graph. The concept of smoothness for graph signals was explained in Section 1.1.1. If the similarity is learned correctly, the resulting structure includes *clusters* comprising all nodes of a certain class. Edges insides a cluster are numerous and have high weights, while edges between clusters are scarce and have low weights.

To predict unknown labels using this model, we represent each test data point with a node. Then, we find the edges connecting the test nodes to training nodes using the optimized definition of similarity. This process is illustrated in Figure 9; new edges connecting a test image to a training or another test image are colored red. After fixing the edges, estimating the unknown labels becomes equivalent to an interpolation problem for graph signals (as opposed to time-domain signals). Thus, the unknown labels can be estimated using the smoothness criterion and the known labels.

This estimation yields real values for labels; for example, we might get 1.2 or 2.37. The real-valued labels can be simply rounded (or otherwise discretized) into whole numbers to obtain class labels. The original real values can also serve as a measure of confidence in the final prediction; for example, if a real-valued label is rather close to a threshold border, then the final prediction has a high probability of error.

To increase the accuracy of our model, nodes can be associated with extracted features of images, instead of their raw values. These features can be arbitrarily simple or complex. For instance, one might use filters of a pre-trained convolutional neural network (CNN) to extract features from the images. The filters can belong to the first layers of CNN which result in simpler (but more generalizable) features, or the last layers which yield more specific and sophisticated features. Alternatively, one can use feature extraction algorithms such as SIFT [87], SURF [88], BRIEF [89], ORB [90], etc.

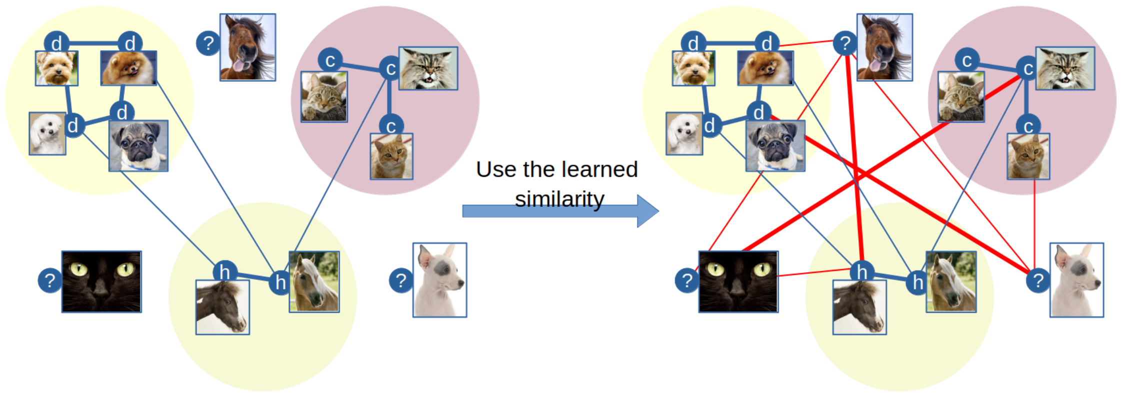


Figure 9: Estimating labels of test images using the optimized similarity graph of Figure 8. Edges connecting at least one test image to another image are colored red.

2.1.2 Machine Learning

The previous section described the outline of a graph-based classification model. This section summarizes the model's advantages compared to existing classification approaches such as deep learning [91], gradient boosting decision trees (GBDT) [92], k-nearest neighbors [93], etc.

There are two main advantages that our model provides: explainability, and the ability to use the test nodes' similarities to predict their labels. The focus of this thesis is providing good explanations of the model's predictions. This is not to say that popular classification approaches like deep learning are 100% opaque and unexplainable. In fact, several tools to explain machine learning models were introduced in Section 1.3. Nevertheless, we develop a transparent model in this thesis so that it enables producing neurobiological interpretations which can be used in the research about retinal visual coding.

A similarity graph is completely explainable and visualizable. Moreover, it enables further analysis via already-developed tools in GSP. Since the definition of similarity can be arbitrarily parameterized, one can choose a widely studied and established formulation. In this thesis, we defined similarity based on the Mahalanobis distance which is well-known and studied in the literature [94, 95, 96]. More explanation of the Mahalanobis distance is presented in the following sections.

Although the similarity graph is completely explainable, the features that are given to the similarity graph may not be. If features are not explainable, then our method is only half-explainable; we can see the similarity graph and the relationship between the features and the prediction, but not the relationship between the features and the input image. However, the choice of features is given to the user. Based on the application, one might use a fully explainable feature set such as SIFT. If the features and the definition of similarity are explainable, then our approach is 100% transparent from start to finish.

The second advantage of our method is the possibility to utilize similarities between the test data, as well as the training data ,to predict the labels. Machine learning models typically predict each test label independently from other test data. Conversely, our proposed model estimates all test labels at once; hence, it can account for the similarities between the test data during prediction.

To estimate test nodes' labels, we found edges between them and training nodes, then considered the whole graph signal and interpolated it on the test nodes based on the smoothness criterion. We can incorporate the similarity between the test nodes into the prediction process by connecting pairs of test nodes via edges. Based on the smoothness criterion, connected test nodes with similar images tend to have similar predictions, whatever that prediction may be. This property allows the model to rely less on the training data which is specially sought after in applications where labeled training data is expensive to obtain. One such application is neurobiological signaling which is discussed in the next section.

2.1.3 Retinal Coding

As described in Section 1.2, each retinal ganglion cell type extracts a specific feature from the eye's input which is crucial to the animal's survival from an evolutionary point of view. Recent advances in building electrode arrays [97] have made it possible to record individual responses of a population of RGCs to an stimulus, such as a naturalistic movie played in front of the retina. Deep learning models have been trained and tested on these datasets (i.e. the stimulus and the responses) [98, 99]. Although deep learning can model the input-output of the retina with high accuracy, it struggles to yield applicable information about the nature of the ganglion cells [99]. This is mostly due to the opaqueness of large convolutional neural networks (CNN) which was discussed in Section 1.3.2. Our model's transparency enables neurobiological interpretations. These interpretations let us select the features to which ganglion cells are most sensitive from a given set of extracted features. Moreover, they can be used to locate each ganglion cell's receptive field (see Section 1.2.1 for a definition of the receptive field). More applications of these interpretations can be explored in the future.

On the other hand, recording ganglion cell responses is expensive so the datasets are small compared to the usual sets used in deep learning applications. Additionally, these responses are sparse since ganglion cells stay silent most of the time; i.e. action potentials are infrequent. The small and sparse datasets pose a challenge for mainstream deep learning approaches; therefore, our model's ability to use test data similarities is convenient to this context. Furthermore, our model has the option of connecting each node to an arbitrary number of nodes from zero to the size of the graph. As a result, it has the flexibility of *caring* more about some nodes compared to the others; for instance, one can set a higher degree for nodes that are associated with action potentials than those associated with silence.

2.2 Comprehensive Picture

2.2.1 Problem Formulation

In its most simplified form, the retina is a system that converts the eye's spatiotemporal input to electrical pulses conveyed by multiple cells through time (Figure 10). The eye's input could be mimicked by a movie. Mathematically, it can be represented by a 3D tensor (see Figure 11). The first two dimensions serve as the width and height of the visual field (i.e. the movie's frames), and the third dimension is time. The tensor's elements delineate the pixels' brightness on the grayscale spectrum. If the eye's input includes color information, then three of these tensors are needed, each for one primary color. The dataset used in this thesis includes a grayscale movie as the eye's input; therefore, we only consider one tensor.

light
Retina
electrical pulses

Figure 10: Retina is simplified as an encoder, converting the visual input to a set of electrical pulses through time

The electrical pulses can be represented as a set of discrete binary signals through time. The time interval between samples depends on how the continuous experimental data is processed. For simplicity, we assume this time interval is equal to the time that each frame takes in the input movie. As a result, the samples of these binary signals are exactly aligned with the frames of the given movie. The signals' samples can either be 1 or -1. If a cell's signal is 1 in a specific time interval, it means that cell had an action potential in that interval, otherwise the signal is recorded as -1. The signals of all cells can be represented as a 2D matrix; each row illustrates one cell's signal through time. Figure 11 shows the tensor and the matrix aligned with each other on the time axis.

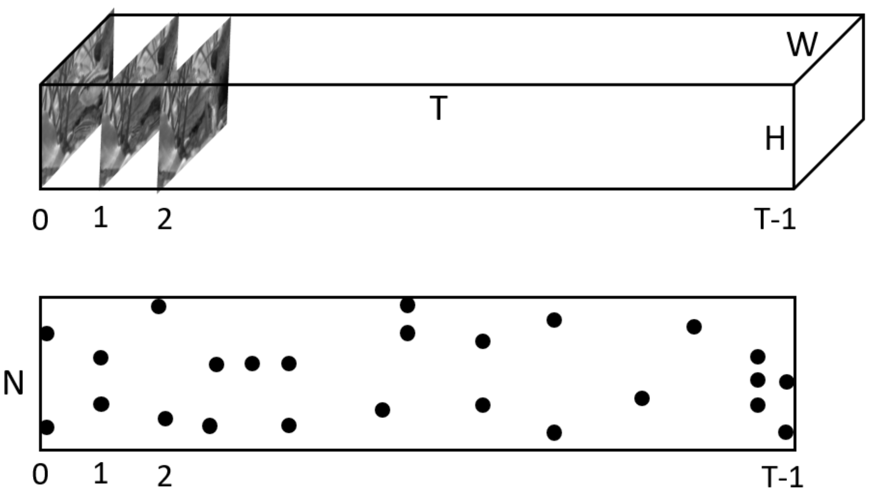


Figure 11: The retina's input is illustrated by a 3D tensor (top), while its output is represented as a 2D matrix (bottom). Black dots on the matrix show entries valued +1, while white spaces show entries valued -1.

As a result, one can mathematically describe the retina as a function from the space of all 3D tensors of the correct size to the space of all matrices of the correct size, i.e.,

$$r : \mathbb{R}^{T\times H\times W} \to \{+1,-1\}^{N\times T}. \tag{16}$$

The experimental dataset can be mathematically formulated as

$$\mathcal{D} = \{\boldsymbol{Q},\boldsymbol{S}\} \quad , \quad \boldsymbol{Q} \in \mathbb{R}^{T\times H\times W}, \boldsymbol{S} \in \{+1,-1\}^{N\times T}. \tag{17}$$

We know that the experimental data is an input-output set of the retina, i.e.,

$$r(\boldsymbol{Q}) = \boldsymbol{S}. \tag{18}$$

The problem is to estimate the function r using the given input-output set $\mathcal{D}$.

Estimating r with our current dataset is difficult since r is a complicated function; therefore, we simplify the problem by making assumptions. The assumptions do not necessarily reflect the real function of the retina; however, we take care not to *over*simplify the problem. The first assumption is that the ganglion cells operate independently from each other; in other words, r is a concatenation of N independent functions, each receiving the same 3D tensor as input, but producing a different discrete binary signal as output,

$$\begin{aligned} &r = [r_0, r_1, r_2, ..., r_{N-1}] \\ &r_i : \mathbb{R}^{T\times H\times W} \to \{+1,-1\}^T \text{ , } \forall i \in \{0,...,N-1\}. \end{aligned} \tag{19}$$

By this definition, we have N pairs of input-output in our dataset,

$$\begin{aligned} &\mathcal{D} = \{(\boldsymbol{Q},\boldsymbol{s_0}),(\boldsymbol{Q},\boldsymbol{s_1}),...,(\boldsymbol{Q},\boldsymbol{s_{N-1}})\} \\ &\boldsymbol{s_i} = \text{row } i \text{ of matrix } \boldsymbol{S}, \end{aligned} \tag{20}$$

each pair belonging to one function,

$$r_i(\boldsymbol{Q}) = \boldsymbol{s_i} \ , \ \forall i \in \{0,...,N-1\}. \tag{21}$$

The second simplifying assumption is that a ganglion cell's response in a certain time interval depends only on a fixed duration of the stimulus prior to the response. Let that duration be M preceding frames of the movie; then, sample t of signal $\boldsymbol{s_i}$ depends only on frames $t-M+1$ to t of the movie,

$$\begin{aligned} &s_i[t] \text{ depends on } \{\boldsymbol{Q_{t-M+1}},\boldsymbol{Q_{t-M+2}},...,\boldsymbol{Q_t}\} \ , \ \forall t \in \{0,...,T-1\} \ , \ i \in \{0,...,N-1\} \\ &s_i[t] \in \{+1,-1\} \ , \ \boldsymbol{Q_t} \in \mathbb{R}^{H\times W} \ , \ \forall t \in \{0,...,T-1\}. \end{aligned} \tag{22}$$

This reduces each function r_i to a simpler function which receives chunks of the whole stimulus and produces one sample of the response at a time,

$$\begin{aligned} &r'_i : \mathbb{R}^{M\times H\times W} \to \{+1,-1\} \\ &r'_i(\boldsymbol{Q_{t-M+1}},...,\boldsymbol{Q_t}) = s_i[t] \ , \ \forall t \in \{0,...,T-1\}. \end{aligned} \tag{23}$$

To obtain r_i from r'_i, one should only feed all chunks of the stimulus to r'_i, and then concatenate the responses,

$$r_i(\boldsymbol{Q}) = [r'_i(\boldsymbol{Q_{t-M+1}},...,\boldsymbol{Q_t}) \ , \ \forall t \in \{0,...,T-1\}] = [s_i[t] \ , \ \forall t \in \{0,...,T-1\}] = \boldsymbol{s_i}. \tag{24}$$

Our input-output pairs are now composed of chunks of the video and binary responses,

$$\begin{aligned} &\mathcal{D} = \{\mathcal{D}_0,\mathcal{D}_1,...,\mathcal{D}_{N-1}\} \\ &, \ \mathcal{D}_i = \{([\boldsymbol{Q_{t-M+1}},...,\boldsymbol{Q_t}],s_i[t]) \ , \ \forall t \in \{0,...,T-1\}\}. \end{aligned} \tag{25}$$

Each $\mathcal{D}_i$ comprises T input-output pairs; therefore, the whole dataset $\mathcal{D}$ consists of $N\times T$ input-output pairs.

Lastly, we split a cell's function into two nested functions,

$$r'_i(\cdot) = g_i(h(\cdot)). \tag{26}$$

The first function h extracts visual features from the video clips which are shared among all ganglion cells,

$$\begin{aligned} &h : \mathbb{R}^{M\times H\times W} \to \mathbb{R}^C \\ &h(\boldsymbol{Q_{t-M+1}},...,\boldsymbol{Q_t}) = \boldsymbol{f_t} \ , \ \forall t \in \{0,...,T-1\}, \end{aligned} \tag{27}$$

where $\boldsymbol{f_t}$ is a feature vector describing M video frames prior to and including frame t. The second function

g_i, which is specific to each ganglion cell, generates the cell's response based on the input features $\boldsymbol{f_t}$,

$$\begin{aligned} &g_i : \mathbb{R}^C \rightarrow \{+1, -1\} \\ &g_i(\boldsymbol{f_t}) = s_i[t]\ ,\ \forall t \in \{0, ..., T-1\}. \end{aligned} \tag{28}$$

Figure 12 illustrates this simplifying step.

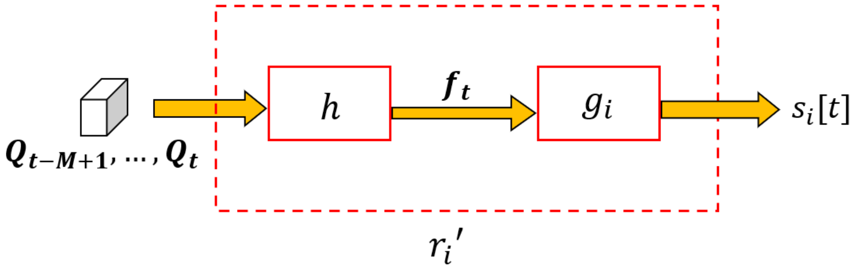


Figure 12: Each neuron's function is composed of two nested functions. The first one (h) extracts visual features from the input, and the second one (g_i) generates the cell's response based on the features.

In this thesis, function h is arbitrarily chosen from existing feature extraction algorithms such as SIFT [87], 3D-SIFT [100], pre-trained filters of a suitable CNN [101], etc. In contrast, function g_i is learned using the dataset $\mathcal{D}_i$. Since function h is chosen rather than learned, the problem is effectively reduced to finding functions g_i , $\forall i \in \{0, ..., N-1\}$. We can learn function h as well in the future of this research.

Since the features $\boldsymbol{f_t}$, $\forall t \in \{0, ..., T-1\}$ are now known, our dataset is composed of pairs of feature vectors and binary responses,

$$\begin{aligned} &\mathcal{D} = \{\mathcal{D}_0, \mathcal{D}_1, ..., \mathcal{D}_{N-1}\} \\ &\mathcal{D}_i = \{(\boldsymbol{f_0}, s_i[0]), (\boldsymbol{f_1}, s_i[1]), (\boldsymbol{f_{T-1}}, s_i[T-1])\}. \end{aligned} \tag{29}$$

After applying these simplifying assumptions, the problem becomes finding N functions g_i based on N datasets $\mathcal{D}_i$. Each dataset comprising T input-output pairs, where the input is a 1D vector and the output is a binary number. This can be seen as N separate binary classification problems. Equation (30) shows the final formulation of the problem:

$$\begin{aligned}
&\text{For each ganglion cell } i: \\
&\qquad g_i : \mathbb{R}^C \to \{+1, -1\} \\
&\qquad g_i(\boldsymbol{f_0}) = s_i[0] \\
&\qquad g_i(\boldsymbol{f_1}) = s_i[1] \\
&\qquad \vdots \\
&\qquad g_i(\boldsymbol{f_{T-1}}) = s_i[T-1] \\
&\qquad \implies \text{Find } g_i.
\end{aligned} \tag{30}$$

For the rest of this section, we focus only on one cell so we can drop the index i.

2.2.2 The Graph

The dataset is composed of T input-output pairs: $\mathcal{D} = \{(\boldsymbol{f_0}, s[0]), \ldots, (\boldsymbol{f_{T-1}}, s[T-1])\}$. Each input-output pair is illustrated by a node in the similarity graph. Each node is associated with a time instant t; hence, each node holds a feature vector $\boldsymbol{f_t}$. In contrast, the cell's *response* at interval t is the *graph signal's sample* on node t. Figure 13 shows a handful of nodes and their signal samples.

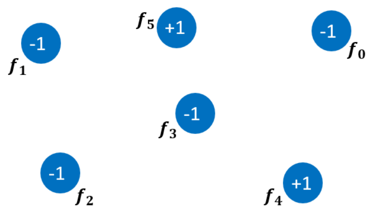


Figure 13: Several nodes of the similarity graph and the signal samples on top.

The similarity graph can be complete, meaning that all nodes are connected to all other nodes. Even so, a complete graph makes the training and validation process computationally expensive; therefore, only a number of edges should be selected for the graph according to a user-selected strategy. For instance, a certain number of edges can be selected at random, or based on the closeness of the nodes' time indices t. Both of the aforementioned strategies are implemented in this thesis which will be thoroughly explained in Section 2.3.

After selecting the graph's edges, the weights of the edges determine the similarity of the node pairs at their ends. In this thesis, the similarity is defined as

$$w_{ij} \triangleq e^{-d_{ij}}, \tag{31}$$

where w_{ij} is the weight of the edge connecting nodes i and j, and d_{ij} is the *distance* of the node pair.

Subsequently, the distance is defined as

$$d_{ij} \triangleq (\boldsymbol{f_i} - \boldsymbol{f_j})^T \boldsymbol{M} (\boldsymbol{f_i} - \boldsymbol{f_j})$$
$$\boldsymbol{f_i}, \boldsymbol{f_j} \in \mathbb{R}^C \ , \ \boldsymbol{M} \in \mathbb{R}^{C \times C}, \tag{32}$$

which is called the *Mahalanobis distance*[94]. $\boldsymbol{f_i}$ and $\boldsymbol{f_j}$ are feature vectors of nodes i and j. The matrix $\boldsymbol{M}$ is a metric matrix; it defines the space where the Mahalanobis distance is calculated. A metric matrix must be positive semi-definite (PSD), meaning that it should satisfy the following constraint,

$$\boldsymbol{x}^T \boldsymbol{M} \boldsymbol{x} \geq 0 \ , \ \forall \boldsymbol{x} \in \mathbb{R}^C \iff \boldsymbol{M} \in \mathbb{R}^{C \times C} \text{ is PSD} \ . \tag{33}$$

This constraint is necessary to ensure that the distance d_{ij} is always non-negative. Negative distance is ill-defined and would cause numerical issues later on. A PSD matrix is denoted $\boldsymbol{M} \succeq 0$. Since $0 \leq d_{ij} < +\infty$, the edge weights w_{ij} will end up in the $(0, 1]$ range, with 1 corresponding to zero distance ($d_{ij} = 0$). As the distance of $\boldsymbol{f_i}$ and $\boldsymbol{f_j}$ increases to $+\infty$, their similarity decreases to 0. Since $\boldsymbol{M}$ is positive *semi*-definite, the distance may be zero for non-equal feature vectors as well. In mathematical terms, we have

$$\boldsymbol{f_i} = \boldsymbol{f_j} \Rightarrow d_{ij} = (\boldsymbol{f_i} - \boldsymbol{f_j})^T \boldsymbol{M} \, (\boldsymbol{f_i} - \boldsymbol{f_j}) = \boldsymbol{0} \, \boldsymbol{M} \, \boldsymbol{0} = 0, \tag{34}$$

but,

$$\text{if } \boldsymbol{M} \succeq 0 \ : \ d_{ij} = (\boldsymbol{f_i} - \boldsymbol{f_j})^T \boldsymbol{M} \, (\boldsymbol{f_i} - \boldsymbol{f_j}) = 0 \not\Rightarrow \boldsymbol{f_i} = \boldsymbol{f_j}. \tag{35}$$

The Mahalanobis distance is simple and visualizable, yet it has enough flexibility to match most of the data distributions in the real world. To illustrate, let us rewrite the definition in Equation (32) as

$$d_{ij} = \sum_{a=0}^{C-1} \sum_{b=0}^{C-1} M_{ab} f_a f_b$$
$$, \boldsymbol{f} \triangleq \boldsymbol{f_i} - \boldsymbol{f_j}. \tag{36}$$

By this notation, one can see that pairs of entries of $\boldsymbol{f_i} - \boldsymbol{f_j}$ are multiplied and linearly combined to determine the distance between nodes i and j. For better visualization, let us consider a 2D example of this; i.e.

$$\boldsymbol{f} = \boldsymbol{f_i} - \boldsymbol{f_j} = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2$$
$$, \boldsymbol{M} \in \mathbb{R}^{2 \times 2}. \tag{37}$$

Thus, Equation (32) becomes

$$d_{ij} = \boldsymbol{f}^T \boldsymbol{M} \boldsymbol{f} = [x, y] \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$= M_{00} x^2 + (M_{01} + M_{10}) xy + M_{11} y^2, \tag{38}$$

which is the parametric equation of conic sections minus the first-degree terms (with only x or only y). The conic sections are the circle, ellipse, hyperbola and parabola. The entries of $\boldsymbol{M}$ can be adjusted based on the data distribution in a way that its contours become any ellipse or circle; however, the contours can not be hyperbolas or parabolas. To prove that any ellipse is feasible, consider the parametric equation of the ellipse,

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1. \tag{39}$$

To shape the contours into ellipses, we only need to define $\boldsymbol{M}$ as

$$\boldsymbol{M} = \begin{bmatrix} \frac{1}{a^2} & 0 \\ 0 & \frac{1}{b^2} \end{bmatrix}. \tag{40}$$

By the definition in Equation (40), $\boldsymbol{M}$ is positive semi-definite,

$$d_{ij} = \boldsymbol{f}^T \boldsymbol{M} \boldsymbol{f} = \frac{x^2}{a^2} + \frac{y^2}{b^2} \geq 0 \quad \forall x, y \in \mathbb{R}. \tag{41}$$

To rotate the ellipse, we only need to multiply the rotation matrix $\boldsymbol{R_\theta}$ by the coordinates,

$$\boldsymbol{R_\theta} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}. \tag{42}$$

Hence, to rotate the elliptical contours, we define the metric matrix $\boldsymbol{M_\theta}$ as in Equation (43),

$$\boldsymbol{M_\theta} \triangleq \boldsymbol{R_\theta}^T \, \boldsymbol{M} \, \boldsymbol{R_\theta}, \tag{43}$$

where $\boldsymbol{M}$ is the same matrix in Equation (40). Now, the contours are derived by Equation (44),

$$d_{ij} = \boldsymbol{f}^T \, \boldsymbol{M_\theta} \, \boldsymbol{f} = \boldsymbol{f}^T \, \boldsymbol{R_\theta}^T \, \boldsymbol{M} \, \boldsymbol{R_\theta} \, \boldsymbol{f} = [x \;\; y] \, \boldsymbol{R_\theta}^T \, \boldsymbol{M} \, \boldsymbol{R_\theta} \begin{bmatrix} x \\ y \end{bmatrix} = const., \tag{44}$$

which is the equation of a rotated ellipse by angle θ. Thus, any ellipse (rotated or unrotated) is feasible.

Parabolas are not feasible because their parametric equation requires the first-degree terms (with only x or only y) which are not present in Equation (38). Hyperbolas are not feasible because their corresponding matrix $\boldsymbol{M}$ is not PSD. To elaborate, consider the parametric equation of the hyperbola,

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1. \tag{45}$$

To shape the contours of the Mahalanobis distance into hyperbolas, we need to define $\boldsymbol{M}$ as

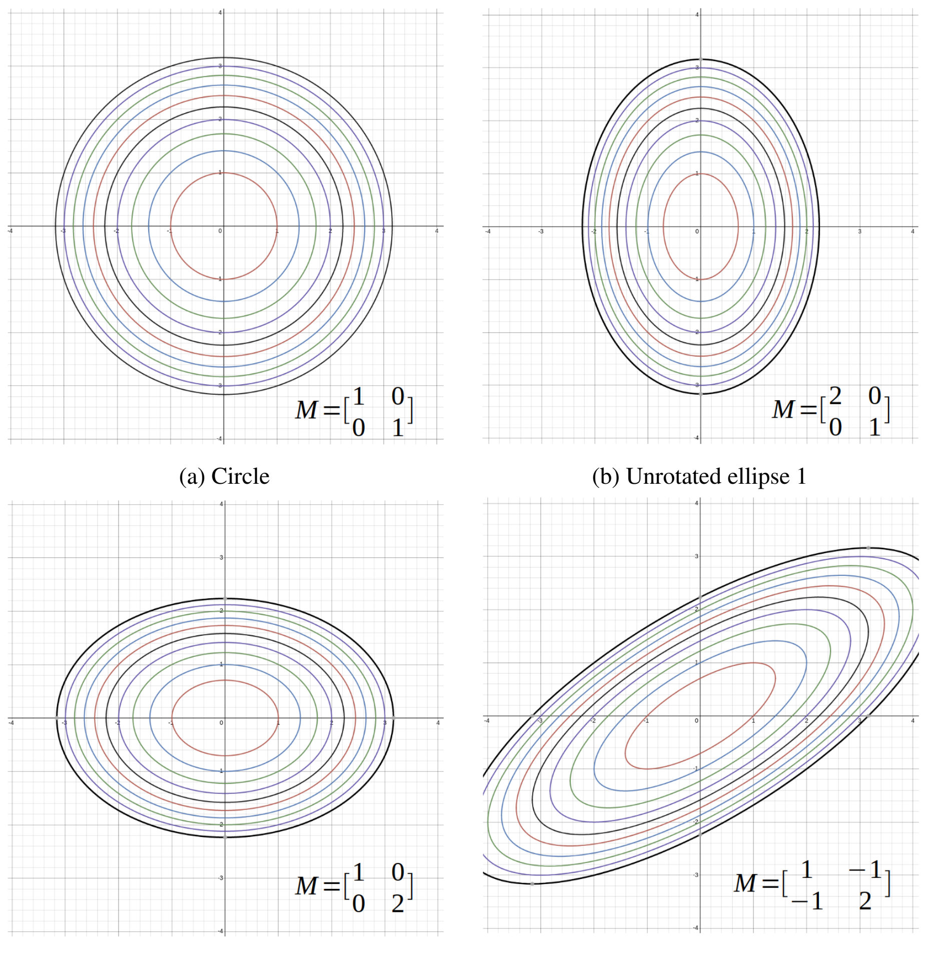
$$\boldsymbol{M} = \begin{bmatrix} \frac{1}{a^2} & 0 \\ 0 & -\frac{1}{b^2} \end{bmatrix}, \tag{46}$$

where a, b are the same coefficients used in Equation (45). Consequently, this means that the distance between any two points on a vertical line is negative:

$$\text{if } \boldsymbol{f} = \boldsymbol{f_i} - \boldsymbol{f_j} = \begin{bmatrix} 0 \\ y \end{bmatrix}, \; y \neq 0 \implies d_{ij} = \frac{0^2}{a^2} - \frac{y^2}{b^2} = -\frac{y^2}{b^2} < 0. \tag{47}$$

Therefore, hyperbolic contours do not satisfy the PSD constraint of $\boldsymbol{M}$.

Figure 14 shows examples of the Mahalanobis distance contours, next to their corresponding metric matrices. The Mahalanobis distance is reduced to the Euclidean distance if $\boldsymbol{M} = \boldsymbol{I}$. In principle, *any* definition of distance can be used for the similarity graph. In this thesis, we use the definition in Equation (32).



(a) Circle

(b) Unrotated ellipse 1

(c) Unrotated ellipse 2

(d) Rotated ellipse

Figure 14: Examples of Mahalanobis distance contours in 2D space, along with their defining metric matrix.

Lastly, it is worth noting that, in our current scheme, the edge selection process did not incorporate anything about the similarity of feature vectors. In fact, edge selection is completely separate from the definition of similarity. As a future work, one can incorporate the edge selection process in the similarity definition as well, leading to more informative similarity graphs.

2.2.3 Training with GLR

To use our model, first we split the given dataset $\mathcal{D}$ into training and validation datasets $\mathcal{D}_t$, $\mathcal{D}_v$ with sizes N_t , N_v respectively. For training, we only consider the training nodes (i.e. input-output pairs). Edges between the nodes are selected based on a user-defined algorithm, and the metric matrix $\boldsymbol{M}$ is initialized. We know that the goal of training is to optimize $\boldsymbol{M}$ in a way that the graph signal (i.e. the class labels) becomes smooth on the resulting graph; in short, we want to maximize the smoothness.

Graph Laplacian Regularizer (GLR) is a commonly used prior for signal smoothness in GSP; it is defined as

$$\text{GLR} \triangleq \boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y}, \tag{48}$$

where $\boldsymbol{y}$ is the graph signal and $\boldsymbol{L}$ is the graph Laplacian. We can rewrite Equation (48) as

$$\text{GLR} = \frac{1}{2} \sum_{i=0}^{N_t-1} \sum_{j=0}^{N_t-1} w_{ij}(y_i - y_j)^2, \tag{49}$$

where w_{ij} is the weight of the edge between nodes i, j and y_i is the graph signal's sample on node i. Refer to Appendix A for a proof.

Equation (49) shows that GLR is a linear combination of edge weights (w_{ij}) multiplied by the square of the difference between their end-nodes' samples ($(y_i - y_j)^2$). Minimizing the GLR is equivalent to *minimizing the similarity* (i.e. edge weight) of the nodes with *different* samples. As the difference of the nodes' samples increases, so does the *importance* of minimizing their similarity. Conversely, if the nodes have equal samples (i.e., $y_i = y_j$), then the weight of the edge connecting them (i.e., w_{ij}) is completely removed from the objective.

One might notice that using GLR as an objective only *minimizes* the similarity where suitable; however, it does not *maximize* the similarity between similar nodes. As a result, if the optimization objective consists only of the GLR, then the metric matrix $\boldsymbol{M}$ would tend to infinity, meaning that all nodes would be interpreted as dissimilar; in other words, all edge weights would become zero. To prevent this, we add a penalty term to the training objective; i.e., the training objective does not *only* consist of the GLR as in Equation (48), rather it is given by

$$E \triangleq \boldsymbol{y_t}^T \boldsymbol{L_t} \boldsymbol{y_t} + \mu \, \text{tr}(\boldsymbol{M}), \tag{50}$$

where $\boldsymbol{y_t}$ and $\boldsymbol{L_t}$ are the graph signal and the graph Laplacian when *only* the training nodes are considered, respectively. μ is a free scalar parameter that is set before the optimization process begins, and $\text{tr}(\boldsymbol{M})$ is the trace of the matrix $\boldsymbol{M}$,

$$\operatorname{tr}(\boldsymbol{M}) = \sum_{i=0}^{C-1} M_{ii}. \tag{51}$$

In the following, we explain how the $\operatorname{tr}(\boldsymbol{M})$ penalty prevents the edge weights tending to zero. Suppose that $\boldsymbol{M}$ is real and symmetric (justification will be given later in the section); therefore, $\boldsymbol{M}$ can be decomposed as

$$\boldsymbol{M} = \boldsymbol{U}^T \boldsymbol{\Lambda} \boldsymbol{U}, \tag{52}$$

where columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{M}$ and $\boldsymbol{\Lambda}$ is a diagonal matrix with the corresponding eigenvalues of $\boldsymbol{M}$ on its diagonal. Substituting this into Equation (32) we have

$$\begin{aligned} d_{ij} &= (\boldsymbol{f_i} - \boldsymbol{f_j})^T \boldsymbol{U}^T \boldsymbol{\Lambda} \boldsymbol{U} (\boldsymbol{f_i} - \boldsymbol{f_j}) \\ &= \boldsymbol{x}^T \boldsymbol{\Lambda} \boldsymbol{x} = \sum_{k=0}^{C-1} \lambda_k x_k^2 \\ &, \boldsymbol{x} \triangleq \boldsymbol{U}(\boldsymbol{f_i} - \boldsymbol{f_j}). \end{aligned} \tag{53}$$

The trace of a $C \times C$ matrix is equal to the sum of its eigenvalues

$$\operatorname{tr}(\boldsymbol{M}) = \sum_{k=0}^{C-1} \lambda_k. \tag{54}$$

Adding $\operatorname{tr}(\boldsymbol{M})$ as a penalty term ensures that this sum stays finite. Let the upper bound of $\operatorname{tr}(\boldsymbol{M})$ be p. Since $\boldsymbol{M}$ is PSD, we also know that all of its eigenvalues are non-negative; therefore,

$$0 \leq \sum_{k=0}^{C-1} \lambda_k \leq p < +\infty. \tag{55}$$

Moreover, entries of $\boldsymbol{U}$ are in $[-1, +1]$ range since all eigenvectors are of unit length; meaning that entries of $\boldsymbol{x}$ are finite as well. Let $\max(\boldsymbol{x})$ show the maximum entry of $\boldsymbol{x}$, then

$$0 \leq d_{ij} = \sum_{k=0}^{C-1} \lambda_k x_k^2 \leq p \,.\, \max(\boldsymbol{x})^2 < +\infty. \tag{56}$$

This is how the penalty term upper bounds the distances.

To ensure that $\boldsymbol{M}$ is PSD, we define it as

$$\boldsymbol{M} \triangleq \boldsymbol{B}^T \boldsymbol{B} \,,\, \boldsymbol{B} \in \mathbb{R}^{C \times C}, \tag{57}$$

where $\boldsymbol{B}$ can be any real matrix. Recalling Equation (33), we can write

$$\forall \boldsymbol{x} \in \mathbb{R}^C : \boldsymbol{x}^T \boldsymbol{M} \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{B}^T \boldsymbol{B} \boldsymbol{x} = \|\boldsymbol{B}\boldsymbol{x}\|_2^2 \geq 0 \implies \boldsymbol{M} \succeq 0. \tag{58}$$

Finally, the whole training process is formulated as follows,

$$\begin{aligned} \boldsymbol{B^*} &= \arg \min_{\boldsymbol{B} \in \mathbb{R}^{C \times C}} E \\ &= \arg \min_{\boldsymbol{B} \in \mathbb{R}^{C \times C}} \boldsymbol{y_t}^T \boldsymbol{L_t} \boldsymbol{y_t} + \mu \operatorname{tr}(\boldsymbol{B}^T \boldsymbol{B}) \\ \boldsymbol{M^*} &= \boldsymbol{B^*}^T \boldsymbol{B^*}, \end{aligned} \tag{59}$$

and $\boldsymbol{y_t}$ and $\boldsymbol{L_t}$ depend on the training data $\mathcal{D}_t$. This optimization problem is solved by the Gradient Descent (GD) algorithm [102] in this thesis. Refer to Appendix B for the derivation of loss (which is required by GD). The time complexity of the training process is analyzed in Appendix B as well. In the rest of the thesis, we refer to this training approach as B-GLR; because the objective is GLR-based and the PSDness is ensured by defining $\boldsymbol{M} = \boldsymbol{B}^T \boldsymbol{B}$. Two more training approaches will be introduced in the following sections.

2.2.4 Graph-based Large Margin Nearest Neighbor (GLMNN)

In Section 2.2.3, we saw that the first term of the training objective (i.e. GLR) only accounts for edges between different-labeled nodes. Recall Equation (49); since the nodes' labels are binary in our dataset, we can separate the summation terms based on whether $y_i = y_j$ or $y_i \neq y_j$, thus we have

$$\begin{aligned} \text{GLR} &= \frac{1}{2} \sum_{i \in \mathcal{P}_{-1}} \sum_{j \in \mathcal{P}_{-1}} w_{ij}(-1-(-1))^2 + \frac{1}{2} \sum_{i \in \mathcal{P}_{-1}} \sum_{j \in \mathcal{P}_{+1}} w_{ij}(-1-(+1))^2 \\ &+ \frac{1}{2} \sum_{i \in \mathcal{P}_{+1}} \sum_{j \in \mathcal{P}_{-1}} w_{ij}(1-(-1))^2 + \frac{1}{2} \sum_{i \in \mathcal{P}_{+1}} \sum_{j \in \mathcal{P}_{+1}} w_{ij}(1-1)^2 \\ &= 2 \left[\sum_{i \in \mathcal{P}_{-1}} \sum_{j \in \mathcal{P}_{+1}} w_{ij} + \sum_{i \in \mathcal{P}_{+1}} \sum_{j \in \mathcal{P}_{-1}} w_{ij} \right], \end{aligned} \tag{60}$$

where $\mathcal{P}_{+1}, \mathcal{P}_{-1}$ are the sets of training nodes labeled +1 and -1, respectively. Since the graph is undirected ($w_{ij} = w_{ji}$), we can write

$$\text{GLR} = 2 \left[\sum_{i \in \mathcal{P}_{-1}} \sum_{j \in \mathcal{P}_{+1}} w_{ij} + \sum_{j \in \mathcal{P}_{-1}} \sum_{i \in \mathcal{P}_{+1}} w_{ji} \right] = 4 \sum_{i \in \mathcal{P}_{-1}} \sum_{j \in \mathcal{P}_{+1}} w_{ij}. \tag{61}$$

To prevent the node distances from tending to infinity, a penalty term was added to the objective,

$$E = 4 \sum_{i \in \mathcal{P}_{-1}} \sum_{j \in \mathcal{P}_{+1}} w_{ij} + \mu \operatorname{tr}(\boldsymbol{M}). \tag{62}$$

However, the distances between the same-labeled nodes (i.e. their feature vectors) are still left unused. Incorporating these unused nodes into the objective can increase the accuracy of our model; hence, we introduce a new objective based on the well-known *Large Margin Nearest Neighbour* (LMNN) objective [103]. We call this new objective *Graph-based Large Margin Nearest Neighbour* (GLMNN),

$$E = \sum_{(i,j)\in\mathcal{E}_t | y_i = y_j} d_{ij}(\boldsymbol{M}) + \rho \sum_{(i,j),(i,l)\in\mathcal{E}_t | y_i = y_j = -y_l} \left[d_{ij}(\boldsymbol{M}) + \gamma - d_{il}(\boldsymbol{M})\right]_+ , \tag{63}$$

where $d_{ij}(\boldsymbol{M})$ is the distance between nodes i, j which is a function of the metric matrix $\boldsymbol{M}$. Moreover, $\mathcal{E}_t$ is the set of edges of the training graph, $\rho, \gamma > 0$ are free scalar parameters, and $[\cdot]_+$ is the Relu function,

$$[a]_+ = \left\{ \begin{array}{ll} a, & a \geq 0 \\ 0, & O.W. \end{array} \right\}. \tag{64}$$

The first term of Equation (63) minimizes the distance between same-labeled nodes, while the second term maximizes the distance between different-labeled nodes. The parameter ρ determines the importance of one term relative to the other term. Figure 15 illustrates the reasoning behind the second term. An explanation of this term follows.

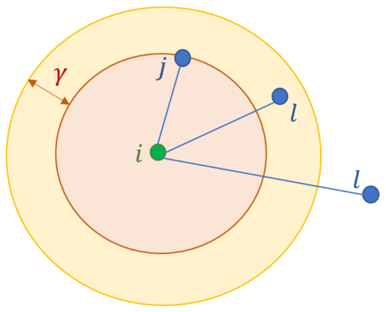


Figure 15: The GLMNN objective accounts for different-labeled nodes by considering a third node, and comparing the distances between the same-labeled pair and the different-labeled pair. The distance between the different-labeled pair should be larger by a specific margin (γ).

For each pair of different-labeled nodes (i,l), all nodes with the same label as i are considered. These third-party nodes are denoted by j. The distance between i and l (i.e. d_{il}) is compared with the distance between i and j (i.e. d_{ij}). If d_{il} is close to d_{ij} by a specific margin (γ), then node l appears in the objective. In other words, if node l falls into the yellow circle in Figure 15, it is considered in the objective; otherwise, node l is deemed *far enough* from node i and left out from the optimization.

The term concerning node l in the objective is $[d_{ij}(\boldsymbol{M}) + \gamma - d_{il}(\boldsymbol{M})]_+$. Minimizing this term means increasing the *difference* of the distances d_{il} and d_{ij} so much so that node l falls outside of the yellow circle in Figure 15. After this happens, the Relu function removes the term concerning node l from the objective; in other words, the objective only considers the different-labeled pairs that are *dangerously* close to each other in a way that might lead to incorrect prediction. This mechanism keeps the objective from becoming infinitely negative by pushing the different-labeled nodes to infinity; i.e., it ensures that the objective is lower bounded.

Another weakness of the GLR-based optimization approach in Section 2.2.3 is the relaxation of the SDP constraint. Recalling Equation (57), we defined $\boldsymbol{M}$ as the multiplication of another matrix $\boldsymbol{B}$ with its own transpose. This might be an over-relaxation as was discussed in Section 2.2.3. Hence, we employ a different technique to ensure $\boldsymbol{M}$'s PSDness in the GLMNN optimization approach. The aforementioned optimization can be summarized as

$$\begin{aligned}\boldsymbol{M}^* = \arg\min_{\boldsymbol{M}\succeq 0} \sum_{(i,j)\in\mathcal{E}_t|y_i=y_j} d_{ij}(\boldsymbol{M}) + \rho \sum_{(i,j),(i,l)\in\mathcal{E}_t|y_i=y_j=-y_l} [d_{ij}(\boldsymbol{M})+\gamma-d_{il}(\boldsymbol{M})]_+ \\ , d_{ij}(\boldsymbol{M}) = (\boldsymbol{f_i}-\boldsymbol{f_j})^T\boldsymbol{M}(\boldsymbol{f_i}-\boldsymbol{f_j}).\end{aligned} \tag{65}$$

First, we rewrite the Mahalanobis distance,

$$\begin{aligned} d_{ij}(\boldsymbol{M}) = (\boldsymbol{f_i}-\boldsymbol{f_j})^T\boldsymbol{M}(\boldsymbol{f_i}-\boldsymbol{f_j}) &= \operatorname{tr}\left((\boldsymbol{f_i}-\boldsymbol{f_j})^T\boldsymbol{M}(\boldsymbol{f_i}-\boldsymbol{f_j})\right) \\ &= \operatorname{tr}\left(\boldsymbol{M}(\boldsymbol{f_i}-\boldsymbol{f_j})(\boldsymbol{f_i}-\boldsymbol{f_j})^T\right) \\ &= \operatorname{tr}(\boldsymbol{M}\boldsymbol{F_{ij}}),\end{aligned} \tag{66}$$

where $\boldsymbol{F_{ij}} \triangleq (\boldsymbol{f_i}-\boldsymbol{f_j})(\boldsymbol{f_i}-\boldsymbol{f_j})^T$. Second, to linearize the Relu function $[\cdot]_+$, we define non-negative auxiliary variable δ_{ijl} that is an upper bound on the distance of the l node to the yellow circle's border in Figure 15,

$$\delta_{ijl} \geq d_{ij}(\boldsymbol{M})+\gamma-d_{il}(\boldsymbol{M}). \tag{67}$$

Then, we replace the Relu function with these upper bounds. If the distance $d_{ij}(\boldsymbol{M})+\gamma-d_{il}(\boldsymbol{M})$ is positive, minimizing its upper bound is equivalent to minimizing itself. However, the upper bound can only decrease to zero since it is non-negative by definition; hence, if the distance is negative, the upper bound stays at zero. This way, negative values of $d_{ij}(\boldsymbol{M})+\gamma-d_{il}(\boldsymbol{M})$ are effectively ignored, which is the same role that $[\cdot]_+$ played before. Finally, the linearized GLMNN optimization is written as

$$\begin{aligned}\boldsymbol{M}^* = \arg &\min_{\boldsymbol{M}\succeq 0,\{\delta_{ijl}\in\mathbb{R}\}} \sum_{(i,j)\in\mathcal{E}_t|y_i=y_j} \operatorname{tr}(\boldsymbol{M}\boldsymbol{F_{ij}}) + \rho \sum_{(i,j),(i,l)\in\mathcal{E}_t|y_i=y_j=-y_l} \delta_{ijl} \\ \text{s.t. } &\delta_{ijl} \geq \operatorname{tr}(\boldsymbol{M}\boldsymbol{F_{ij}})+\gamma-\operatorname{tr}(\boldsymbol{M}\boldsymbol{F_{il}}) \\ &\delta_{ijl} \geq 0.\end{aligned} \tag{68}$$

Equation (68) is a semi-definite program (SDP) [104], meaning it has a linear objective with linear constraints plus a PSD cone constraint for the metric matrix $\boldsymbol{M}$. Many SDP solvers are already developed and implemented such as SeDuMi [105], SDPT3 [106], Gurobi [107], etc. In our experiments, we used SeDuMi implemented in a Matlab package called CVX [108]. From now on, we call this training approach SDP-GLMNN.

SeDuMi's time complexity is $O(C^3+C^{2.5}X+C^{0.5}X^{2.4})$ where C is the number of features ($\boldsymbol{M}\in\mathbb{R}^{C\times C}$), and X is the number of δ_{ijl} variables which depends on the number of edges between same-labeled and different-labeled nodes [109]. If the maximum degree of the nodes is fixed to D_t, then X is smaller than $D_t^2\, N_t$.

Hence, we can rewrite the complexity as $O(C^3 + C^{2.5}N_tD_t^2 + C^{0.5}N_t^{2.4}D_t^{4.8})$. This runtime is intractable for large datasets. The time complexity is reduced by employing a technique called Gershgorin Disc Perfect Alignment (GDPA) [110]. The next section elaborates on this technique.

2.2.5 GDPA Speed-up

Gershgorin Disc Perfect Alignment (GDPA) approximates an SDP with iterative linear programs (LP) to reduce its time-complexity. This technique is introduced in [110], and a Matlab implementation is provided in [111].

GDPA leverages a theorem called the Gershgorin Circle Theorem (GCT) [112]. The GCT states that each real eigenvalue λ of a real symmetric matrix $\boldsymbol{M}$ resides inside at least one *Gershgorin disc* Ψ_i corresponding to row i, with center $c_i \triangleq M_{ii}$ and radius $r_i \triangleq \sum_{j\neq i} |M_{ij}|$; i.e.,

$$\exists i \quad \text{s.t.} \quad c_i - r_i \leq \lambda \leq c_i + r_i. \tag{69}$$

The corollary is that the smallest eigenvalue $\lambda_{\min}$ is lower bounded by the smallest disc left end $\lambda^-_{\min}$; i.e.,

$$\lambda^-_{\min}(\boldsymbol{M}) \triangleq \min_i \; M_{ii} - \sum_{j\neq i} |M_{ij}| \leq \lambda_{\min}(\boldsymbol{M}). \tag{70}$$

Thus, to ensure $\mathbf{M} \succeq 0$, one can enforce linear constraints,

$$\lambda^-_{\min}(\boldsymbol{M}) \geq 0, \tag{71}$$

i.e., all disc left-ends are at least zero.

However, the aforementioned constraints might pose an overrelaxation since many PSD matrices have negative λ^-_{min}'s. This overrelaxation might lead to a low quality optimized metric matrix $\boldsymbol{M^*}$, ultimately causing the model's failure. To address this issue, we take a similarity transform of $\boldsymbol{M}$,

$$\boldsymbol{P} = \boldsymbol{SMS^{-1}}. \tag{72}$$

$\boldsymbol{P}$ shares all the eigenvalues of $\boldsymbol{M}$. We choose $\boldsymbol{S}$ in a way that all disc left ends of $\boldsymbol{P}$ are aligned (i.e. equal to each other). Then, we force all disc left ends of $\boldsymbol{P}$ to be non-negative. This will ensure $\boldsymbol{M}$ is PSD, while not being too restrictive. Substituting $\boldsymbol{P}$ into Equation (71), we have

$$\lambda^-_{\min}(\boldsymbol{P}) \geq 0 \implies M_{ii} - \sum_{j|j\neq i} \left| \frac{s_i M_{ij}}{s_j} \right| \geq 0, \;\; \forall i \in \{0, ..., C-1\}. \tag{73}$$

To ensure the alignment of $\boldsymbol{P}$'s Gershgorin discs, the matrix $\boldsymbol{S}$ should be defined as

$$\begin{aligned} &\boldsymbol{S} = \operatorname{diag}(s_1, ..., s_C) \\ &, s_i = \frac{1}{v_i^*}, \ \forall i \in \{0, ..., C-1\}, \end{aligned} \tag{74}$$

where $\boldsymbol{v^*}$ is the first eigenvector (i.e. corresponding to the smallest eigenvalue) of the optimized metric matrix $\boldsymbol{M^*}$ [110]. This presents a dilemma since we need $\boldsymbol{S}$ to obtain $\boldsymbol{M^*}$, and we need $\boldsymbol{M^*}$ to obtain $\boldsymbol{S}$. As a result, the linear program is solved in an iterative manner, where each step's optimized metric matrix $\boldsymbol{M_k^*}$ determines the next step's similarity transform $\boldsymbol{S_{k+1}}$.

Finally, we can approximate the SDP in Equation (68) as a linear program (LP),

$$\begin{aligned} \boldsymbol{M^*} = \arg & \min_{\boldsymbol{M} \in \mathbb{R}^{C \times C}, \{\delta_{ijl} \in \mathbb{R}\}} \sum_{(i,j) \in \mathcal{E}_t | y_i = y_j} \operatorname{tr}(\boldsymbol{M}\boldsymbol{F_{ij}}) + \rho \sum_{(i,j),(i,l) \in \mathcal{E}_t | y_i = y_j = -y_l} \delta_{ijl} \\ \text{s.t. } & \delta_{ijl} \geq \operatorname{tr}(\boldsymbol{M}\boldsymbol{F_{ij}}) + \gamma - \operatorname{tr}(\boldsymbol{M}\boldsymbol{F_{il}}) \\ & \delta_{ijl} \geq 0 \\ & M_{ii} - \sum_{j | j \neq i} \left| \frac{s_i M_{ij}}{s_j} \right| \geq 0, \ \ \forall i \in \{0, ..., C-1\}, \end{aligned} \tag{75}$$

which is iteratively solved until the minimization objective converges to a constant value.

One linear program takes $O((C+X)^{2.055})$ to compute, where C is the number of features and X is the number of δ_{ijl} variables [113]. If the maximum node degree is fixed to D_t, then X is smaller than $D_t^2 N_t$; thus, we can rewrite the complexity as $O((C+N_t D_t^2)^{2.055})$. In our experiments, we demonstrate that the number of LPs required to approximate the SDP remains constant for various sizes of the dataset; hence, the runtime of the whole algorithm is effectively reduced to $O((C+N_t D_t^2)^{2.055})$. This approach is referred to as GDPA-GLMNN in the rest of the thesis. In conclusion, we have introduced three objectives to use during training: B-GLR(Equation (59)), SDP-GLMNN(Equation (68)), and GDPA-GLMNN(Equation (75)). Any of these objectives can be used in the model.

2.2.6 Validation

After the training, we have obtained the optimized metric matrix $\boldsymbol{M^*}$. In the validation phase, we predict the labels of the validation data $\mathcal{D}_v$, and compare our predictions with the ground-truth labels to compute the accuracy.

Figure 16 illustrates the graph which is used for validation. The blue nodes and edges show the training nodes and the edges between pairs of training nodes; these edges were selected in the training phase. For the validation phase, we add the validation nodes to the graph as well; hence, new edges should be selected which involve at least one validation node. These new edges and the validation nodes are colored red in Figure 16.

Weights of all edges (blue and red) are computed using the nodes' feature vectors and the optimized metric matrix $\boldsymbol{M^*}$. In our current scheme, the selected edges between pairs of *training* nodes do **not** change in the validation phase.

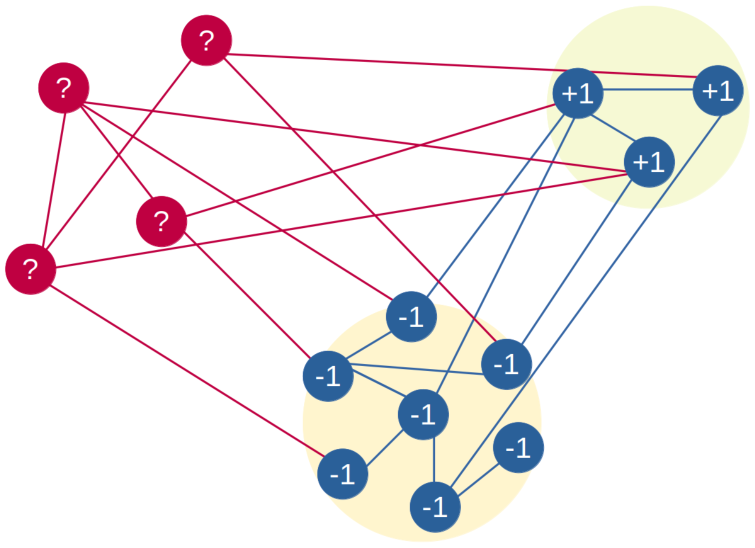


Figure 16: The graph used for validation. Blue nodes and edges were selected in the training phase. Red nodes and edges are newly added for the validation phase.

To estimate the validation labels, we interpolate the graph signal on the validation nodes using the reasonable assumption that the whole signal is smooth. Again, we use the GLR as a smoothness quantity; therefore, the validation loss becomes

$$E' \triangleq \boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y} = \left[\boldsymbol{y_t}^T \; \boldsymbol{y_v}^T\right] \boldsymbol{L} \begin{bmatrix} \boldsymbol{y_t} \\ \boldsymbol{y_v} \end{bmatrix}, \tag{76}$$

where $\boldsymbol{y}$ is the graph signal shown in Figure 16 including both the training nodes' samples $\boldsymbol{y_t}$ and the validation nodes' samples $\boldsymbol{y_v}$, and $\boldsymbol{L}$ is the same graph's Laplacian. This time, the GLR is minimized with respect to the validation labels,

$$\begin{aligned} \boldsymbol{y_v^*} &= \arg \min_{\boldsymbol{y_v} \in \mathbb{R}^{N_v}} E' \\ &= \arg \min_{\boldsymbol{y_v} \in \mathbb{R}^{N_v}} \left[\boldsymbol{y_t}^T \; \boldsymbol{y_v}^T\right] \boldsymbol{L} \begin{bmatrix} \boldsymbol{y_t} \\ \boldsymbol{y_v} \end{bmatrix}. \end{aligned} \tag{77}$$

Since $\boldsymbol{y_v^*}$ has real entries, we discretized the entries to obtain the final binary predictions $\boldsymbol{\hat{y}_v}$,

$$(\hat{y}_v)_i = \left\{ \begin{array}{ll} +1 & (y_v^*)_i \geq 0 \\ -1 & O.W. \end{array} \right\} \quad \forall i \in \{0, ..., N_v - 1\}. \tag{78}$$

At the end, we compute the accuracy by comparing the predictions with the ground-truth labels $\boldsymbol{y_v}$,

$$acc = \frac{\text{sum}(I(\hat{\boldsymbol{y}}_{\boldsymbol{v}} = \boldsymbol{y_v}))}{N_v}. \tag{79}$$

To understand Equation (77) better, let us consider a simple case with only one validation node. Figure 17 illustrates this case.

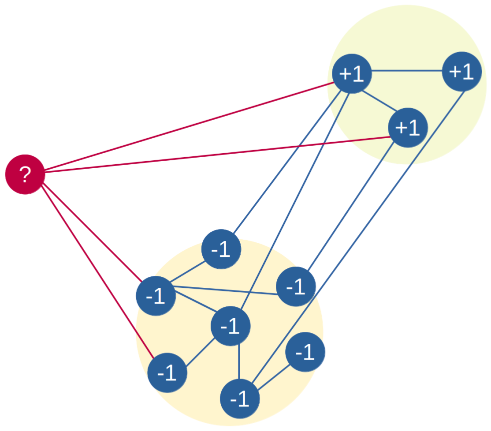


Figure 17: Simple example of a graph with just one validation node.

Using Equation (49), we can rewrite the validation loss as

$$\begin{aligned} E' &= \left[\boldsymbol{y_t}^T y_v\right] \boldsymbol{L} \begin{bmatrix} \boldsymbol{y_t} \\ y_v \end{bmatrix} \\ &= \frac{1}{2}\sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} w_{ij}(y_{t,i}-y_{t,j})^2 + \frac{1}{2}\sum_{i=0}^{N_t-1} w_{i\,N_t}(y_{t,i}-y_v)^2 + \frac{1}{2}\sum_{j=0}^{N_t-1} w_{N_t\,j}(y_v-y_{t,j})^2 + \frac{1}{2}w_{N_t\,N_t}(y_v-y_v)^2. \end{aligned} \tag{80}$$

Since the graph is undirected, we have $w_{ij} = w_{ji} \;\; \forall i, j$; thus, we can simplify Equation (80) as

$$E' = \frac{1}{2}\sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} w_{ij}(y_{t,i}-y_{t,j})^2 + \sum_{i=0}^{N_t-1} w_{i\,N_t}(y_{t,i}-y_v)^2. \tag{81}$$

The first term of Equation (81) only involves the training nodes; therefore, it can be removed from the optimization objective; hence, the optimization formula becomes

$$\begin{aligned} y_v^* &= \arg\min_{y_v\in\mathbb{R}} E' \\ &= \arg\min_{y_v\in\mathbb{R}} \sum_{i=0}^{N_t-1} w_{i\,N_t}(y_{t,i}-y_v)^2. \end{aligned} \tag{82}$$

Let $\mathcal{P}_{+1}, \mathcal{P}_{-1}$ denote the sets of nodes labeled +1 and -1, respectively. Then, we can separate these nodes in the objective as follows,

$$y_v^* = \arg \min_{y_v \in \mathbb{R}} \sum_{i \in \mathcal{P}_{-1}} w_{i\,N_t}(-1-y_v)^2 + \sum_{i \in \mathcal{P}_{+1}} w_{i\,N_t}(+1-y_v)^2, \tag{83}$$

hence, the objective is just a weighted sum of the weights of the edges connecting the validation node to training nodes. To illustrate this even further, one can expand the polynomials in Equation (83) and write the objective as

$$y_v^* = \arg \min_{y_v \in \mathbb{R}} \; y_v^2 \left[\sum_{i=0}^{N_t-1} w_{i\,N_t}\right] + 2y_v \left[\sum_{i \in \mathcal{P}_{-1}} w_{i\,N_t} - \sum_{i \in \mathcal{P}_{+1}} w_{i\,N_t}\right]. \tag{84}$$

The first term in Equation (84) keeps the magnitude of y_v^* in a reasonable range, while the second term determines its value based on its cumulative similarity to +1 nodes or -1 nodes. As a result, the closer y_v^* is to an end of the interval $[-1,1]$, the more confident we are in the prediction. y_v^* is then thresholded with boundary 0 to obtain $\hat{y_v}$ which is either -1 or +1.

Equation (77) has a closed form solution [114],

$$\begin{aligned}\boldsymbol{y_v^*} &= \arg \min_{\boldsymbol{y_v} \in \mathbb{R}^{N_v}} \left[\boldsymbol{y_t}^T \, \boldsymbol{y_v}^T\right] \begin{bmatrix} \boldsymbol{L}_{11} & \boldsymbol{L}_{12} \\ \boldsymbol{L}_{21} & \boldsymbol{L}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{y_t} \\ \boldsymbol{y_v} \end{bmatrix} \\ &\implies \boldsymbol{y_v^*} = -(\boldsymbol{L}_{22})^{-1}\, \boldsymbol{L}_{21}\, \boldsymbol{y_t},\end{aligned} \tag{85}$$

which can be viewed as filtering the graph signal $\boldsymbol{y_t}$ by the filter $\boldsymbol{G}$ to obtain a new graph signal $\boldsymbol{y_v^*}$,

$$\boldsymbol{G} \triangleq -(\boldsymbol{L}_{22})^{-1}\, \boldsymbol{L}_{21}. \tag{86}$$

2.3 Programming Notes

2.3.1 Data

The data used in this thesis, accessible in [115], is collected as part of the study in [116]. A tiger salamander was euthanized and the retina was separated from its eye under dim illumination. Then, the retina was flattened and connected to a multi-electrode array from its ganglion cell side. A movie was played on a screen in front of the retina (where the retina could "see" it) while the multi-electrode array recorded the response at the end of the ganglion cells.

The movie is grayscale and shown at 60 frames per second. It is made to imitate a salamander's point of view. To illustrate, six random frames of the movie are shown in Figure 18. There are 1141 frames in total corresponding to a 19-second-long video clip. This clip was repeated 297 times during the experiment. The height and width of frames are 120×200 pixels; therefore, all of the input video can be stored as a Numpy array of size $1141 \times 120 \times 200$.

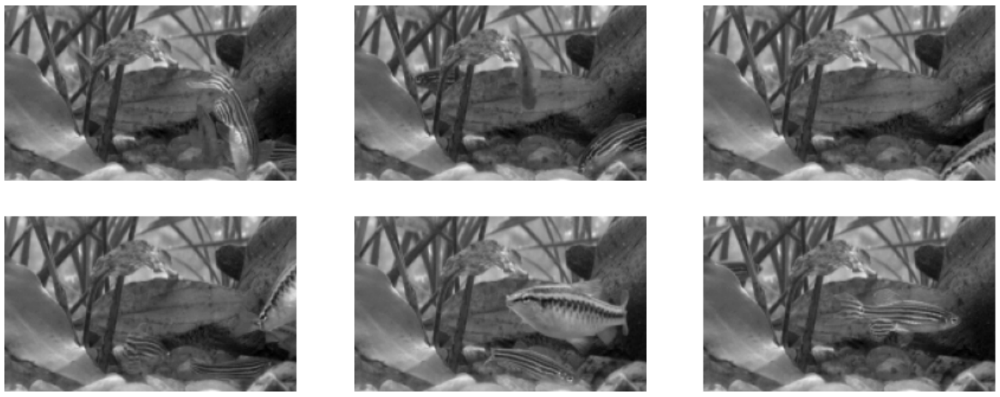


Figure 18: Random frames of the stimulus shown to the retina

The response signals were later sorted by a custom software to delineate each neuron's spikes through time. The software was specifically developed to be used with the multi-electrode array [97]. 234 neurons passed the standard tests for waveform stability and lack of refractory period violations. Of those, 160 cells whose firing rates were most stable across stimulus repeats were made accessible in [115]. Of these remaining neurons, 113 cells which had the highest quality are selected. This final selection of cells might be because of decreases in cells' responsiveness over time due to photobleaching, increases in their spiking as the cells die, etc.

Afterwards, the signals are discretized into $\frac{1}{60}$ second time bins and synced to the frames, so that a different frame is playing during each time bin. Moreover, the responses are binarized; each time bin with at least one action potential is given the value +1, while all other time bins are given the value -1. Consequently, all of the responses are stored in a binary tensor with dimensions of "repeat $\times$ frame in a repeat $\times$ cell", i.e., $297 \times 1141 \times 113$.

We ranked the neurons based on the amount of information in their response signals, measured by the following equation,

$$I(\text{cell};\text{stimulus}) = \frac{1}{T}\sum_t \frac{r(t)}{\bar{r}}\ \log_2\left(\frac{r(t)}{\bar{r}}\right), \tag{87}$$

where T is the total number of time bins , $r(t)$ is the firing rate in each bin (calculated by taking the mean response over all repetitions), and $\bar{r}$ is the mean response over the entire movie. Justification for this formula can be found in [117].

The quantity in Equation (87) is calculated for each cell, and cells are ranked by this measure in descending order. Top ranked cells contain the most amount of information in their spike signals; therefore, they are most suitable for analysis. Figure 19 shows responses of the 10 most informative cells during all 1141 frames of the first repeat of the movie. Black dots mark +1 values while white spaces represent -1. The neurons' identifying indices in the dataset are shown on the y-axis; neurons at the top are more informative than neurons at the bottom.

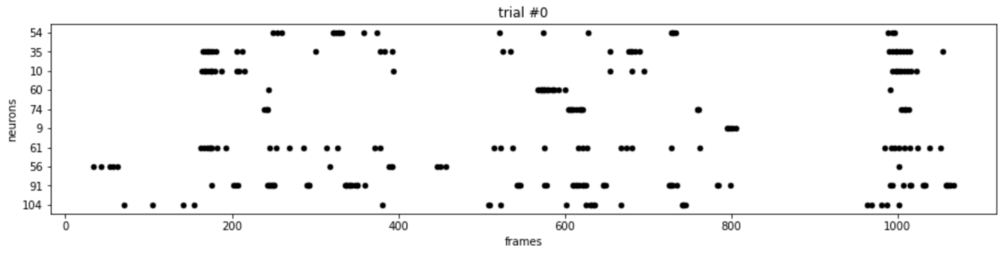


Figure 19: Responses of the most informative neurons during the first repeat of the movie. Black dots represent spikes. Y-axis shows the neurons' indices in the dataset.

Grouping

Only 3.00% of the time bins are labeled +1 among all neurons and all repeats of the whole movie. This sparsity complicates the assessment of our model's accuracy, since one can achieve a staggering 97% accuracy by simply labeling all time bins -1. As a result, we considered a group of all 113 neurons. The group's response in a specific time bin is defined as +1 if any of the neurons spikes in that time bin, and -1 otherwise. In other words, the group's response is equivalent to the OR of all neurons' responses. The group's response is +1 in 68.47% of the time bins which makes a more balanced dataset.

Nevertheless, one can devise better grouping techniques. For example, grouping ganglion cells based on their type leads to more meaningful results since each type responds to a specific feature of the visual input [24]. However, to group cells based on their type, their responses to random noise (i.e. checkerboard stimulus) is required which is currently out of our reach.

2.3.2 Feature Extraction

The group's response (as described in Section 2.3.1) comprises a dataset of 297×1141 time bins each holding a binary label ($\{+1, -1\}$). As explained in Section 2.2.1, each time bin must be associated with a number of frames preceding and including the frame synced to the time bin in question. We call these frames the "frame batch" of the time bin. Various features are extracted from the frame batches to form the feature vectors that are given to the similarity graph (Recall Equation (29)). The number of frames in a batch is determined based on the utilized feature extraction algorithm.

This thesis uses three methods of feature extraction. The first method is filtering the batches with the pre-trained first-layer filters of a CNN called "slowfast-r50" accessible in PyTorchVideo Library [118]. The CNN is introduced in an accompanying paper [119]. The second method is a 3D version of the well-known *Scale Invariant Feature Transform* (SIFT) [87], introduced in [100]. The code is made publicly accessible on GitHub [120]. The third method is using the filters of a neural network developed for simultaneous audio-video texture analysis called SOE-Net [121]. SOE-Net's code is accessible in [122]. CNN features and SIFT features both use batches of 32 frames. SOE-Net uses batches of 42 frames.

Lastly, we needed a completely different dataset to compare the fish movie's result with. Therefore, we extracted SIFT features from the MNIST dataset [123], which consists of images of hand-written digits from 0 to 9 and their labels. Since the fish movie dataset has binary labels, we only selected 0 and 1 digits from MNIST. The SIFT algorithm works by finding keypoints in the image which are invariant to rotation, scaling and illumination. These keypoints are also resistant to changes in contrast and the viewpoint's angle to some degree. Each keypoint is then assigned a descriptor. A descriptor is a 128-dimensional vector which describes a local area around the keypoint. The SIFT algorithm is thoroughly described in Section 2.3.3.

An image can have any number of keypoints. This causes an issue for our model since we need all feature vectors to be of the same length. To solve this issue, we disregarded all images that had fewer than two keypoints, and kept exactly two keypoints in the remaining images. To choose the keypoints, we computed the distance of all keypoints in an image to the top-left and bottom-right corners; then, we kept one keypoint closest to each corner. The reason behind this decision was to urge the descriptors to cover the largest possible area of the image. Nevertheless, more sophisticated approaches for keypoint selection can be made in the future. After selecting the keypoints, their descriptors are concatenated to form 256-dimensional feature vectors. These feature vectors are then used for training and validation.

The 3D-SIFT method follows a similar approach. The only difference is that the keypoints are found in 3D blocks representing frame batches instead of 2D images. Moreover, the descriptors are 768 dimensional and describe a local *cube* around each keypoint. The 3D-SIFT algorithm is described in Section 2.3.3 in more depth. Predictably, the same issue regrading the number of keypoints arises with 3D-SIFT; therefore, we used a similar keypoint selection approach to address it. For the 3D-SIFT features, we chose only one keypoint closest to the center of the frame batch.

For the CNN features, we downloaded the pretrained model. Then, replaced all layers after the first one with identity layers (i.e. output equals input). Then, used the model's output as features. All feature vectors (in all feature extraction methods) are subsampled as required to increase processing speed.

After equipping each time bin with a feature vector, random data points (i.e. time bins) between a minimum and a maximum time index are chosen to create the training and validation datasets. An option is provided in the code to ensure that each set contains an equal number of +1 and -1 nodes. By changing the minimum and maximum time index, one can choose only a certain repeat of the 19-seconds video clip for the model instead of the whole dataset.

2.3.3 SIFT and 3D-SIFT

To realize the advantage of our model's explainability, we need to know how the features are extracted. This section explains two utilized feature extraction algorithms in this thesis which are fully explainable; SIFT and 3D-SIFT.

A) SIFT

Scale-Invariant Feature Transform (SIFT) was first introduced in 2004 [87]. SIFT is usually used for image matching; i.e. determining whether two images are taken from the same object. SIFT is invariant to rotation and scaling. Moreover, it is resistant to changes in illumination and small changes in the viewpoint's angle. Furthermore, SIFT features are local; hence, if parts of the object are obstructed by clutter in the scene, SIFT can still match the visible parts.

There are four steps in SIFT:

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

In the first step, the image is blurred with a Gaussian kernel at different scales (i.e. σ). The scale is multiplied by a constant k at each step ($\sigma_{i+1} = k\sigma_i$). Following this, the image is subsampled with 2:1 rate and blurred again at various scales. This process continues for a given number of times. Each set of blurred images with a certain size is called an "octave". The Difference of Gaussian (DoG) for consecutive pairs of images are calculated in all octaves. Figure 20 illustrates two octaves, different scales in each octave, and the calculated DoGs.

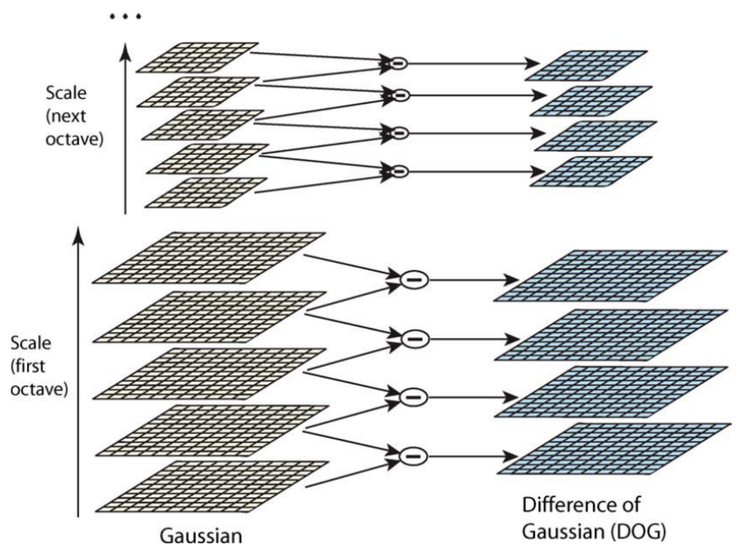


Figure 20: The Scale-space and Difference of Gaussian in SIFT.

After the DoGs are obtained, they are searched for local extrema. Each pixel is compared with 26 other pixels; its 8 neighbors in the same scale, 9 in its previous scale and 9 in its next scale. If the pixel in question

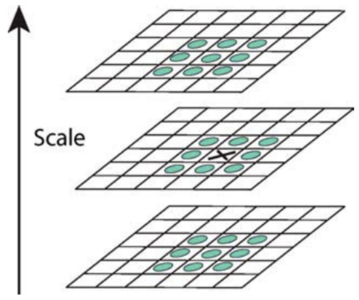


Figure 21: Search for extrema in the DoG pyramid.

is a local extremum, it is flagged as a potential keypoint. Figure 21 illustrates this search for extrema. The paper provides empirically-found optimum parameters for this step: number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6$, $k = \sqrt{2}$.

In the second step (keypoint localization), the potential keypoints that were found in the first step are refined. SIFT utilizes Taylor series expansion of the scale-space to obtain more accurate location of extrema, and if the intensity at a certain extremum is lower than a threshold (0.03 in the paper), it is rejected. Furthermore, some of the extrema found in the first step may be located on edges (which are undesirable) instead of corners. SIFT computes a 2×2 Hessian matrix and its eigenvalues. If the extremum is on an edge, one eigenvalue is significantly larger than the other. SIFT removes all extrema for which the ratio of the eigenvalues is larger than a threshold (10 in the paper). As a result, low-contrast extrema and edge extrema are removed; therefore, only strong keypoints remain.

In the third step, an orientation is assigned to each keypoint to achieve invariance to rotation. A neighborhood around each keypoint is considered, and the gradient magnitude and direction are calculated in that region. The size of the neighborhood depends on the keypoint's scale. An orientation histogram with 36 bins covering 360 degrees is created. The highest peak of the histogram determines the keypoint's orientation. If other bins in the histogram are above 80% of the highest bin, other keypoints are created in the same location and scale, but with different orientations. If the image is rotated by θ degrees, all the keypoints' orientations are also rotated by θ; hence, SIFT can build invariance to rotation.

Figure 22 shows SIFT keypoints on an image. The size of the circles around the keypoints are proportional to the keypoints' scales. The keypoints' orientations are illustrated with lines inside the circles. It is observed that some circles have more than one line; these show multiple keypoints in the same location and scale but with different orientations. Figure 22 is taken from [124].

In the fourth step, a 128-dimensional vector is assigned to each keypoint which describes a neighborhood around the keypoint; the vector is called the keypoint's "descriptor". A 16×16 neighbourhood around each keypoint is considered. Then, it is divided into 16 sub-blocks of 4×4 size. The gradient magnitude and di-

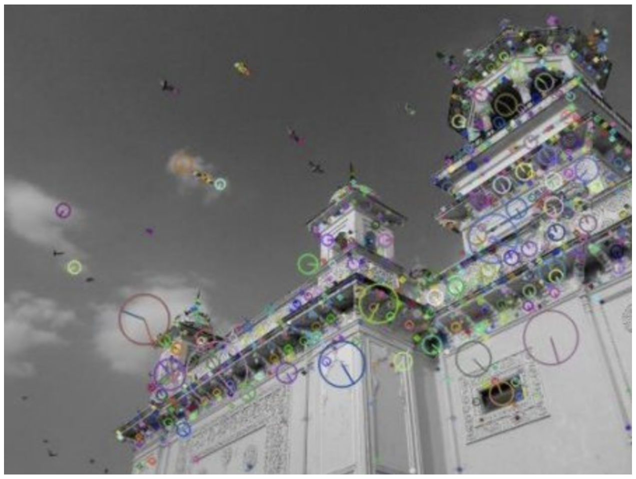


Figure 22: SIFT keypoints with various scales and orientations.

rection, relative to the keypoint's orientation, are calculated in all sub-blocks. Then, an orientation histogram is created for each sub-block with 8 bins covering 360 degrees. All the histograms are concatenated to create the descriptor. Since there are 16 histograms each with 8 bins, the resulting descriptor has 128 dimensions.

Figure 23 illustrates the calculation of the descriptor. The keypoint is located in the center of the 16×16 block at the left. Black arrows in the middle block represent gradient vectors. In the rightmost block, orientation histograms are drawn inside their relevant sub-block. The magnitude of an arrow in a histogram shows the accumulated magnitude of all gradients in that direction. Figures 20, 21, and part of Figure 23 are taken from the original paper [87].

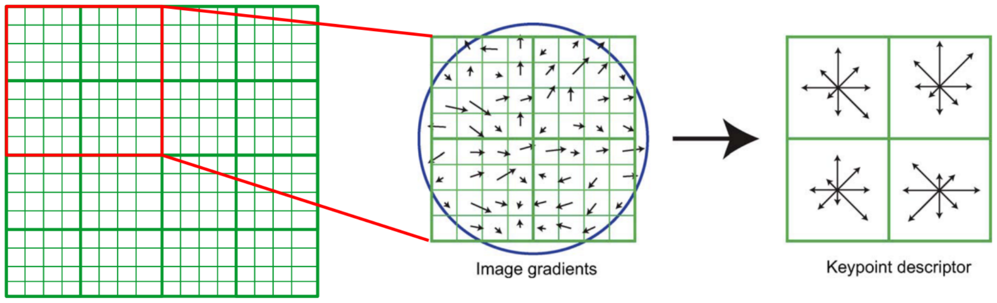


Figure 23: A 16×16 neighborhood around the keypoint is divided into sixteen 4×4 sub-blocks. Orientation histograms are created for all sub-blocks, each with 8 bins covering 360 degrees.

One application of SIFT is matching an image to another image in a given set. To carry out this task, the descriptors of the test image are matched to their nearest neighbors in all of the given images. Sometimes, a keypoint in the test image does not have any correct match in the given set because it arises from background clutter or noise. To prevent false matches, the ratio of the distance to the closest neighbor and second-closest neighbor is computed. If the ratio is greater than 0.8 the match is rejected. This approach eliminates around 90% of false matches while discarding only 5% of correct matches, as per the paper.

B) 3D-SIFT

3D-SIFT finds keypoints and descriptors in 3D blocks of pixels representing 3D images (or short video clips) in the same fashion that SIFT finds them on 2D images. Several algorithms have been introduced as a 3D version of SIFT [125, 126, 127, 128]. In this thesis, we use the one introduced in [100], and implemented in [120].

In the first step of the 3D-SIFT algorithm, the scale-space is created by blurring the 3D image with Gaussian kernels of various scales, and subsampling the 3D image between octaves. Then, the extrema of the DoG pyramid are flagged as potential keypoints. Small modifications are made in this step compared to SIFT.

In the second step, potential keypoints whose intensities are lower than a threshold are rejected. The threshold depends on the contrast of the whole 3D image instead of being a fixed number as in SIFT. The proposed 3D-SIFT in [100] does not use Taylor series expansion for better localization of keypoints since it did not significantly improve their results.

In the third step, a neighborhood around each keypoint is considered; the size of the neighborhood depends on the keypoint's scale. Then, gradient components are computed in this neighborhood, and the correlation of gradient components (otherwise known as the "structure tensor") is obtained by the following equation,

$$\boldsymbol{K} = \int \omega(\boldsymbol{x})\ \nabla I(\boldsymbol{x})\ \left(\nabla I(\boldsymbol{x})\right)^T\ d(\boldsymbol{x}) \quad ,\boldsymbol{K} \in \mathbb{R}^{3\times 3}, \tag{88}$$

where $\nabla I(\boldsymbol{x})$ is the gradient of image I at location $\boldsymbol{x}$, and $\omega(\boldsymbol{x})$ is a Gaussian window centered at the keypoint, the width of which is a constant multiple of the keypoint scale. Afterwards, the eigendecomposition of $\boldsymbol{K}$ is calculated,

$$\boldsymbol{K} = \boldsymbol{R}\,\boldsymbol{\Lambda}\,\boldsymbol{R}^T \quad ,\boldsymbol{R} \in \mathbb{R}^{3\times 3}, \tag{89}$$

and matrix $\boldsymbol{R}$ is assigned to the keypoint as its orientation in 3D space. Additional notes are given in the paper to make sure that $\boldsymbol{R}$ is unique, and to remove degenerate keypoints which cannot be reliably oriented.

In the fourth step, the descriptors are constructed. A neighborhood around the keypoint is considered; the size of the neighborhood depends on the keypoint's scale. The neighborhood is divided into a $4 \times 4 \times 4$ array of cubical sub-regions. These sub-regions are illustrated in Figure 24. Gradient magnitude and direction are computed and an orientation histogram is formed in each sub-region. Histogram bins represent vertices of a regular icosahedron. An image of the icosahedron is provided in Figure 25a.

When a gradient vector intersects with one of the icosahedron's faces, its magnitude is interpolated onto the three vertices of that face; hence, parts of the gradient's magnitude are considered for all three bins corresponding to the three vertices. This interpolation is illustrated in Figure 25b. Both images in Figure 25

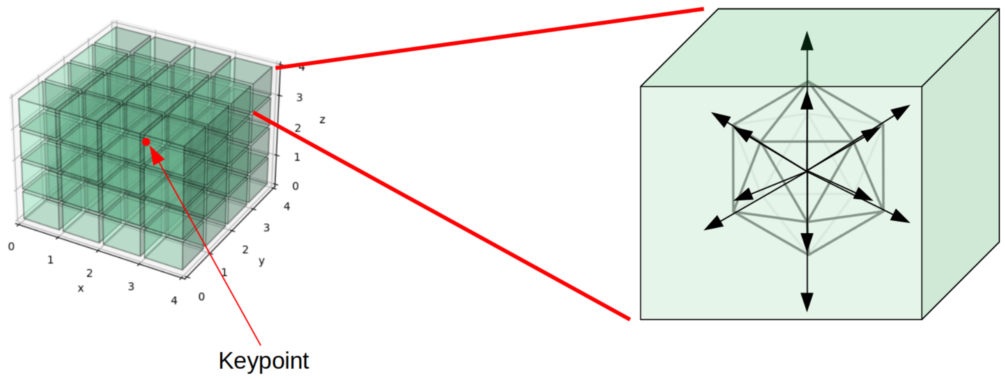


Figure 24: 64 sub-regions around the keypoint are considered in 3D-SIFT. Orientation histograms are computed for all of them and concatenated to form the descriptor. Histogram bins correspond to vertices of a regular icosahedron.

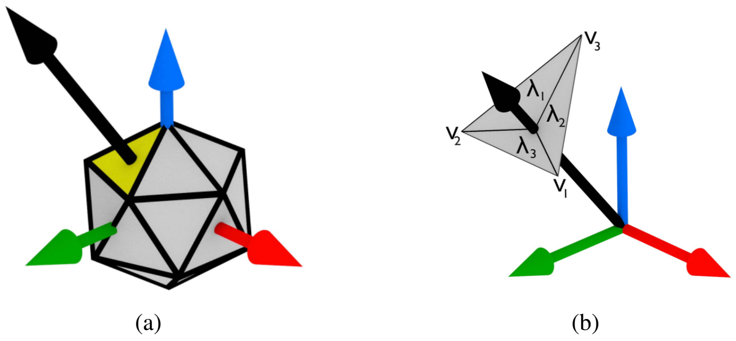


Figure 25: (a) Vertices of a regular icosahedron are used as bins in orientation histograms to create 3D-SIFT descriptors. (b) The gradient vector is interpolated onto the three vertices of the intersected face. The interpolated values are accumulated for all vertices to form the orientation histogram.

are taken from the paper [100]. The orientation histograms are concatenated to form the descriptor. Since the icosahedron has 12 vertices, and there are 64 sub-regions around the keypoint, the descriptor has 768 dimensions.

2.3.4 Training Graph Construction

As discussed in Section 2.2.2, if all pairs of training nodes are connected by an edge (i.e. we have a complete training graph), the training process becomes prohibitively time-consuming; therefore, only a certain number of edges should be selected for the graph. Two strategies for selecting edges are developed in this thesis. In both strategies, the degrees of all nodes are upper-bounded by a given number called D_t; therefore, the number of all edges in the training graph is less equal to $\frac{1}{2}D_tN_t$ where N_t is the number of training nodes. Both strategies are described below.
Algorithm 1 selects edges randomly, while Algorithm 2 selects edges based on the temporal closeness of their end-nodes; the closer the end-nodes' corresponding time bins are, the better.

Algorithm 1 Random Edge Selection Between Pairs of Training/Validation Nodes

```
Require: $N_t, D_t$
Ensure: $0 \leq D_t \leq N_t$
1: for $node_i \leftarrow 1$ to $N_t$ do
2:     candidates $\leftarrow$ list of all training nodes (excluding $node_i$) with a lower degree than $D_t$
3:     if len(candidates) $\leq D_t - degree(node_i)$ then
4:         connect all nodes in the candidates list to $node_i$
5:     else if len(candidates) $> D_t - degree(node_i)$ then
6:         $temp \leftarrow D_t - degree(node_i)$ randomly chosen nodes from the candidates list
7:         connect all nodes in $temp$ to $node_i$
8:     end if
9: end for
```

Algorithm 2 Time-based Edge Selection Between Pairs of Training/Validation Nodes

```
Require: $N_t, D_t$
Ensure: $0 \leq D_t \leq N_t$
1: for $node_i \leftarrow 1$ to $N_t$ do
2:     candidates $\leftarrow$ list of all training nodes (excluding $node_i$) with a lower degree than $D_t$
3:     if len(candidates) $\leq D_t - degree(node_i)$ then
4:         connect all nodes in the candidates list to $node_i$
5:     else if len(candidates) $> D_t - degree(node_i)$ then
6:         define $time(node_i) \triangleq$ the index of the time bin corresponding to $node_i$
7:         $t \leftarrow |time(node_i) - time(node_j)| \; \forall j \leftarrow 1$ to $N_t, j \neq i$
8:         sort all nodes in the candidates list based on their corresponding value in $t$
9:         connect the first $D_t - degree(node_i)$ nodes in the sorted candidates list to $node_i$
10:     end if
11: end for
```

The training graph is denoted by $\mathcal{G}_t$

$$\mathcal{G}_t = \{\mathcal{V}_t, \mathcal{E}_t\}, \tag{90}$$

where $\mathcal{V}_t$ is the list of all training nodes, and $\mathcal{E}_t$ is the list of all edges between training nodes.

2.3.5 Validation Graph Construction

As discussed in Section 2.2.6, after the training is done, new edges should be selected which involve at least one validation node. These edges are separated into two categories: edges between pairs of validation nodes, and edges between one validation node and one training node. To select edges connecting pairs of validation nodes, similar algorithms to Algorithm 1 or Algorithm 2 are used. The only difference is that the parameter determining the maximum degree is noted by D_v instead of D_t.

To select edges between validation and training nodes, an equal number of training nodes from each label ($\{+1,-1\}$) are selected; the training nodes can be selected randomly or based on their temporal closeness to the validation nodes. The number of all training nodes connected to each validation node is given by D_{vt}. Algorithms 3, 4 illustrate the edge selection process. The sets of training nodes labeled +1 and -1 are represented by $\mathcal{P}_{+1}, \mathcal{P}_{-1}$, respectively.

Algorithm 3 Random Edge Selection Between a Validation Node and Training Nodes

```
Require: $\mathcal{P}_{+1}, \mathcal{P}_{-1}, N_v, D_{vt}$
1: $N_{+1} \leftarrow len(\mathcal{P}_{+1})$ and $N_{-1} \leftarrow len(\mathcal{P}_{-1})$
2: Ensure $0 \le \lceil \frac{D_{vt}}{2} \rceil \le N_{-1}$
3: Ensure $0 \le \lfloor \frac{D_{vt}}{2} \rfloor \le N_{+1}$
4: for $node_i \leftarrow 1$ to $N_v$ do
5:     candidates $\leftarrow$ list of $\lceil \frac{D_{vt}}{2} \rceil$ randomly chosen training nodes with label -1
6:     connect $node_i$ to all nodes in the candidates list
7:     candidates $\leftarrow$ list of $\lfloor \frac{D_{vt}}{2} \rfloor$ randomly chosen training nodes with label +1
8:     connect $node_i$ to all nodes in the candidates list
9: end for
```

After selecting the new edges, we have an extended graph comprising of both training and validation nodes as in Figure 16. The upper bound of a training node's degree in this extended graph is $D_t + D_v$, and the upper bound of a validation node's degree is $D_v + D_{vt}$.

As a last note, the objective's free parameter μ is optimized by a cursory trial and error; i.e., we tried various values and chose one that yielded the best results. In the future, more sophisticated hyperparameter optimization should be carried out.

Algorithm 4 Time-based Edge Selection Between a Validation Node and Training Nodes

```
Require: $\mathcal{P}_{+1}, \mathcal{P}_{-1}, N_v, D_{vt}$
 1: $N_{+1} \leftarrow len(\mathcal{P}_{+1})$ and $N_{-1} \leftarrow len(\mathcal{P}_{-1})$
 2: Ensure $0 \leq \lceil \frac{D_{vt}}{2} \rceil \leq N_{-1}$
 3: Ensure $0 \leq \lfloor \frac{D_{vt}}{2} \rfloor \leq N_{+1}$
 4: for $node_i \leftarrow 1$ to $N_v$ do
 5:     define $time(node_i) \triangleq$ the index of the time bin corresponding to $node_i$
 6:     $t \leftarrow |time(node_i) - time(node_j)|$ , $\forall j \in \mathcal{P}_{-1}$
 7:     sort all nodes in $\mathcal{P}_{-1}$ based on their corresponding value in $t$
 8:     connect $node_i$ to the first $\lceil \frac{D_{vt}}{2} \rceil$ nodes of the sorted $\mathcal{P}_{-1}$
 9:     $t \leftarrow |time(node_i) - time(node_j)|$ , $\forall j \in \mathcal{P}_{+1}$
10:     sort all nodes in $\mathcal{P}_{+1}$ based on their corresponding value in $t$
11:     connect $node_i$ to the first $\lceil \frac{D_{vt}}{2} \rceil$ nodes of the sorted $\mathcal{P}_{+1}$
12: end for
```

2.3.6 Validation Time-Complexity

To directly solve Equation (85), one can use the row reduction method to compute the inverse of $\boldsymbol{L_{22}}$ which takes $O(N^3)$ operations [129], then multiply $\boldsymbol{L_{22}}^{-1}$ by $\boldsymbol{L_{21}}$ by $\boldsymbol{y_t}$ which takes $O(N^3)$ again. To reduce complexity, we take advantage of the fact that $\boldsymbol{L}$ (and consequently $\boldsymbol{L_{22}}$ and $\boldsymbol{L_{21}}$) are sparse. We can write Equation (85) as a system of linear equation,

$$\boldsymbol{L_{22}}\, \boldsymbol{y_v^*} = -\boldsymbol{L_{21}}\, \boldsymbol{y_t}. \tag{91}$$

Multiplying $\boldsymbol{L_{21}}$ by $\boldsymbol{y_t}$ takes $O(m)$ time where m is the number of non-zero elements in $\boldsymbol{L_{21}}$. Since $\boldsymbol{L_{21}}$ is sparse, we have $m \in O(N_v)$.

Now, we have a system of linear equations whose coefficient matrix ($\boldsymbol{L_{22}}$) is sparse, positive definite, symmetric and real. In this case, we can use the *Conjugate Gradient Algorithm* [130] to compute $\boldsymbol{y_v^*}$ in $O(n\sqrt{\kappa})$ time, where n is the number of non-zero elements in $\boldsymbol{L_{22}}$ and κ is the condition number of $\boldsymbol{L_{22}}$ [131]. Typically, $\kappa \in O(N_v^{\frac{2}{d}})$ for d-dimensional spaces [131]. In our case, we have a N_v-dimensional space, so the complexity of the whole algorithm is $O(N_v^{1+\frac{2}{N_v}})$ which tends to $O(N_v)$ for large N_v.

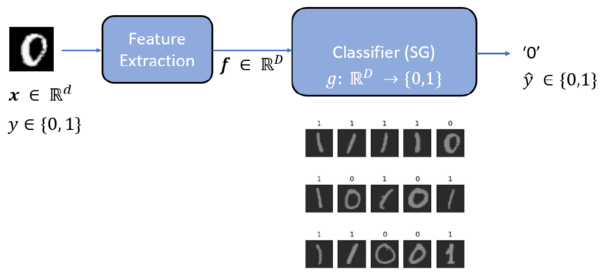
3 Experiments

At the beginning of this research, we implemented a simple version of our model to gauge its potential. The results of our experiment with this preliminary implementation are given in Section 3.1. Since the experiment's results were promising, we continued to implement a large and modular version of the model, with various visualization and experimentation tools. The quantities that are used to assess the model's performance in its final implementation are explained in Section 3.2.

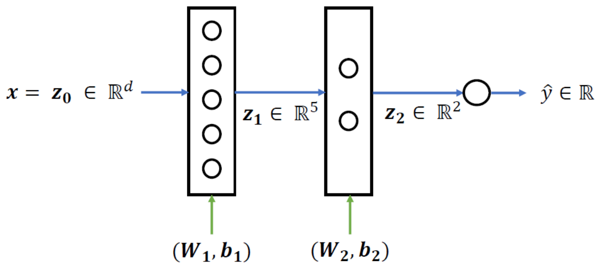
Section 3.3 explores the advantages of our model's explainability using two feature sets: SIFT and 3D-SIFT. Conversely, Sections 3.4, 3.5 and 3.6 are about performance; our model's performance is investigated on all three training optimizations (see Section 2.2), all feature sets in Section 2.3.2, and several benchmark approaches. Finally, Sections 3.7, 3.8, 3.9, and 3.10 explore the effect of changing the hyperparameters on performance.

3.1 Preliminary Experiments

To test our model, we first implemented a quick and simple example of our model and compared it with an equivalently simple neural network. Figure 26 illustrates the architectures of both methods. Figure 26a show the similarity graph model. It comprises two blocks; feature extraction and classification. The data used in this experiment consist of 0 and 1 digits from the MNIST [123] dataset; thus, the labels are binary ($\{0,1\}$). The feature extraction block finds SIFT [87] keypoints in images, chooses two keypoints for each image and concatenates their descriptors to form 256-dimensional feature vectors as described in Section 2.3.2.



(a) Block diagram of the similarity graph model



(b) The neural network

Figure 26: A simple example of the similarity graph model is compared with a simple neural network to benchmark its performance

The second block in Figure 26a illustrates the similarity graph model which is described in Section 2.2. In this experiment, the training graph is complete; meaning that all pairs of training nodes are connected by edges. The validation graph is also complete; meaning that all validation nodes are connected to all training nodes and all other validation nodes. The optimization formula solved during the training is presented in Equation (59). This formulation is explained in Section 2.2.3. Furthermore, Equation (85) is solved during validation. In this experiment, the inverse of $\boldsymbol{L_{22}}$ is computed which is time-consuming; therefore, this part is replaced with the Conjugate Gradient algorithm (see Section 2.3.6) in the final implementation.

Figure 26b illustrates the neural network that benchmarks our model. It has a simple architecture with only one hidden layer and five hidden neurons. Hidden neurons' activation function is Relu as defined in Equation (64). The last layer has a Softmax activation instead of Relu; the Softmax function is defined in Equation (92),

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{92}$$

where $\sigma(z)_i$ shows the ith element of the function's output vector. In Figure 26b, $\boldsymbol{z} = \boldsymbol{W_2}\, \boldsymbol{z_1} + \boldsymbol{b_2}$, and K is the dimensionality of $\boldsymbol{z}$ which is 2. The last circle in Figure 26b, right before the prediction, represents a maximizing function where the label with the highest corresponding value in the last layer is chosen as the final prediction.

The loss function of this neural network is cross-entropy, a commonly used loss for machine learning. $\boldsymbol{W_1}, \boldsymbol{W_2}$ and $\boldsymbol{b_1}, \boldsymbol{b_2}$ are the weights that get optimized during training. The optimization is done by the Gradient Descent (GD) algorithm [102]. This algorithm finds a local minimum by iteratively moving in the opposite direction of the objective's gradient. The number of iterations run in each GD optimization is set to 10 for this experiment. Correspondingly, the similarity graph's training optimization in Equation (59) is also solved by the GD algorithm with 10 iterations. We chose 10 iterations because the objectives seem to stay almost constant after 10 iterations.

Figure 27 shows the accuracies obtained by the two methods for various sizes of the validation dataset. The ratio of the training set size to the validation set size is 3 to 2; in other words, as the validation set size grows, so does the training set size. However, only the validation set size is shown on the plot. To randomize the experiment, we repeat the experiment many times for each validation set size. Each blue point on the plot shows the mean accuracy obtained over 10000 runs of the neural network. The blue error bars show the standard deviation of the obtained accuracies. The same is true for orange points and error bars; the only difference is that the similarity graph experiments are repeated 500 times for each point instead of 10000. The reason is the similarity graph model's long runtime.

One might notice that the similarity graph model has horizontal error lines as well as vertical ones. The horizontal lines are due to the fact that some MNIST images are discarded during feature extraction because the contain fewer than two keypoints. Hence, the final size of the validation set my be smaller than the initially chosen set. That difference in size is illustrated by horizontal error lines. It is witnessed that the similarity graph model performs comparably with the benchmark.

In the preliminary implementation, the feature extraction block is considered part of our proposed model; hence, the neural network is trained and validated on raw pixel values of the images. However, in the final

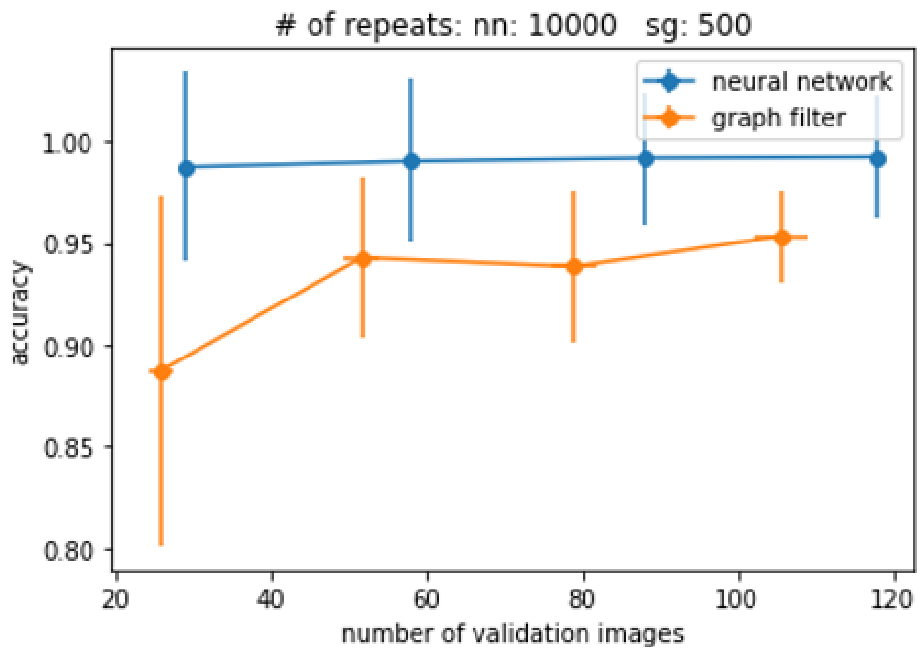


Figure 27: Accuracies obtained by two different methods for various sizes of the dataset.

implementation, the feature extraction block is considered separate from the similarity graph model; thus, the benchmarks in the next sections are trained and validated on the same features that are associated with the similarity graph's nodes.

3.2 Assessment Quantities

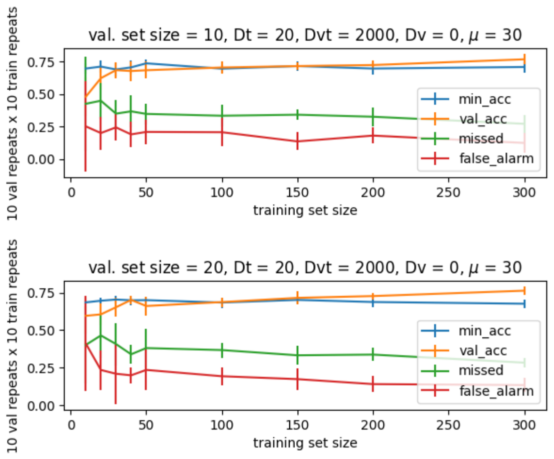
After achieving promising results in the preliminary experiment, a modular and expanded version of the code was implemented, which is made publicly accessible on GitHub [132]. All features of the code and instructions on how to run them are explained in Appendix C. To summarize, we can train the similarity graph with three approaches:

1. B-GLR: the objective is GLR plus trace of the metric matrix, and it is optimized by Gradient Descent
2. SDP-GLMNN: the objective is GLMNN, and it is optimized as an SDP
3. GDPA-GLMNN: the objective is GLMNN, and it is optimized using GDPA approximation

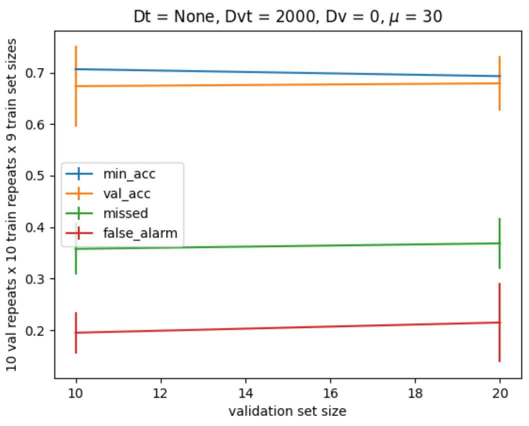
Furthermore, we have three sets of features:

1. Slowfast: first-layer filters of "slowfast-r50"
2. 3D-SIFT: descriptor of one keypoint in each frame batch
3. SOE-Net: second layer outputs of SOE-Net

For our first experiment, we predicted the group response of all ganglion cells (as explained in Section 2.3.1) using the B-GLR model and Slowfast features. Figure 28 illustrates the results.



(a) Accuracy vs. training set sizes



(b) Accuracy vs. validation set sizes

Figure 28: Performance of the similarity graph when predicting a single neuron's response.

Figure 28a plots four performance curves with respect to the training set size. Two subplots exists in the figure; the only difference between the subplots is the size of the validation datasets. To obtain each plot-point and its error bar, the mean performance and its standard deviation are calculated over 100 experiments. The training and validation datasets are randomly created for these experiments; 10 random training sets are paired with 10 random validation sets. The validation sets are newly created for each new training set. The number of random training and validation sets are observed next to the plots.

The mean and standard deviation of all four quantities in Figure 28a are calculated for each validation set size, and plotted in Figure 28b. This figure serves to analyze the effect of the validation set size on our performance assessment. Logically, there should not be a considerable difference between our model's performance on various validation set sizes; hence, if we see a difference, it means our experiment setting should change.

The performance quantities shown in Figure 28 include: validation accuracy (val_acc), minimum accuracy (min_acc), missed detection rate (missed), and false alarm rate (false_alarm). Validation accuracy is calculated by dividing the number of correct predictions (i.e. the predicted label is equal to the ground-truth label) by the number of all predictions,

$$\text{Accuracy} \triangleq \frac{\text{\# of correct predictions}}{\text{\# of val. data points}}. \tag{93}$$

Minimum accuracy (min_acc) is obtained when all the data points are given the same prediction; therefore, the minimum accuracy rate depends on the distribution of labels in the dataset. For example, if 70% of the dataset is labeled +1 and 30% is labeled -1, the minimum accuracy is 70% and it is obtained when all labels are predicted as +1.

Missed detection rate is defined as the number of data points that are labeled +1 but detected as -1 by the model, divided by the number of all +1 labeled data points,

$$\text{Missed detection} \triangleq \frac{\text{\# of val. data points predicted as -1, while truly labeled +1}}{\text{\# of val. data points labeled +1}}. \tag{94}$$

False alarm rate is defined as the number of data points that are labeled -1 but detected as +1 by the model, divided by the number of all -1 labeled data points,

$$\text{False alarm} \triangleq \frac{\text{\# of val. data points predicted as +1, while truly labeled -1}}{\text{\# of val. data points labeled -1}}. \tag{95}$$

Figure 29 illustrates these concepts; the horizontal axis shows the true labels of validation data points, while the vertical axis shows the predicted labels.

In Figure 28, the minimum accuracy (min_acc) is about 75% which is quite high. A high minimum accuracy leaves little space for improvement to our model. This happens because about 70% of our dataset (described in Section 2.3.1) is labeled +1; so when data points are randomly chosen to form the training and validation datasets, about 70% of them will be labeled +1. To fix this issue, we started making sure that our training and validation datasets are equally distributed between the two labels. In other words, 50% of a set is randomly chosen from the +1 nodes, and 50% from the -1 nodes. All of our following experiments are done with such balanced sets.

In addition to the quantities in Figure 28, we also measure the experiments' runtime. An example of a

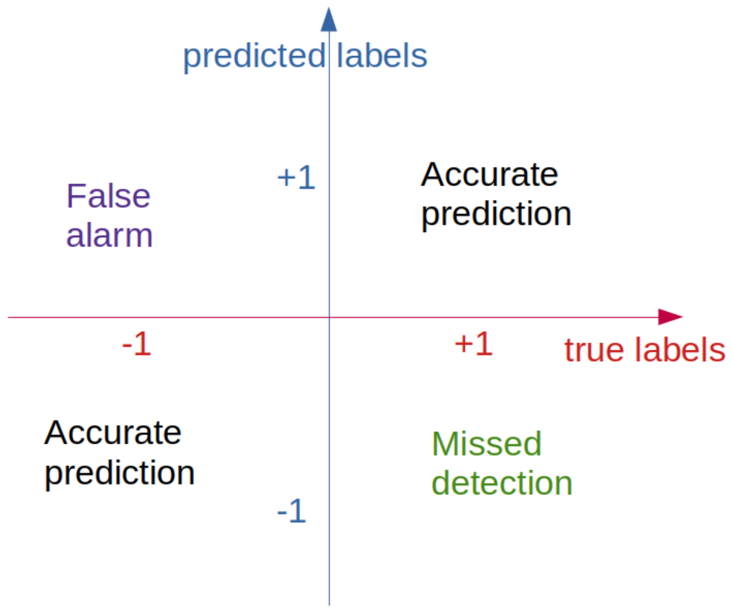


Figure 29: Illustration of the concepts of accuracy, missed detection rate and false alarm rate.

runtime plot is provided in Figure 30. This figure shows the runtime of an experiment with a *complete* graph. As expected, the time complexity is $O(N^2)$ where N is the number of training nodes. This complexity is mostly due to computing the derivation of loss in GD. A justification is given in Appendix B.

Lastly, we visualize the optimized metric matrices. Figure 31 illustrates three examples. We mark the relatively large entries with red dots. The definition of "large entry" can vary from experiment to experiment. However, in most of our experiments, all entries within 30% of the largest entry are marked.

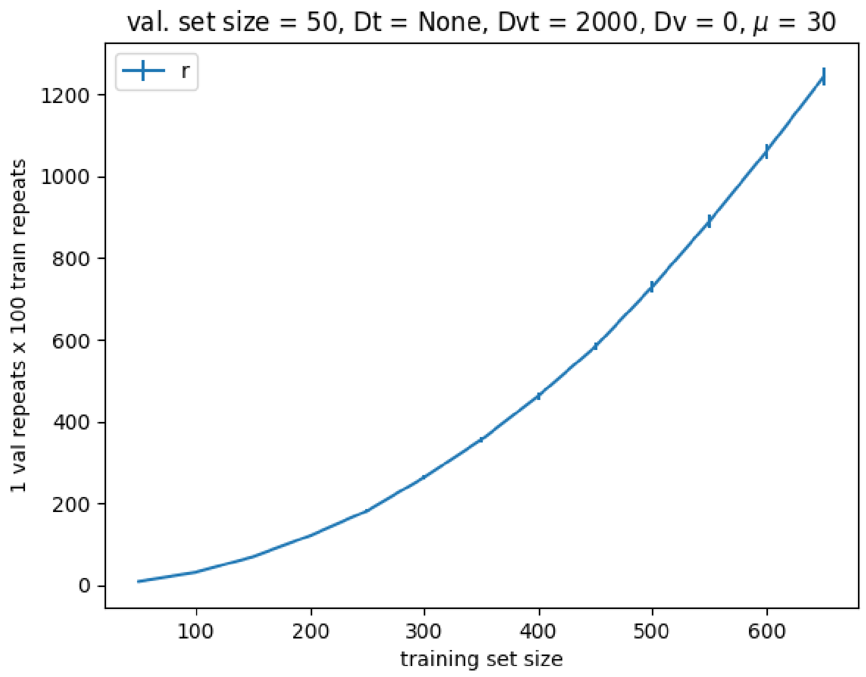
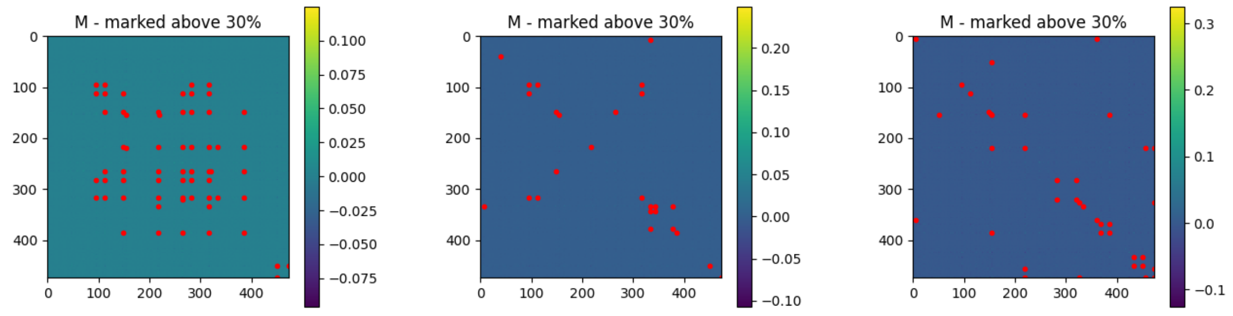


Figure 30: Runtime of a randomized experiment.



(a) Repeat 1

(b) Repeat 2

(c) Repeat 3

Figure 31: Visualizations of three optimized metric matrices. All entries within 30% of the maximum entry are marked with red dots.

3.3 Explainability

3.3.1 SIFT on the MNIST dataset

Figure 32 shows several examples of SIFT keypoints found on MNIST images. Two keypoints are selected in each image; one closest to the top-left corner, and the other closest to the bottom-right corner. These keypoints' descriptors are concatenated to form the feature vector of an image.

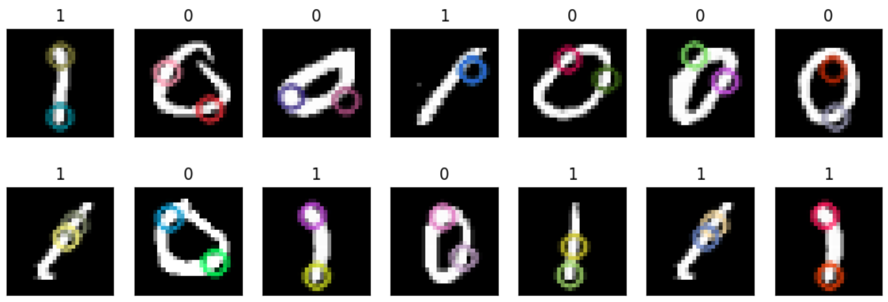
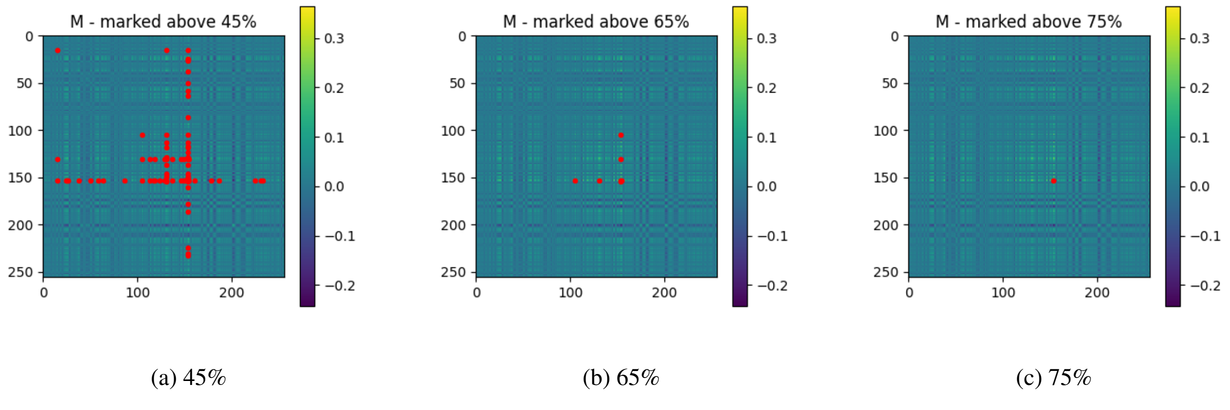


Figure 32: Examples of SIFT keypoints on MNIST images.

The B-GLR model is trained on these features. Then, the optimized metric matrix ***M****** is analyzed to gain insight into the prediction process. Figure 33 shows the optimized metric matrix, where all entries within a certain percentage of the maximum entry are marked with red dots. That certain percentage is stated below the images; we have 45%, 65% and 75%.



(a) 45%

(b) 65%

(c) 75%

Figure 33: The optimized metric matrix by B-GLR on MNIST-SIFT features. Entries within a certain percentage of the maximum entry are marked with red dots. The percentage is stated below the images.

First, we analyze the *diagonal* entries of ***M****** . Recall Equation (36). We can separate the diagonal and non-diagonal elements in Equation (36) and write it as

$$\begin{aligned} d_{ij} &= \sum_{a=0}^{C-1} M_{aa} f_a^2 + \sum_{a=0}^{C-1} \sum_{b=0, b\neq a}^{C-1} M_{ab} f_a f_b \\ \boldsymbol{f} &= \boldsymbol{f_i} - \boldsymbol{f_j}. \end{aligned} \tag{96}$$

Thus, we observe that the diagonal entries of $\boldsymbol{M^*}$ determine the weights of single entries of the feature vector in the calculation of distance. Features with larger corresponding entries in $\boldsymbol{M^*}$ have larger effects on the prediction; therefore, one can view the diagonal entries of $\boldsymbol{M^*}$ as contribution scores assigned to the features. Figure 34 shows the diagonal entries of the matrix in Figure 33. The red horizontal line delineates 45% of the maximum entry. All entries above the red line are marked with red numbers; the numbers indicate the order from the largest entry to the smallest one. Feature 153 has the largest corresponding $\boldsymbol{M^*}$ entry, and feature 131 is the second largest.

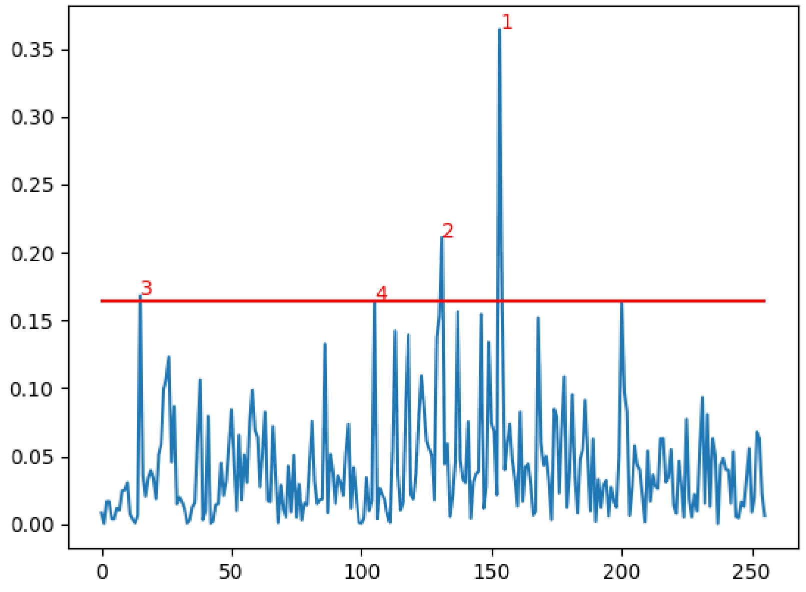
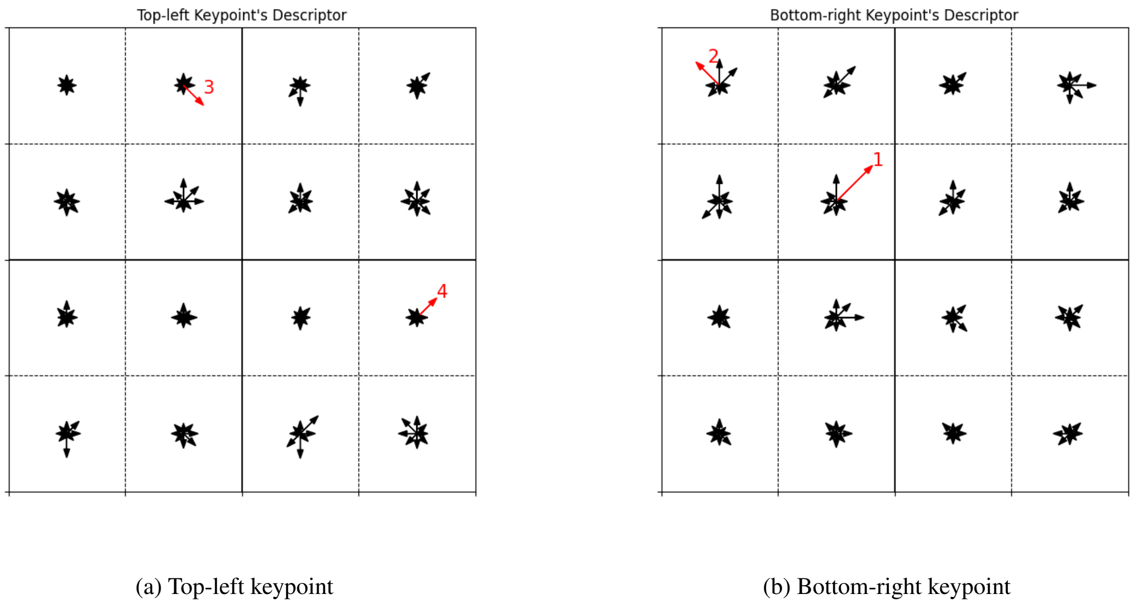


Figure 34: Diagonal entries of $\boldsymbol{M^*}$. The red line marks 45% of the maximum entry. All entries above the red line are numbered starting from the largest entry to the smallest.

The diagonal entries can be separated into two 128-dimensional vectors, each corresponding to one keypoint's descriptor. As described in Section 2.3.3, SIFT descriptors consist of histograms of local image gradients around the keypoint; hence, one can visualize a descriptor by drawing the histograms as in Figure 23. In Figure 35 a similar visualization is made out of the diagonal $\boldsymbol{M^*}$ entries. The left image illustrates the first half of the diagonal entries corresponding to the top-left keypoint, and the right image illustrates the second half corresponding to the bottom-right keypoint.

The top diagonal entries which were marked in Figure 34 are also marked in Figure 35 with the same numbers. Both of the largest entries belong to the bottom-right keypoint. Moreover, the median of the first half of the entries is 0.0246 which is 34% smaller than the median of the second half (= 0.0373). Based on these facts, we conclude that the bottom-right keypoint is generally more informative than the top-left keypoint.



(a) Top-left keypoint

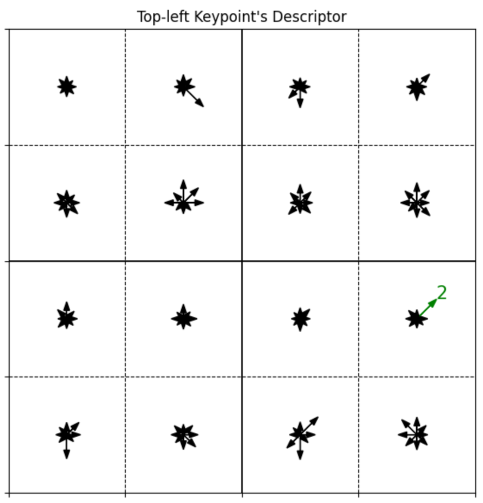
(b) Bottom-right keypoint

Figure 35: Visualization of the diagonal entries of $\boldsymbol{M^*}$ corresponding to local gradients of the image. The same entries marked in Figure 34 are colored red and numbered.

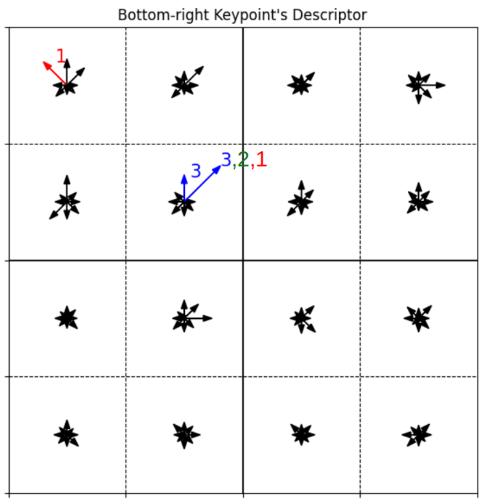
By observing Figure 35, we can easily conclude that the feature that contributes most to the prediction is the accumulation of the gradient magnitudes between 22.5 and 67.5 degrees in a region of 4×4 pixels starting from the keypoint and going to the left and top. Moreover, the keypoint in question is the one closest to the bottom-right corner of the image. We can describe the next most contributing features in the same manner.

Figure 35 may be counter-intuitive because the top contributing features have inclined orientations. One might expect the vertical orientations to be more important because number 1 is a vertical line and 0 is a circle which does not prioritize any direction over the others. We can analyze "how" these features can distinguish 1 from 0, and whether this leads to new algorithms for distinguishing 1 from 0 that do not immediately occur to humans. This can be explored further in the future.

The non-diagonal entries of $\boldsymbol{M^*}$ can be interpreted as contribution scores assigned to *pairs* of features, or as the covariance of the pair. The largest non-diagonal entries in Figure 33b are (153, 105), (153, 131) and (153, 154). Each of these pairs is distinguished by a different color in Figure 36.



(a) Top-left keypoint



(b) Bottom-right keypoint

Figure 36: Highly informative *pairs* of features which correspond to the largest non-diagonal entries of $\boldsymbol{M}^*$ are distinguished by red, green and blue arrows. The magnitude of the arrows correspond to diagonal entries of $\boldsymbol{M}^*$ as in Figure 35.

As a possible future analysis, one can create an image based on the local gradients given by the diagonal entries of $\boldsymbol{M}^*$. In other words, one can craft feature vectors that get the highest (or lowest) possible distance by the formula in Equation (96). These crafted features may reveal further insight about the reasoning behind the predictions.

3.3.2 3D-SIFT on the RGC dataset

Every 32 consecutive frames of the fish movie in the RGC dataset are aggregated in 3D tensors which are given to 3D-SIFT as input. 3D-SIFT finds multiple keypoints in each of these 3D tensors which we will call "frame batches". Of all the keypoints, one which is closest to the frame batch's center is selected and kept while all the other keypoints are discarded. These selected keypoints are illustrated in Figure 37 on the same frames where they are located. The descriptors of these keypoints are used as feature vectors describing the frame batches.

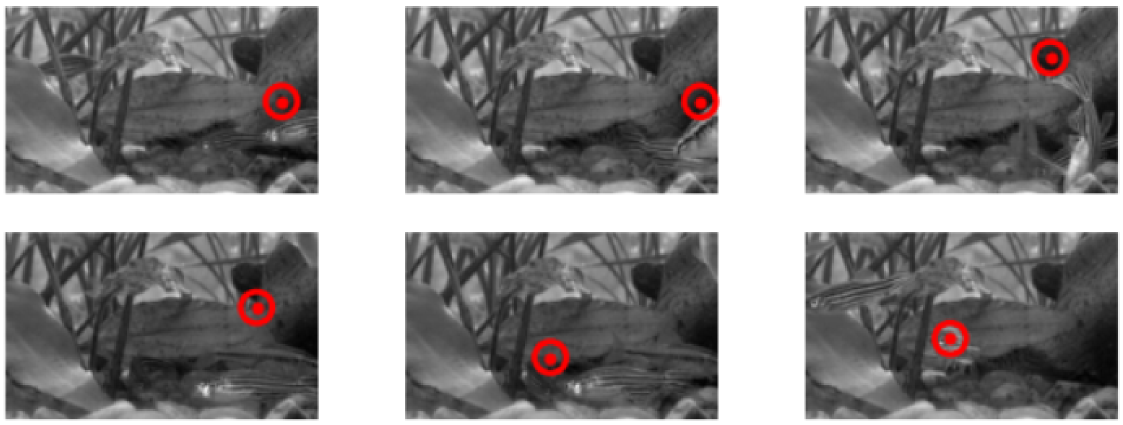
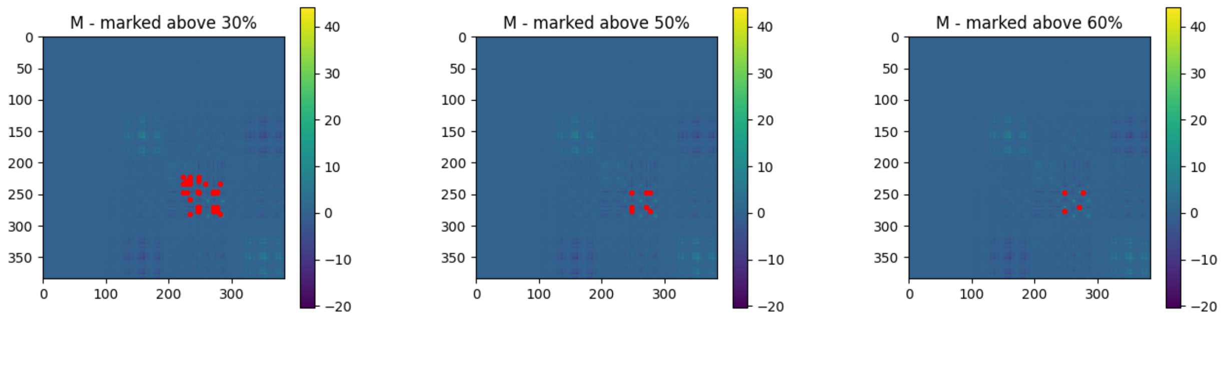


Figure 37: Selected 3D-SIFT keypoints in various frame batches of the RGC dataset.

We trained the B-GLR model on these features; however, we subsampled the feature vectors with 2:1 rate to cut our experiment's runtime to $\frac{1}{4}$. The relationship between our model's time complexity to the number of features is given in Appendix B. As a result, half of the features are not used in the experiment; even so, our model still achieves an acceptable 75% accuracy as seen in Figure 45. The optimized metric matrix is shown in Figure 38. Large entries are marked with red dots similar to Figure 33.



(a) 30%

(b) 50%

(c) 60%

Figure 38: The optimized metric matrix by B-GLR on 3D-SIFT features. Entries within a certain percentage of the maximum entry are marked with red dots; the percentage is stated below each image.

Analogous to Section 3.3.1, we start by analyzing the diagonal entries of $\boldsymbol{M}^*$; these entries are plotted in Figure 39. The red horizontal line delineates 50% of the maximum entry. All entries above the red line are numbered from the largest to the smallest; these entries are the 247th, 270th, and 277th, respectively. The corresponding sub-regions are illustrated in Figure 40; numbers on top of the sub-regions denote the same features as in Figure 39.

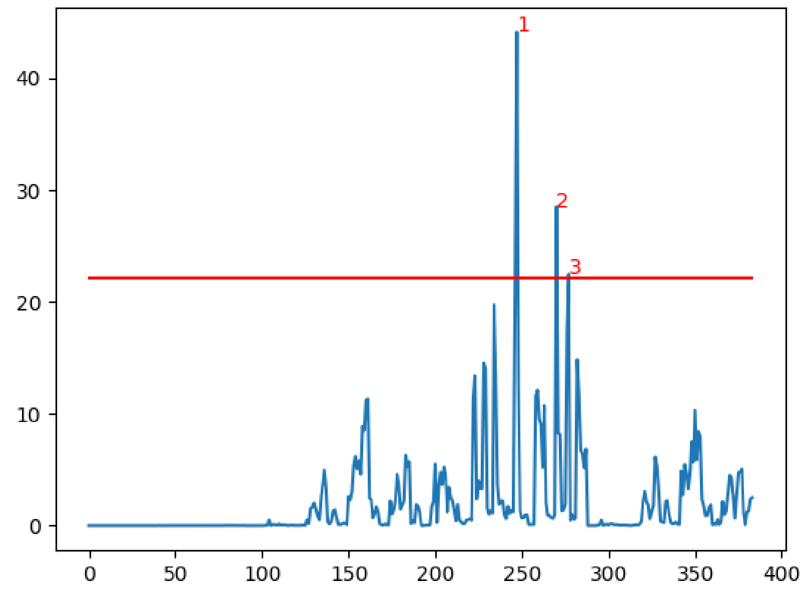


Figure 39: Diagonal entries of the optimized metric matrix $\boldsymbol{M}^*$ which is shown in Figure 38. The red line marks 50% of the maximum entry. All entries above the red line are numbered from the largest to the smallest.

We computed the medians of diagonal $\boldsymbol{M}^*$ entries corresponding to all sub-regions. These medians are plotted in Figure 41. The red line marks 70% of the peak, and all bins above the red line are numbered in order. The numbered sub-regions are also highlighted in Figure 42 bearing the same numbers.

Similar to Section 3.3.1, we can conclude from Figure 40 that the most contributing feature appears in a sub-region of a specific size in $(x, y, z) = (0, 1, 1)$ coordinates relative to the keypoint's location. Moreover, this feature shows the accumulation of gradient magnitudes in a specific direction described by vertex #2 of the icosahedron in that sub-region. Other highly contributing features are similarly describable. Consequently, we can analyze the model further to find out "how" the top features can distinguish between 1 and -1 labeled frame batches.

Orientation histograms for all sub-regions are drawn in Figure 43. The histograms show diagonal entries of

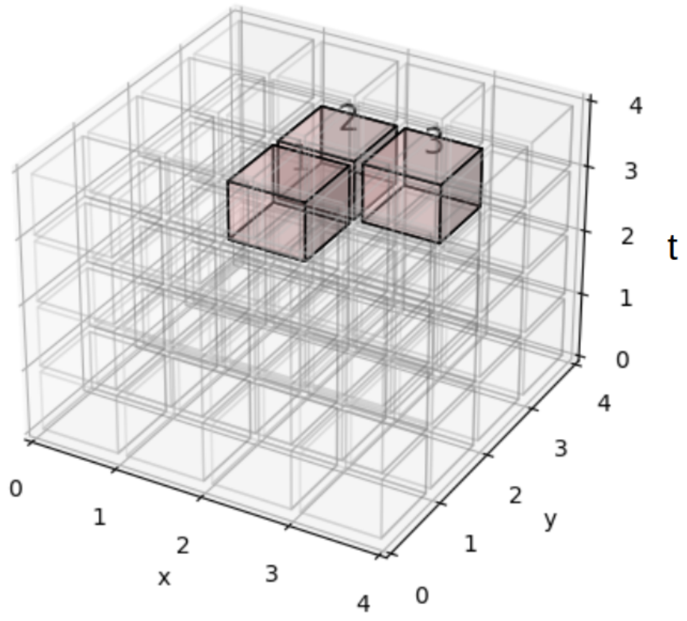


Figure 40: Sub-regions of the most contributing features. The numbers on top of the sub-regions denote the same features as in Figure 39.

M* instead of any 3D image's gradients. Since we subsampled the feature vectors during training, only 6 directions are still included in the histograms (the other 6 are discarded). The largest non-diagonal elements of the matrix in Figure 38b are (247, 270) and (247, 277). These pairs are distinguished in Figure 43 with green and red colors, respectively. The relevance of the features and ***M**** entries to the prediction can be further explored in the future of this research, realizing the full potential of our model's explainability.

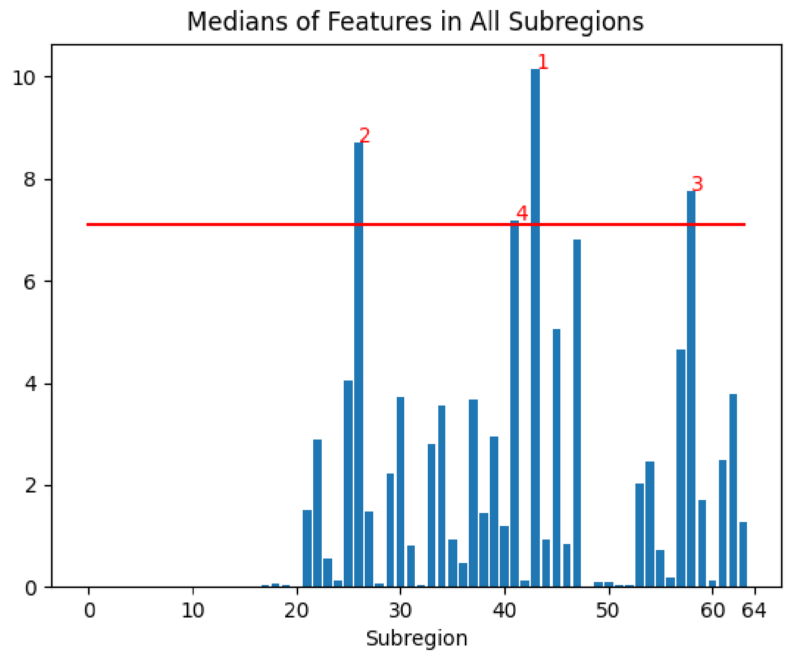


Figure 41: Medians of diagonal ***M**** entries corresponding to all sub-regions. The red line marks 70% of the peak. All bins above the red line are numbered from the largest to the smallest.

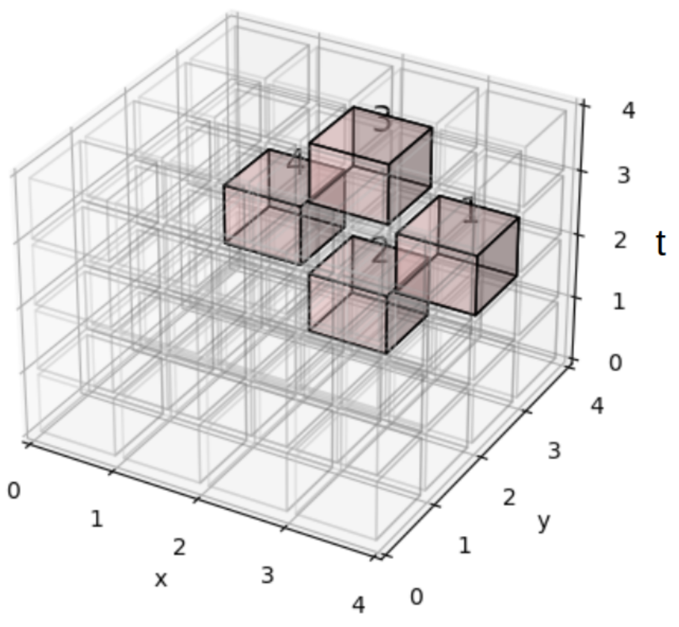
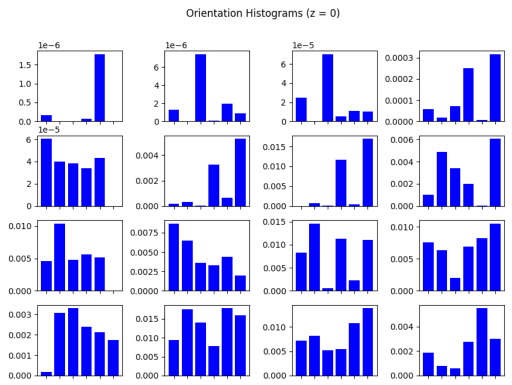
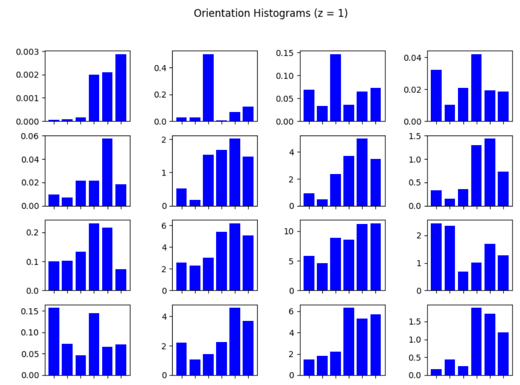


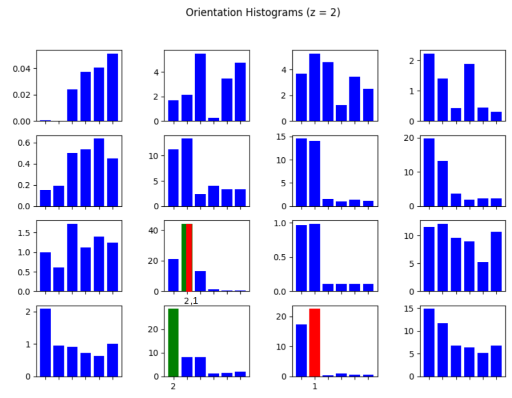
Figure 42: Sub-regions with the largest medians of corresponding diagonal ***M**** entries. Numbers indicate the same regions as in Figure 41.



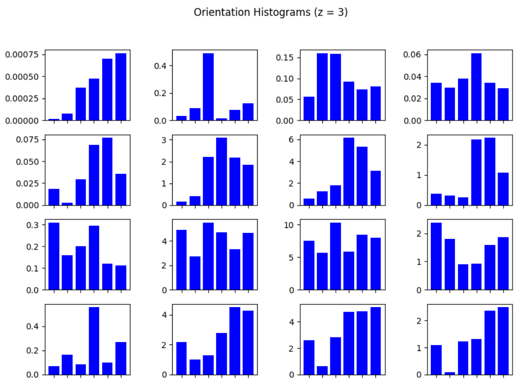
(a) Histograms of the lowest plane of sub-regions (z = 0)



(b) Histograms of the second-lowest plane of sub-regions (z = 1)



(c) Histograms of the second-highest plane of sub-regions (z = 2)

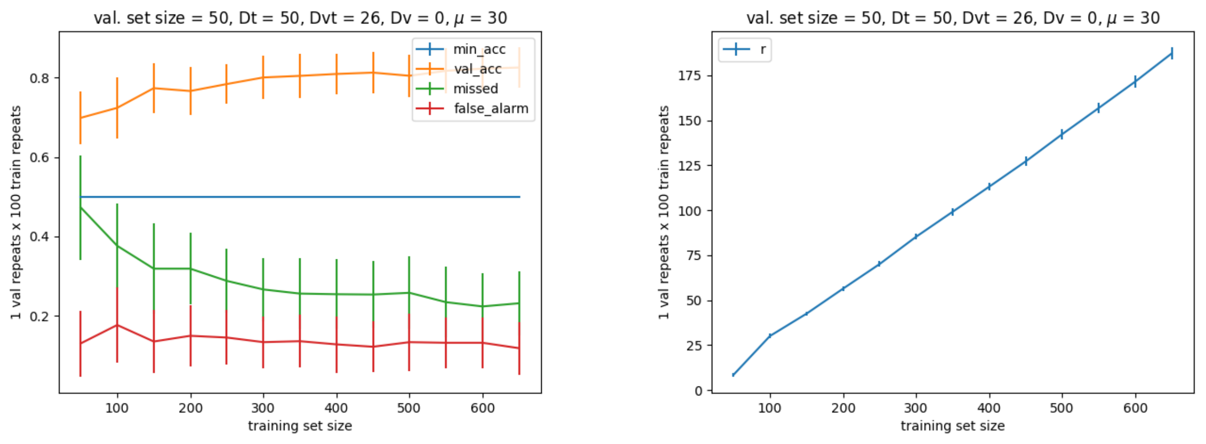


(d) Histograms of the highest plane of sub-regions (z = 4)

Figure 43: Orientation histograms of all sub-regions in 3D-SIFT. Histograms show the diagonal entries of $\boldsymbol{M}^*$ instead of image gradients. Highly correlated pairs of features are distinguished by different colors and numbers.

3.4 B-GLR with All Sets of Features

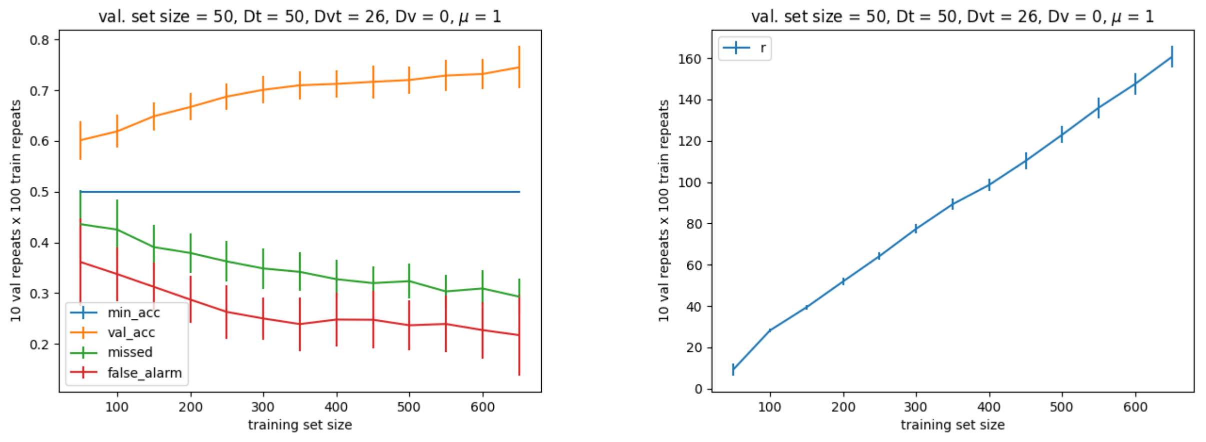
We trained and validated the B-GLR method on all four feature sets described in Section 2.3.2. Figure 44 illustrates the results of the Slowfast features. The Slowfast feature vectors have 4732 dimensions. Running the experiment with the original feature vectors is prohibitively time-consuming; therefore, the feature vectors are subsampled into 474 dimensions. The experiment took about 3 days with 474-dimensional vectors. Since the time complexity has a quadratic relationship with the number of features (see Appendix B), using the original feature vectors would extend the experiment's runtime to about 12 days.



(a) Performance (b) Runtime

Figure 44: Results of the B-GLR model on Slowfast features

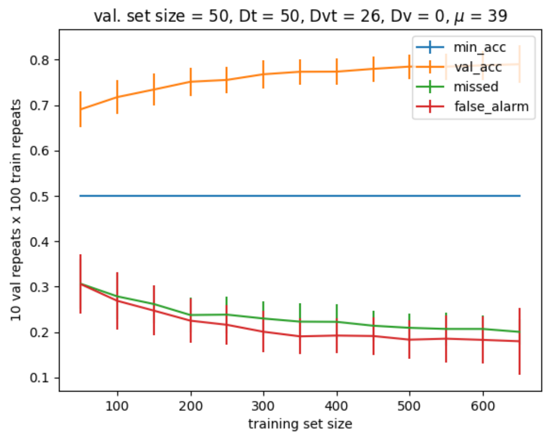
Figure 45 illustrates the results of the 3D-SIFT features. The original 3D-SIFT feature vectors have 768 dimensions. In this experiment, they are subsampled into 384 dimensions.



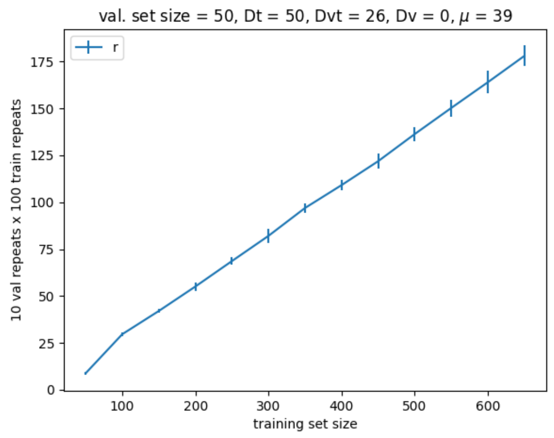
(a) Performance (b) Runtime

Figure 45: Results of the B-GLR model on 3D-SIFT features

Figure 46 illustrates the results of the SOE-Net features. SOE-Net features have different dimensions based on which layer of the SOE-Net they come from. The first through fifth layer feature vectors have dimension 20, 400, 800, 1600, 3200, respectively. In this experiment, second layer feature vectors are used without subsampling.



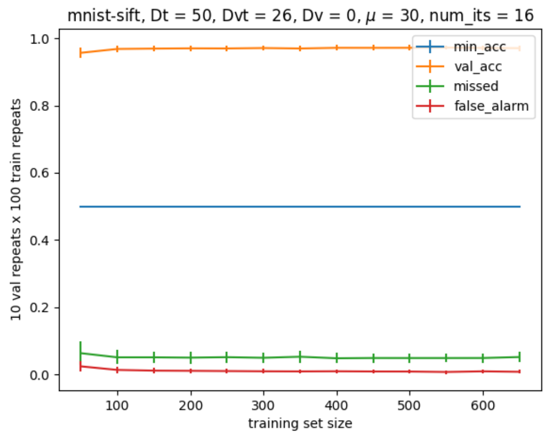
(a) Performance



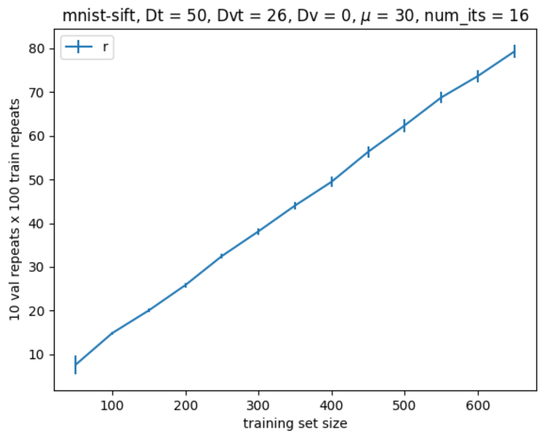
(b) Runtime

Figure 46: Results of the B-GLR model on SOE-Net features

Figure 47 illustrates the results of the SIFT features extracted from MNIST images. Feature vectors are 256-dimensional, and they are not subsampled for the experiment in Figure 47. We used this dataset to see how our model performs on a dataset which is known to be “easy” for machine learning models. As observed, our model achieves an acceptable 97% accuracy.



(a) Performance



(b) Runtime

Figure 47: Results of the B-GLR model on MNIST-SIFT features

3.5 Benchmarks

Figure 48 compares the performance of the B-GLR model with several bechmarking methods on Slowfast features. The first benchmark is XGBoost or eXtreme Gradient Boosting [92], a popular machine learning algorithm specially famous for winning Kaggle challenges [133]. This algorithm iteratively trains an ensemble of shallow decision trees where each new decision tree reduces the prediction error in a formalized manner similar to gradient descent. The final prediction is a weighted sum of all the tree predictions. The number of boosting rounds (i.e. number of decision trees) for the curve in Figure 48 is 16. We also tried training with 10000 boosting rounds which did not make much improvement over 16 rounds; therefore, 16 rounds is the chosen value for future experiments.

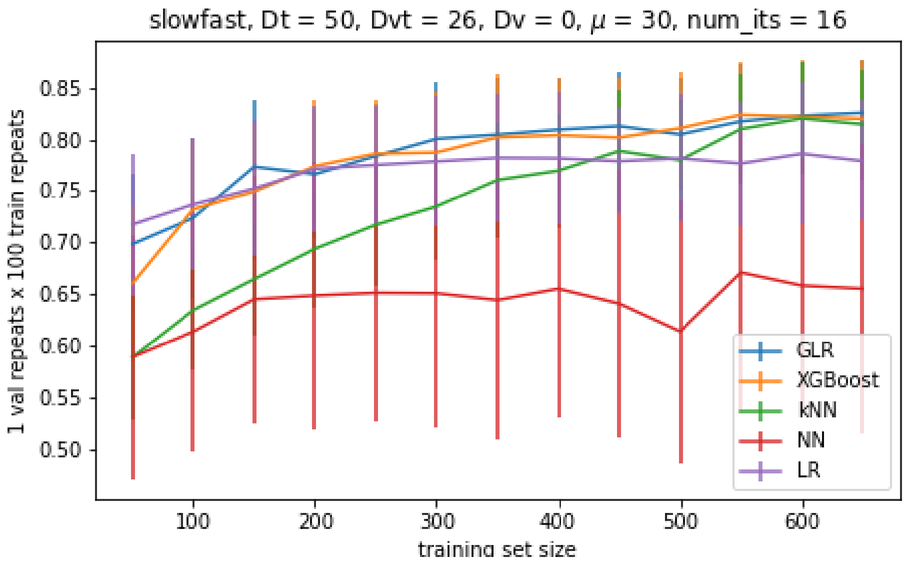


Figure 48: B-GLR model on Slowfast features along with several benchmark methods comprising XGBoost, K-nearest neighbors (kNN), neural network (NN) and logistic regression (LR)

The second benchmark is K-nearest neighbors (kNN) [93]. This classifier estimates the label of a query point by taking a simple majority vote of its nearest neighbors. The number of voting neighbors is K which is set to 3 in Figure 48. This value is optimized by cross-validation. Figure 49 shows the accuracy score for various values of K. It is observed that the best performance is obtained when $K = 3$.

The third benchmark is neural network (NN). The network in Figure 48 has 2 layers each with 10 neurons. By increasing the number of layers and neurons in the network, the accuracy increases. However, it did not surpass XGBoost in any of our experiments with various network architectures. The fourth benchmark is Logistic Regression (LR) [134]. The only hyperparameter of this model is the number of gradient descent iterations that it takes to minimize the loss which is set to 16 in Figure 48.

Figure 50 illustrates the performance of the B-GLR model on 3D-SIFT features compared with the four benchmarks. The neural network in this figure has 5 layers each with 20 neurons. By observing Figures 48

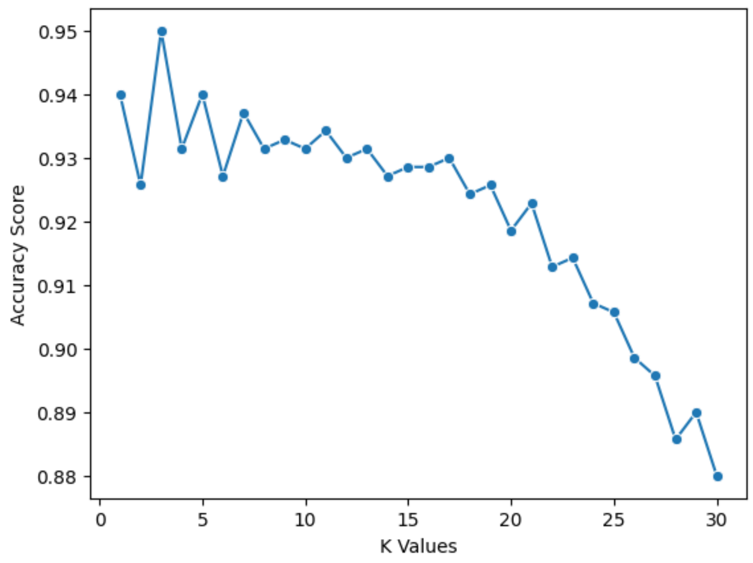


Figure 49: Accuracy scores for various values of K are shown for the K-nearest neighbors benchmark. $K = 3$ yields the best performance.

and 50, we can conclude that B-GLR performs comparably with the benchmarks. It is worth mentioning that while our benchmarks' hyperparameters are tuned using cross-validation, the B-GLR's hyperparameters (μ, D_t, D_v, D_{vt}) are tuned by manually trying several values. We were not able to run cross-validation for B-GLR since it was prohibitively time-consuming.

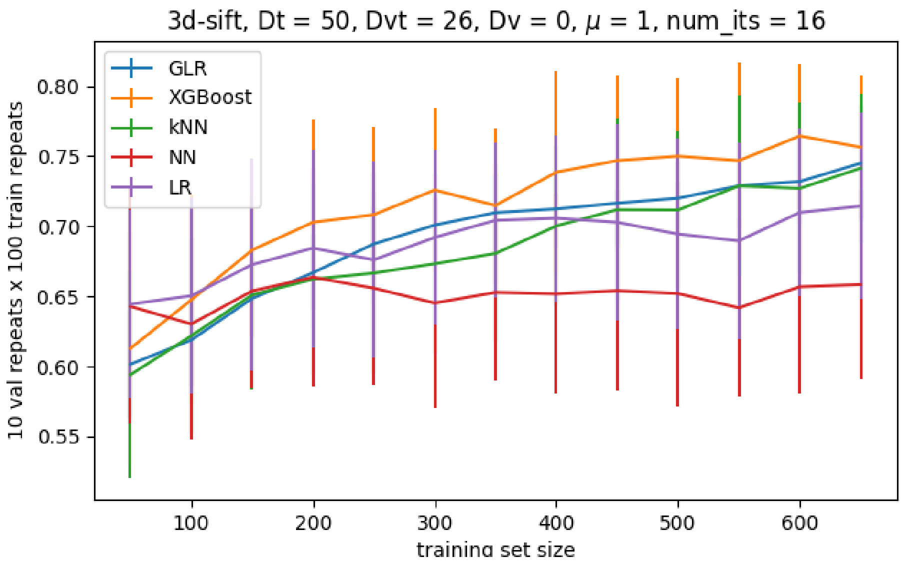


Figure 50: B-GLR model on 3D-SIFT features along with several benchmark methods comprising XGBoost, K-nearest neighbors (kNN), neural network (NN) and logistic regression (LR)

3.6 SDP-GLMNN and GDPA-GLMNN

Figure 51 shows the obtained accuracies of the B-GLR model and the SDP-GLMNN model. This experiment is conducted on SIFT features extracted from MNIST images, and with a sparse similarity graph. As expected, SDP-GLMNN achieves higher accuracies since GLMNN incorporates an additional requirement into the optimization: that the distance between same-labeled nodes should be small. Refer to Section 2.2.4 for more explanation.

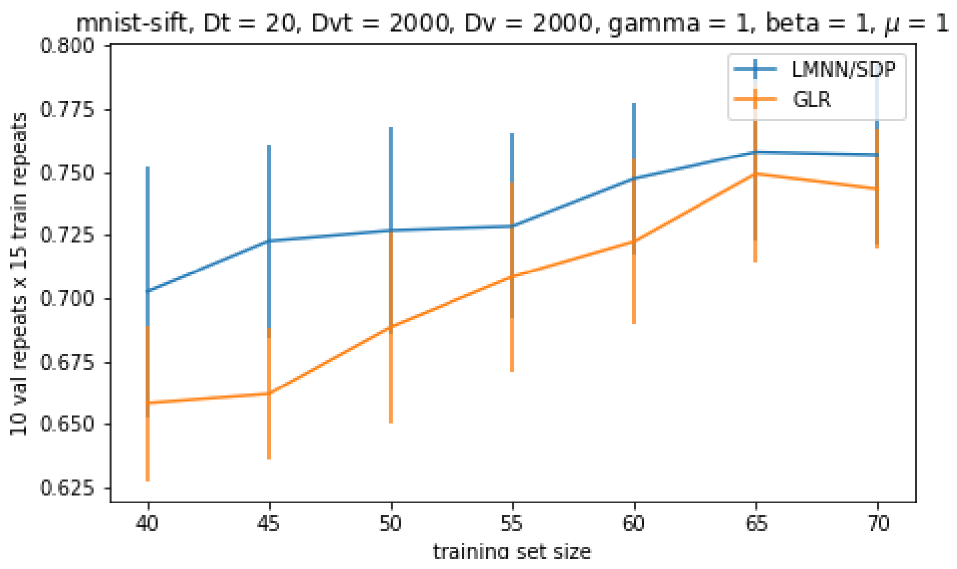


Figure 51: Accuracy of B-GLR and SDP-GLMNN models

Figure 52 compares the accuracies of the SDP-GLMNN and the GDPA-GLMNN optimizations. Both of these approaches optimize the same objective with the same semi-definite constraint. However, SDP solves the program exactly while GDPA relaxes the semi-definite constraint to speed up the computation (see Section 2.2.5 for more information). Although GDPA-GLMNN is a relaxation, Figure 52 shows that it achieves comparable accuracy with SDP-GLMNN.

To compare the models' runtimes, we counted the number of linear programs (LP) that are solved in GDPA-GLMNN until it converges to an answer. Figure 53 shows this number for various sizes of the training dataset. SDP-GLMNN's solver, SeDuMi [105], takes $O(C^3 + C^{2.5}N_tD_t^2 + C^{0.5}N_t^{2.4}D_t^{4.8})$ to find the optimization's answer, while each LP takes $O((C+N_tD_t^2)^{2.055})$. Figure 53 shows that the number of LPs remains almost constant for various dataset sizes. As a result, the time complexity of GDPA-GLMNN is $O(cnst \times (C+N_tD_t^2)^{2.055}) = O((C+N_tD_t^2)^{2.055})$. In other words, GDPA reduces time complexity from $O(C^3 + C^{2.5}N_tD_t^2 + C^{0.5}N_t^{2.4}D_t^{4.8})$ to $O((C+N_tD_t^2)^{2.055})$.

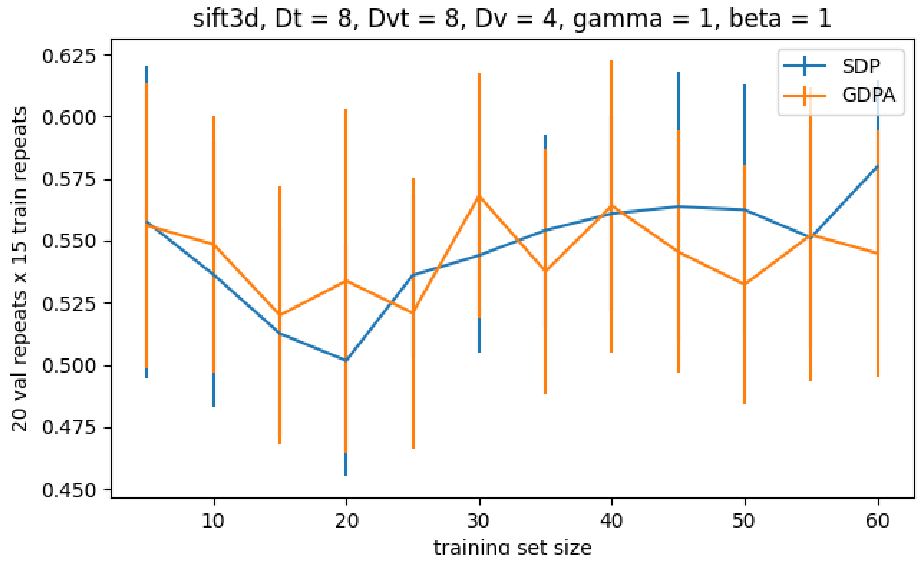


Figure 52: Accuracy of SDP-GLMNN and GDPA-GLMNN

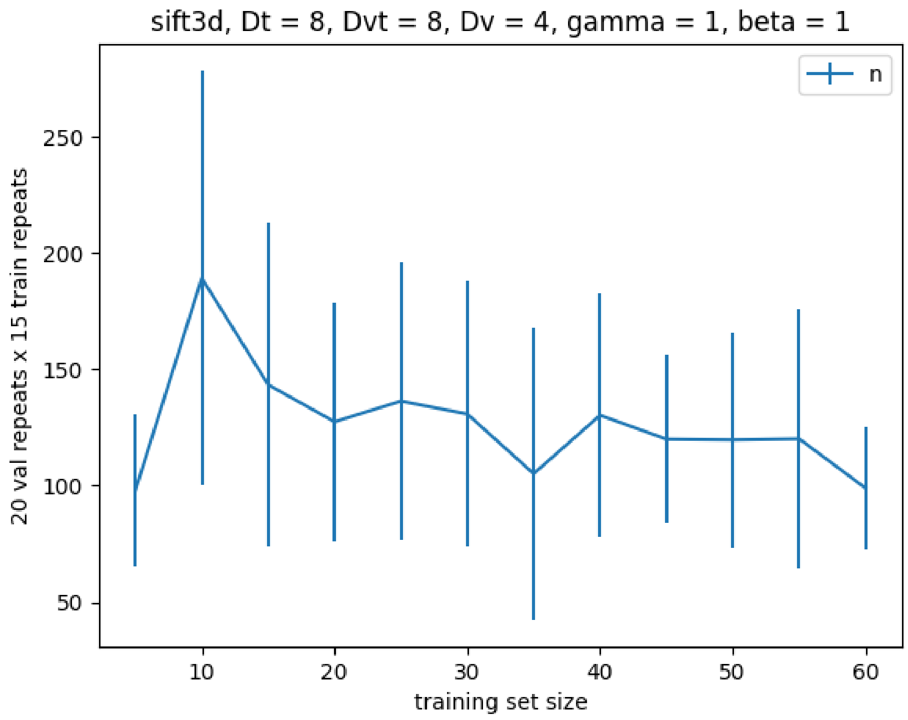
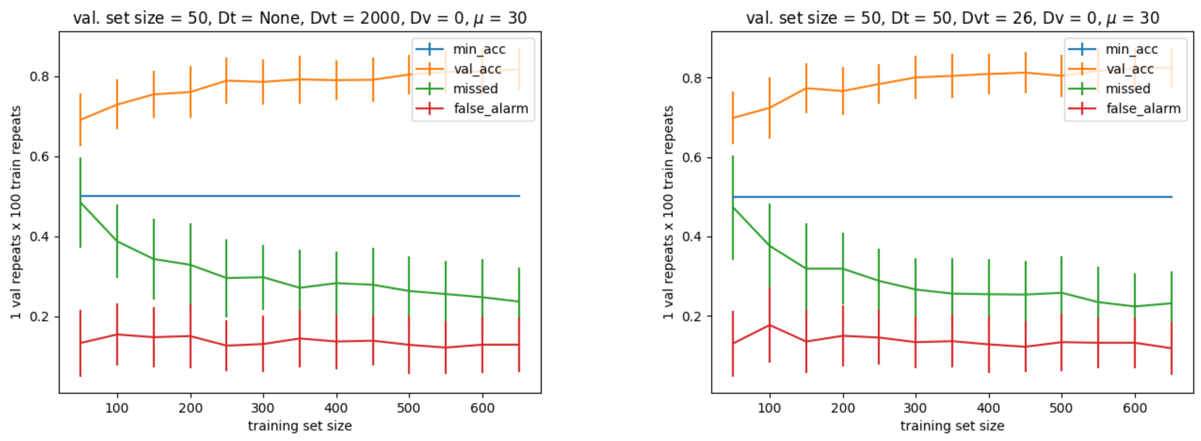


Figure 53: Number of linear programs solved in GDPA-GLMNN to approximate the SDP formulation of GLMNN.

3.7 Complete Graph vs. Sparse Graph

To demonstrate the benefit of edge selection, we trained and tested the B-GLR model on the Slowfast features with the same parameters except for graph structure. In our first experiment, the similarity graph is complete, meaning that all pairs of nodes are connected by edges. In the second experiment, the maximum degree of the nodes is fixed; training node degree (D_t) is 50, the number of edges connecting each validation node to training nodes ($2\ D_{vt}$) is 52, and there are no edges between pairs of validation nodes ($D_v = 0$)). Figure 54 shows the performance plots of both experiments. Figure 55 puts similar curves of both experiments on the same plots. Performance-related quantities are virtually maintained, but runtime is reduced by a substantial amount.



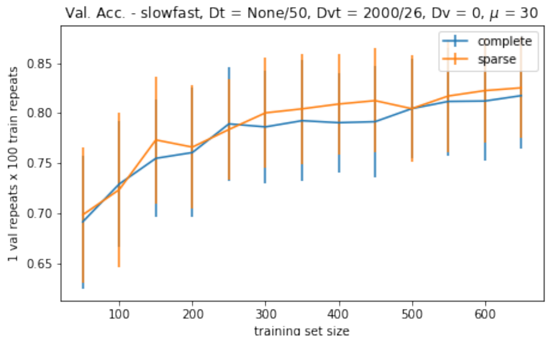
(a) Complete graph

(b) Sparse graph

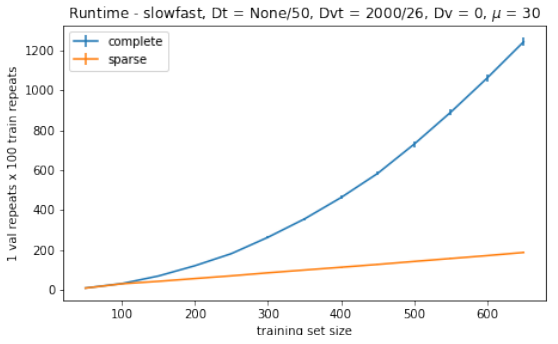
Figure 54: Three performance assessment quantities are compared between two experiments where the features and the objective remain the same, but the graph's structure changes from a complete graph to a sparse graph where the maximum degree of the nodes is fixed.

This section answers a question that comes up about this work: *Why do we need a graph?* The data points can be interpreted as points in a high-dimensional space instead of nodes of a graph. The edge weights represent the distance between pairs of points by some definition of distance in the high-dimensional space. Moreover, the training and validation processes are formulated as optimization problems which have well-studied linear algebraic interpretations. So, where in the process do we need a graph interpretation?

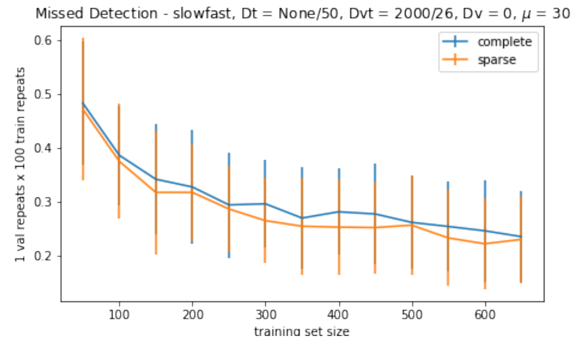
The linear algebraic interpretation seems sufficient when we consider a *complete* graph; however, the need for a graph becomes apparent when we start removing some of the edges. In the linear algebraic interpretation, all pairs of nodes have a distance; therefore, there is no way to distinguish between a distance that we want to remove from calculations and a distance that we want to retain. To keep a record of the retained distances, we need to represent those distances with edges, which means we now have a graph instead of scattered points in a space. As a result, the graph interpretation allows us to reduce the time complexity while maintaining the performance as demonstrated in Figure 55. It might even enable the model to achieve a better accuracy if the edges are chosen in a more sophisticated manner than our current scheme.



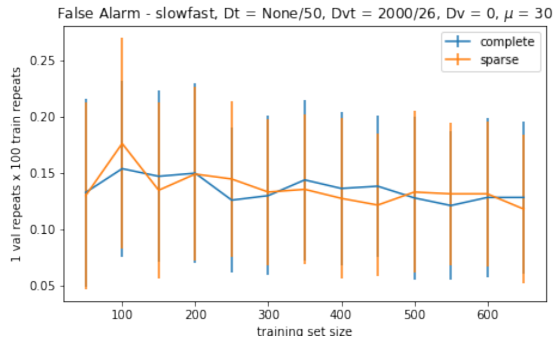
(a) Accuracy



(b) Runtime



(c) Missed detection



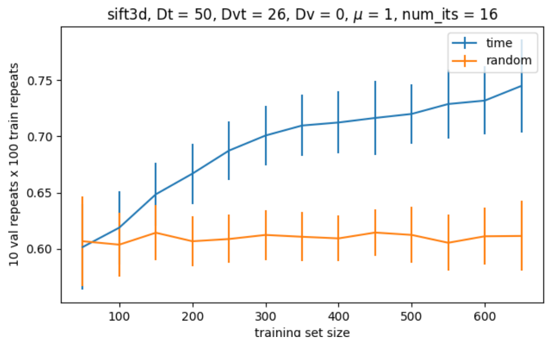
(d) False alarm

Figure 55: Performance and runtime of a complete similarity graph is compared with a sparse similarity graph. Runtime is greatly reduced while performance is virtually maintained at the same level.

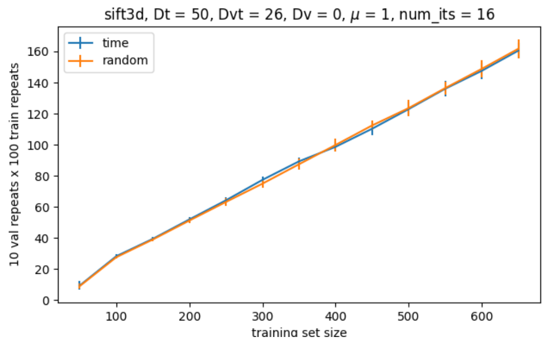
3.8 Time-based vs. Random Edge Selection

In this section, we compare our two proposed strategies of edge selection during the similarity graph's construction. We train and validate the B-GLR model on 3D-SIFT features (subsampled into 384 dimensions) with two different graph structures. The maximum degrees of nodes (D_t, D_v, D_{vt}) are equal between the two experiments. The free scalar parameter (μ) is set to 1 in both, and all parameters of the gradient descent are the same as well. The only difference is the algorithms used to select edges.

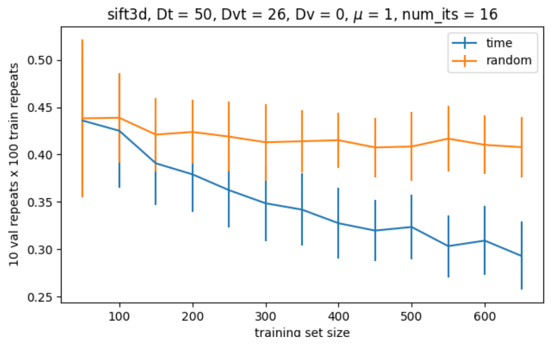
In the first experiment, edges are selected based on the proximity of their end-nodes' corresponding time bins. To elaborate, edges between pairs of training and pairs of validation nodes are selected with Algorithm 2, and edges between training and validation nodes are selected with Algorithm 4. In the second experiment, edges are selected randomly. In other words, Algorithm 1 is used to select edges between pairs of training/validation nodes, and Algorithm 3 is utilized to select edges between training and validation nodes. Figure 56 illustrates the performance and runtime curves of the two experiments. It is witnessed that time-based edge selection greatly increases the accuracy of our model, while the runtime remains virtually the same.



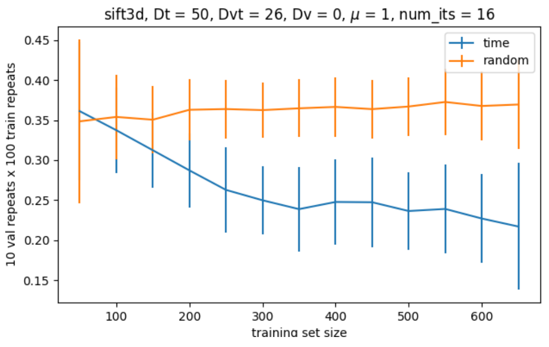
(a) Accuracy



(b) Runtime



(c) Missed detection



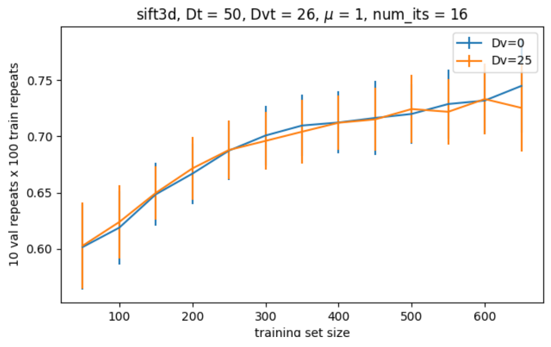
(d) False alarm

Figure 56: Performance and runtime of the B-GLR model with two different similarity graphs; one where edges are selected randomly and one where edges are selected based on their end-nodes' corresponding time bins.

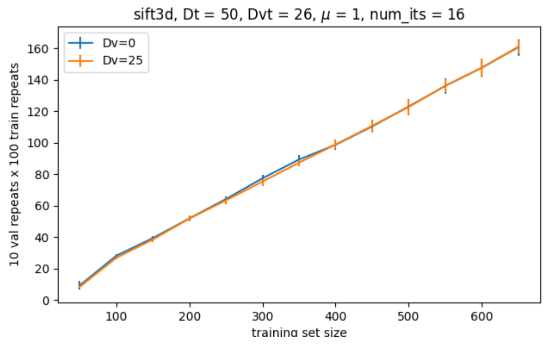
3.9 $D_v = 0$ vs. $D_v > 0$

One advantage of our model is that it does not ignore the similarities between the validation nodes, unlike deep learning models. As explained in Section 2.1.2, if edges exist between pairs of validation nodes, their similarity to each other affects the objective as well as their similarity to the training nodes. To test this property, we ran two experiments with the B-GLR model on 3D-SIFT features. All parameters between the two experiments remain the same except for D_v.

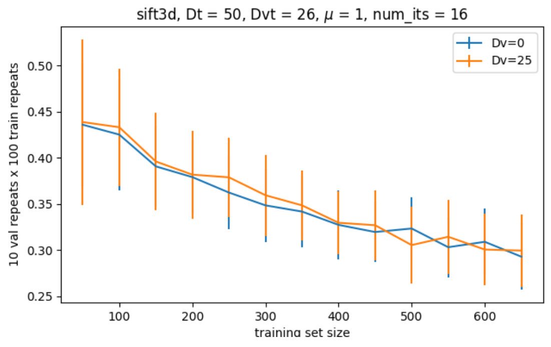
In the first experiment, $D_v = 0$ which means no edges exist between pairs of validation nodes. In this case, the similarity of validation nodes to each other is ignored; in other words, the validation nodes' labels are estimated only based on their similarity to training nodes. In the second experiment, $D_v = 25$, which means each validation node is connected to at most 25 other validation nodes. Figure 57 illustrates the performance and runtime curves of the two experiments.



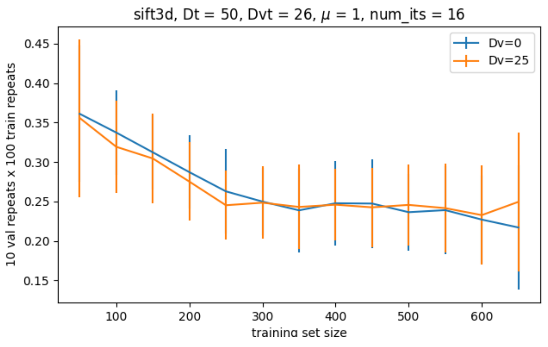
(a) Accuracy



(b) Runtime



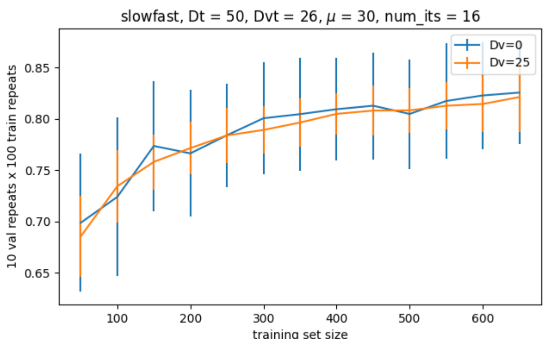
(c) Missed detection



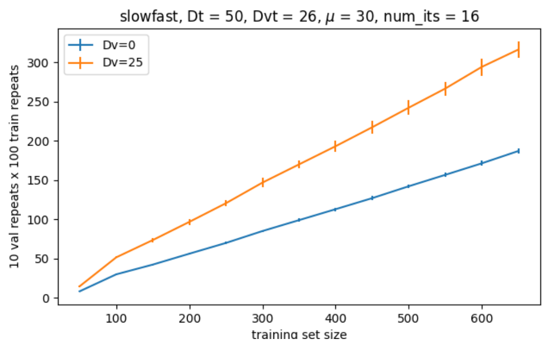
(d) False alarm

Figure 57: Performance and runtime of two similarity graph models where the only difference is the connectivity between pairs of validation nodes (3D-SIFT features).

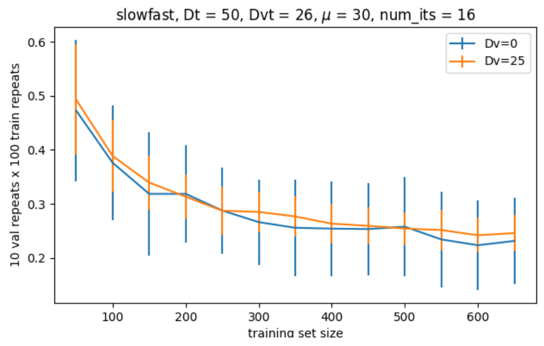
Contrary to our expectation, the accuracy did not improve by connecting the validation nodes. This means that the similarity between pairs of validation nodes effectively holds no new information about the labels. We run the same experiment with Slowfast features as well, whose results are observed in Figure 58. Again, the similarity between validation nodes does not add any new information to the model. Moreover, connecting the validation nodes to each other increases the runtime. This experiment may have a different result if we experiment with other feature sets, or other edge selection strategies.



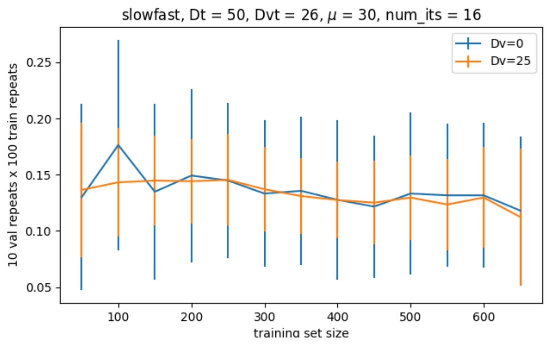
(a) Accuracy



(b) Runtime



(c) Missed detection



(d) False alarm

Figure 58: Performance and runtime of two similarity graph models where the only difference is the connectivity between pairs of validation nodes (Slowfast features).

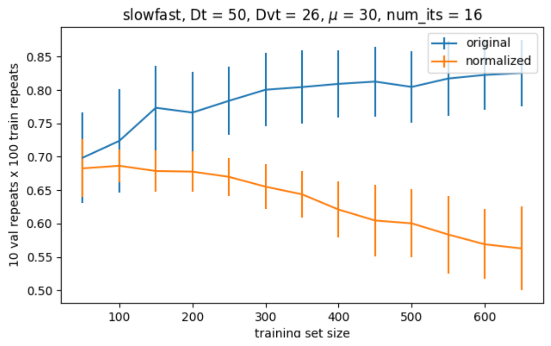
3.10 Feature Normalization

In this section, we test the effect of normalizing the feature vectors. We tested the B-GLR model on the Slowfast features. All model parameters remain the same between the two experiments. The only difference is that in one of the experiments, we normalized the feature vectors with Algorithm 5. The performance and runtime of both experiments are compared in Figure 59.

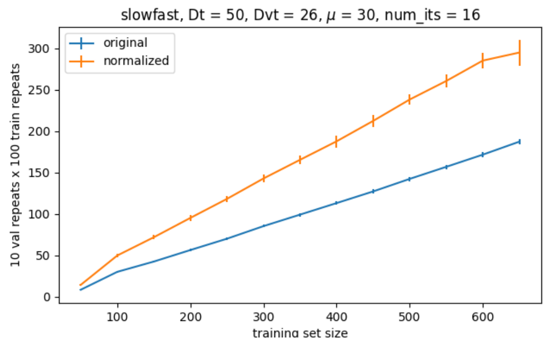
Algorithm 5 Feature Normalization

```
Require: $\mathcal{F} = \{\boldsymbol{f_i} | i = 0, ..., N_t - 1\}, \sigma_f, l_s$
Ensure: $0 \leq \sigma_f, l_s$
1: Feature-wise normalization:
2: find $\boldsymbol{m} = [m_i | i = 0, ..., C - 1]^T$    where $m_i$ is the mean of the $i$th entry of all feature vectors in $\mathcal{F}$.
3: $\boldsymbol{f_i} \leftarrow \boldsymbol{f_i} - \boldsymbol{m} \quad \forall i = 0, ..., N_t - 1$.
4: find $\boldsymbol{\Sigma} = [\sigma_i | i = 0, ..., C - 1]^T$    where $\sigma_i$ is the standard deviation of the $i$th entry of all feature vectors in $\mathcal{F}$.
5: $\boldsymbol{f_i} \leftarrow \sigma_f \, (\boldsymbol{f_i} \oslash \boldsymbol{\Sigma}) \quad \forall i = 0, ..., N_t - 1$,    where $\oslash$ stands for element-wise division and $\sigma_f$ is the desired standard deviation for features.
6: Sample-wise normalization:
7: $\boldsymbol{f_i} \leftarrow l_s \, (\boldsymbol{f_i} / \|\boldsymbol{f_i}\|_2)$,    where $l_s$ is the desired norm for each feature vector.
```

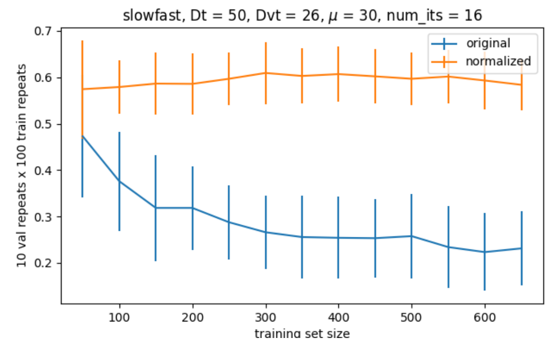
It is observed that normalizing the Slowfast features reduces the accuracy as the dataset size grows. Moreover, it increases the runtime as expected. The decrease in accuracy is mostly due the the increase in false detection rate rather than the missed detection rate. In conclusion, we should not normalize the extracted Slowfast features.



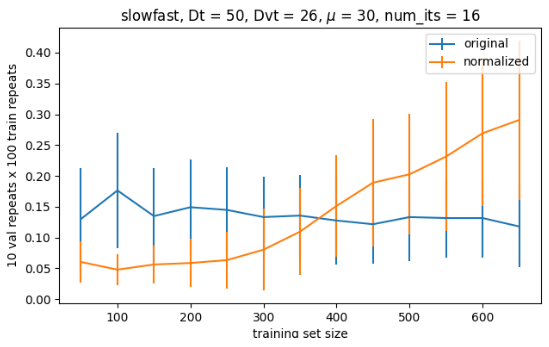
(a) Accuracy



(b) Runtime



(c) Missed detection



(d) False alarm

Figure 59: Performance and runtime of the B-GLR model with the same parameters on normalized Slowfast features compared with original Slowfast features.

4 Conclusion

This thesis demonstrates a classification model based on a similarity graph which provides more explainability than state-of-the-art methods, while achieving a comparable accuracy. The model learns a metric matrix defining the distance between the data points. The data points are represented as nodes in the similarity graph, and their distance to each other as the weights of the edges connecting them. By strategically choosing edges to make a sparse similarity graph, the model's complexity is reduced to $O(N_t\, C^2)$ where N_t is the number of training nodes and C is the number of features.

Each node of the similarity graph is assigned a label; the set of all labels form a *graph signal* which must be smooth, i.e. have a small graph Laplacian regularizer (GLR), to make a sensible similarity graph. More explanation can be found in Section 2.1. Based on this criterion, a training process is proposed in this thesis which is formalized in Equation (59). An alternative training process is proposed based on the well-studied Large Margin Nearest Neighbor approach which we call Graph-based Large Margin Nearest Neighbor (GLMNN). This process is formalized in Equation (68). The GLMNN can be solved as a semi-definite program (SDP) whose time complexity is $O(C^3 + C^{2.5}N_tD_t^2 + C^{0.5}N_t^{2.4}D_t^{4.8})$.

To reduce the time complexity, we use a technique called Gershgorin Disc Perfect Alignment (GDPA). More information about this technique is found in Section 2.2.5. The GDPA-infused GLMNN is formalized in Equation (75). This approach effectively reduces time complexity to $O((C + N_tD_t^2)^{2.055})$. We say "effectively" because the complexity depends on the constant number of linear programs (LP) that are solved during the GDPA-infused GLMNN. This constant number is experimentally demonstrated in Figure 53; however, we have no theoretical proof that the number of LPs will remain constant.

The model's explainability relies on the ability to analyze the optimized metric matrix. The diagonal elements of this matrix show the contribution of each single entry of the feature vector to the label's prediction; hence, they can be used as contribution scores similar to the scores that are calculated in several other Explainable AI approaches. Furthermore, the off-diagonal elements of the metric matrix show the relevance of pairs of feature vector entries to each other. We can interpret these elements as the learned covariance of the features.

In addition, the high valued elements of the metric matrix can serve to separate the most relevant features to the prediction from the irrelevant ones. Figure 31 shows several examples of this separation by marking the high-value elements with red dots. If the features are explainable themselves (i.e. if an explainable feature extraction algorithm is utilized), visualizing the highly informative features can lead to further discoveries about the prediction process. As an example, Figure 37 illustrates keypoints of the 3D-SIFT features used in a model. The location of these keypoints can unveil a reasoning behind their relevance to the prediction.

We extracted three different sets of features from our original dataset as described in Section 2.3.1. The first set of features is extracted by using the first-layer filters of a pre-trained CNN called "slowfast-r50". The “slowfast-r50” network is originally trained for action recognition in videos. The second set of features is a 3D version of the well-known Scale-Invariant Feature Transform (SIFT). A description of this algorithm was given in Section 2.3.3. The third and last feature set is extracted using pre-trained filters of a CNN used for simultaneous audio-video texture analysis called SOE-Net. Section 3.4 illustrates our model’s performance on all three feature sets. The performance on all feature sets is generally in the same range.

On the other hand, we compared the performance of our model with four benchmarks: XGBoost, K-Nearest Neighbors, Neural Network and Logistic Regression. Section 3.5 illustrates the accuracy of our model with the first proposed training process (called B-GLR) next to the benchmarks. Our model achieves comparable accuracy with the benchmarks. Running the same experiments in Section 3.5 with the SDP-GLMNN and GDPA-GLMNN approaches was not feasible since these methods are quite time-consuming. Hence, the performance of these two methods is compared with the B-GLR model and each other using smaller datasets in Section 3.6. It is witnessed that the GLMNN-based methods achieve a higher accuracy than B-GLR. Moreover, the time efficiency of GDPA-GLMNN is demonstrated compared to SDP-GLMNN.

Our model can take advantage of the similarities between the validation data points (as well as training data points) to predict the validation labels. The effectiveness of this property is tested in Section 3.9. Additionally, the effect of the similarity graph’s structure is explored in Sections 3.7 and 3.8. Section 3.10 examines the effect of normalizing the feature vectors on the prediction accuracy.

4.1 Future Work

For the continuation of this project, the proposed model can be used to predict the responses of smaller subgroups of neurons. Instead of one group which comprises all neurons, we can group neurons based on their type. RGCs of the same type respond to similar features in the stimulus; hence, analyzing the trained model for these smaller subgroups can reveal more useful information about the features that are extracted by the cells, or the parts of the visual input to which they are sensitive (i.e. their receptive fields). Since our current experiments consider a large group of all neurons, the analysis is not as useful or meaningful as it can be for neurobiological purposes. Taking this idea a step forward, we can even experiment on single ganglion cells instead of subgroups.

So far, we have focused on predicting single response signals whether that signal belongs to a single cell or it is a mix of several cells’ responses. However, it is observed that *patterns* of responses in a neuron population appear repeatedly regarding a stimulus; therefore, information may be encoded in the *pattern* rather than independent neuron responses. To account for this dependence, we can define a similar classification problem where the time bins’ labels are the *number* of spiking neurons in that specific moment; thus, the labels will no longer be binary. Rather, they will be whole numbers between zero to the number of all neurons in the dataset. Taking this idea one step further, we can label the time bins with vectors (instead of whole numbers) that contain the responses of *all* neurons in that specific moment. This would allow the model to differentiate between all the possible patterns that might arise in the population’s response to the stimulus.

Bibliography

[1] F. Wild, "Outline of a computational theory of human vision," in *KI 2005 Workshop 7 Mixed-reality as a challenge to image understanding and artificial intelligence*, p. 55, Citeseer, 2005.

[2] B. Belgio, A. P. Salvetti, S. Mantero, and F. Boschetti, "The evolution of fabrication methods in human retina regeneration," *Applied Sciences*, vol. 11, no. 9, p. 4102, 2021.

[3] D. Heeger, "Perception lecture notes: retinal ganglion cells." https://www.cns.nyu.edu/ david/courses/perception/lecturenotes/ganglion/ganglion.html, 2006.

[4] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[5] S. Fitzpatrick, "Linear Algebra: A second course, featuring proofs and Python." https://opentext.uleth.ca/Math3410/subsec-ortho-diag.html, 2022. Accessed: 2022-12-18.

[6] A. Ortega, *Introduction to graph signal processing*. Cambridge University Press, 2022.

[7] *Appendix C: Positive Semidefinite and Positive Definite Matrices*, pp. 259–263. John Wiley Sons, Ltd, 2007.

[8] A. Ortega, P. Frossard, J. Kovavcević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[9] W. Huang, T. A. W. Bolton, J. D. Medaglia, D. S. Bassett, A. Ribeiro, and D. Van De Ville, "A graph signal processing perspective on functional brain imaging," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 868–885, 2018.

[10] J. D. Medaglia, W. Huang, E. A. Karuza, A. Kelkar, S. L. Thompson-Schill, A. Ribeiro, and D. S. Bassett, "Functional alignment with anatomical networks is associated with cognitive flexibility," *Nature human behaviour*, vol. 2, no. 2, pp. 156–164, 2018.

[11] L. Rui, H. Nejati, S. H. Safavi, and N.-M. Cheung, "Simultaneous low-rank component and graph estimation for high-dimensional graph signals: Application to brain imaging," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4134–4138, IEEE, 2017.

[12] M. Ménoret, N. Farrugia, B. Pasdeloup, and V. Gripon, "Evaluating graph signal processing for neuroimaging through classification and dimensionality reduction," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 618–622, IEEE, 2017.

[13] L. Goldsberry, W. Huang, N. F. Wymbs, S. T. Grafton, D. S. Bassett, and A. Ribeiro, "Brain signal analytics from graph signal processing perspective," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 851–855, IEEE, 2017.

[14] C. Hu, L. Cheng, J. Sepulcre, K. A. Johnson, G. E. Fakhri, Y. M. Lu, and Q. Li, “A spectral graph regression model for learning brain connectivity of alzheimer’s disease,” *PloS one*, vol. 10, no. 5, p. e0128136, 2015.

[15] H. Behjat, N. Leonardi, L. Sörnmo, and D. Van De Ville, “Anatomically-adapted graph wavelets for improved group-level fmri activation mapping,” *NeuroImage*, vol. 123, pp. 185–199, 2015.

[16] N. Leonardi and D. Van De Ville, “Tight wavelet frames on multislice graphs,” *IEEE Transactions on Signal Processing*, vol. 61, no. 13, pp. 3357–3367, 2013.

[17] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974.

[18] I. Rotondo, G. Cheung, A. Ortega, and H. E. Egilmez, “Designing sparse graphs via structure tensor for block transform coding of images,” in *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pp. 571–574, IEEE, 2015.

[19] H. E. Egilmez, Y.-H. Chao, and A. Ortega, “Graph-based transforms for video coding,” *IEEE Transactions on Image Processing*, vol. 29, pp. 9330–9344, 2020.

[20] “Introduction to graph signal processing,” *Signals and Communication Technology*, pp. 3–108, 2019.

[21] A. Buades, B. Coll, and J.-M. Morel, “A non-local algorithm for image denoising,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 2, pp. 60–65, IEEE, 2005.

[22] W. Hu, X. Li, G. Cheung, and O. Au, “Depth map denoising using graph-based transform and group sparsity,” in *2013 IEEE 15th international workshop on multimedia signal processing (MMSP)*, pp. 001–006, IEEE, 2013.

[23] U. S. Kim, O. A. Mahroo, J. D. Mollon, and P. Yu-Wai-Man, “Retinal ganglion cells—diversity of cell types and clinical relevance,” *Frontiers in Neurology*, vol. 12, 2021.

[24] T. Gollisch and M. Meister, “Eye smarter than scientists believed: neural computations in circuits of the retina,” *Neuron*, vol. 65, no. 2, pp. 150–164, 2010.

[25] H. Barlow, R. Hill, and W. Levick, “Retinal ganglion cells responding selectively to direction and speed of image motion in the rabbit,” *The Journal of physiology*, vol. 173, no. 3, p. 377, 1964.

[26] W. R. Taylor and D. I. Vaney, “New directions in retinal research,” *Trends in neurosciences*, vol. 26, no. 7, pp. 379–385, 2003.

[27] J. B. Demb, “Cellular mechanisms for direction selectivity in the retina,” *Neuron*, vol. 55, no. 2, pp. 179–186, 2007.

[28] S. Hecht, S. Shlaer, and M. H. Pirenne, “Energy at the threshold of vision,” *Science*, vol. 93, no. 2425, pp. 585–587, 1941.

[29] B. Sakitt, “Counting every quantum,” *The Journal of Physiology*, vol. 223, no. 1, pp. 131–150, 1972.

[30] D. A. Baylor, T. Lamb, and K.-W. Yau, “Responses of retinal rods to single photons.,” *The Journal of physiology*, vol. 288, no. 1, pp. 613–634, 1979.

[31] D. M. Schneeweis and J. L. Schnapf, “Photovoltage of rods and cones in the macaque retina,” *Science*, vol. 268, no. 5213, pp. 1053–1056, 1995.

[32] H. Barlow, W. Levick, and M. Yoon, “Responses to single quanta of light in retinal ganglion cells of the cat,” *Vision research*, vol. 11, pp. 87–101, 1971.

[33] P. Sterling, M. A. Freed, and R. G. Smith, “Architecture of rod and cone circuits to the on-beta ganglion cell,” *Journal of Neuroscience*, vol. 8, no. 2, pp. 623–642, 1988.

[34] M. A. Freed, R. G. Smith, and P. Sterling, “Rod bipolar array in the cat retina: Pattern of input from rods and gaba-accumulating amacrine cells,” *Journal of Comparative Neurology*, vol. 266, no. 3, pp. 445–455, 1987.

[35] Y. Tsukamoto, K. Morigiwa, M. Ueda, and P. Sterling, “Microcircuits for night vision in mouse retina,” *Journal of Neuroscience*, vol. 21, no. 21, pp. 8616–8623, 2001.

[36] C. Enroth-Cugell and J. G. Robson, “The contrast sensitivity of retinal ganglion cells of the cat,” *The Journal of physiology*, vol. 187, no. 3, pp. 517–552, 1966.

[37] S. Hochstein and R. Shapley, “Linear and nonlinear spatial subunits in y cat retinal ganglion cells.,” *The Journal of physiology*, vol. 262, no. 2, pp. 265–284, 1976.

[38] J. Caldwell and N. Daw, “New properties of rabbit retinal ganglion cells.,” *The Journal of Physiology*, vol. 276, no. 1, pp. 257–276, 1978.

[39] E. Kaplan and R. Shapley, “X and y cells in the lateral geniculate nucleus of macaque monkeys.,” *The Journal of Physiology*, vol. 330, no. 1, pp. 125–143, 1982.

[40] J. B. Demb, L. Haarsma, M. A. Freed, and P. Sterling, “Functional circuitry of the retinal ganglion cell’s nonlinear receptive field,” *Journal of Neuroscience*, vol. 19, no. 22, pp. 9756–9767, 1999.

[41] D. Petrusca, M. I. Grivich, A. Sher, G. D. Field, J. L. Gauthier, M. Greschner, J. Shlens, E. Chichilnisky, and A. M. Litke, “Identification and characterization of a y-like primate retinal ganglion cell type,” *Journal of Neuroscience*, vol. 27, no. 41, pp. 11019–11027, 2007.

[42] S. Martinez-Conde, S. L. Macknik, and D. H. Hubel, “The role of fixational eye movements in visual perception,” *Nature reviews neuroscience*, vol. 5, no. 3, pp. 229–240, 2004.

[43] P. Hammond and A. Smith, “On the sensitivity of complex cells in feline striate cortex to relative motion,” *Experimental Brain Research*, vol. 47, no. 3, pp. 457–460, 1982.

[44] B. Frost and K. Nakayama, "Single visual neurons code opposing motion independent of direction," *Science*, vol. 220, no. 4598, pp. 744–745, 1983.

[45] R. T. Born and R. B. Tootell, "Segregation of global and local motion processing in primate middle temporal visual area," *Nature*, vol. 357, no. 6378, pp. 497–499, 1992.

[46] J. Y. Lettvin, H. R. Maturana, W. S. McCulloch, and W. H. Pitts, *What the frog's eye tells the frog's brain*. na, 1965.

[47] B. P. Ölveczky, S. A. Baccus, and M. Meister, "Segregation of object and background motion in the retina," *Nature*, vol. 423, no. 6938, pp. 401–408, 2003.

[48] T. A. Münch, R. A. Da Silveira, S. Siegert, T. J. Viney, G. B. Awatramani, and B. Roska, "Approach sensitivity in the retina processed by a multifunctional neural circuit," *Nature neuroscience*, vol. 12, no. 10, pp. 1308–1316, 2009.

[49] D. Baylor, A. Hodgkin, and T. Lamb, "The electrical response of turtle cones to flashes and steps of light," *The Journal of Physiology*, vol. 242, no. 3, pp. 685–727, 1974.

[50] J. Schnapf, B. Nunn, M. Meister, and D. Baylor, "Visual transduction in cones of the monkey macaca fascicularis.," *The Journal of physiology*, vol. 427, no. 1, pp. 681–713, 1990.

[51] R. M. Shapley and J. D. Victor, "The effect of contrast on the transfer properties of cat retinal ganglion cells.," *The Journal of physiology*, vol. 285, no. 1, pp. 275–298, 1978.

[52] J. D. Victor, "The dynamics of the cat retinal x cell centre.," *The Journal of physiology*, vol. 386, no. 1, pp. 219–246, 1987.

[53] S. A. Baccus and M. Meister, "Fast and slow contrast adaptation in retinal circuitry," *Neuron*, vol. 36, no. 5, pp. 909–919, 2002.

[54] M. J. Berry, I. H. Brivanlou, T. A. Jordan, and M. Meister, "Anticipation of moving stimuli by the retina," *Nature*, vol. 398, no. 6725, pp. 334–338, 1999.

[55] T. H. Bullock, M. H. Hofmann, F. K. Nahm, J. G. New, and J. C. Prechtl, "Event-related potentials in the retina and optic tectum of fish," *Journal of neurophysiology*, vol. 64, no. 3, pp. 903–914, 1990.

[56] T. H. Bullock, S. Karamürsel, J. Z. Achimowicz, M. C. McClune, and C. Bacsar-Eroglu, "Dynamic properties of human visual evoked and omitted stimulus potentials," *Electroencephalography and clinical neurophysiology*, vol. 91, no. 1, pp. 42–53, 1994.

[57] J. J. McAnany and K. R. Alexander, "Is there an omitted stimulus response in the human cone flicker electroretinogram?," *Visual neuroscience*, vol. 26, no. 2, pp. 189–194, 2009.

[58] G. Schwartz, R. Harris, D. Shrom, and M. J. Berry, "Detection and prediction of periodic patterns by the retina," *Nature neuroscience*, vol. 10, no. 5, pp. 552–554, 2007.

[59] G. Schwartz and M. J. Berry 2nd, “Sophisticated temporal pattern recognition in retinal ganglion cells,” *Journal of neurophysiology*, vol. 99, no. 4, pp. 1787–1798, 2008.

[60] B. Werner, P. B. Cook, and C. L. Passaglia, “Complex temporal response patterns with a simple retinal circuit,” *Journal of neurophysiology*, vol. 100, no. 2, pp. 1087–1097, 2008.

[61] F. J. Montáns, F. Chinesta, R. Gómez-Bombarelli, and J. N. Kutz, “Data-driven modeling and learning in science and engineering,” *Comptes Rendus Mécanique*, vol. 347, no. 11, pp. 845–855, 2019.

[62] R. O. Duda, “Modeling head related transfer functions,” in *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pp. 996–1000, IEEE, 1993.

[63] Z.-S. Hou and Z. Wang, “From model-based control to data-driven control: Survey, classification and perspective,” *Information Sciences*, vol. 235, pp. 3–35, 2013.

[64] F. Mazzocchi, “Could big data be the end of theory in science? a few remarks on the epistemology of data-driven science,” *EMBO reports*, vol. 16, no. 10, pp. 1250–1255, 2015.

[65] C. Anderson, “The end of theory: The data deluge makes the scientific method obsolete,” *Wired magazine*, vol. 16, no. 7, pp. 16–07, 2008.

[66] S. Leonelli, “What difference does quantity make? on the epistemology of big data in biology,” *Big data & society*, vol. 1, no. 1, p. 2053951714534395, 2014.

[67] M. T. Ribeiro, S. Singh, and C. Guestrin, “” why should i trust you?” explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[68] C. Otte, “Safe and interpretable machine learning: A methodological review,” *Computational intelligence in intelligent data analysis*, pp. 111–122, 2013.

[69] N. Bostrom and E. Yudkowsky, “The Ethics of Artificial Intelligence,” *Artificial Intelligence Safety and Security*, pp. 57–69, jul 2018.

[70] B. Goodman and S. Flaxman, “European union regulations on algorithmic decision-making and a “right to explanation”,” *AI magazine*, vol. 38, no. 3, pp. 50–57, 2017.

[71] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, “Hidden technical debt in machine learning systems,” *Advances in neural information processing systems*, vol. 28, 2015.

[72] L. Jiang, J. D. Hwang, C. Bhagavatula, R. L. Bras, M. Forbes, J. Borchardt, J. Liang, O. Etzioni, M. Sap, and Y. Choi, “Delphi: Towards machine ethics and norms,” *arXiv preprint arXiv:2110.07574*, 2021.

[73] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” *arXiv preprint arXiv:1702.08608*, 2017.

[74] L. K. Hansen and L. Rieger, “Interpretability in intelligent systems–a new concept?,” in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pp. 41–49, Springer, 2019.

[75] T. Miller, “Explanation in artificial intelligence: Insights from the social sciences,” *Artificial Intelligence*, vol. 267, pp. 1–38, 2019.

[76] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, “Explainable ai: A review of machine learning interpretability methods,” *Entropy*, vol. 23, no. 1, p. 18, 2020.

[77] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” *arXiv preprint arXiv:1312.6034*, 2013.

[78] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” *Advances in neural information processing systems*, vol. 30, 2017.

[79] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian, “Certifying and removing disparate impact,” in *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 259–268, 2015.

[80] B. H. Zhang, B. Lemoine, and M. Mitchell, “Mitigating unwanted biases with adversarial learning,” in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 335–340, 2018.

[81] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, “Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index,” *Computer physics communications*, vol. 181, no. 2, pp. 259–270, 2010.

[82] R. Cukier, C. Fortuin, K. E. Shuler, A. Petschek, and J. Schaibly, “Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. i theory,” *The Journal of chemical physics*, vol. 59, no. 8, pp. 3873–3878, 1973.

[83] M. D. Morris, “Factorial sampling plans for preliminary computational experiments,” *Technometrics*, vol. 33, no. 2, pp. 161–174, 1991.

[84] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[85] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

[86] Y. Li, L. Li, L. Wang, T. Zhang, and B. Gong, “Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks,” in *International Conference on Machine Learning*, pp. 3866–3876, PMLR, 2019.

[87] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[88] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008. Similarity Matching in Computer Vision and Multimedia.

[89] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*, pp. 778–792, Springer, 2010.

[90] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[91] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[92] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[93] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[94] P. C. Mahalanobis, "On the generalized distance in statistics," *Sankhyā: The Indian Journal of Statistics, Series A (2008-)*, vol. 80, pp. S1–S7, 2018.

[95] S. Xiang, F. Nie, and C. Zhang, "Learning a mahalanobis distance metric for data clustering and classification," *Pattern recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

[96] H. Ghorbani, "Mahalanobis distance and its application for detecting multivariate outliers," *Facta Universitatis, Series: Mathematics and Informatics*, pp. 583–595, 2019.

[97] O. Marre, D. Amodei, N. Deshmukh, K. Sadeghi, F. Soo, T. E. Holy, and M. J. Berry, "Mapping a complete neural population in the retina," *Journal of Neuroscience*, vol. 32, no. 43, pp. 14859–14873, 2012.

[98] L. McIntosh, N. Maheswaranathan, A. Nayebi, S. Ganguli, and S. Baccus, "Deep learning models of the retinal response to natural scenes," *Advances in neural information processing systems*, vol. 29, 2016.

[99] H. Tanaka, A. Nayebi, N. Maheswaranathan, L. McIntosh, S. Baccus, and S. Ganguli, "From deep learning to mechanistic understanding in neuroscience: the structure of retinal prediction," *Advances in neural information processing systems*, vol. 32, 2019.

[100] B. Rister, M. A. Horowitz, and D. L. Rubin, "Volumetric image registration from invariant keypoints," *IEEE Transactions on Image Processing*, vol. 26, no. 10, pp. 4900–4910, 2017.

[101] S. Niu, Y. Liu, J. Wang, and H. Song, "A decade survey of transfer learning (2010–2020)," *IEEE Transactions on Artificial Intelligence*, vol. 1, no. 2, pp. 151–166, 2020.

[102] C. Lemaréchal, “Cauchy and the gradient method,” *Doc Math Extra*, vol. 251, no. 254, p. 10, 2012.

[103] K. Q. Weinberger, J. Blitzer, and L. Saul, “Distance metric learning for large margin nearest neighbor classification,” in *Advances in Neural Information Processing Systems* (Y. Weiss, B. Schölkopf, and J. Platt, eds.), vol. 18, MIT Press, 2005.

[104] Z.-q. Luo, W.-k. Ma, A. M.-c. So, Y. Ye, and S. Zhang, “Semidefinite relaxation of quadratic optimization problems,” *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 20–34, 2010.

[105] J. F. Sturm, “Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones,” *Optimization methods and software*, vol. 11, no. 1-4, pp. 625–653, 1999.

[106] R. H. Tütüncü, K.-C. Toh, and M. J. Todd, “Solving semidefinite-quadratic-linear programs using sdpt3,” *Mathematical programming*, vol. 95, pp. 189–217, 2003.

[107] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023.

[108] I. CVX Research, “CVX: Matlab software for disciplined convex programming, version 2.0.” http://cvxr.com/cvx, Aug. 2012.

[109] H. Jiang, T. Kathuria, Y. T. Lee, S. Padmanabhan, and Z. Song, “A faster interior point method for semidefinite programming,” in *2020 IEEE 61st annual symposium on foundations of computer science (FOCS)*, pp. 910–918, IEEE, 2020.

[110] C. Yang, G. Cheung, and W. Hu, “Signed graph metric learning via gershgorin disc perfect alignment,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 7219–7234, 2021.

[111] C. Yang, “Signed Graph Metric Learning via Gershgorin Disc Alignment.” https://github.com/bobchengyang/SGML, 2021.

[112] L. Elsner, “Gervsgorin and his circles,” 2006.

[113] S. Jiang, Z. Song, O. Weinstein, and H. Zhang, “Faster dynamic matrix inverse for faster lps,” *arXiv preprint arXiv:2004.07470*, 2020.

[114] E. M. Gene Cheung, *Graph Spectral Image Processin*. ISTE Ltd, 2021. Chapter 9.2.1.

[115] O. Marre, G. Tkacik, D. Amodei, E. Schneidman, W. Bialek, and M. Berry, “Multi-electrode array recording from salamander retinal ganglion cells,” 2017.

[116] G. Tkavcik, O. Marre, D. Amodei, E. Schneidman, W. Bialek, and M. J. Berry, “Searching for collective behavior in a large network of sensory neurons,” *PLoS computational biology*, vol. 10, no. 1, p. e1003408, 2014.

[117] N. Brenner, S. P. Strong, R. Koberle, W. Bialek, and R. R. d. R. v. Steveninck, “Synergy in a neural code,” *Neural computation*, vol. 12, no. 7, pp. 1531–1552, 2000.

[118] H. F. et al., "PyTorchVideo: A deep learning library for video understanding," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021. https://pytorchvideo.org/.

[119] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6202–6211, 2019.

[120] B. Rister, "Sift3d." https://github.com/bbrister/SIFT3D, 2019.

[121] I. Hadji and R. P. Wildes, "A spatiotemporal oriented energy network for dynamic texture recognition," in *ICCV*.

[122] I. Hadji, "SOE-Net." https://github.com/hadjisma/SOE-Net, 2020. Accessed in year 2023.

[123] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[124] OpenCV, "Introduction to SIFT (Scale-Invariant Feature Transform)." https://docs.opencv.org/4.x/da/df5/tutorial$_{py_sift_intro.html,}$2023.

[125] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 357–360, 2007.

[126] D. Ni, Y. Qu, X. Yang, Y. P. Chui, T.-T. Wong, S. S. Ho, and P. A. Heng, "Volumetric ultrasound panorama based on 3d sift," in *International conference on medical image computing and computer-assisted intervention*, pp. 52–60, Springer, 2008.

[127] W. Cheung and G. Hamarneh, "*n*-sift: *n*-dimensional scale invariant feature transform," *IEEE Transactions on Image Processing*, vol. 18, no. 9, pp. 2012–2021, 2009.

[128] A. Klaser, M. Marszalek, and C. Schmid, "A spatio-temporal descriptor based on 3d-gradients," in *BMVC 2008-19th British Machine Vision Conference*, pp. 275–1, British Machine Vision Association, 2008.

[129] R. W. Farebrother, *Linear least squares computations*. Routledge, 2018.

[130] J. L. Nazareth, "Conjugate gradient method," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 3, pp. 348–353, 2009.

[131] J. R. Shewchuk *et al.*, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[132] Y. Parhizkar, "Binary classification via a similarity graph." https://github.com/yasamanparhizkar/simgraph, 2023.

[133] NVIDIA, "XGBoost." https://www.nvidia.com/en-us/glossary/data-science/xgboost/, 2023.

[134] M. Maalouf, “Logistic regression in data analysis: an overview,” *International Journal of Data Analysis Techniques and Strategies*, vol. 3, no. 3, pp. 281–299, 2011.

[135] M. O. Documentation, “Call MATLAB from Python.” https://www.mathworks.com/help/matlab/matlab-engine-for-python.html?category=matlab-engine-for-pythons$_t id = CRUX_t opnav,$ 2023.

Appendices

A Proof of Equation (49)

This section proves that the GLR (Equation (48)) can be written in summation format as Equation (49).

$$
\begin{aligned}
GLR = \boldsymbol{y^T}\, \boldsymbol{L y} &= \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} L_{ij} y_i y_j = \sum_{i=0}^{N_t-1}\sum_{j=0, j\neq i}^{N_t-1} -W_{ij} y_i y_j + \sum_{i=0}^{N_t-1} (D_{ii} - W_{ii}) y_i^2 \\
&= \sum_{i=0}^{N_t-1}\sum_{j=0, j\neq i}^{N_t-1} -W_{ij} y_i y_j \sum_{i=0}^{N_t-1} \left(\sum_{p=0}^{N_t-1} W_{ip} - W_{ii} \right) y_i^2 \\
&= -\sum_{i=0}^{N_t-1}\sum_{j=0, j\neq i}^{N_t-1} W_{ij} y_i y_j - \sum_{i=0}^{N_t-1} W_{ii}\, y_i^2 + \sum_{i=0}^{N_t-1}\sum_{p=0}^{N_t-1} W_{ip} y_i^2 \\
&= -\sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} W_{ij} y_i y_j + \sum_{i=0}^{N_t-1}\sum_{p=0}^{N_t-1} W_{ip} y_i^2 .
\end{aligned}
\tag{97}
$$

On the other hand, we know that the graph is undirected; therefore, its adjacency matric $\boldsymbol{W}$ is symmetric; hence, we can write

$$
\begin{aligned}
\sum_{i=0}^{N_t-1}\sum_{p=0}^{N_t-1} W_{ip} y_i^2 &= \sum_{p=0}^{N_t-1}\sum_{i=0}^{N_t-1} W_{pi} y_p^2 \\
&= \sum_{p=0}^{N_t-1}\sum_{i=0}^{N_t-1} W_{ip} y_p^2 = \sum_{i=0}^{N_t-1}\sum_{p=0}^{N_t-1} W_{ip} y_p^2 .
\end{aligned}
\tag{98}
$$

From Equation (97) and (98), we can conclude that

$$
\begin{aligned}
2\, GLR &= -2 \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} W_{ij} y_i y_j + \sum_{i=0}^{N_t-1}\sum_{p=0}^{N_t-1} W_{ip} y_i^2 + \sum_{i=0}^{N_t-1}\sum_{p=0}^{N_t-1} W_{ip} y_p^2 \\
&= -2 \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} W_{ij} y_i y_j + \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} W_{ij} y_i^2 + \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} W_{ij} y_j^2 \\
&= \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} W_{ij} \left(y_i^2 + y_j^2 - 2\, y_i\, y_j \right) \\
&= \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} W_{ij} \left(y_i - y_j \right)^2 \ . \blacksquare
\end{aligned}
\tag{99}
$$

B Derivation of Loss In Equation (59) For Gradient Descent

Equation (59), which describes one possible approach to train the similarity graph model, is solved via the Gradient Descent (GD) method in this thesis. GD requires the derivative of the objective with respect to the optimization variables. This section calculates the aforementioned derivative to be used in GD. In addition, the time complexity of computing the derivative is analyzed.

B.1 Problem Description

$$\begin{aligned}
\text{Given}\quad & \boldsymbol{B} \ \text{ s.t. } \boldsymbol{B} \in \mathbb{R}^{C\times C} \ , && (100)\\
& \boldsymbol{M} = \boldsymbol{B}^T \boldsymbol{B} \ , && (101)\\
& \text{The graph's adjacency matrix: } \boldsymbol{W} \in \mathbb{R}^{N_t \times N_t} \ , && (102)\\
& W_{ij} = e^{-d_{ij}} \ , && (103)\\
& d_{ij} = \left(\boldsymbol{f}^{(i)} - \boldsymbol{f}^{(j)}\right)^T \boldsymbol{M} \left(\boldsymbol{f}^{(i)} - \boldsymbol{f}^{(j)}\right) \ , && (104)\\
& \boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} = \operatorname{diag}(\boldsymbol{W}\mathbf{1}) - \boldsymbol{W} \ , && (105)\\
& r = \boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y} \ , && (106)\\
& l = \mu \operatorname{tr}(\boldsymbol{M}) \ , && (107)\\
& E = r + l \ . && (108)\\
\text{Find}\quad & \frac{dE}{d\boldsymbol{B}}. && (109)
\end{aligned}$$

B.2 Solution

$\frac{dE}{d\boldsymbol{B}}$ is defined as

$$\frac{dE}{d\boldsymbol{B}} = \left[\frac{dE}{dB_{ij}} \quad \forall i,j = 0,\ldots,C-1\right] = \begin{bmatrix} \frac{dE}{dB_{00}} & \frac{dE}{dB_{01}} & \cdots & \frac{dE}{dB_{0(C-1)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dE}{dB_{(C-1)0}} & \frac{dE}{dB_{(C-1)1}} & \cdots & \frac{dE}{dB_{(C-1)(C-1)}} \end{bmatrix} \in \mathbb{R}^{C\times C}. \quad (110)$$

By applying the sum rule of derivation, we have

$$\frac{dE}{d\boldsymbol{B}} = \frac{dr}{d\boldsymbol{B}} + \frac{dl}{d\boldsymbol{B}}. \quad (111)$$

Therefore, this solution has three parts.
In Part 1, we prove $\frac{dl}{d\boldsymbol{B}} = 2\mu\boldsymbol{B}$.
In Part 2, we prove $\frac{dr}{d\boldsymbol{B}} = \boldsymbol{B}\left[\frac{dr}{d\boldsymbol{M}} + \left(\frac{dr}{d\boldsymbol{M}}\right)^T\right]$.
In Part 3, we prove $\frac{dr}{dM_{st}} = -\frac{1}{2}\operatorname{sum}\left(\boldsymbol{W} \odot \boldsymbol{Y} \odot \boldsymbol{F_s} \odot \boldsymbol{F_t}\right)$, where $\odot$ stands for element-wise multiplication.

B.3 Part 1

$$l = \mu\, tr(\boldsymbol{M}) = \mu \sum_{i=0}^{C-1} M_{ii}. \tag{112}$$

If we show column i of $\boldsymbol{B}$ with $\boldsymbol{b_i}$, then we have

$$\boldsymbol{M} = \boldsymbol{B}^T\boldsymbol{B} \implies M_{ii} = \|\boldsymbol{b_i}\|_2^2, \tag{113}$$

which implies

$$\frac{dM_{ii}}{dB_{st}} = \left\{ \begin{array}{cc} 0\,, & t \neq i \\ 2\,B_{st}\,, & t = i \end{array} \right\}. \tag{114}$$

Thus, we can write

$$\begin{aligned} &\frac{dl}{dB_{st}} = \mu \sum_{i=0}^{C-1} \frac{dM_{ii}}{dB_{st}} = 2\,\mu\,B_{st} \\ \implies &\frac{dl}{d\boldsymbol{B}} = \left[\frac{dl}{dB_{st}} \quad \forall s,t = 0,...,C-1\right] = 2\mu\,[B_{st} \quad \forall s,t = 0,...,C-1] = 2\,\mu\,\boldsymbol{B}\,.\blacksquare \end{aligned} \tag{115}$$

B.4 Part 2

According to the chain rule of derivation, we have

$$\frac{dr}{dB_{ij}} = \sum_{s=0}^{C-1}\sum_{t=0}^{C-1} \frac{dr}{dM_{st}} \times \frac{dM_{st}}{dB_{ij}}. \tag{116}$$

On the other hand, we can write

$$\boldsymbol{M} = \boldsymbol{B}^T\boldsymbol{B} \implies M_{st} = \sum_{p=0}^{C-1} B_{ps}\,B_{pt} \implies \frac{dM_{st}}{dB_{ij}} = \left\{ \begin{array}{cc} 0\,, & j \neq s, j \neq t \\ B_{it}\,, & j = s, j \neq t \\ B_{is}\,, & j \neq s, j = t \\ 2\,B_{ij}\,, & j = s, j = t \end{array} \right\}. \tag{117}$$

Hence,

$$
\begin{aligned}
\frac{dr}{dB_{ij}} &= \sum_{s=0,s\neq j}^{C-1}\sum_{t=0,t\neq j}^{C-1}\frac{dr}{dM_{st}}\times 0+\sum_{s=0,s\neq j}^{C-1}\frac{dr}{dM_{sj}}\times\frac{dM_{sj}}{dB_{ij}}+\sum_{t=0,t\neq j}^{C-1}\frac{dr}{dM_{jt}}\times\frac{dM_{jt}}{dB_{ij}}+\frac{dr}{dM_{jj}}\times\frac{dM_{jj}}{dB_{ij}}\\
&= \sum_{s=0,s\neq j}^{C-1}\frac{dr}{dM_{sj}}\times B_{is}+\sum_{t=0,t\neq j}^{C-1}\frac{dr}{dM_{jt}}\times B_{it}+\frac{dr}{dM_{jj}}\times(2\,B_{ij}) \qquad (118)\\
&= \sum_{s=0}^{C-1}\frac{dr}{dM_{sj}}\times B_{is}+\sum_{t=0}^{C-1}\frac{dr}{dM_{jt}}\times B_{it}.
\end{aligned}
$$

We notice that various terms of Equation (118) correspond to rows and columns of known matrices. Equation (119) illustrates the resemblance,

$$
\begin{aligned}
&\left[\frac{dr}{dM_{sj}}\quad \forall s=0,\ldots,C-1\right]=j\text{ 'th column of }\frac{dr}{d\boldsymbol{M}}\\
&\left[B_{is}\quad \forall s=0,\ldots,C-1\right]=i\text{ 'th row of }\boldsymbol{B}\\
&\left[\frac{dr}{dM_{jt}}\quad \forall t=0,\ldots,C-1\right]=j\text{ 'th row of }\frac{dr}{d\boldsymbol{M}}\\
&\left[B_{it}\quad \forall t=0,\ldots,C-1\right]=i\text{ 'th row of }\boldsymbol{B}.
\end{aligned}
\qquad (119)
$$

Using Equation (119), we can write Equation (118) as matrix multiplications. Thus, we have

$$
\frac{dr}{d\boldsymbol{B}}=\left[\frac{dr}{dB_{ij}}\quad \forall i,j=0,\ldots,C-1\right]=\boldsymbol{B}\,\frac{dr}{d\boldsymbol{M}}+\boldsymbol{B}\left(\frac{dr}{d\boldsymbol{M}}\right)^{T}=\boldsymbol{B}\left[\frac{dr}{d\boldsymbol{M}}+\left(\frac{dr}{d\boldsymbol{M}}\right)^{T}\right].\blacksquare \qquad (120)
$$

B.5 Part 3

Using Equation (49), we can write

$$
\frac{dr}{dW_{ij}}=\frac{1}{2}\left(y_i-y_j\right)^2. \qquad (121)
$$

Let us define

$$
\boldsymbol{f}^{(i)}-\boldsymbol{f}^{(j)}\triangleq\boldsymbol{f}^{(ij)}. \qquad (122)
$$

Then, we can use the definition of matrix multiplication to write

$$
d_{ij}=\left(\boldsymbol{f}^{(ij)}\right)^{T}\boldsymbol{M}\,\boldsymbol{f}^{(ij)}=\sum_{s=0}^{C-1}\sum_{t=0}^{C-1}M_{st}\,f_s^{(ij)}\,f_t^{(ij)}, \qquad (123)
$$

which implies

$$
\frac{dW_{ij}}{dM_{st}}=-e^{-d_{ij}}\,f_s^{(ij)}\,f_t^{(ij)}. \qquad (124)
$$

Using Equations (121) and (124), we can write

$$\begin{aligned}\frac{dr}{dM_{st}} &= \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} \frac{dr}{dW_{ij}} \times \frac{dW_{ij}}{dM_{st}} = \sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} -\frac{1}{2}\,(y_i-y_j)^2\; e^{-d_{ij}}\; f_s^{(ij)}\; f_t^{(ij)} \\ &= -\frac{1}{2}\sum_{i=0}^{N_t-1}\sum_{j=0}^{N_t-1} (y_i-y_j)^2\; W_{ij}\; f_s^{(ij)}\; f_t^{(ij)}.\end{aligned} \tag{125}$$

Let us define the following matrices.

$$\boldsymbol{Y} \triangleq \left[(y_i-y_j)^2 \quad \forall i,j=0,...,N_t-1\right] \tag{126}$$

$$\boldsymbol{F_s} \triangleq \left[\left(f_s^{(i)}-f_s^{(j)}\right) \quad \forall i,j=0,...,N_t-1\right] \tag{127}$$

$$\boldsymbol{F_t} \triangleq \left[\left(f_t^{(i)}-f_t^{(j)}\right) \quad \forall i,j=0,...,N_t-1\right]. \tag{128}$$

Using the above definitions, we can rewrite Equation (125) in a compact way as in Equation 129,

$$\frac{dr}{dM_{st}} = -\frac{1}{2}\,\mathrm{sum}\left(\boldsymbol{Y}\odot\boldsymbol{W}\odot\boldsymbol{F_s}\odot\boldsymbol{F_t}\right), \tag{129}$$

where $\odot$ means element-wise multiplication and sum($\boldsymbol{A}$) adds all elements of $\boldsymbol{A}$. ■

B.6 Time Complexity

In this section, we count the number of operations to evaluate $\frac{dE}{d\boldsymbol{B}}$.

1. First, we compute $\frac{dr}{dM_{st}}$ as in Equation (129), which comprises three element-wise multiplications (i.e. $3\,N_t^2$ operations) and one sum($\cdot$) (i.e. N_t^2 operations).

2. Now, we obtain $\frac{dr}{d\boldsymbol{M}} = \left[\frac{dr}{dM_{st}}\ \forall s,t=0,...,C-1\right]$. This means that step 1 is repeated C^2 times.

3. We obtain $\frac{dr}{d\boldsymbol{B}} = \boldsymbol{B}\left[\frac{dr}{d\boldsymbol{M}} + \left(\frac{dr}{d\boldsymbol{M}}\right)^T\right]$. C^2 summations are done inside the bracket, plus C^2 multiplications on the bracket's result and $\boldsymbol{B}$.

4. Finally, we evaluate $\frac{dE}{d\boldsymbol{B}} = \frac{dr}{d\boldsymbol{B}} + 2\mu\boldsymbol{B}$. Computing the second term takes C^2 multiplications. Moreover, C^2 summations are made between the two terms.

In total, $4N_t^2C^2 + 4C^2$ operations are done to evaluate $\frac{dE}{d\boldsymbol{B}}$ for one iteration of GD, where N_t is the number of training nodes and C is the number of features.

Since we run GD for a fixed number of iterations in our approach, the total time complexity of our model's training, with B-GLR optimization, is $O(4N_t^2C^2)$. The other computations in our approach (constructing training and validation datasets and edge selection) are less complex; therefore, our model's complexity is decided by derivation evaluation.

The above analysis assumes that the graph is complete; i.e., all entries of $\boldsymbol{W}$ can be non-zero. If we fix the degree of the nodes, the number of edges in the graph will have a linear relationship with the number of nodes; i.e., only a maximum of $2D_tN_t$ elements of $\boldsymbol{W}$ are non-zero. Hence, step 1 will take only $O(N_t)$ operations, instead of $O(N_t^2)$. As a result, the time complexity of the derivation will be reduced to $O(4N_tC^2+4C^2)$ which is linear with respect to N_t. In this case, if our edge selection takes linear time, the whole training will have a linear time complexity.

In our current implementation of time-based edge selection (Algorithms 2, 4), we sort the edges by their time indices which takes $O(N_tlog(N_t))$ to compute; hence, sorting becomes the bottleneck and increases the model's complexity to $O(N_tlog(N_t))$. However, if we assume that the feature vectors are given to the similarity graph according to their temporal order, then sorting is not necessary; hence, the model's time complexity remains $O(N_t)$.

C How to run the scripts?

The developed model in this thesis, and all the experiments, can be accessed on GitHub [132]. This section provides instructions on how to use the model in your own work. Furthermore, we introduce several helpful Jupyter notebooks; these notebooks can be read as tutorials of using various features of our model, extracting features from a video, or to reproduce the same results presented in this thesis. All directories in this section are given relative to the GitHub repository's root directory in [132].

C.1 Main Scripts

The core functionalities of our code are stored in the following directory: */my_packages/*. This folder contains four sub-directories: */dataprocess/*, */simgraph/*, */assessment/*, and */package_tests/*. Figure 60 shows the structure of the main scripts.

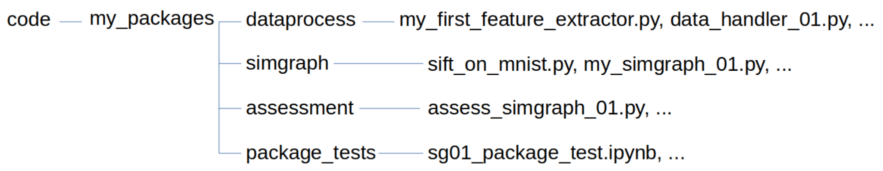

code
my_packages
dataprocess
my_first_feature_extractor.py, data_handler_01.py, ...
simgraph
sift_on_mnist.py, my_simgraph_01.py, ...
assessment
assess_simgraph_01.py, ...
package_tests
sg01_package_test.ipynb, ...

Figure 60: The structure of the scripts implementing the similarity graph model

Inside */my_packages/dataprocess/*, four Python scripts are listed: *my_first_feature_extractor.py* belongs to the preliminary experiment. It downloads MNIST images and prepares training and validation datasets for the similarity graph model as explained in Section 3.1. The next three files (*data_handler_01.py*, etc.) belong to the expanded (i.e. official) code; they represent three versions in chronological order. A description of each version's modifications is written at the top of the script, plus extensive documentation of all the APIs inside. Two APIs for creating random training and validation sets are provided. One of these APIs makes sure that the sets are equally distributed between the labels, while the other one just randomly picks data points from the whole dataset.

/my_packages/simgraph/ contains the model's core files. *sift_on_mnist.py* belongs to the preliminary experiment, while all the other files (*my_simgraph_01.py*, etc.) belong to the expanded code. Again, numbers at the end of filenames indicate chronological order; the main version, which is used in most of the experiments, is *my_simgraph_06.py*. Documentation for all the APIs and the package as a whole is written inside the script. *my_simgraph_06.py* only implements the first proposed training optimization as in Equation (59). The GLMNN-based optimizations are implemented in Matlab and accessed from Python notebooks using a Matlab Engine [135]. SDP-GLMNN is implemented in */15_cheng/lmnn-offtheshelf/lmnn_cvx_python.m*, and GDPA-GLMNN is implemented in */15_cheng/lmnn-gdpa-2/lmnn_gdpa_python.m*

Most of the APIs in *my_simgraph_06.py* are used for edge selection, implementing all proposed strategies in this thesis (Algorithm 1,2,3,4). Then, there are three APIs that implement the loss (see Equation (50)), the

Gradient Descent optimization [102], and estimation of validation labels (see Equation (85)) by using the Conjugate Gradient algorithm (see Section 2.3.6). Furthermore, two auto-run APIs are provided: *fit_graph* which automates the training process, and *get_acc* which automates the validation process. Lastly, several utility APIs are supplied for analysis and visualization of the trained model.

/my_packages/assessment/ holds scripts that automate the randomized experiments (*assess_simgraph_01.py*, *assess_simgraph_02.py*, etc.). They run numerous experiments with various sizes of training and validation datasets and plot the results. To form training and validation datasets with different sizes, random data points are chosen from the whole dataset.

assess_simgraph_02.py appraises the first proposed training optimization as in Equation (59). The scripts following *assess_simgraph_02.py* each appraise and compare two methods together; both methods are trained and validated on the exact same datasets. The methods' names are included in the script's name; for instance, *assess_simgraph_03_gdpaobj2_lgrg.py* runs randomized experiments with GDPA-GLMNN and Logistic Regression [134] using the same datasets for both methods. Then, it compares the results of the two models. Appendix C.2 gives a detailed account of all the plots and files generated by an assessment script.

C.2 Assessment Script Outputs

The assessment scripts run randomized experiments with all three training approaches and all the mentioned features in this thesis. These scripts create several plots and files which are all stored in a user-given directory. Figure 61 illustrates a tree structure of all the files and directories created by an assessment script.

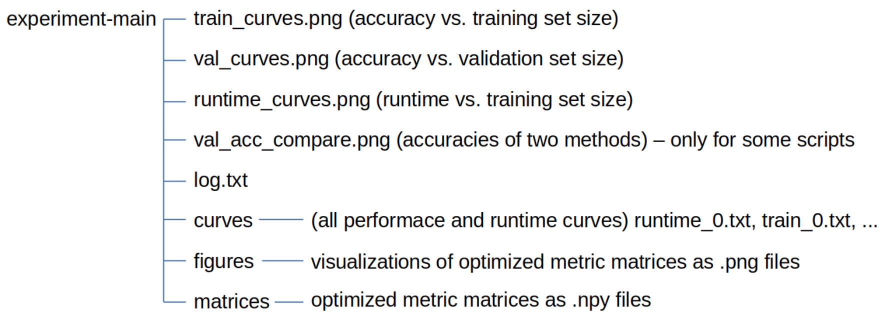

experiment-main
train_curves.png (accuracy vs. training set size)
val_curves.png (accuracy vs. validation set size)
runtime_curves.png (runtime vs. training set size)
val_acc_compare.png (accuracies of two methods) – only for some scripts
log.txt
curves
(all performace and runtime curves) runtime_0.txt, train_0.txt, ...
figures
visualizations of optimized metric matrices as .png files
matrices
optimized metric matrices as .npy files

Figure 61: All generated files by an assessment script.

The first three images in Figure 61 are explained in Section 3.2. *train_curves.png* plots four performance quantities with respect to various training set sizes. An example is given in Figure 28a. *val_curves.png* computes the mean and standard deviation of the performance quantities over all training set sizes and plots them with respect to the validation set sizes as seen in Figure 28b. *runtime_curves.png* plots the experiments' runtimes for each training set size. An example is provided in Figure 30.

Assessment scripts that compare two methods generate an extra plot illustrating the obtained accuracies of both methods. This extra plot is stored as *val_acc_compare.png*. All performance quantities and runtimes that are plotted in the aforementioned images are also stored in text files in the */curves/* directory. */figures/* contains visualizations of the optimized metric matrices of all repeats of the experiment the likes of which are given in Figure 31. Additionally, the optimized metric matrices are stored as Numpy arrays in */matrices/*. Lastly, the performance-related quantities of all repeats of the experiment are stored in a file named *log.txt*.

C.3 Helpful Jupyter Notebooks

Several Jupyter notebooks are provided that serve as tutorials of how to use the scripts. Please note that older Jupyter notebooks use older versions of the main scripts; you can not use a newer version of a script with older notebooks. The first notebook can be accessed in the following directory:
/14_comparison/08_cvxlmnn_gdpalmnn_factobj1.ipynb.
This notebook runs a single experiment (as opposed to multiple randomized experiments) on all three proposed training objectives; i.e. B-GLR (Equation (59)), SDP-GLMNN (Equation (68)) and GDPA-GLMNN (Equation (75)). Several other Jupyter notebooks demonstrate how to use the assessment scripts to run randomized experiments, including the following:
/14_comparison/09_cvxlmnn_factobj1_assess.ipynb
/14_comparison/10_cvxlmnn_gdpalmnn_assess.ipynb

These notebooks start by importing suitable versions of the main scripts explained in Appendix C.1. Then, they load the neurons' spike data (see Figure 19) on RAM. Afterwards, they set the parameters required to retrieve the correct set of features for the model's input. Four types of features are provided as described in Section 2.3.2; Slowfast, 3D-SIFT and SOE-Net features that describe the fish movie, and a fourth set of SIFT features describing MNIST images. The parameters for all feature sets are written in the notebook; one only needs to uncomment the corresponding lines to use the features.

For further information about the process of extracting the aforementioned features, refer to the following Jupyter notebook.
For Slowfast:

- */07_slowfast/02_slowfast_ft.ipynb*
- */07_slowfast/03_half_slowfast_ft.ipynb*
- */07_slowfast/04_slowfast_ft_sanity.ipynb*

For 3D-SIFT:

- */09_sift3d/01_sift3d_ft.ipynb*
- */09_sift3d/02_sift3d_kpselection.ipynb*
- */09_sift3d/05_save_framebatch_data.ipynb*

- */09_sift3d/08_visualize_sift_keypoints.ipynb*

For SOE-Net:

- */13_soenet/01_soenet_ft.ipynb*
- */13_soenet/03_merge_and_convert_to_numpy.ipynb*

For SIFT features from the MNIST dataset:

- */03_mnist/03_sift_on_mnist/sift_on_mnist_06.ipynb*
- */03_mnist/04_nn_vs_sift/sift_vs_nn_on_mnist.ipynb*

After setting up the data parameters, the scripts continue to set up model-related parameters. These include parameters that describe the similarity graph's connectivity, the Gradient Descent algorithm, free scalars in the optimization objectives, the random seed, and parameters about the metric matrix visualization as seen in Figure 31. Assessment notebooks determine an extra set of parameters in this step regarding the randomization of the experiment. Moreover, assessment notebooks determine the directory where the experiment's results will be stored.

We used the code found in [111] for GDPA-GLMNN which is written in Matlab. Consequently, the code for SDP-GLMNN was written by making changes to GDPA-GLMNN's code; hence, SDP-GLMNN is also written in Matlab. To run these modules from Python, we use a Matlab Engine [135]. Since data files need to be transferred between our Matlab and Python scripts, we designate certain directories to save and load data files. We call these directories "swap directories". Swap directories are also given to the model as parameters. There are several additional steps in *08_cvxlmnn_gdpalmnn_factobj1.ipynb* which compare the results of all three training approaches with Logistic Regression.

To reproduce the preliminary experiment (Section 3.1), refer to the following notebook:
/03_mnist/04_nn_vs_sift/sift_vs_nn_on_mnist.ipynb
Two more Python scripts can be found in the same directory containing the implementations of the similarity graph model and the neural network benchmark.

To run random experiments with all four benchmarks introduced in Section 3.5, refer to the following notebook.
code/11_knn_and_lgrg/07_knn_xgb_nn_lgrg_assess.ipynb
Additionally, this notebook plots the results next to the similarity graph's performance curves.

The following Jupyter notebook analyzes the B-GLR model on 2D synthetic data as in Section **??**
/05_simple_features_toy_examples/07_toy_example.ipynb

To produce feature visualizations akin to the ones in Section 3.3, refer to the following notebooks:
/16_mnist_sift_on_sg/10_visualize_features.ipynb
/09_sift3d/08_visualize_sift_keypoints.ipynb