Unraveling Optic Nerve Shapes: A Topological Dive into Intracranial Idiopathic Hypertension (IIH) through Fundus Photo Analysis

Josue Antonio Consultant: Brad Turow
Department of Mathematics, Northeastern University

December 9th 2023

Abstract

Idiopathic Intracranial Hypertension (IIH) is a rare condition that affects the optic nerve head, often resulting in severe vision impairment. The objective of this project is to leverage Topological Data Analysis (TDA) techniques to extract topological information from retinal scans of IIH patients. This information will then be used as input vectors for a classification model. The model's goal is to accurately determine the severity of IIH based on a patient's scan. By pursuing this approach, we aim to identify distinctive shape features associated with the optic nerve head and the changes that manifest as the disease progresses. Our ultimate aim is to contribute to more effective diagnoses for patients with IIH.

1 Introduction

Intracranial Idiopathic Hypertension (IIH), with an incidence of approximately 1 per 100,000 population per year [1], presents symptoms like blurred vision, double vision, and vision loss. Diagnosis involves *Fundus photography* and optical coherence tomography (OCT) scans, with specialists assigning a Frisen grade (ranging from 1 to 5, where 5 indicates severe cases) based on these images (refer to Figure 1).

The manual analysis of these images is laborious, time-consuming, subjective, and prone to errors, highlighting the pressing need for automated diagnostic methods. Unfortunately, IIH has received less attention compared to more prevalent diseases like glaucoma in the realm of medical image analysis.

Topological data analysis' ultimate objective is to extract insightful knowledge from complex data, often challenging to attain through traditional statistical or machine learning approaches. By using topological data analysis, we aim to bridge this gap. Our focus is to identify unique shape characteristics of the

optic nerve head–bright circular structure at the center of each fundus photo–linked to IIH and track these changes over time, enabling better understanding and management of the condition.

2 Data Source

Our dataset is provided by the NORDIC group at Mount Sinai Hospital, under the direction of Dr. Mark Kupersmith, a renowned expert in IIH and related conditions. The dataset originates from the Idiopathic Intracranial Hypertension Treatment Trial, which is the largest prospectively analyzed cohort of untreated IIH patients [1]. This study enrolled 165 IIH patients from 38 NORDIC sites in the United States and Canada over a 3-year period. Fundus photography is a type of medical imaging where pictures of the back part of the eye, called the fundus, are taken. The dataset consists of 6,566 fundus photographs, each labeled with Frisen grades ranging from 0 to 5. Note that grade 0 fundus photos correspond a control group made up of scans from healthy eyes. It's also important to note that the dataset exhibits a significant class imbalance (to be resolved in section 6), with grade 5 images accounting for less than 2 percent of the data, while grade 2 images constitute about a third of the data.

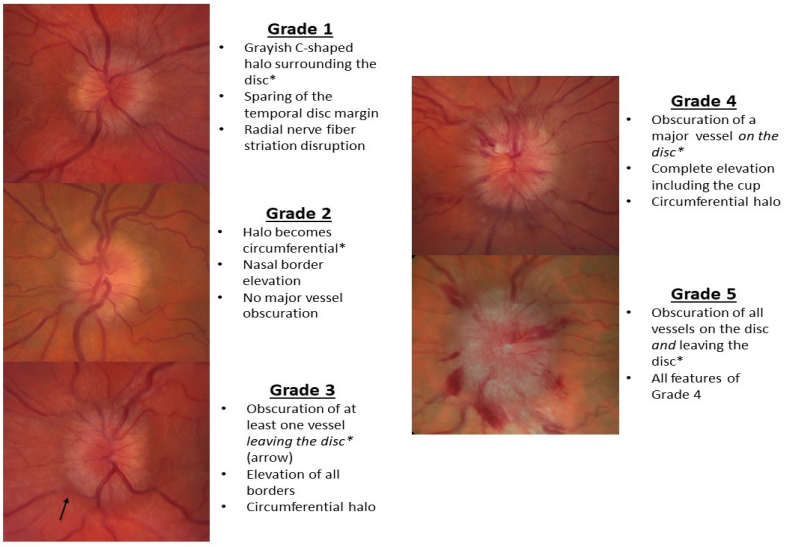


Figure 1: Modified Frisen scale for grading of IIH. Key features of each grade are marked with an asterisk (*). Scale from the Case Report of IIH found in [4].

3 Data Preprocessing

Data preprocessing is an important step in the analysis of any dataset. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific data analysis task. In particular, to work with the fundus photos the first step is to read, resize, and convert each of the images to gray scale. For this step, I wrote the following Python function:

```python
def preprocess_images(path, new_size):
    """ Process all images (png's) found in given file path
        by resizing and converting them to grayscale
        Args: location of images (path -- string),
              new img size (new_size -- int)
    """

    # Initialize list to store all images
    images = []

    # Loop through all files in given path
    for filename in os.listdir(path):

        # Check if file is a fundus image
        if filename.endswith(".png"):

            # Read fundus image
            img_path = os.path.join(path, filename)
            img = Image.open(img_path)

            # Resize image to (new_size x new_size) size
            img_resized = img.resize((new_size, new_size))

            # Convert to grayscale
            img_gray = ImageOps.grayscale(img_resized)

            # Add preprocessed image to images list
            images.append(np.array(img_gray))
    return np.stack(images, axis=-1)
```

Figure 2: Function to preprocess fundus photos

We iterate through all fundus photos, resize and convert them to gray scale. We then store the preprocessed photos (as arrays of numbers) in a list. Note: We are currently storing the whole images but we will attempt to store the optic nerve head region only, by cropping and isolating this region from each fundus photo, at a later time.

After preprocessing all the images of each (Frisen) grade, we now have lists containing all the photos and we're ready to extract topological features from them.

4 Topological Data Analysis

Topological data analysis (TDA) involves employing techniques from algebraic topology to analyze high-dimensional, incomplete, and noisy datasets. At the intersection of algebraic topology and data science, TDA provides methods to study the structure of data using low-dimensional topological invariants.

One way of achieving this is through the use of *Persistent homology*, which, roughly speaking, captures how topological features of the data persist across different scales. Homology is one way in which topologists have formalized the idea of measuring the presence of n-dimensional holes in a topological space. Persistent homology is the multi-scale version of homology when applied to data. Connectedness refers to the connected components of a space, and it's captured by the 0-th dimensional homology group, H_0. Loops (1-dimensional holes) are given by H_1, and voids (2-dimensional holes) are captured by H_2. Higher-dimensional homology groups exist, but understanding them in terms of visualizing "holes" (components, loops, voids) is more challenging compared to

the more intuitive concepts of holes in dimensions 0, 1, and 2. Since our images are not topological spaces, we have to "topologize" the data.

A set of the form

$$L_c(f) = \{(x_1, \cdots, x_n) : f(x_1, \cdots, x_n) \leq c\}$$

is called a sublevel set of f. One way of topologizing our data is by constructing simplicial complexes according to the *Sublevel sets* of the following function

$$f : V \to \mathbb{R} \tag{1}$$

that assigns each of the image's pixels to their corresponding intensities, where V is the set of vertices (we can think of pixels as vertices) of a given image. A simplicial complex is a set composed of *n-simplices* (vertices, line segments, triangles, and their n-dimensional counterparts) corresponding to a particular scale given by (1). In general, if you're working with n-dimensional data, the maximum dimension of the simplices is also n, which only allows for $(n-1)$-dimensional features to be observed. In our analysis of two-dimensional fundus photos, we only consider 0- and 1-dimensional features, that is (connected) components and loops.

4.1 "Topologizing" Images

Given a two-dimensional gray scale image I, we start with the empty complex K. We then assign a vertex to each pixel in I and add these vertices (0-simplices) to K. We place a line segment (1-simplex) between two vertices that are horizontally, vertically, or diagonally adjacent. In this context, adjacent vertices refer to those that correspond to pixels that are right next to each other in I. K is now a fully-connected simplicial complex of I consisting of all vertices in I, and all the corresponding line segments connecting adjacent vertices. We now define our filtration function and construct the sublevel set filtration of I.

Let V be set of vertices of K and $f : V \to [0, 255]$ be the pixel intensity function assigning an intensity value for each pixel in I. Notice that we replaced the codomain of f from $\mathbb{R}$ to $[0, 255]$ since we're working with gray scale images. Let f_{min}, f_{max} denote the minimum and maximum values of f, i.e., 0 and 255, respectively. Let $a_1 < a_2 < \cdots$ be an increasing sequence of positive real numbers.

Then

$$K_i = \{\sigma \in K : \forall v \in \sigma, f(v) \leq i\}.$$

In words, each complex (K_i) is made from a subset of pixels (σ) that all fall below a pixel intensity value (pixels that are dimmer relative to the intensity value i).

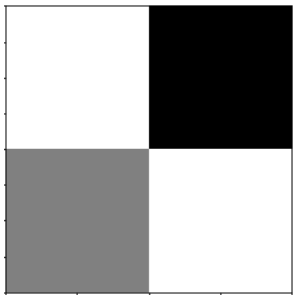
Then the sequence of subcomplexes:

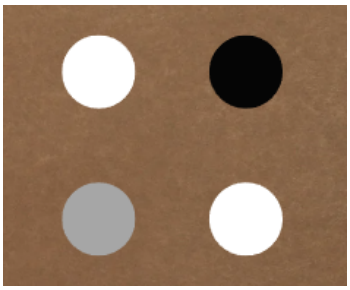
$$K_{f_{min}=0} \subseteq K_{0+a_1} \subseteq K_{0+a_2} \subseteq \cdots \subseteq K_{f_{max}=255} = K$$

is a *filtration* of K called the sublevel set filtration. We can think of a filtration as a collection of subcomplexes K_i of some complex K such that if $i \leq j$, then $K_i \subseteq K_j$.

4.1.1 Sublevel Set Filtration Example

Consider the following gray scale image with two pixels with intensity 255 (white), one pixel with intensity 0 (black), and one pixel with intensity 127.5 (gray).

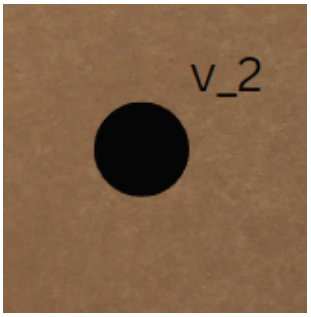




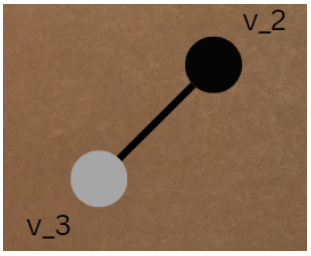
Let $V = \{v_1, v_2, v_3, v_4\}$ be the set of vertices of the image and let K be the fully-connected complex defined in Section 4.1 including all four vertices/pixels of our gray scale image.

Then:

$$K_0 = \{\sigma \in K : \forall v \in \sigma, f(v) \leq 0\} = \{v_2\}$$

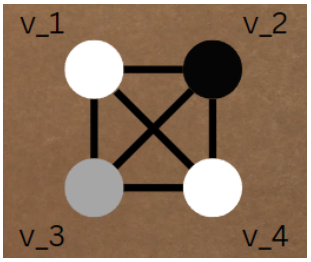


$$K_{127.5} = \{\sigma \in K : \forall v \in \sigma, f(v) \leq 127.5\} = \{v_2, v_3\}$$



Notice that we now have two 0-simplices and one 1-simplex since we have (diagonally) adjacent vertices.

$$K_{255} = \{\sigma \in K : \forall v \in \sigma, f(v) \leq 255\} = \{v_1, v_2, v_3, v_4\}$$



Notice that when the pixel intensity scale is high enough, all four vertices are part of our simplex and it becomes fully connected.

Hence,

$$K_0 \subseteq K_{127.5} \subseteq K_{255} = K$$

is the sublevel set filtration on pixel intensity of K, which is a family of topological representations of our image.

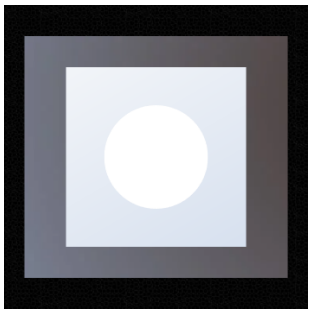
4.2 Persistence Diagrams

After the construction of each complex for a particular intensity scale, we compute homology groups of the complex and observe the changes in homology across all the complexes for all scales. We then describe these changes in terms of *birth* and *death* of each topological feature captured by homology. A feature (component or loop) is said to be born when it's first captured at some scale, and it's said to die when that feature no longer appears at some other scale.

The lifetime of these features is presented as a persistence diagram. A persistence diagram is a 2D representation, where each point represents a (born, death) pair. The x or *birth* axis signifies the particular scale (in our case pixel intensity) when a feature is born, while the y or *death* axis represents at which scale the feature died. In the diagram, points with high y-values, situated far above the diagonal line $y = x$, correspond to persistent features that exist over a wide range of intensity scales. Conversely, points near the diagonal represent features with short lifespans. It's essential to note that points considerably above the diagonal line represent significant topological features within your data space, providing valuable insights into its underlying structure. In contrast, points near the diagonal typically signify noise or less influential variations. This concise yet powerful representation enables us to visually and quantitatively grasp the enduring and noteworthy topological features, making it easier to distinguish them from inconsequential elements as we explore the data across various scales.

Let us tie the ideas of computing the sublevel set filtration of a fundus photo and computing the corresponding persistence diagram. The diagram captures features across the varying scales of the filtration. For simplicity's sake, assume that in a fundus photo, pixel intensities increase as you get closer to the optic nerve head, which we will assume corresponds to white pixels.

Let our simplified fundus photo be:

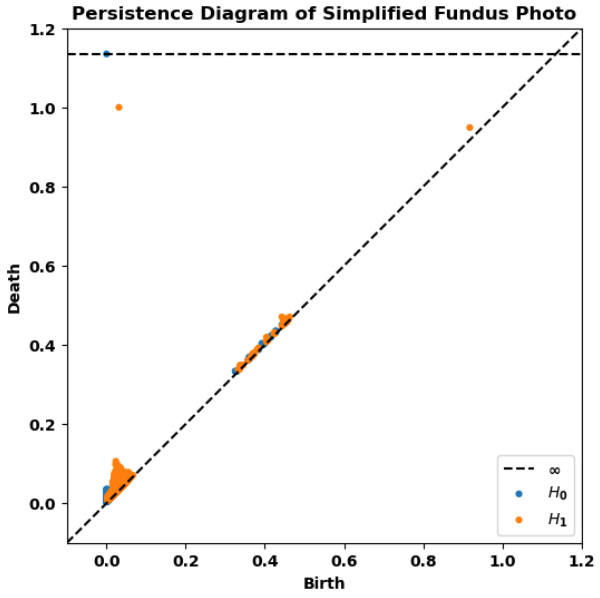


If we imagine what the sublevel set filtration of this image looks like, we can say that for small scales, homology will detect the bright circular region as a loop (1-dimensional hole). This happens because the intensity scale isn't large enough for the pixels in this region to be part of the simplicial complexes at these small scales.

As you increase the scale, more and more pixels become part of these complexes, but the optic nerve head will still be detected as a loop. When the scale is sufficiently large (at least equal to 255 = white), the pixels from the optic nerve head are included, and the complex contains all the image's pixels, making it fully connected.

From this analysis, we expect that in dimension 1, the corresponding persistence diagram will have one point far from the diagonal. This point represents the persistence of the bright region across many different scales. We would also see points near the diagonal, representing the gradient of colors from black to brighter ones, which we can clearly observe in the image.

Computing the persistence diagram for our simplified fundus photo yields



and we see that in dimension 1 (represented by the orange points), there's one point far from the diagonal indicating the presence of the optic nerve head (hole) in our image. At this point, our primary focus will be on 1-dimensional homology for machine learning purposes. With this, I invite you to consider why we observe only one persistent point in dimension 0 (represented by the blue points), indicating that our image has one connected component. Now, we have a step-by-step process that allows us to go from a photo to a simplicial complex to a filtration of complexes, and finally to a persistence diagram summarizing the information gathered from the filtration. The next step involves extracting

a feature vector from the diagram, which we can then input into a machine learning classification model.

Unfortunately, even though persistence diagrams provide valuable insights into topological features, they present a unique challenge when we try to use them in machine learning. This challenge arises because persistence diagrams don't fit into the usual framework of vector spaces that machine learning models rely on. Machine learning models typically require an inner product space, and thus directly incorporating persistence diagrams can be tricky. While persistence diagrams form a kind of metric space, using measures like the bottleneck distance or p-Wasserstein distance, many machine learning algorithms need more than just a metric space. For example, how do you calculate the average of a collection of persistence diagrams? As discussed in reference [6], this average might not be unique. If you want to learn more about the space of persistence diagrams, please read source [7]. The fact that persistence diagrams lack a vector space structure makes it challenging to integrate these valuable topological insights into machine learning pipelines.

4.3 Featurization Methods

Now, let's tackle a pivotal challenge: Featurization methods. These are the key to transforming persistence diagrams into numerical feature vectors. As we just stated, despite the valuable insights they offer, these diagrams don't fit neatly into the vector space typically used in machine learning.

4.3.1 Persistence Landscapes

The first method we'll explore transforms persistence diagrams into collections of piecewise linear functions. These linear functions are then discretized, creating feature vectors for our classification model. Discretization involves selecting specific points along the function to capture essential information. In the context of persistence landscapes, this simplification process condenses each landscape into key data points, making it more manageable for analysis and computation.

From [5] (Bubenik), for a persistence diagram $D = \{(a_i, b_i)\}_{i \in I}$ (recall a_i refers to when a feature was born, and b_i refers to when the feature died), we can define the persistence landscape as follows. First, for $a < b$, define

$$f_{(a,b)}(t) = max(0, min(a + t, b - t)) \tag{2}$$

Then

$$\lambda_k(t) = kmax\{f_{(a_i,b_i)}(t)\}_{i \in I},$$

where $k$max denotes the k-th largest element.

The persistence landscape is the sequence of piecewise linear functions, $\lambda_1, \lambda_2, \cdots : \mathbb{R} \to \mathbb{R}$. Bubenik shows desirable properties for working with persistence landscapes in statistical modeling, in particular that even if unique means do not exist in the set of persistence diagrams, persistence landscapes do have

unique means and the mean landscape converges to the expected persistence landscape [5] (Bubenik).

Consider the persistence diagram with $(0.01, 1)$ and $(0.01, 0.02)$ as the only points. Then

$$f_{(0.01,1)}(t) = max(0, min(0.01 + t, 1 - t))$$
$$f_{(0.01,0.02)}(t) = max(0, min(0.01 + t, 0.02 - t))$$

The maximum value of the function defined in (3) occurs when $t = \frac{b-a}{2}$. We get this by setting $a + t = b - t$ and solving for t. We take the largest possible $(b - a)$ difference, and in this case: $t = \frac{1-0.01}{2} = \frac{0.99}{2} = 0.495$ In order to discretize each of the linear functions, let $t \in \{0, 0.2475, 0.495, 0.7425\}$. Hence, for $t = 0$, we have

$$f_{(0.01,1)}(0) = max(0, min(0.01, 1)) = 0.01$$
$$f_{(0.01,0.02)}(0) = max(0, min(0.01, 0.02)) = 0.01$$
$$\lambda_1(0) = 1max\{f_{(0.01,1}(0), f_{(0.01,0.02)}(0)\} = 1max\{0.01, 0.01\} = 0.01$$
$$\lambda_2(0) = 2max\{f_{(0.01,1}(0), f_{(0.01,0.02)}(0)\} = 2max\{0.01, 0.01\} = 0.01$$

For $t = 0.2475$, we have

$$f_{(0.01,1)}(0.2475) = max(0, min(0.01 + 0.2475, 1 - 0.2475)) = 0.2575$$
$$f_{(0.01,0.02)}(0.2475) = max(0, min(0.01 + 0.2475, 0.02 - 0.2475)) = 0$$
$$\lambda_1(0.2475) = 1max\{f_{(0.01,1}(0.2475), f_{(0.01,0.02)}(0.2475)\} = 1max\{0, 0.2575\} = 0.2575$$
$$\lambda_2(0.2475) = 2max\{f_{(0.01,1}(0.2475), f_{(0.01,0.02)}(0.2475)\} = 2max\{0, 0.2575\} = 0$$

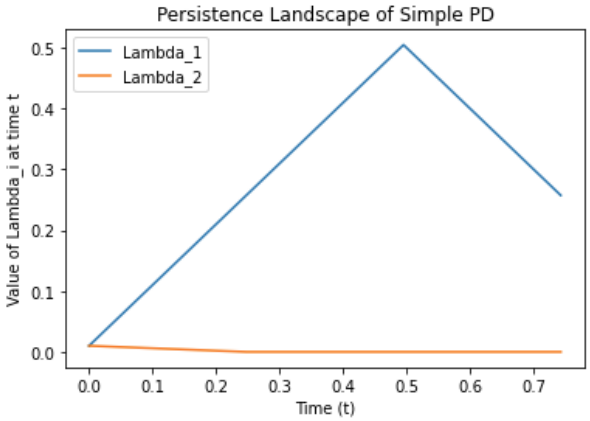
For $t = 0.495$, we have

$$f_{(0.01,1)}(0.495) = max(0, min(0.01 + 0.495, 1 - 0.495)) = 0.505$$
$$f_{(0.01,0.02)}(0.495) = max(0, min(0.01 + 0.495, 0.02 - 0.495)) = 0$$
$$\lambda_1(0.495) = 1max\{f_{(0.01,1)}(0.495), f_{(0.01,0.02)}(0.495)\} = 1max\{0, 0.505\} = 0.505$$
$$\lambda_2(0.495) = 2max\{f_{(0.01,1)}(0.495), f_{(0.01,0.02)}(0.495)\} = 2max\{0, 0.505\} = 0$$

Lastly, for $t = 0.7425$, we have

$$f_{(0.01,1)}(0.7425) = max(0, min(0.01 + 0.7425, 1 - 0.7425)) = 0.2575$$
$$f_{(0.01,0.02)}(0.7425) = max(0, min(0.01 + 0.7425, 0.02 - 0.7425)) = 0$$
$$\lambda_1(0.7425) = 1max\{f_{(0.01,1)}(0.7425), f_{(0.01,0.02)}(0.7425)\} = 1max\{0, 0.2575\} = 0.2575$$
$$\lambda_2(0.7425) = 2max\{f_{(0.01,1)}(0.7425), f_{(0.01,0.02)}(0.7425)\} = 2max\{0, 0.2575\} = 0$$

We now have $\lambda_1(0)$, $\lambda_1(0.2475)$, $\lambda_1(0.495)$, $\lambda_1(0.7425)$, and $\lambda_2(0)$, $\lambda_2(0.2475)$, $\lambda_2(0.495)$, $\lambda_2(0.7425)$. Therefore, our persistence landscape would look something like this



Recall the diagram points are $(0.01, 1), (0.01, 0.02)$, and notice how only the first one has a large y-value, i.e., it took many different scales for the feature to die, and hence it's a point representing a significant topological feature of our made up data. This checks out when we look at the persistence landscape since we can see only λ_1 is capturing information while λ_2 starts with a small y-value and it quickly becomes a zero constant function.

We've learned to construct persistence landscapes to encapsulate topological information from persistence diagrams. By transforming these landscapes into discrete feature vectors (discretizing), we can readily feed them into machine learning models.

4.3.2 Persistence Images

The second method we'll explore converts a persistence diagram into a 2D $n \times m$ image called persistence image which we can then *flatten* to obtain a feature vector with $n \times m$ elements.

From [8] (Adams et al.), let $D = \{(a_i, b_i)\}_{i \in I}$ be a persistence diagram in birth-death coordinates. Let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be the linear transformation $T(a, b) = (a, b-a)$, and let $T(D)$ be the transformed multiset in birth-persistence coordinates, where each point $(a, b) \in D$ corresponds to a point $(a, b-a) \in T(D)$. Let $\phi_u : \mathbb{R}^2 \to \mathbb{R}$ be a differentiable probability distribution with mean $u = (u_a, u_b) \in \mathbb{R}^2$. For our classification model purposes, we'll choose this distribution to be the normalized symmetric Gaussian $\phi_u = g_u$ with mean u and variance σ^2 defined as

$$g_u(a, b) = \frac{1}{2\pi\sigma^2} e^{\frac{-[(a-u_a)^2+(b-u_b)^2]}{2\sigma^2}}$$

but other distributions may be used. We then fix a nonnegative weighting function $f : \mathbb{R}^2 \to \mathbb{R}$ that's zero along the horizontal axis, continuous, and piecewise differentiable. For our purposes, we'll use a weighting function which only depends on the *death* coordinate b.

Let $n \in \mathbb{Z}^+$ and let $f_n(a, b) = b^n$. We choose this weighting function because we want to attribute more *weight* to PD points that persist longer, i.e., points corresponding to pixel intensity values that represent significant topological features. Note that the parameter n determines the *strength* of the weight assigned to more persistent features.

We're now one step closer to converting a persistence diagram into a persistence image.

Definition 1 *Let D be a PD, then the corresponding persistence surface $\rho_D : \mathbb{R}^2 \to \mathbb{R}$ is the function*

$$\rho_D(z) = \sum_{u \in T(D)} f(u)\phi_u(z)$$

With this surface, we now discretize a relevant subdomain of $\mathbb{R}^2$ and integrate the surface $\rho_D(z)$ over each region in the discretization to reduce the original surface to a finite-dimensional vector. In particular, the subdomain will be a fixed grid in the plane with n boxes (pixels) and we assign to each the integral of $\rho_D(z)$ over that region. We're now ready to define the persistence image of a diagram D.

Definition 2 *Let D be a PD, then its persistence image is the collection of pixels $I(\rho_D)_p = \iint_p \rho_D dbda$.*

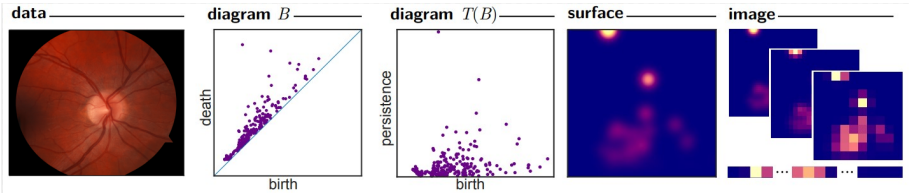


Figure 3: Algorithm pipeline to transform data into a persistence image at different resolutions. Figure adapted from [8] p.9

This section draws extensively from §4 in [8]. For a more comprehensive understanding of persistence images, readers are encouraged to refer to §5 of the referenced work. In that section, Adams et al. discuss the stability of persistence images with different choices of the probability distribution function, weighting function, and resolution (grid) of the image.

5 Random Forest Classification Model

In the landscape of machine learning, Random Forests emerge as a formidable strategy, particularly in the context of our medical image classification framework. This ensemble method, rooted in the decision tree family, brings a collective strength to predictive modeling.

Decision Trees, the foundational components of Random Forests, are intuitive models that make decisions based on the values of different features within the data. Think of them as a series of questions (if-statements) that lead to a final decision or prediction. However, while Decision Trees offer transparency in their decision-making process, they often risk *overfitting*. Overfitting occurs when a model captures too much detail from the training data, which is the subset of data used to train or teach the machine learning model. In this context, the model may memorize noise and specific details from the training data, potentially misrepresenting the broader patterns of the full dataset. This phenomenon emphasizes the delicate balance between capturing intricate details in the training phase and ensuring the model's ability to generalize well to unseen data, and it's referred as the *Bias–variance tradeoff*.

To address this, we turn to the ensemble approach embodied by Random Forests. Rather than relying on a single Decision Tree, a Random Forest consists of a multitude of trees, each trained on a random subset of features and a *bootstrapped* sample of the data. Here, bootstrapped refers to the statistical technique of sampling with replacement from the dataset. This means that each tree in the Random Forest is trained on a subset of the data created by randomly selecting observations from the dataset, allowing some instances to be sampled multiple times while others may not be included at all. The bootstrapped sampling introduces a crucial element of randomness during both the training and prediction phases, promoting diversity within the ensemble and enhancing the overall robustness of the model. When it comes to making predictions, the ensemble averages or votes on the individual tree predictions, resulting in a more stable and accurate overall model. The step of taking into account the majority voting from all the trees is called *aggregation*.

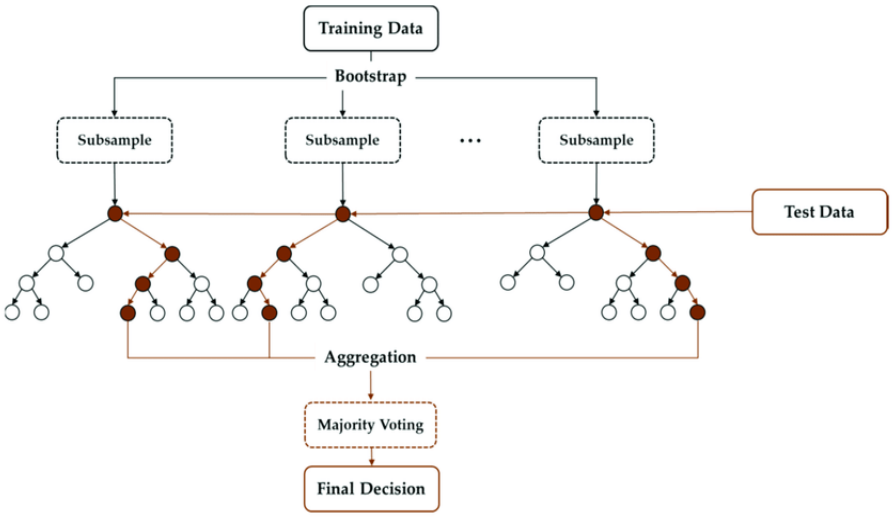


Figure 4: The structure of a random forest with bootstrap aggregating. Starting from the top with all the training data, bootstrap samples are created to then train independent decision trees. Each tree makes a decision and most *popular* decision is taken as the random forest's final decision through a process called Aggregation. Figure found in [9].

Building on earlier sections, our attention shifts to predicting Frisen grades in fundus photos from patients with IIH. Central to our approach is the utilization of the Random Forest model, chosen for its prowess in navigating complex relationships within high-dimensional data while addressing concerns of overfitting.

Transitioning smoothly, let's reexamine the pivotal connection between computing the sublevel set filtration of a fundus photo and generating the corresponding persistence diagram. This step, elucidated in previous sections, is essential for extracting topological features that provide valuable insights into disease severity. Integrating this topological perspective with featurization methods—such as persistence landscapes or persistence images—seamlessly transforms our persistence diagrams into vectorized representations. These representations serve as input vectors for our Random Forest model, facilitating the translation of topological insights into machine-learning-compatible features.

Furthermore, the proficiency of Random Forests in handling large datasets with intricate relationships aligns seamlessly with the challenges posed by our fundus photos analysis. An integral component in this process is the feature importances method within Random Forests, playing a pivotal role in identifying the subset of the most crucial features for our predictions. Here, featurization methods yield a comprehensive set of features, and it's through the feature importances method that we discern the most influential subset of original features

when predicting the Frisen grade of a fundus photo. These selected features correspond to entries in our vector, representing specific pixel intensity ranges encapsulating persistent and significant topological features captured through homology.

6 Results

Now, we get to the section we've all been waiting for—the results! Our current approach employs Persistence Images as the featurization method for vectorizing our diagrams. We chose images over landscapes because the former proved better in the realm of interpretability, which we'll discuss shortly.

First, let's remind ourselves about the pipeline. To create input vectors for a Random Forest model, we've built the following process so far: We start with a (gray scale) fundus photo, compute its corresponding sublevel set filtration on pixel intensity, generate the persistence diagram corresponding to this filtration, and lastly, compute the corresponding persistence image. We obtain the desired "vectorized" diagram, which will serve as the input vector for our classification model [1].

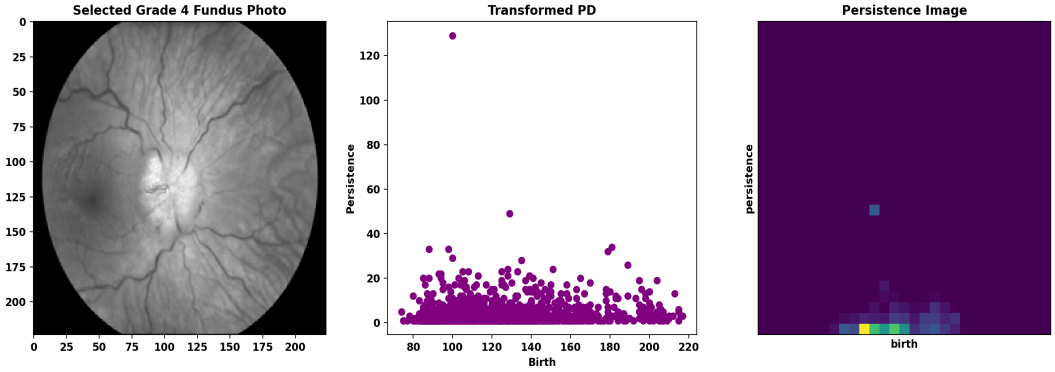


Figure 5: Here is a visualization of our pipeline, where we we demonstrate going from a gray scale photo to a persistence image, by computing the corresponding persistence diagram in (birth, persistence) coordinates.

Now that we have an input vector for each photo, we can concatenate them all and store them in an array called X. The concatenation process can be visualized as stacking each input vector on top of the other to create X. Consequently, each row in the array corresponds to a specific input vector obtained from a persistence image, which, in turn, relates to a particular fundus photo.

Subsequently, another array (y), named the labels array, is created. This array contains the Frisen grades corresponding to each row/image in X. X, along

[1] Each persistence image vector is first *flattened* and then fed into our random forest model.

with y, is fed into our Random Forest model to learn about the classification of fundus photos.

It's crucial to note that we combined grade 4 and grade 5 images before generating the persistence images. This step was taken due to the significant data imbalance highlighted in Section 2. Classification models often struggle when one class has significantly fewer samples than the others. Fortunately, we can create an additional model specifically to distinguish grade 4 from grade 5 images, and combining this model with the first one, will enables us to distinguish fundus photos across all grades effectively.

```
rf = RandomForestClassifier(n_estimators=200, max_depth= None, max_features='sqrt',
                            min_samples_leaf=1, random_state=4400)
```

Figure 6: Specific hyperparameters used for our random forest model. What each of these does or how they affect the model is outside the scope of this paper. But if you want a taste, the first says that our model uses 200 decision trees, the second says each tree can grow as much as it wants to, and so on.

Our current random forest model attains a 62% accuracy, where accuracy, expressed as a percentage, evaluates the model's ability to correctly predict sample labels in the dataset. It represents the fraction of correctly classified labels over the total number of samples, offering a comprehensive measure of correctness across all classes. In our case, a 62% accuracy means our model correctly classifies the Frisen grade in 62 out of 100 unlabeled images.

It is also possible to provide a measure of accuracy for individual severity grades using other metrics such as *precision, recall, and f1-score* but that would we part of future work which will be discussed in the last section of this paper called "Moving Forward."

The achieved accuracy appears reasonable given our choice of a specific filtration to capture topological information. However, it's important to note that this chosen filtration may not necessarily maximize accuracy. In contrast, more sophisticated models, such as *Convolutional Neural Networks (CNNs)*, can achieve accuracy values in the high 90s. Yet, the challenge with these models lies in understanding what exactly they are learning and how they leverage this information for accurate image classification.

Therefore, this project was motivated by the idea of constructing an interpretable model. We aimed to develop a model that allowed us to understand precisely what it is learning and identify the specific regions in the input images that it focuses on. This aspect of interpretability will be explored further in the next section.

6.1 Most Important Features

At the end of Section 5, we emphasized the significance of the feature importance method in Random Forests to identify the subset of crucial features for our predictions. Initially, each flattened persistence image vector had $29 \times 29 = 841$

components, each corresponding to a particular pixel in the persistence image. As we will discuss shortly, each of these pixels corresponds to a particular point in the corresponding persistence diagram, which, in turn, yields a (birth, death) coordinate revealing a particular range of pixel intensities.

Using the feature importance method, we assessed the importance of each of the 841 features as a fraction in the model's final prediction and picked the top 19 features since they accounted for 20% of the overall importance. These important features allowed us to identify the top 19 most *predictive* pixel intensity ranges across all of our images.

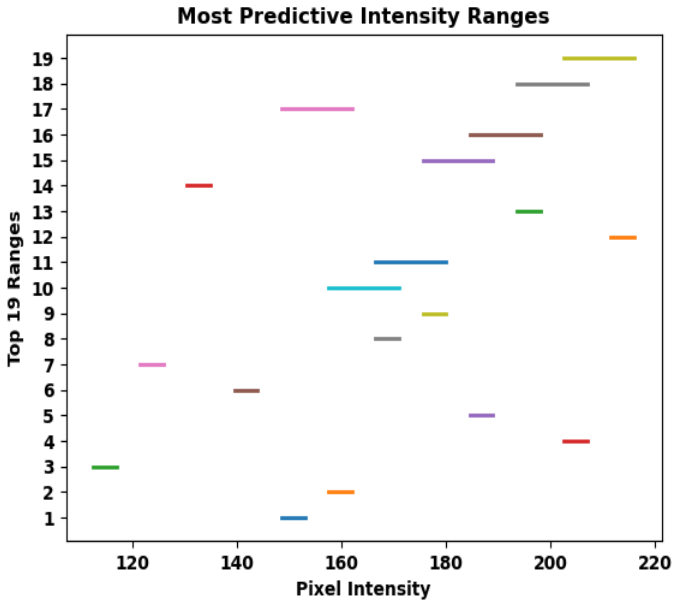


Figure 7: The 19 most predictive intensity ranges (colored lines) across all images, where blue line at the bottom represents most predictive range.

Now armed with the most *predictive* intensity ranges, we can display *masked* fundus photos, showcasing only the pixels in the original photos that fall within the union of all 19 most predictive ranges. This process effectively emphasizes regions where the model learned the most, and these regions contribute the most to the prediction of Frisen grades using topological methods.

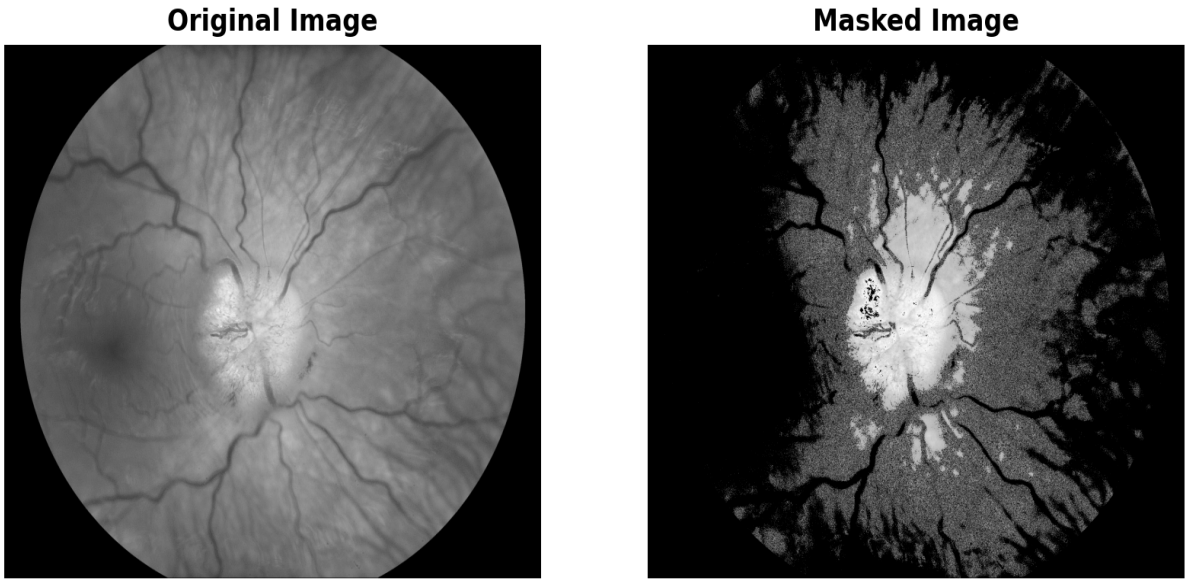


Figure 8: Masked grade 4 fundus photo, displaying only pixels falling within the union of the 19 most predictive ranges.

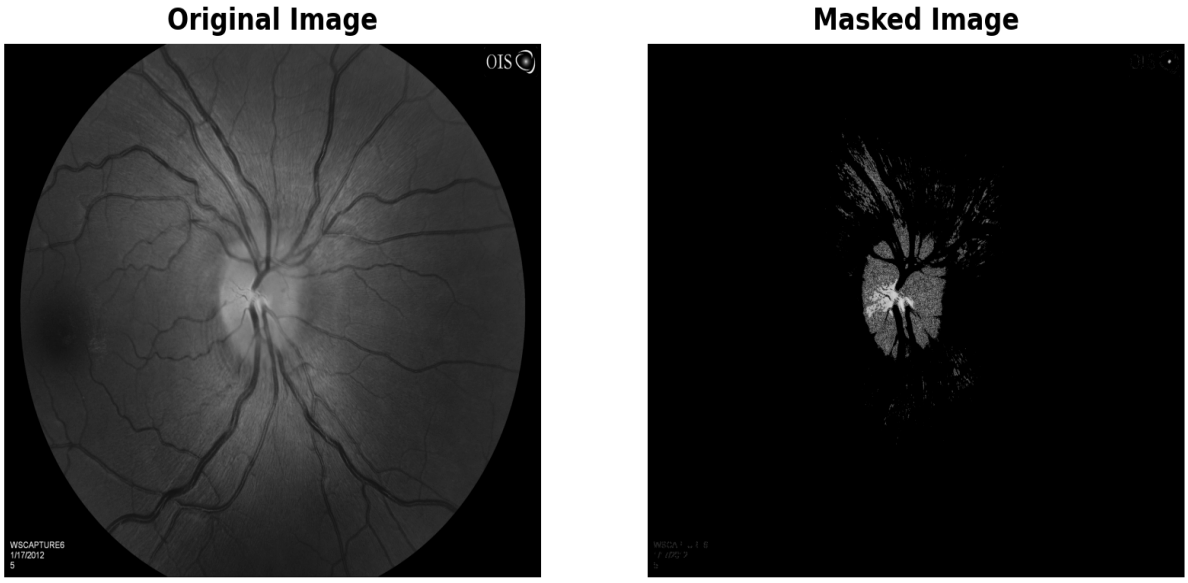


Figure 9: Masked grade 2 fundus photo, displaying only pixels falling within the union of the 19 most predictive ranges.

From the two figures above, we hypothesize that the model identifies more predictive regions when learning from high-grade images, contrasting with fewer identified regions when learning from low-grade images. If this holds true for all images, it aligns with our understanding from Figure 1, where the most severe stages of the disease affect more regions of the eye, including the obscuration of vessels going in and coming out of the optic nerve head. This is in contrast to changes primarily in the optic nerve head, as explained for the least severe grades of the disease.

We now conclude this rich exploration of fundus photography through topological methods, wherein we've built a moderately accurate classification model for predicting IIH severity from a given fundus photo. Due to the interpretable nature of our chosen model, we are able to identify the most predictive regions in each image by finding the pixel intensity ranges ranked as having the highest importance to the model's predictions. The next section will provide the reader with initial thoughts on potential next steps, such as transitioning from our current *masking* of individual images to potentially *masking* groups of images, such as grouping by Frisen grade.

7 Moving Forward

- Investigate methods for masking groups of photos instead of individual images.
- Work with colored images by splitting them into three channels. Apply the analysis procedure independently to each channel to leverage color information.
- Consider different types of filtration.
- Explore multi-parameter persistence for a comprehensive topological analysis. Consider combining pixel intensity with additional parameters for enhanced insights.

Glossary

Fundus photography Taking an image of the retina of the eye with a fundus camera – specialized low power microscope with an attached camera designed to photograph the interior surface of the eye, including the retina, retinal vasculature, optic disc (optic nerve head), macula, and posterior pole (i.e. the fundus). 1

Persistent homology Persistent homology is a method for computing topological features of a space at different *Spatial resolution*. More persistent features are detected over a wide range of spatial scales and are deemed more likely to represent true features of the underlying space rather than artifacts of sampling, noise, or particular choice of parameters. 4

Spatial resolution Spatial resolution is the ability to distinguish between two neighboring structures as separated. 21

Sublevel sets A set of the form $L_c(f) = \{(x_1, \cdots, x_n) : f(x_1, \cdots, x_n) \leq c\}$ is called a sublevel set of f. 5

8 Materials

1. Wall M, Kupersmith MJ, Kieburtz KD, Corbett JJ, Feldon SE, Friedman DI, Katz DM, Keltner JL, Schron EB, McDermott MP; NORDIC Idiopathic Intracranial Hypertension Study Group. *"The idiopathic intracranial hypertension treatment trial: clinical profile at baseline."* JAMA Neurol, 71 (2014), 6, 693-701.

2. Barnes, D., Polanco, L., and Perea, J. A. *"A Comparative Study of Machine Learning Methods for Persistence Diagrams."* Frontiers in Artificial Intelligence-Machine Learning and Artificial Intelligence, 2-7 (2021)

3. Jensen RH, Radojicic A, Yri H. *"The diagnosis and management of idiopathic intracranial hypertension and the associated headache."* Ther Adv Neurol Disord, 2016; 9(4): 317-326

4. *"Moran CORE | Case Report of Idiopathic Intracranial Hypertension and Frisen Scale Papilledema Grading."* Moran CORE, https://morancore.utah.edu/section-05-neuro-ophthalmology/idiopathic-intracranial-hypertension-iih-and-papilledema-grading/ (Accessed 10/31/2023).

5. Bubenik, P. *"The Persistence Landscape and Some of its Properties."* Topological Data Anal, 15 (2020), 3-6.

6. Yuriy Mileyko, Sayan Mukherjee, and John Harer. *"Probability measures on the space of persistence diagrams."* Inverse Problems, 27 (2011), 12, 124007. DOI: 10.1088/0266-5611/27/12/124007

7. Jose A. Perea, Elizabeth Munch, and Firas A. Khasawneh. *"Approximating Continuous Functions on Persistence Diagrams Using Template Functions."* Foundations of Computational Mathematics, 23 (2022), 4, 6-10.

8. Adams, H., Chepushtanova, S., Emerson, T., Hanson, E., Kirby, M., Motta, F., et al. *Persistence images: A stable vector representation of persistent homology.* arXiv.org, July 11, 2016. `https://arxiv.org/abs/1507.06217`. §4 and 5

9. Dong-Hwa, J., Se-Eun, K., Woo-Hyeok, C., & Seong-Ho, A. *A Comparative Study on the Influence of Undersampling and Oversampling Techniques for the Classification of Physical Activities Using an Imbalanced Accelerometer Dataset. ResearchGate | Find and Share Research.* §2.3 p. 9

A Appendix A

Definition of "lower_star_img()" Python function to compute the sublevel set filtration of a given image. It uses several Python libraries and in particular, it leverages the Ripser library to compute persistence diagrams.

```
def lower_star_img(img):
    """
    Construct a lower star filtration on an image
    Parameters
    ----------
    img: ndarray (M, N)
        An array of single channel image data
    Returns
    -------
    I: ndarray (K, 2)
        A 0-dimensional persistence diagram corresponding to the sublevelset filtration
    """
    m, n = img.shape
    idxs = np.arange(m * n).reshape((m, n))
    I = idxs.flatten()
    J = idxs.flatten()
    V = img.flatten()

    # Connect 8 spatial neighbors
    tidxs = np.ones((m + 2, n + 2), dtype=np.int64) * np.nan
    tidxs[1:-1, 1:-1] = idxs
    tD = np.ones_like(tidxs) * np.nan
    tD[1:-1, 1:-1] = img

    for di in [-1, 0, 1]:
        for dj in [-1, 0, 1]:
            if di == 0 and dj == 0:
                continue
            thisJ = np.roll(np.roll(tidxs, di, axis=0), dj, axis=1)
            thisD = np.roll(np.roll(tD, di, axis=0), dj, axis=1)
            thisD = np.maximum(thisD, tD)

            # Deal with boundaries
            boundary = ~np.isnan(thisD)
            thisI = tidxs[boundary]
            thisJ = thisJ[boundary]
            thisD = thisD[boundary]
            I = np.concatenate((I, thisI.flatten()))
            J = np.concatenate((J, thisJ.flatten()))
            V = np.concatenate((V, thisD.flatten()))

    sparseDM = sparse.coo_matrix((V, (I, J)), shape=(idxs.size, idxs.size))
    return ripser(sparseDM, distance_matrix=True, maxdim=1)["dgms"]
```

Definition of "pers_pt_centroid_and_bottom_left_coords()" function to compute coordinates of bottom left point and centroid point of area in pers. dgm. corresponding to a pixel of interest in pers. img.

```
def pers_pt_centroid_and_bottom_left_coords(pixel_idx, birth_range=(-3.5, 257.5),
                                            pers_range=(-2.5, 258.5), pers_img_shape=(29, 29)):
    """
    Computes coordinates of bottom left point and centroid point of rectangular area
    generated (in persistence diagram) by given pixel (in persistence image)
    Parameters
    ----------
    pixel_idx: tuple of (row, col) index of pixel of interest
    birth_range: tuple of births range such that all pers. pairs (in dgm) are minimally
                 enclosed across pers. pairs from one or more pers. dgms.
    pers_range: tuple of persistences range such that all pers. pairs (in dgm) are minimally
                enclosed across pers. pairs from one or more pers. dgms.
    pers_img_shape: tuple of pers. img's dimensions

    """
    # Compute scale factor to go images' axes to persistence diagrams' axes
    delta_b = (birth_range[1] - birth_range[0]) / pers_img_shape[1]
    delta_p = (pers_range[1] - pers_range[0]) / pers_img_shape[0]

    # Compute coords. of bottom left point of rectangle generated (in persistence diagram)
    # by the given pixel (in persistence image)
    bottom_left_coords = (pixel_idx[1] * delta_b, (pers_img_shape[0] - pixel_idx[0]) * delta_p)

    # Compute coords. of centroid of rectangle generated
    centroid_coords = ((pixel_idx[1] + 0.5) * delta_b, (pers_img_shape[0] - pixel_idx[0] + 0.5) * delta_p)

    return bottom_left_coords, centroid_coords
```

Definition of "display_original_vs_masked2()" function to display a given fundus photo, and a masked version where only pixels falling within given intensity ranges, are shown.

```
def display_original_vs_masked2(img, inten_ranges, output_filename='default_name', save=False):
    """
    Display original and masked image side-by-side. Masks image based on a given list of intensity ranges
    Parameters
    ----------
    img: image of interest (as an array)
    inten_ranges: list of predictive intensity ranges
    output_filename: file name to use when saving figure
    save: boolean to determine whether figure will be saved
    """

    # Initialize an empty mask with the same shape as the image
    mask = np.zeros_like(img, dtype=bool)

    # Iterate through the list of intensity ranges and apply masks with logical OR
    for lower, upper in inten_ranges:
        current_mask = np.logical_and(img >= lower, img <= upper)
        mask = np.logical_or(mask, current_mask)

    # Apply the final mask to the original image
    result_image = img * mask

    # Display the original and masked images side by side
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(result_image, cmap='gray')
    plt.title('Masked Image')
    plt.axis('off')

    # Saving figure
    if save:
        plt.savefig(output_filename, dpi=200, bbox_inches='tight')
    plt.show()
```