Loading and Visualizing structural connectome Openneuro/FPII

In [3]:
```python
#Loading FairparkII matrices
import os
import numpy as np

def load_pd_matrices(directory, subjects):
    """
    Load the matrices from "parcels_coreg.csv" for specific subjects in the specified directory.

    Args:
    directory (str): The directory where your datasets are located.
    subjects (list): The list of subjects to load matrices for.
    """
    # Create an empty dictionary to store the matrices
    matrices = {}

    # Iterate over the specified subjects
    for subject_id in subjects:
        # Construct the path to the subject's directory
        subject_directory = os.path.join(directory, subject_id)

        # Check if the directory exists for the current subject
        if os.path.isdir(subject_directory):
            # Create a dictionary to hold the matrices for each session
            session_matrices = {}

            # Define the session directories
            sessions = ['W00', 'W36']

            # Iterate over the session directories within the subject's directory
            for session in sessions:
                # Construct the path to the session's directory
                session_directory = os.path.join(subject_directory, session)

                # Construct the path to the .csv file
                csv_file = os.path.join(session_directory, 'parcels_coreg.csv')

                # Check if the .csv file exists
                if os.path.isfile(csv_file):
                    # Load the matrix from the .csv file
                    matrix = np.loadtxt(open(csv_file, "rb"), delimiter=",", skiprows=0)

                    # Add the matrix to the dictionary using the session as the key
                    session_matrices[session] = matrix

            # Add the session matrices to the main dictionary using the subject ID as the key
            matrices[subject_id] = session_matrices

    return matrices
```

In [4]:
```python
# Loading the Pd's matrices
pd_list = ["101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
pd_matrices = load_pd_matrices("/media/imane/DATA/FPII/Fairparkpreprocessed/Lille/", pd_list)
pd_matrices['101018JW']
```

Out[4]:
```
{'W00': array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 7.35907096e-04],
        ...,
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 6.12915969e+00, 3.11312096e-03],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         6.12915969e+00, 0.00000000e+00, 1.39777430e-04],
        [0.00000000e+00, 0.00000000e+00, 7.35907096e-04, ...,
         3.11312096e-03, 1.39777430e-04, 0.00000000e+00]]),
 'W36': array([[0.00000000e+00, 0.00000000e+00, 1.28486910e-04, ...,
         5.86121818e-05, 0.00000000e+00, 2.14683935e-05],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [1.28486910e-04, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 8.63461999e-04],
        ...,
        [5.86121818e-05, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 6.31223792e+00, 2.60735966e-03],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         6.31223792e+00, 0.00000000e+00, 1.65842975e-04],
        [2.14683935e-05, 0.00000000e+00, 8.63461999e-04, ...,
         2.60735966e-03, 1.65842975e-04, 0.00000000e+00]])}
```

In [5]:

```python
import os
import numpy as np

#loading openneuro matrices
def load_control_matrices(bids_directory, subjects, sessions):
    """
    Load the subject matrices from "parcels_coreg.csv" in the 'dwi' folder of each specified subject's session i

    Args:
    bids_directory (str): The directory where your BIDS datasets are located.
    subjects (list): The list of subjects.
    sessions (list): The list of sessions.
    """
    matrices = {}

    # Iterate over the specified subjects and sessions
    for subject in subjects:
        for session in sessions:
            subject_path = os.path.join(bids_directory, "sub-" + subject, "ses-" + session, "dwi")

            # Check if the 'dwi' directory exists in the current subject directory
            if os.path.isdir(subject_path):

                # File name of the subject matrix
                file_name = "parcels_coreg.csv"

                file_path = os.path.join(subject_path, file_name)

                # Check if the 'parcels_coreg.csv' file exists
                if os.path.exists(file_path):
                    # Load the matrix from the .csv file
                    matrix = np.loadtxt(open(file_path, "rb"), delimiter=",", skiprows=0)

                    # Add the matrix to the dictionary using the subject and session as the key
                    matrices[(subject, session)] = matrix
                else:
                    print(f"File '{file_name}' doesn't exist in the directory {subject_path}")
            else:
                print(f"'dwi' directory doesn't exist in the directory {subject_path}")

    return matrices
```

In [ ]:

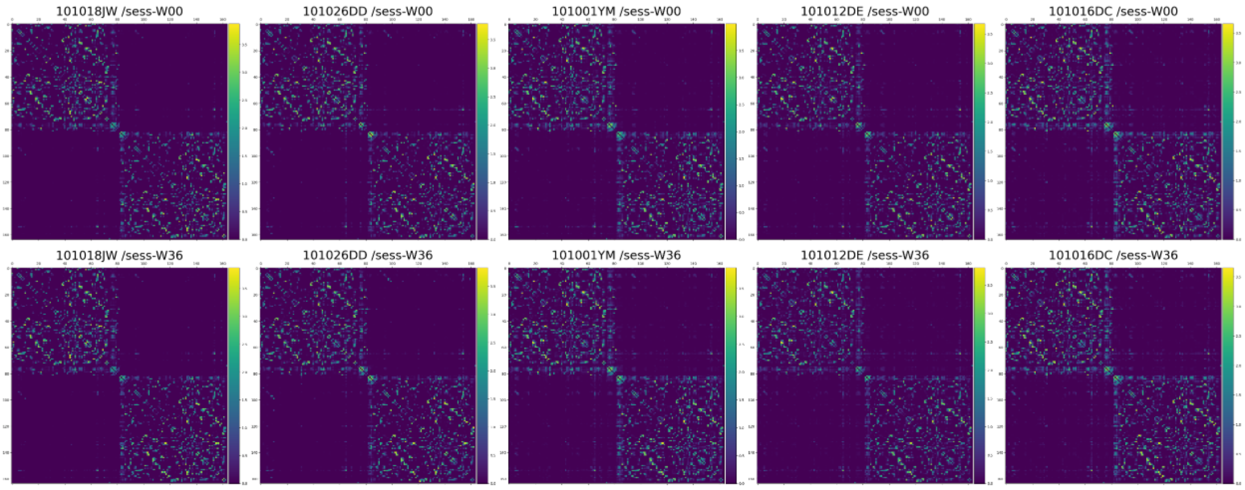
```python
#Loading the control's matrices
control_list = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
sessions = ["1"]
control_matrices = load_control_matrices('/media/imane/DATA/openneuro/ds001907/', control_list, sessions)
control_matrices[('RC4107','1')] #matrix of subject RC4107 session 1,
```

In [7]:

```python
#Visualizing PD matrices
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.axes_grid1 import make_axes_locatable

def visualize_pd_matrices(matrices, subjects_to_plot, sessions_to_plot):
    # Define the colormap range
    vmin = None
    vmax = None

    # Define the number of columns and rows for the subplots
    n_cols = len(subjects_to_plot)
    n_rows = len(sessions_to_plot)

    # Create a new figure with specified size
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(10 * n_cols, 10 * n_rows))  # Adjust size as needed

    for i, session in enumerate(sessions_to_plot):
        for j, subject_id in enumerate(subjects_to_plot):
            # Retrieve the matrix if it exists
            if subject_id in matrices and session in matrices[subject_id]:
                matrix = matrices[subject_id][session]

                # Plot the matrix on the subplot
                ax = axs[i, j]
                im = ax.matshow(np.log1p(matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
                ax.set_title(f'{subject_id} /sess-{session}', fontsize=35)

                # Create a new axes with a smaller height and add the colorbar to this axes
                divider = make_axes_locatable(ax)
                cax = divider.append_axes("right", size="5%", pad=0.05)
                plt.colorbar(im, cax=cax)

    # Show the figure
    plt.tight_layout()
    plt.show()
```

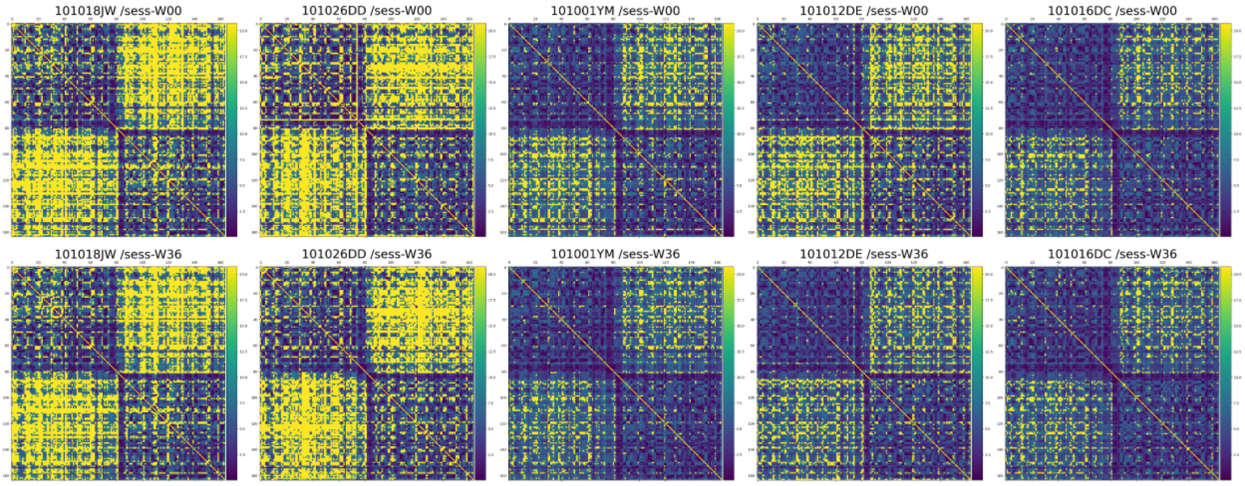
In [8]:

```python
#plotting pd's matrices
sessions_to_plot = ["W00", "W36"]
subjects_to_plot = ["101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
visualize_pd_matrices(pd_matrices, pd_list, sessions_to_plot)
```



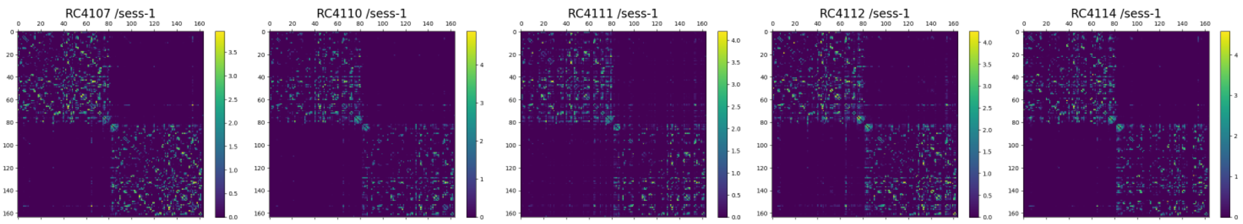
In [9]:

```
def visualize_and_collect_pd_inverse_matrices_separately(matrices, subjects_to_plot, sessions_to_plot):
    # Define the colormap range
    vmin = None
    vmax = None

    # Define the number of columns and rows for the subplots
    n_cols = len(subjects_to_plot)
    n_rows = len(sessions_to_plot)

    # Create a new figure with specified size
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(10 * n_cols, 10 * n_rows))  # Adjust size as needed

    # Dictionaries to store the inverse matrices of each session
    spd_matrices_W00 = []
    spd_matrices_W36 = []

    for i, session in enumerate(sessions_to_plot):
        for j, subject_id in enumerate(subjects_to_plot):
            # Retrieve the matrix if it exists
            if subject_id in matrices and session in matrices[subject_id]:
                matrix = matrices[subject_id][session]

                # Calculate the inverse of the matrix
                inverse_matrix = 1 / (matrix + 1e-9)

                # Add the inverse matrix to the corresponding list
                if session == "W00":
                    spd_matrices_W00.append(inverse_matrix)
                elif session == "W36":
                    spd_matrices_W36.append(inverse_matrix)

                # Plot the matrix on the subplot
                ax = axs[i, j]
                im = ax.matshow(np.log1p(inverse_matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
                ax.set_title(f'{subject_id} /sess-{session}', fontsize=35)

                # Create a new axes with a smaller height and add the colorbar to this axes
                divider = make_axes_locatable(ax)
                cax = divider.append_axes("right", size="5%", pad=0.05)
                plt.colorbar(im, cax=cax)

    # Show the figure
    plt.tight_layout()
    plt.show()

    return spd_matrices_W00, spd_matrices_W36

subjects_to_plot = [ "101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
sessions_to_plot = ["W00", "W36"]
pd_inversematrices_W00, pd_inversematrices_W36 = visualize_and_collect_pd_inverse_matrices_separately(pd_matrice
```

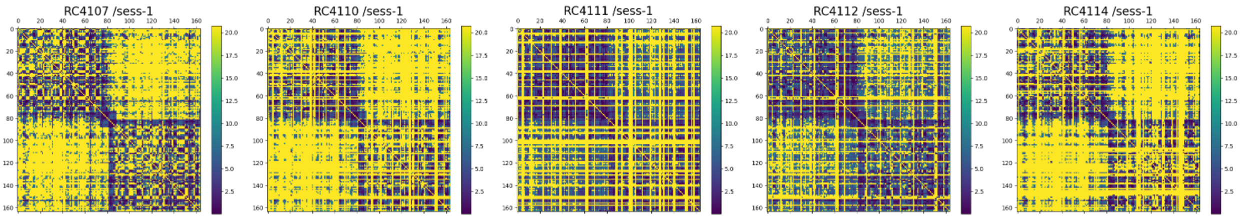


In [10]:

```
#Visualizing control matrices
import matplotlib.pyplot as plt
import numpy as np

def visualize_control_matrices(matrices, subjects_to_plot):
    # Define the number of columns and rows for the subplots
    n_cols = 5
    n_rows = 1

    # Initialize a figure and axes object
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(30, 5))
    axs = axs.flatten()  # Flatten to easily iterate over

    # Define the colormap range
    vmin = None
    vmax = None

    # Iterate over the matrices
    i = 0
    for (subject_id, session), matrix in matrices.items():
        # Skip subjects not in the subjects_to_plot list
        if subject_id not in subjects_to_plot:
            continue

        # Plot the matrix on the subplot
        cax = axs[i].matshow(np.log1p(matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
        axs[i].set_title(f'{subject_id} /sess-{session}', fontsize=20)

        # Add a colorbar to the subplot
        fig.colorbar(cax, ax=axs[i])

        # Increment the subplot index
        i += 1

    # If there are more axes than matrices, remove the unused axes
    if i < len(axs):
        for j in range(i, len(axs)):
            fig.delaxes(axs[j])

    # Show the figure
    plt.tight_layout()
    plt.show()

#plotting control's matrices

subjects_to_plot = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
visualize_control_matrices(control_matrices, subjects_to_plot)
```



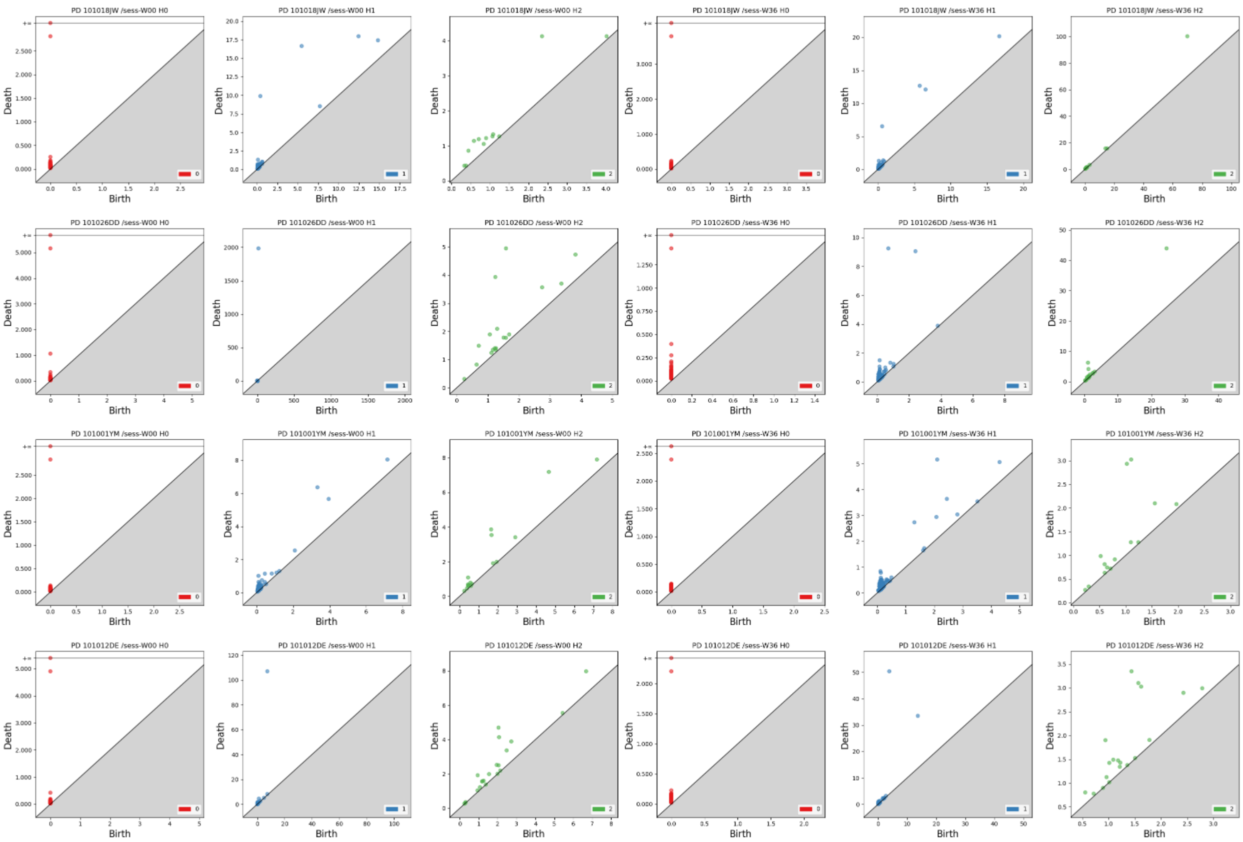
In [11]:
```python
#Visualizing control element-wise inverse matrices
import matplotlib.pyplot as plt
import numpy as np

def visualize_and_collect_control_inverse_matrices(matrices, subjects_to_plot):
    # Define the number of columns and rows for the subplots
    n_cols = 5
    n_rows = 1

    # Initialize a figure and axes object
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(30, 5))
    axs = axs.flatten()  # Flatten to easily iterate over

    # Define the colormap range
    vmin = None
    vmax = None

    # List to store the inverse matrices
    icc_matrices = []

    # Iterate over the matrices
    i = 0
    for (subject_id, session), matrix in matrices.items():
        # Skip subjects not in the subjects_to_plot list
        if subject_id not in subjects_to_plot:
            continue

        # Calculate the inverse of the matrix
        inverse_matrix = 1 / (matrix + 1e-9)

        # Plot the matrix on the subplot
        cax = axs[i].matshow(np.log1p(inverse_matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
        axs[i].set_title(f'{subject_id} /sess-{session}', fontsize=20)

        # Add a colorbar to the subplot
        fig.colorbar(cax, ax=axs[i])

        # Add the inverse matrix to the list
        icc_matrices.append(inverse_matrix)

        # Increment the subplot index
        i += 1

    # If there are more axes than matrices, remove the unused axes
    if i < len(axs):
        for j in range(i, len(axs)):
            fig.delaxes(axs[j])

    # Show the figure
    plt.tight_layout()
    plt.show()

    return icc_matrices

subjects_to_plot = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
icc_matrices = visualize_and_collect_control_inverse_matrices(control_matrices, subjects_to_plot)
```

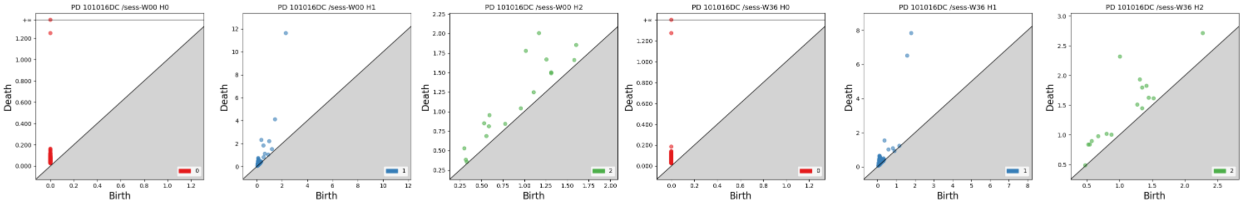


Topological data analysis : persistent homology

In [74]:

```
import matplotlib.pyplot as plt
import gudhi
import numpy as np
## compute separate lists of pd diagrams for each homology degree gudhi:
def compute_and_plot_pd_persistence_diagrams_gudhi(matrices_W00, matrices_W36, subjects_to_plot):
    H0_pd_diagrams_W00 = []
    H1_pd_diagrams_W00 = []
    H2_pd_diagrams_W00 = []
    H0_pd_diagrams_W36 = []
    H1_pd_diagrams_W36 = []
    H2_pd_diagrams_W36 = []

    for i, (matrix_W00, matrix_W36) in enumerate(zip(matrices_W00, matrices_W36)):
        subject_id = subjects_to_plot[i]

        # Create a figure with subplots for each homology dimension (H0, H1, H2) and session (W00, W36)
        fig, axs = plt.subplots(nrows=1, ncols=6, figsize=(30, 5))

        # Compute and plot the persistence diagrams for session W00
        compute_and_plot_session(matrix_W00, subject_id, "W00", axs[:3], H0_pd_diagrams_W00, H1_pd_diagrams_W00

        # Compute and plot the persistence diagrams for session W36
        compute_and_plot_session(matrix_W36, subject_id, "W36", axs[3:], H0_pd_diagrams_W36, H1_pd_diagrams_W36

        plt.tight_layout()
        plt.show()

    return (H0_pd_diagrams_W00, H1_pd_diagrams_W00, H2_pd_diagrams_W00), (H0_pd_diagrams_W36, H1_pd_diagrams_W3(

def compute_and_plot_session(matrix, subject_id, session, axs, H0_pd_diagrams, H1_pd_diagrams, H2_pd_diagrams):
    rips_complex = gudhi.RipsComplex(distance_matrix=matrix, max_edge_length=np.inf)
    simplex_tree = rips_complex.create_simplex_tree(max_dimension=3)
    diagrams = simplex_tree.persistence()

    for dim in range(3):
        diagram_dim = [point for point in diagrams if point[0] == dim]
        gudhi.plot_persistence_diagram(diagram_dim, axes=axs[dim])
        axs[dim].set_title(f'PD {subject_id} /sess-{session} H{dim}')

        # Store diagrams in respective lists
        if dim == 0:
            H0_pd_diagrams.append(diagram_dim)
        elif dim == 1:
            H1_pd_diagrams.append(diagram_dim)
        else:  # dim == 2
            H2_pd_diagrams.append(diagram_dim)

subjects_to_plot = ["101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
(H0_pd_diagrams_W00, H1_pd_diagrams_W00, H2_pd_diagrams_W00), (H0_pd_diagrams_W36, H1_pd_diagrams_W36, H2_pd_dia
```





In [ ]:

```
import gudhi as gd
## compute separate lists of control diagrams for each homology degree with gudhi:
def plot_and_store_control_persistence_diagrams(distance_matrices, subjects):

    H0_cc_diagrams = []
    H1_cc_diagrams = []
    H2_cc_diagrams = []


    fig, axs = plt.subplots(len(subjects), 3, figsize=(15, 4 * len(subjects)))

    for i, (distance_matrix, subject) in enumerate(zip(distance_matrices, subjects)):
        # Compute the Rips complex
        rips_complex = gd.RipsComplex(distance_matrix=distance_matrix)

        # Compute the simplex tree
        simplex_tree = rips_complex.create_simplex_tree(max_dimension=3)

        # Compute the persistence diagram
        diag = simplex_tree.persistence()

        # Separate the diagrams by dimension
        diag_H0 = [p for p in diag if p[0] == 0]
        diag_H1 = [p for p in diag if p[0] == 1]
        diag_H2 = [p for p in diag if p[0] == 2]

        # Add diagrams to respective lists
        H0_cc_diagrams.append(diag_H0)
        H1_cc_diagrams.append(diag_H1)
        H2_cc_diagrams.append(diag_H2)

        # Plot H0 diagram
        gd.plot_persistence_diagram(diag_H0, axes=axs[i, 0])
        axs[i, 0].set_title(f'H0 - {subject} - sess-1')

        # Plot H1 diagram
        gd.plot_persistence_diagram(diag_H1, axes=axs[i, 1])
        axs[i, 1].set_title(f'H1 - {subject} - sess-1')

        # Plot H2 diagram
        gd.plot_persistence_diagram(diag_H2, axes=axs[i, 2])
        axs[i, 2].set_title(f'H2 - {subject} - sess-1')

    #Adjust the layout
    plt.tight_layout()
    plt.show()

    return H0_cc_diagrams, H1_cc_diagrams, H2_cc_diagrams

subjects = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
H0_cc_diagrams, H1_cc_diagrams, H2_cc_diagrams = plot_and_store_control_persistence_diagrams(icc_matrices, subj
```

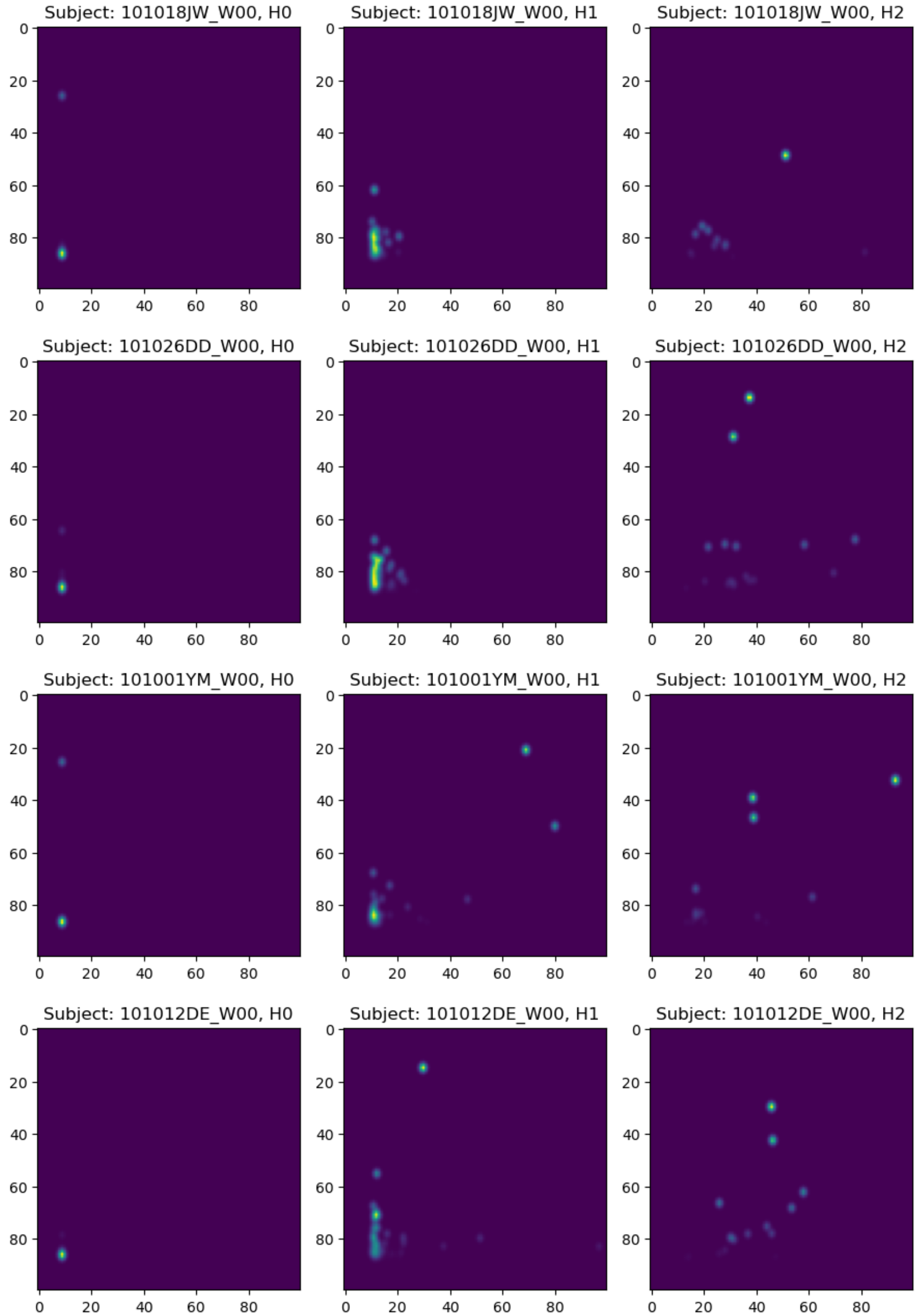
In [ ]:

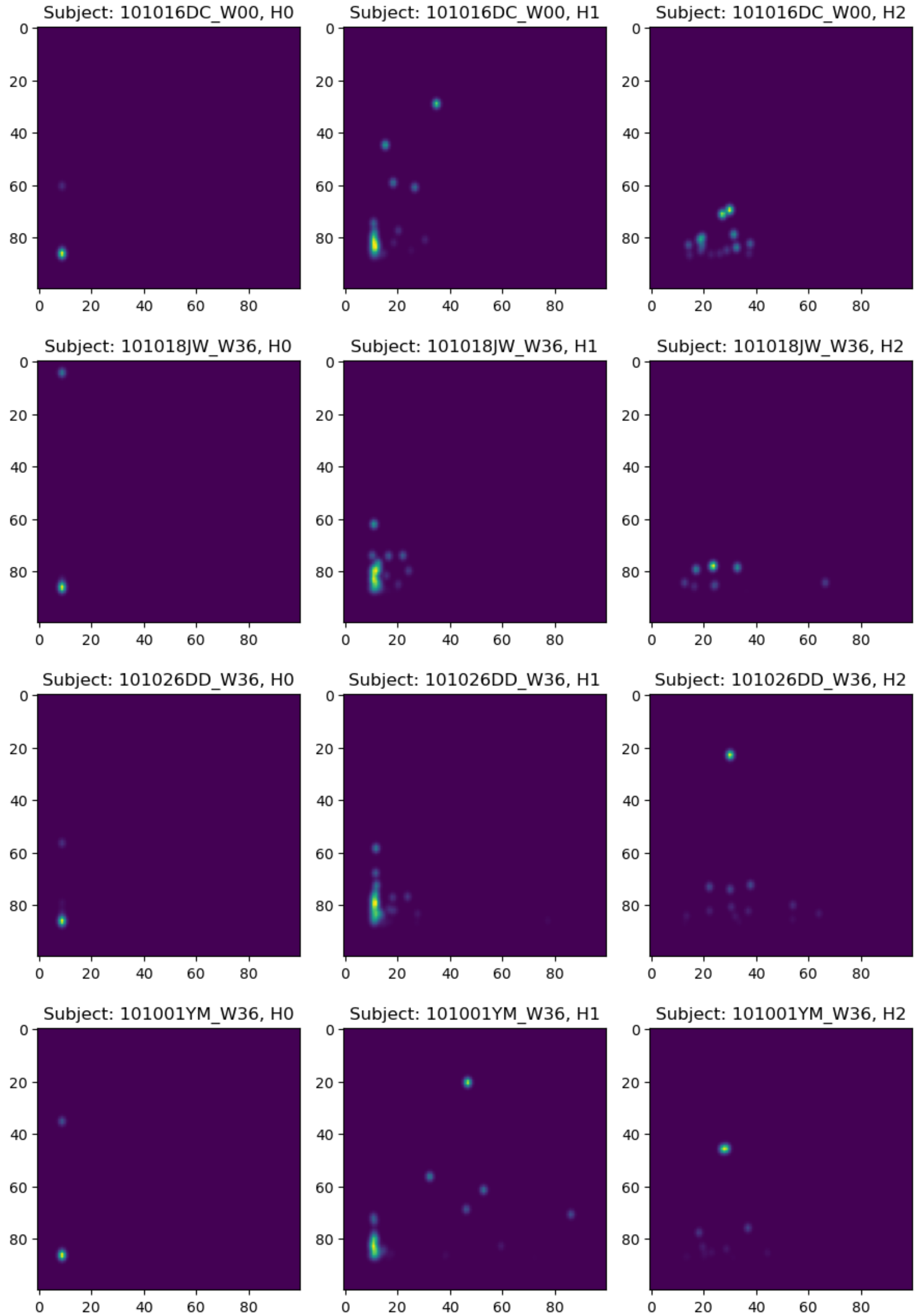
```
#example of a persistence diagram data
H1_cc_diagrams[0]
```

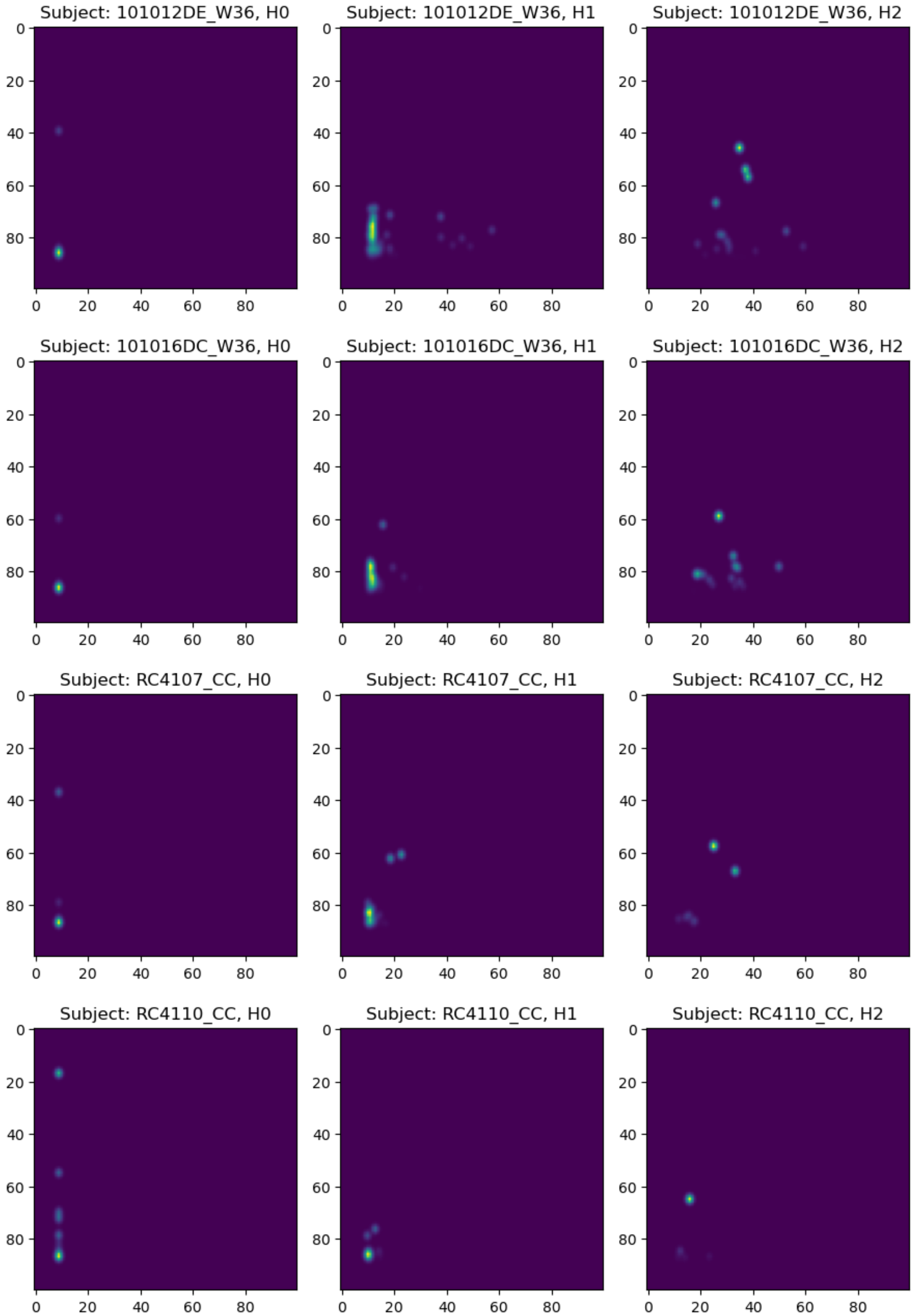
Conversion of Persistence Diagrams into Persistence Images

In [133…

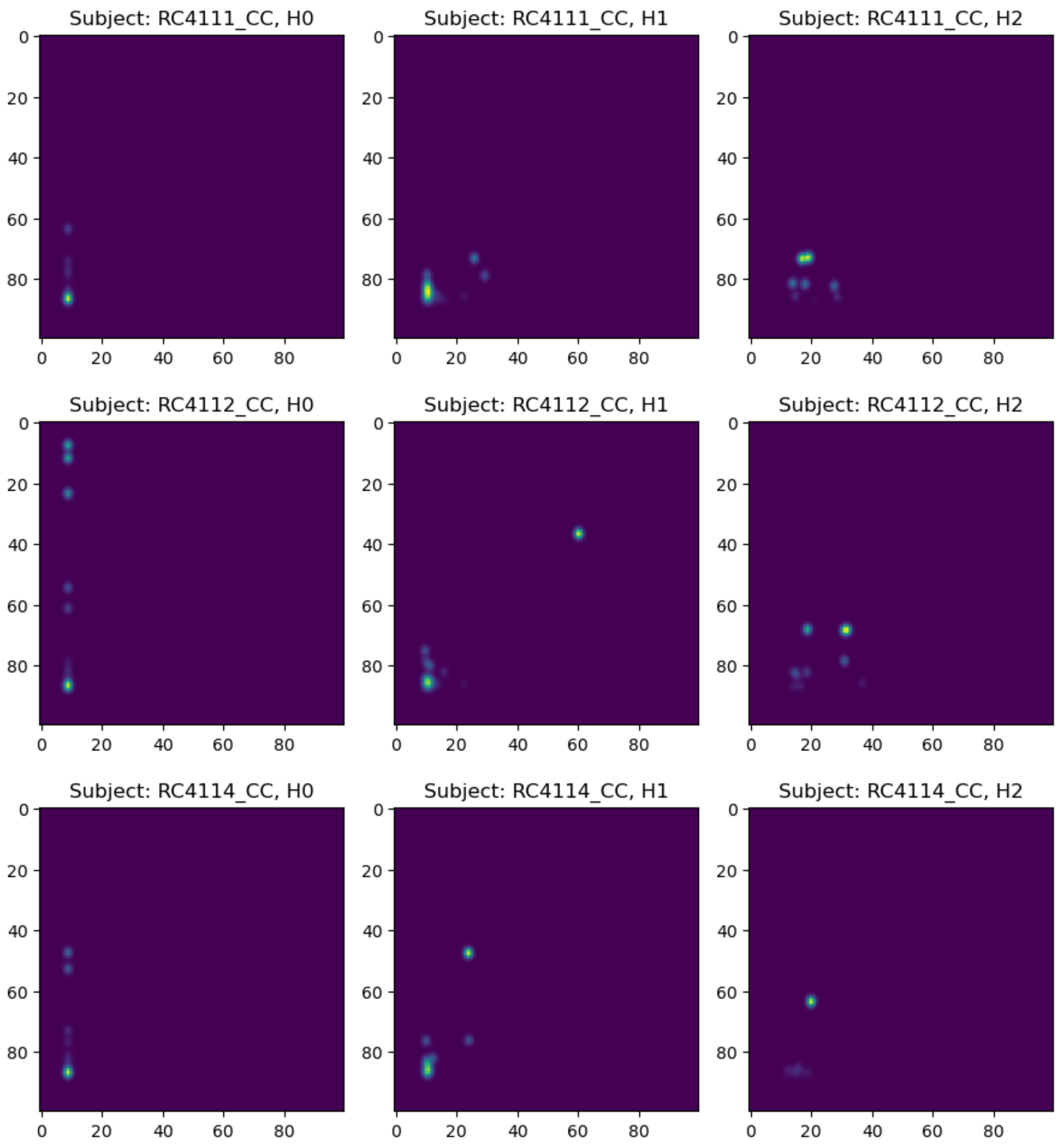
```python
#compute and plot persistence images
import numpy as np
import gudhi.representations
import matplotlib.pyplot as plt
import gudhi as gd

def filter_infinite_points(diagram):
    return [(birth, death) for dim, (birth, death) in diagram if np.isfinite(death) and death != 999999999.9999


PI = gd.representations.PersistenceImage(bandwidth=5*1e-2, weight=lambda x: x[1]**1, im_range=[-.5,5,-.5,4],reso

# Transform diagrams to images
H0_pd_images_W00 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H0_pd_diagrams_W00]
H1_pd_images_W00 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H1_pd_diagrams_W00]
H2_pd_images_W00 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H2_pd_diagrams_W00]

H0_pd_images_W36 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H0_pd_diagrams_W36]
H1_pd_images_W36 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H1_pd_diagrams_W36]
H2_pd_images_W36 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H2_pd_diagrams_W36]

H0_cc_images = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H0_cc_diagrams]
H1_cc_images = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H1_cc_diagrams]
H2_cc_images = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H2_cc_diagrams]


# Construct a dictionary
subjects_dict = {
    "101018JW_W00": [H0_pd_images_W00[0], H1_pd_images_W00[0], H2_pd_images_W00[0]],
    "101026DD_W00": [H0_pd_images_W00[1], H1_pd_images_W00[1], H2_pd_images_W00[1]],
    "101001YM_W00": [H0_pd_images_W00[2], H1_pd_images_W00[2], H2_pd_images_W00[2]],
    "101012DE_W00": [H0_pd_images_W00[3], H1_pd_images_W00[3], H2_pd_images_W00[3]],
    "101016DC_W00": [H0_pd_images_W00[4], H1_pd_images_W00[4], H2_pd_images_W00[4]],
    "101018JW_W36": [H0_pd_images_W36[0], H1_pd_images_W36[0], H2_pd_images_W36[0]],
    "101026DD_W36": [H0_pd_images_W36[1], H1_pd_images_W36[1], H2_pd_images_W36[1]],
    "101001YM_W36": [H0_pd_images_W36[2], H1_pd_images_W36[2], H2_pd_images_W36[2]],
    "101012DE_W36": [H0_pd_images_W36[3], H1_pd_images_W36[3], H2_pd_images_W36[3]],
    "101016DC_W36": [H0_pd_images_W36[4], H1_pd_images_W36[4], H2_pd_images_W36[4]],
    "RC4107_CC": [H0_cc_images[0], H1_cc_images[0], H2_cc_images[0]],
    "RC4110_CC": [H0_cc_images[1], H1_cc_images[1], H2_cc_images[1]],
    "RC4111_CC": [H0_cc_images[2], H1_cc_images[2], H2_cc_images[2]],
    "RC4112_CC": [H0_cc_images[3], H1_cc_images[3], H2_cc_images[3]],
    "RC4114_CC": [H0_cc_images[4], H1_cc_images[4], H2_cc_images[4]],
}


# Calculate number of columns for subplot
n_images_per_subject = 3  # H0, H1, H2
n_cols = n_images_per_subject

# Define homologies
homologies = ['0', '1', '2']

# Iterate over subjects and display all images
for i, (subject, images) in enumerate(subjects_dict.items()):
    # Create a new figure for each subject
    fig, axs = plt.subplots(1, n_cols, figsize=(3*n_cols,5))

    for j, image in enumerate(images, start=0):
        # Reshape the image to 2D (as per your PI resolution), and flip it vertically
        reshaped_image = image[0].reshape(100,100)
        flipped_image = np.flip(reshaped_image, 0)

        # Add a subplot for the image
        ax = axs[j]
        ax.imshow(flipped_image)

        # Add title with subject and homology information
        ax.set_title(f"Subject: {subject}, H{homologies[j]}")

    # Show the plot
    plt.tight_layout()
    plt.show()
```


Subject: 101018JW_W00, H0
Subject: 101018JW_W00, H1
Subject: 101018JW_W00, H2
Subject: 101026DD_W00, H0
Subject: 101026DD_W00, H1
Subject: 101026DD_W00, H2
Subject: 101001YM_W00, H0
Subject: 101001YM_W00, H1
Subject: 101001YM_W00, H2
Subject: 101012DE_W00, H0
Subject: 101012DE_W00, H1
Subject: 101012DE_W00, H2


Subject: 101016DC_W00, H0
Subject: 101016DC_W00, H1
Subject: 101016DC_W00, H2
Subject: 101018JW_W36, H0
Subject: 101018JW_W36, H1
Subject: 101018JW_W36, H2
Subject: 101026DD_W36, H0
Subject: 101026DD_W36, H1
Subject: 101026DD_W36, H2
Subject: 101001YM_W36, H0
Subject: 101001YM_W36, H1
Subject: 101001YM_W36, H2
0
20
40
60
80



Subject: 101012DE_W36, H0
Subject: 101012DE_W36, H1
Subject: 101012DE_W36, H2
Subject: 101016DC_W36, H0
Subject: 101016DC_W36, H1
Subject: 101016DC_W36, H2
Subject: RC4107_CC, H0
Subject: RC4107_CC, H1
Subject: RC4107_CC, H2
Subject: RC4110_CC, H0
Subject: RC4110_CC, H1
Subject: RC4110_CC, H2



Classification

In [107…

```
#Discriminating the subjects pd vs control
from sklearn.model_selection import train_test_split

def create_dataset(images_cc, images_pd):
    X = np.concatenate((images_cc, images_pd))
    y = np.concatenate((np.zeros(len(images_cc)), np.ones(len(images_pd)))) # 0 for CC, 1 for PD
    X = [img[0].reshape(-1) for img in X] # Flatten the images
    return train_test_split(X, y, test_size=0.2, random_state=42) # Split into training and test sets
```

In [108…

```
#training classifiers for each homology to discriminate PD vs Control
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

def train_logistic_regression(X_train, y_train, X_test, y_test):
    clf = LogisticRegression()
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print("Logistic Regression Accuracy:", accuracy)
```

In [109…
```python
from sklearn.svm import SVC

def train_svm(X_train, y_train, X_test, y_test):
    clf = SVC(kernel='linear')
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print("SVM Accuracy:", accuracy)
```

In [112…
```python
for homology, images_cc, images_pd in zip(['H0', 'H1', 'H2'], [H0_cc_images, H1_cc_images, H2_cc_images], [H0_p
    print(f"Training classifiers for {homology}")
    X_train, X_test, y_train, y_test = create_dataset(images_cc, images_pd)
    train_logistic_regression(X_train, y_train, X_test, y_test)
    train_svm(X_train, y_train, X_test, y_test)
```

```
Training classifiers for H0
Logistic Regression Accuracy: 1.0
SVM Accuracy: 1.0
Training classifiers for H1
Logistic Regression Accuracy: 1.0
SVM Accuracy: 1.0
Training classifiers for H2
Logistic Regression Accuracy: 0.5
SVM Accuracy: 0.5
```

In [113…
```python
#Discriminating between pd sessions (W00 vs W36)
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

def train_classifiers(images_W00, images_W36, homology_name):
    # Combine images and labels
    images_combined = images_W00 + images_W36
    labels = [0] * len(images_W00) + [1] * len(images_W36)

    # Reshape images
    images_flatten = [img.flatten() for img in images_combined]

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(images_flatten, labels, test_size=0.2, random_state=42)

    # Logistic Regression
    lr = LogisticRegression()
    lr.fit(X_train, y_train)
    lr_accuracy = accuracy_score(y_test, lr.predict(X_test))

    # SVM
    svm = SVC()
    svm.fit(X_train, y_train)
    svm_accuracy = accuracy_score(y_test, svm.predict(X_test))

    print(f"Training classifiers for {homology_name}")
    print(f"Logistic Regression Accuracy: {lr_accuracy}")
    print(f"SVM Accuracy: {svm_accuracy}")

# Call the function for H0, H1, and H2 homologies
train_classifiers(H0_pd_images_W00, H0_pd_images_W36, "H0")
train_classifiers(H1_pd_images_W00, H1_pd_images_W36, "H1")
train_classifiers(H2_pd_images_W00, H2_pd_images_W36, "H2")
```

```
Training classifiers for H0
Logistic Regression Accuracy: 0.5
SVM Accuracy: 0.5
Training classifiers for H1
Logistic Regression Accuracy: 0.5
SVM Accuracy: 0.5
Training classifiers for H2
Logistic Regression Accuracy: 1.0
SVM Accuracy: 1.0
```

In [114…

```python
#3 classes : pds W00 VS pds W36 VS controls
#before paramater tunning
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import numpy as np

def train_SVM(cc_images, pd_images_W00, pd_images_W36):
    # Concatenate the images and create labels
    all_images = np.concatenate([cc_images, pd_images_W00, pd_images_W36])
    all_images = all_images.reshape(all_images.shape[0], -1) # Flatten each image
    labels = [0] * len(cc_images) + [1] * len(pd_images_W00) + [2] * len(pd_images_W36)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(all_images, labels, test_size=0.2, random_state=42)

    # Apply scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Create and fit the SVM model
    clf = svm.SVC(kernel='linear')
    clf.fit(X_train, y_train)

    # Predict the test set results
    y_pred = clf.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy

# Training the classifiers for each homology degree
accuracy_H0 = train_SVM(H0_cc_images, H0_pd_images_W00, H0_pd_images_W36)
accuracy_H1 = train_SVM(H1_cc_images, H1_pd_images_W00, H1_pd_images_W36)
accuracy_H2 = train_SVM(H2_cc_images, H2_pd_images_W00, H2_pd_images_W36)

print("SVM Accuracy for H0:", accuracy_H0)
print("SVM Accuracy for H1:", accuracy_H1)
print("SVM Accuracy for H2:", accuracy_H2)
```

```
SVM Accuracy for H0: 0.3333333333333333
SVM Accuracy for H1: 0.6666666666666666
SVM Accuracy for H2: 0.0
```

In [ ]:

```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Function to perform hyperparameter tuning for the images
def tune_hyperparameters(cc_images, pd_images_W00, pd_images_W36):
    # Concatenate images and labels
    X = np.concatenate([cc_images, pd_images_W00, pd_images_W36], axis=0)
    y = np.array([0] * len(cc_images) + [1] * (len(pd_images_W00) + len(pd_images_W36)))

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Apply scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train.reshape(X_train.shape[0], -1))
    X_test = scaler.transform(X_test.reshape(X_test.shape[0], -1))

    # Define the parameter grid
    param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'linear']}

    # Create a GridSearchCV object
    grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=2, cv=min(3, len(np.unique(y))))

    # Fit the model
    grid.fit(X_train, y_train)

    # Print the best parameters
    print("Best Parameters: ", grid.best_params_)

    # Print the test accuracy
    print("Test Accuracy: ", grid.score(X_test, y_test))

# Tuning hyperparameters for each homology degree
print("Tuning for H0:")
tune_hyperparameters(H0_cc_images, H0_pd_images_W00, H0_pd_images_W36)

print("\nTuning for H1:")
tune_hyperparameters(H1_cc_images, H1_pd_images_W00, H1_pd_images_W36)

print("\nTuning for H2:")
tune_hyperparameters(H2_cc_images, H2_pd_images_W00, H2_pd_images_W36)
```

In [119…

```python
#after parameter tunning
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import numpy as np

def train_SVM(cc_images, pd_images_W00, pd_images_W36, kernel, C, gamma):
    # Concatenate the images and create labels
    all_images = np.concatenate([cc_images, pd_images_W00, pd_images_W36])
    all_images = all_images.reshape(all_images.shape[0], -1) # Flatten each image
    labels = [0] * len(cc_images) + [1] * len(pd_images_W00) + [2] * len(pd_images_W36)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(all_images, labels, test_size=0.2, random_state=42)

    # Apply scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Create and fit the SVM model with the given kernel, C, and gamma values
    clf = svm.SVC(kernel=kernel, C=C, gamma=gamma)
    clf.fit(X_train, y_train)

    # Predict the test set results
    y_pred = clf.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy

# Training the classifiers for each homology degree with the best parameters
accuracy_H0 = train_SVM(H0_cc_images, H0_pd_images_W00, H0_pd_images_W36, kernel='linear', C=0.1, gamma=1)
accuracy_H1 = train_SVM(H1_cc_images, H1_pd_images_W00, H1_pd_images_W36, kernel='linear', C=0.1, gamma=1)
accuracy_H2 = train_SVM(H2_cc_images, H2_pd_images_W00, H2_pd_images_W36, kernel='rbf', C=0.1, gamma=1)

print("SVM Accuracy for H0:", accuracy_H0)
print("SVM Accuracy for H1:", accuracy_H1)
print("SVM Accuracy for H2:", accuracy_H2)
```

```
SVM Accuracy for H0: 0.3333333333333333
SVM Accuracy for H1: 0.6666666666666666
SVM Accuracy for H2: 0.3333333333333333
```