1. Import Required Libraries

pip install torch torchvision torch-geometric scikit-image opencv-python scikit-learn numpy

2. Data Pre-processing and Conversion to Graphs

a. Image Segmentation (SLIC Superpixels)

```
import numpy as np
from skimage.segmentation import slic
from skimage import io
import cv2

def segment_image(image_path):
    # Load image
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply SLIC segmentation
    segments = slic(image, n_segments=100, compactness=10)

    return segments
```

b. Convert Segments to Graphs

```
from skimage.future.graph import rag_mean_color, RAG
from torch_geometric.data import Data

def image_to_graph(image_path):
    image = io.imread(image_path)
    segments = segment_image(image_path)

    # Create RAG (Region Adjacency Graph)
    rag = RAG(segments, connectivity=2)

    # Create node features (mean intensity in this case)
    node_features = []
    for region in rag.nodes:
        mask = segments == region
        mean_intensity = image[mask].mean()
        node_features.append([mean_intensity])

    node_features = torch.tensor(node_features, dtype=torch.float)
```

```
    # Create edges
    edges = []
    for (node1, node2) in rag.edges:
        edges.append([node1, node2])

    edge_index = torch.tensor(np.array(edges).T, dtype=torch.long)

    # Create graph data object
    data = Data(x=node_features, edge_index=edge_index)

    return data
```

c. Process All Images

```
import os

def process_dataset(image_dir):
    graphs = []
    labels = []

    for label, class_dir in enumerate(['subdural', 'epidural', 'subarachnoid']):
        class_path = os.path.join(image_dir, class_dir)
        for image_name in os.listdir(class_path):
            image_path = os.path.join(class_path, image_name)
            graph = image_to_graph(image_path)
            graphs.append(graph)
            labels.append(label)

    return graphs, labels
```

3. GCNN Model Implementation

a. Define the GCNN Model

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, global_max_pool

class GCNNModel(nn.Module):
    def __init__(self, input_features, hidden_channels):
        super(GCNNModel, self).__init__()
       self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        self.conv4 = nn.Conv2d(128, 256, 3, padding=1)
        self.conv5 = nn.Conv2d(256, 512, 3, padding=1)
        self.conv6 = nn.Conv2d(512, 1024, 3, padding=1)

        self.conv7 = nn.Conv2d(1024, 2048, 3, padding=1)
        self.conv8 = nn.Conv2d(2048, 4096, 3, padding=1)
        self.conv9 = nn.Conv2d(4096, 8192, 3, padding=1)

        self.fc1 = nn.Linear(8192 * 16 * 16, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 3)

    def forward(self, data):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = self.pool(F.relu(self.conv5(x)))
        x = self.pool(F.relu(self.conv6(x)))
        x = self.pool(F.relu(self.conv7(x)))
        x = self.pool(F.relu(self.conv8(x)))
        x = self.pool(F.relu(self.conv9(x)))
        x = x.view(-1, 8192 * 16 * 16)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

b. Training the GCNN Model

```
def train_gcnn(model, train_loader, optimizer, criterion, num_epochs=50):
    model.train()
    for epoch in range(num_epochs):
        total_loss = 0
        for data in train_loader:
            data = data.to(device)
            optimizer.zero_grad()
            out = model(data)
            loss = criterion(out, data.y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f'Epoch {epoch+1}/{num_epochs}, Loss: {total_loss/len(train_loader):.4f}')
```

4. Evaluation and Metrics

a. Evaluate the Model

```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score

def evaluate_gcnn(model, test_loader):
    model.eval()
    all_labels = []
    all_preds = []

    with torch.no_grad():
        for data in test_loader:
            data = data.to(device)
            out = model(data)
            pred = out.argmax(dim=1)
            all_labels.extend(data.y.cpu().numpy())
            all_preds.extend(pred.cpu().numpy())

    # Convert lists to numpy arrays
    all_labels = np.array(all_labels)
    all_preds = np.array(all_preds)

    # Calculate metrics
    accuracy = accuracy_score(all_labels, all_preds)
    f1 = f1_score(all_labels, all_preds, average='weighted')
    precision = precision_score(all_labels, all_preds, average='weighted')
    recall = recall_score(all_labels, all_preds, average='weighted')
    cm = confusion_matrix(all_labels, all_preds)

    print(f'Accuracy: {accuracy * 100:.2f}%')
    print(f'F1 Score: {f1:.2f}')
    print(f'Precision: {precision:.2f}')
    print(f'Recall: {recall:.2f}')

    # Print confusion matrix
    print('Confusion Matrix:')
    print(cm)
```