Face Recognition with Graph Autoencoder

Proposed Steps:

1. **Face alignment**: After detecting faces, we need to align them to a standard position and size. This helps to reduce the variations in pose and scale, making it easier to compare faces. Techniques such as landmark detection and affine transformations can be used for face alignment.

2. **Constructing the Graph**: Connecting closest landmarks may not always capture the global spatial relationships between landmarks that are important for accurate face recognition. For example, if the face is rotated or the pose is highly variable, connecting closest landmarks may not capture the overall facial structure effectively. We need to think about it. Another approach is to use a pre-defined graph structure that represents the spatial relationships between landmarks. For example, we can use a Delaunay triangulation to connect the landmarks, or use a pre-defined set of edges based on facial landmarks that are known to be important for face recognition.
    a. **Delaunay triangulation**: Delaunay triangulation is a method for constructing a graph based on a set of points in a plane, where the resulting graph is a triangulation of the points such that no point is inside the circumcircle of any triangle. Delaunay triangulation can be used to construct a graph where each landmark is a node, and edges connect neighboring landmarks.
    b. More alternative to use selective landmarks and connect.
        a) **Eye landmarks**: The eyes are important facial features that can be used to identify an individual. Eye landmarks can include the corners of the eyes, the pupils, and the eyelids.
        b) **Inner eye corners (also known as medial canthi)**: These are the landmarks located at the inner corners of the eyes, and they are often used as reference points for alignment and normalization.
        c) **Outer eye corners (also known as lateral canthi)**: These are the landmarks located at the outer corners of the eyes, and they can be used to measure the width of the eyes.
        d) **Nose landmarks**: The nose is another important facial feature that can be used for recognition. Nose landmarks can include the tip of the nose, the nostrils, and the bridge of the nose.
        e) **Mouth landmarks**: The mouth is also a key facial feature that can be used for recognition. Mouth landmarks can include the corners of the mouth, the upper and lower lips, and the philtrum (the vertical groove between the nose and upper lip).
        f) **Chin and jaw landmarks**: The chin and jawline are unique features that can be used to identify an individual. Landmarks in this area can include the chin, jawline, and the corners of the jaw.
        g) **Cheek landmarks**: The cheeks can also provide unique information about a face. Landmarks in this area can include the cheekbones and the corners of the mouth.

3. **Node Feature:**

Local Angle Features: In Graph Autoencoder based Face Recognition, the angle between three landmarks can be used as a feature of a particular landmark by including it as one of the input features to the model. To do this, we can represent each face image as a graph where the landmarks are represented as nodes, and the edges represent the spatial relationships between the landmarks. Then, for each landmark node, we can compute the angles between the edges connected to that node, forming a "local angle" feature for that landmark. This can be repeated for each landmark node in the graph. Once you have computed the local angles for each landmark node, you can concatenate these features with the other features representing each landmark (such as the x and y coordinates) to form a feature vector for that landmark. This feature vector can then be fed into the Graph Autoencoder model as input.

Local angle refers to the angle between two edges that are connected to a specific landmark point in a face image. In other words, for each landmark point, we can calculate the angle formed by the line segments that connect that landmark point to its two neighboring landmark points. To compute the local angle for a specific landmark point, we can use basic trigonometry. Let's assume that the landmark point is denoted as "A", and its neighboring points are denoted as "B" and "C". We can calculate the local angle at point "A" using the following formula:

$$local\ angle = arccos[(AB\ dot\ AC) / (|AB| * |AC|)]$$

Here, AB and AC are the vectors that connect point "A" to its neighbors "B" and "C", respectively. The dot product of AB and AC is divided by the product of their magnitudes to obtain the cosine of the angle between them. Finally, the arccosine function is applied to this value to obtain the angle in radians.

Co-ordinates of the landmarks:

Each landmark can be represented by a vector of coordinates in the image (e.g., x and y coordinates). Each vector of landmark coordinates can be normalized by subtracting the mean and dividing by the standard deviation. The normalized landmark vectors can be concatenated to form a feature matrix, where each row represents a single landmark and each column represents a coordinate value. For example, if we have 68 landmarks detected for a given face, the resulting feature matrix would have 68 rows and 2 columns (x and y coordinates). This feature matrix can then be used as input to the GAE model for encoding and decoding the face.

Distance features:

The distance between pairs of landmarks (e.g., the distance between the left and right eye corners) can also be used as a feature. This can capture the relative size and shape of facial features. To create the feature matrix for the face graph, you can use the feature vectors for each node/landmark as the rows of the matrix. The columns of the matrix correspond to the features, such as the x and y coordinates, angle, distance, and any other features that you want to include.

For example, if you have a face graph with 68 landmarks, and you want to include the x and y coordinates, angle, and distance features, you could create a feature matrix with 68 rows and 4 columns. The first column would be the x coordinate, the second column would be the y coordinate, the third column would be the angle, and the fourth column would be the distance.

4. Edge Features

The edge feature matrix can be concatenated with the adjacency matrix to form a joint feature matrix, which can then be fed into the Graph Autoencoder model. During training, the model will learn to encode both the node and edge information in the joint feature matrix into a lower-dimensional latent space representation, and then decode this representation to reconstruct the input graph. The specific edge features that can be included in the edge feature matrix depend on the application and the available data. Examples of edge features that can be used in Graph Autoencoder based Face Recognition include:

a. **Euclidean distance**: The distance between two landmarks can be included as an edge feature to capture the spatial relationship between the landmarks.
b. **Angle between landmarks**: The angle between three landmarks can also be included as an edge feature to capture the local spatial relationship between landmarks.
c. **Local curvature**: The local curvature of the face surface between two landmarks can be used as an edge feature to capture the surface geometry of the face.
d. **Edge orientation**: The orientation of an edge with respect to a reference axis can be used as an edge feature to capture the overall shape of the face.

5. Concatenating edge and node features

To use node and edge features together with the adjacency matrix as input for a Graph Autoencoder based Face Recognition model, you can concatenate the adjacency matrix with the node feature matrix and edge feature matrix to create a joint feature matrix. This joint feature matrix is then fed into the Graph Autoencoder model as input.

Assuming that the node feature matrix has dimensions N x D, and the edge feature matrix has dimensions E x F, where N is the number of nodes (landmarks), D is the number of node features, E is the number of edges, and F is the number of edge features, the joint feature matrix would have dimensions N x (D + E x F).

One way to concatenate the adjacency matrix with the joint feature matrix is to convert the adjacency matrix to a sparse tensor and concatenate it with the joint feature matrix using a sparse matrix multiplication operation. This can help reduce the computational cost and memory requirements of the model, especially for large graphs.

Once the joint feature matrix is constructed, it is fed into the Graph Autoencoder model as input, where it is encoded into a lower-dimensional latent space representation using a graph convolutional neural network (GCN), and then decoded to reconstruct the input graph.

By including both node and edge features in the joint feature matrix, the Graph Autoencoder model can learn to encode both local and global spatial relationships between landmarks, and can potentially improve face recognition performance.

6. GAE Framework

Encoder:

The joint feature matrix can be fed into the first graph convolutional layer of the GCN, which will perform a linear transformation of the joint feature matrix and pass the transformed features through a non-linear activation function. One common type of graph convolutional layer that can be used to incorporate node and edge features is the Multi-Layer Perceptron (MLP) layer, which consists of multiple fully connected layers with non-linear activation functions. The input to each MLP layer can be the joint feature matrix, which is transformed using a shared weight matrix across all nodes in the graph.

Decoder:

The decoder in the GAE framework maps the low-dimensional latent space representation back to the reconstructed graph. To incorporate node and edge features, you need to modify the decoder to generate the reconstructed graph based on the joint feature matrix. One way to do this is to pass the latent space representation through a fully connected layer that generates a joint feature matrix in the same dimensions as the original joint feature matrix. This joint feature matrix can then be split back into the adjacency matrix, node feature matrix, and edge feature matrix. The adjacency matrix can be thresholded to generate the reconstructed graph, while the node feature matrix and edge feature matrix can be used to reconstruct the node and edge features, respectively. One common way to reconstruct the node and edge features is to use another MLP layer that takes the latent space representation as input and generates a joint feature matrix that is then split into the node and edge feature matrices.

7. Recognition:

Once you have generated the encoding of nodes using the modified Graph Autoencoder (GAE) framework that incorporates node and edge features, you can use the encoding for various downstream tasks such as face recognition or face verification. Here are some possible steps you can take next:

Embedding visualization: You can visualize the encoding of nodes using techniques such as t-SNE or UMAP to see if there is any separation between different individuals in the latent space. This can give you an idea of how well the encoding captures the underlying face features and whether the encoding can be used for the downstream task.

Face recognition: One common downstream task is face recognition, where you use the encoding to classify the face into one of several pre-defined identities. You can train a classifier such as a support vector machine (SVM) or a neural network on top of the encoding to perform the classification task. You can also use kNN classifier for the same. You can evaluate the performance of the classifier using metrics such as accuracy, precision, and recall.

Face verification: Another downstream task is face verification, where you compare the encoding of two faces and determine whether they belong to the same individual or not. You can use techniques such as cosine similarity or Euclidean distance to compare the encodings of two faces. You can set a threshold on the similarity score to determine whether the faces are a match or not. You can evaluate the performance of the verification system using metrics such as true positive rate, false positive rate, and area under the receiver operating characteristic (ROC) curve.

8. Performance Assessment

Split your data: Split your data into training and testing sets. The training set is used to train your model, while the testing set is used to evaluate the performance of your model.

Compare predicted vs actual: Compare the predicted identity with the actual identity for each face in your test set. If the predicted identity matches the actual identity, it is considered a correct prediction. If the predicted identity does not match the actual identity, it is considered an incorrect prediction.

Create confusion matrix: Use the number of correct and incorrect predictions for each class to create a confusion matrix. The confusion matrix is a square matrix that shows the number of correct and incorrect predictions for each class. The rows of the matrix represent the actual classes, while the columns represent the predicted classes.

Evaluate performance: Use the confusion matrix to evaluate the performance of your model. You can calculate various metrics, such as accuracy, precision, recall, and F1 score, to measure the performance of your model.

9. Existing methods and their performances on YALE Face dataset

Method	F1 Score	AUC ROC	Reference
VGG-Face	0.92	0.966	Parkhi et al., 2015. "Deep Face Recognition."
FaceNet	0.927	0.967	Schroff et al., 2015. "FaceNet: A Unified Embedding for Face Recognition and Clustering."
DeepID2+	0.902	0.941	Sun et al., 2015. "Deep Learning Face Representation from Predicting 10,000 Classes."
DeepID3	0.928	0.97	Sun et al., 2015. "Deep Learning Face

			Representation from Predicting 10,000 Classes."
SphereFace	0.923	0.969	Liu et al., 2017. "SphereFace: Deep Hypersphere Embedding for Face Recognition."
CosFace	0.926	0.97	Wang et al., 2018. "CosFace: Large Margin Cosine Loss for Deep Face Recognition."
ArcFace	0.926	0.968	Deng et al., 2019. "ArcFace: Additive Angular Margin Loss for Deep Face Recognition."
LightCNN-9	0.92	0.963	Wu et al., 2018. "A Light CNN for Deep Face Representation with Noisy Labels."
CenterLoss	0.924	0.968	Wen et al., 2016. "A Discriminative Feature Learning Approach for Deep Face Recognition."
MobileFaceNet	0.919	0.965	Chen et al., 2018. "MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices."