```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class AGES(nn.Module):
    def __init__(self, audio_feature_extractor, visual_feature_extractor, num_classes):
        super(AGES, self).__init__()
        self.audio_feature_extractor = audio_feature_extractor
        self.visual_feature_extractor = visual_feature_extractor
        self.num_classes = num_classes
        self.fc = nn.Linear(1024, num_classes)  # Assuming the combined feature dimension is 1024

    def forward(self, audio_inputs, visual_inputs):
        # 1. Extract features
        local_audio_features, holistic_audio_features, visual_features = self.extract_features(audio_inputs, v
isual_inputs)

        # 2. Apply self-attention mechanism
        local_audio_attention_features = self.apply_self_attention(local_audio_features)
        holistic_audio_attention_features = self.apply_self_attention(holistic_audio_features)
        visual_attention_features = self.apply_self_attention(visual_features)
```

```python
        # 3. Construct multimodal graph
        adjacency_matrix, combined_features = self.construct_graph(local_audio_attention_features, holistic_au
dio_attention_features, visual_attention_features)

        # 4. Fuse using Graph Convolutional Network (GCN)
        fused_features = self.gcn_fusion(adjacency_matrix, combined_features)

        # 5. Classification
        output = self.fc(fused_features)
        return output

    def extract_features(self, audio_inputs, visual_inputs):
        local_audio_features = self.audio_feature_extractor.local_audio(audio_inputs)
        holistic_audio_features = self.audio_feature_extractor.holistic_audio(audio_inputs)
        visual_features = self.visual_feature_extractor(visual_inputs)
        return local_audio_features, holistic_audio_features, visual_features

    def apply_self_attention(self, features):
        attention = torch.softmax(torch.matmul(features, features.transpose(-1, -2)), dim=-1)
        weighted_features = torch.bmm(features, attention).squeeze()
        return weighted_features
```

```python
    def construct_graph(self, local_audio_features, holistic_audio_features, visual_features):
        adjacency_matrix = torch.zeros(local_audio_features.size(0), local_audio_features.size(0))
        adjacency_matrix += 1  # Add an identity matrix
        combined_features = torch.cat([local_audio_features, holistic_audio_features, visual_features], dim=1)
        return adjacency_matrix, combined_features

    def gcn_fusion(self, adjacency_matrix, combined_features):
        x = combined_features
        A = adjacency_matrix

        x = torch.spmm(A, x)
        x = F.relu(x)
        x = torch.spmm(A, x)

        return x
```