**DOCUMENTATION**

**(Face Analysis App)**

**Overview**:

The Skin Analysis page of the Face Analysis App allows users to upload an image of their face and receive analysis on facial features, specifically wrinkles, spots, texture, and an estimation of skin age.

**Workflow (Functionalities)**:

1. **Image Upload**:
   * The user is prompted to upload an image (**.jpg**, **.png**, or **.jpeg** formats are supported).
   * Once uploaded, the image is displayed back to the user.
2. **Facial Feature Analysis**:
   * After uploading, the app starts analyzing the image.
   * The **process\_image** function seems to handle the analysis, returning details about the face's wrinkles, spots, and texture.
   * The results for wrinkles, spots, and texture are then scaled between 0 to 100 using Min-Max scaling for standardized scoring. Placeholder maximum values are currently used for scaling, which should be adjusted based on dataset or domain knowledge.
   * The standardized scores for wrinkles, spots, and texture are displayed to the user.
3. **Skin Age Analysis**:
   * The uploaded image undergoes skin age analysis.
   * The **model\_predict** function is called to estimate the age based on the uploaded image.
   * The predicted age is then displayed to the user.

**Explanation:**

1. **Facial Landmark Detection**:

Facial landmarks, also known as facial keypoints, are specific points on a face like the corners of the eyes, nose tip, mouth corners, etc. Detecting these landmarks is crucial for various applications such as face alignment, emotion recognition, and more.

The function appears to use a method or library named face\_mesh. Given the naming and common practices, this likely refers to a model or methodology that is trained to detect a dense set of 3D facial landmarks, which is more detailed than typical 2D landmarks.

* + At its core, facial landmark detection aims to locate key points on a human face, like the corners of the eyes, tip of the nose, mouth edges, etc.
  + The MediaPipe library, developed by Google, offers pre-trained models for various tasks, including facial landmark detection. The model in use, **face\_mesh**, seems to be their solution for detecting a large number of facial landmarks, potentially hundreds.
  + The architecture of such models often involves convolutional neural networks (CNNs). CNNs are adept at processing image data and recognizing patterns. For facial landmarks, the model is trained on vast datasets with labeled landmark points. Over time, the model learns to recognize facial structures and predict the landmark locations.

1. **Mesh Creation:**
   * Once the landmarks are detected, creating a mesh involves connecting these points in a meaningful manner. The mesh can represent the face's topology.
   * The **mp\_face\_mesh.FACEMESH\_TESSELATION** seems to contain predefined connections between landmark points to form the mesh. This tesselation ensures that the mesh represents facial structures accurately.
   * The function then draws lines between connected points to visualize this mesh on the face.
2. **Wrinkle and Spot Detection:**

The skin's texture and topology around wrinkles and spots are different from the surrounding skin. Edge detection can capture the fine lines of wrinkles, while segmentation techniques can isolate spots. The landmarks and mesh provide a guide to focus these techniques on regions where wrinkles and spots are commonly found. These are significant indicators of skin health, age, and other factors like sun damage.

**Steps and detailed explanation:**

1. **Converts the ROI to grayscale**:
   * **Why?**: Color information is often unnecessary for detecting features like edges or textures. Working with grayscale images simplifies computations and makes subsequent operations more effective.
2. **Uses bilateral filtering for noise reduction while keeping the edges sharp**:
   * **Why?**: While traditional filters, like Gaussian, blur the edges while reducing noise, bilateral filtering preserves edges. Preserving edges is crucial because wrinkles are essentially edges or abrupt changes in pixel intensities.
3. **Applies Canny edge detection to find edges, which correspond to wrinkles**:
   * **Why?**: Canny edge detection is a multi-step process that detects a wide range of edges in images. Wrinkles on the skin can be thought of as edges because they represent areas where there's a rapid change in pixel intensity. The Canny algorithm is especially suitable because it suppresses noise while finding genuine edges.
4. **Counts the detected wrinkles (edges)**:
   * **Why?**: After edge detection, the number of edge pixels can act as a proxy to the number of wrinkles. A higher count of edge pixels might indicate more wrinkles or more pronounced wrinkles.
5. **Uses adaptive thresholding to segment the skin spots**:
   * **Why?**: Spots on the skin, like freckles or blemishes, can vary in intensity and might not have a uniform appearance across the image. Adaptive thresholding considers small windows of pixels and determines a threshold for each window, making it more effective in segmenting non-uniform regions like skin spots.
6. **Performs morphological operations to refine the thresholded image**:
   * **Why?**: Morphological operations like opening (erosion followed by dilation) and closing (dilation followed by erosion) are used to reduce image noise. They can remove small white or black spots, respectively, ensuring that only significant spots are counted.
7. **Finds contours in the thresholded image to detect spots**:
   * **Why?**: Once the image is thresholded, contiguous groups of white pixels can be treated as potential skin spots. Finding contours helps in identifying and isolating these groups.
8. **Filters out very small contours to reduce noise and counts the spots**:
   * **Why?**: Not every white pixel group after thresholding is a skin spot. Noise or minor imperfections can also lead to small contours. By setting a threshold on contour size (i.e., contour area), we can filter out insignificant detections, ensuring that only genuine spots are counted.

**Returns**: Counts of wrinkles and spots, which can then be used for further analysis or reporting in the application.

**Note**: In essence, the function leverages classical computer vision techniques to extract meaningful features from the skin, which can then be used for various applications like skincare recommendations, aging analysis, or health assessments.

1. **Texture Detection:**

This function aims to describe the texture of an image, a prominent feature in skin analysis. Texture can provide insights into skin conditions, hydration levels, age, and other factors.

**Steps and detailed explanation:**

1. **Converts the image to grayscale**:
   * **Why?**: LBP operates on single-channel images, primarily grayscale. The texture in an image often relates to the changes in intensity rather than color. Grayscale simplifies the data while preserving texture information.
2. **Computes LBP of the grayscale image**:
   * **Why?**: LBP works by comparing a central pixel with its surrounding neighbors in a local neighborhood. For each neighbor, if its value is greater than or equal to the central pixel's value, it's assigned a value of 1; otherwise, 0. This creates a binary pattern. This process captures local texture variations.
     + For instance, consider a 3x3 window in an image. The central pixel is compared with its 8 neighbors, resulting in an 8-digit binary number (pattern). This pattern is representative of the local texture.
3. **Computes a histogram of the LBP values to capture texture information**:
   * **Why?**: Once the LBP values are calculated for all pixels, a histogram provides a global representation of the texture. The histogram bins correspond to specific LBP patterns, and the bin counts indicate the frequency of each pattern in the image. A uniform texture would have one or few dominant patterns, while a complex texture would have a more evenly distributed histogram.
4. **Returns**: Sum of the LBP histogram values, representing the texture strength of the image.
   * **Why?**: The sum of the histogram values gives a singular metric representing the overall texture strength or complexity. While the histogram itself provides a detailed texture profile, the sum provides a simplified measure for easy comparisons or further analysis.

**LBP and its Variant "Uniform"**: LBP, as a descriptor, is robust against monotonic gray-scale changes (like illumination variations). The "uniform" variant of LBP further refines the descriptor. An LBP pattern is deemed "uniform" if it contains at most 2 bitwise transitions (0-to-1 or 1-to-0) when viewed as a circular binary string. For example, patterns **00000000**, **11111111**, **01111111**, and **10000000** are uniform, but **01010101** is not.

**Why did we use the "uniform" variant**?

1. It greatly reduces the dimensionality of the LBP histogram. Instead of having 2^*P* bins for a P-sized neighborhood, most of the patterns are categorized into "non-uniform" bins, and only the uniform patterns get individual bins. This makes the histogram more focused on common (and often more meaningful) texture patterns.
2. Uniform patterns tend to capture microstructures and primitive structural information which are often more useful in applications like face recognition and skin analysis.

In the context of skin analysis, texture is pivotal. Different skin conditions, aging, hydration levels, and other factors can influence skin texture. Using LBP, especially the uniform variant, offers a computationally efficient yet powerful way to capture and quantify this texture.

1. **Age Prediction:**
   1. **Dataset Used: WIKI**

It consists of 62308 images crawled from all profile images from pages of people from Wikipedia with the same meta information. In the Wikipedia dataset, the age labels were assigned by first removing the images without timestamp (the date when the photo was taken). Then, assuming that the images with single faces are likely to show the personality and that the timestamp and date of birth are correct, a biological (real) age was assigned to each such image.

* 1. **Preprocessing:**

Steps Undergone:

1. The WIKI dataset for Age Estimation was utilized.
2. The Matlab file was loaded, and the CSV file was extracted from it.
3. The DOB was derived from its datenum format, resulting in data for 62,328 images.
4. Images with more than one face and those without any face were removed.
5. Images of insufficient quality were identified using the 'face\_score' column. A threshold of 3.0 was initially set, yielding data for 22,578 images.
6. Another model was trained with a reduced threshold of 1.75, allowing for the inclusion of 34,200 images.
7. All retained images were confirmed to have ages between 0 and 100.
8. All unnecessary columns were subsequently discarded.
9. For age estimation, models were approached in two distinct ways:
   * Exact pixel values of the images were obtained, normalized, and reshaped into the desired dimensions (number of images, 180, 180, 1). The train\_test\_split() function from the scikit-learn library was employed to divide the data.
   * The ImageDataGenerator class in Keras was utilized to load only certain portions of the dataset into memory and to augment the training set.
10. Only the training set underwent augmentation.
11. It was observed that the latter design yielded superior results.
12. A significant imbalance in the dataset was noted, with 20,422 out of 22,578 images (or 90.4% of the data) falling between the ages of 21 & 75.
13. To address this imbalance, the model was also trained on a modified dataset containing only images between the ages of 21 & 75.

**3. Model Training and Experimentation:**

1. The problem is defined as a regression task with the objective of predicting an individual's exact age.
2. The model architecture for age/gender classification is as follows:
   * The input size is set to 180 x 180 x 3.
   * The loss function utilized is MSE (Mean Squared Error), with Adam as the optimizer. The metrics used for evaluation are MSE and MAE (Mean Absolute Error).
   * A LearningRateScheduler is implemented with an initial learning rate (lr) of 0.006, which is reduced by half every 12 epochs.
   * Training is conducted for a total of 95 epochs, split into two rounds of 45 and 50 epochs.
   * A new initial lr of 0.002 is established for the second round of training after observing minimal performance improvement at the end of the first round.
3. Layer-specific details include:
   * A consistent filter size of 3x3 is employed throughout the architecture.
   * Every Fully Connected (FC) layer is followed by a Dropout layer with rates ranging between 0.2 to 0.3. All layers incorporate BatchNormalization.
   * Convolution layers with 128 or more filters incorporate Spatial Dropout ranging between 0.1 to 0.2.
   * The activation function for all layers is ReLU (Rectified Linear Unit).
4. The above designs were finalized after extensive research, experimentation, consultations, and reviews of numerous research papers.
5. Key areas of experimentation:
   * **Model Structure**:
     + Experimentation was conducted with various configurations of FC layers, convolutional layers, filter sizes, and neuron counts in FC layers.
     + A consistent observation was that 3x3 convolutional filters, increasing in number with depth and in powers of 2, yielded optimal results.
     + Various configurations of MaxPool2D filters were tested, but results were suboptimal.
     + The replacement of Conv2D filters with SeparableConv2D filters, combined with SpatialDropout2D, led to a drastic reduction in model overfitting and a slight performance boost.
   * **Regularization**:
     + Various regularization techniques were explored, from Dropout to L2 regularization. Optimal performance was achieved using Dropout with drop probabilities between 0.2 to 0.35. L2 regularization and its variants produced suboptimal results.
   * **Learning Rate**:
     + Different learning rates, typically between 1e-3 and 1e-5, were trialed. Initially, a step decay approach was adopted for the learning rate. Later, Keras's LearningRateScheduler was used, allowing for learning rate reductions after certain epochs, leading to superior results.
   * **Batch Size**:
     + Different batch sizes, primarily powers of 2 (32, 64, 128, 256), were tested. Optimal results were often achieved with batch sizes of 32 or 64.
   * **Data Augmentation**:
     + Various data augmentation techniques available in Keras's ImageDataGenerator class were tested. It's crucial to apply these augmentations only to the training set and not the validation set.
   * **Dataset**:
     + The WIKI dataset was observed to have a heavy bias towards ages between 20 & 75. Addressing this bias was challenging, and methods like class\_weight in Keras or data augmentation proved ineffective.
     + One approach was to train a model exclusively on images of individuals aged between 21 & 75, though it didn't yield the best real-world results.
     + Another approach that enhanced generalization involved training on a larger dataset of 34,200 images (by reducing the image quality threshold) and employing the ImageDataGenerator for data augmentation.

A diagram of a computer

Description automatically generated

Fig: Model architecture for age estimation