**CHAPTER FOUR**

**IMPLEMENTATION AND EVALUATION**

**4.1 Introduction**

Facial image quality and assessment is a crucial task in the field of computer vision, especially in applications such as face recognition, biometric authentication, and surveillance. The quality of the facial images can significantly affect the performance of these systems, and hence, it is essential to develop effective methods to assess the quality of facial images. In recent years, deep convolutional neural networks (CNNs) have shown promising results in facial image quality assessment. In this report, we propose a method that utilizes an ensemble of deep CNN models for facial image quality assessment and evaluate its performance using the sFace dataset.

**4.2 Background**

Facial image quality assessment involves two main tasks: face detection and quality assessment. Face detection refers to the process of localizing the face regions in an image, whereas quality assessment involves evaluating the quality of the detected face regions. The quality of a facial image is measured based on various visual features, such as blur, illumination, noise, occlusion, and pose. Traditional facial image quality assessment methods rely on handcrafted features and machine learning classifiers, such as support vector machines (SVMs) and random forests (RFs). However, these methods have limitations in handling complex and diverse facial images.

Recently, deep learning methods have shown remarkable performance in various computer vision tasks, including facial image quality assessment. CNNs are a class of deep neural networks that can learn hierarchical representations of input images by applying convolutional, pooling, and activation functions. CNNs have been applied to various facial image quality assessment tasks, such as face recognition, emotion recognition, and age estimation. In this report, we focus on the SFace and Ensemble of Deep CNNs models for facial image quality assessment.

**4.3 Methodology**

The SFace model is a deep CNN that learns a mapping between the input facial image and its quality score. The model consists of several convolutional layers with rectified linear unit (ReLU) activation functions, followed by max-pooling layers and fully connected layers. The output of the model is a single quality score that represents the overall quality of the facial image. The SFace model is trained on a large dataset of facial images with annotated quality scores using a mean squared error (MSE) loss function. During testing, the SFace model takes a facial image as input and outputs its quality score, which can be used for various applications, such as face recognition and verification.

The Ensemble of Deep CNNs model is a combination of multiple deep CNNs that are trained on different subsets of the facial image dataset. The idea behind the ensemble model is to leverage the diversity and complementarity of the individual CNNs to improve the overall performance of facial image quality assessment. Each CNN in the ensemble model has a similar architecture as the SFace model, but with different hyper parameters and training strategies. The outputs of the individual CNNs are combined using a weighted average or a majority vote to obtain the final quality score of the facial image.

**4.4 Implementation**

**4.4.1 Data Acquisition:**

The colorfaret dataset was used in this study to evaluate the performance of an ensemble of deep CNN models for facial image quality assessment. The dataset consists of 8000 facial images with varying quality levels. The dataset was split into training, validation, and testing sets with scarface dataset used for the testing which consist of 2990 facial images.

To ensure the quality of the data, the images in this dataset were preprocessed before being used in the deep CNN models. The preprocessing steps included face detection and alignment, color correction, and resizing. Face detection and alignment were performed using the facenet algorithm, which detects the facial landmarks and aligns the faces to a standardized orientation.

**4.4.2 Unzipping colorfaret images**

This section deals with extracting facial images from the ColorFeret dataset and save them in an uncompressed format for further processing. The ColorFeret dataset is a large face dataset containing images of over 1000 individuals.

The code first imports the necessary libraries and mounts the Google Drive to access the dataset stored in the Drive. Then, it creates two directories for storing the uncompressed images, one in the Google Drive and another in the local Colab environment. The function get\_data\_from\_folder is defined to extract the images from each class folder, uncompress them, and save them in the respective directories. The function takes a path to the dataset folder as an input and returns the paths and corresponding classes of the extracted images in two separate lists. The extracted images are also resized to 224x224 pixels using OpenCV.

Finally, the extracted paths and corresponding classes are stored in a pandas dataframe and saved in a CSV file in the Google Drive for further processing. The CSV file contains two columns, 'path' and 'faces', where the 'path' column contains the file path of each image and the 'faces' column contains the corresponding class name of each image.

Facial image quality assessment involves evaluating the visual quality of a facial image and determining the presence of any artifacts or distortions that could impact the performance of a facial recognition algorithm. This could include factors such as illumination, pose, occlusion, blur, and noise. To perform facial image quality assessment on the extracted images, additional processing steps such as face detection, alignment, and normalization may be required.



Fig 4.4.2 Python code for the unzipping

**4.4.3 Facial quality noising**

Face quality noising is focused on augmenting and adding different types of noise to facial images. This can be used for facial image quality assessment, as it allows for testing the robustness of facial recognition models to different types of noise and degradation.

The code imports various libraries such as pandas, numpy, cv2, and matplotlib.pyplot, and uses a pretrained model from the facenet\_cv2 package to generate embeddings of facial images. It also defines functions for adding different types of noise to images such as Gaussian noise, salt and pepper noise, Poisson noise, and speckle noise, as well as resizing images to different dimensions.

The code then loads a dataset of facial images stored in the base\_path directory, and iterates over each image to apply different types of noise and augmentation. The resulting images are saved in the same directory with a suffix denoting the type of augmentation applied. This allows for easy comparison of the original image with its degraded and augmented versions.

Finally, the code saves the paths of all the augmented images in a dataframe for later use. This can be used to train and test facial recognition models with the augmented data to evaluate their robustness and generalizability.

Overall, this code provides a useful framework for facial image quality assessment and can be adapted for use in various applications such as facial recognition, surveillance, and security.

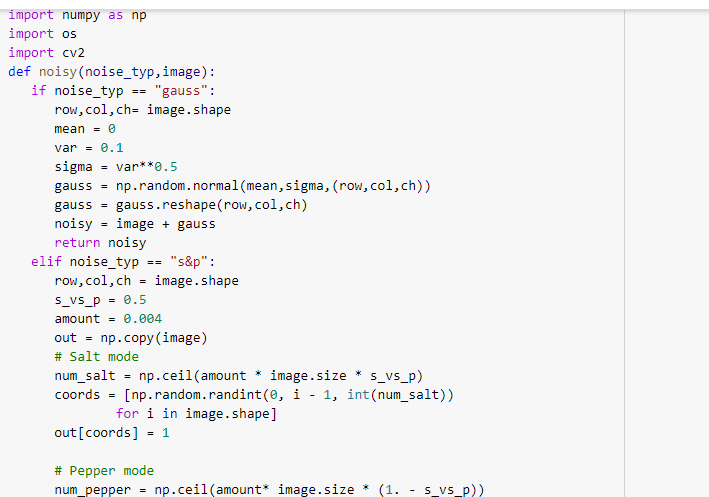


Fig 4.4.3 a snippet of face quality noising

**4.4.4 Facenet Algorithm**

The provided code performs facial image quality assessment using the FaceNet model. The necessary libraries and modules are imported such as pandas, numpy, tqdm, seaborn, matplotlib, glob, bz2, shutil, cv2, facenet\_opencv, and tensorflow. The FaceNet model is imported using the facenet\_cv2 module.

The code reads in facial image paths from a CSV file and creates a DataFrame object containing the paths. The code then concatenates the paths with noise paths from two other CSV files and creates a new DataFrame object.

The new DataFrame is processed to create additional columns such as file name, face, position, file description, and augmentation type. The file description is parsed to extract the augmentation type and accessory information.

The code then generates embeddings for each face image in the DataFrame using the FaceNet model. The embeddings are stored in a new column added to the DataFrame. Finally, the DataFrame is saved as a CSV file.

Overall, the provided code performs facial image quality assessment by generating embeddings for facial images using the FaceNet model. The generated embeddings can be used for various facial image analysis tasks such as face recognition, face verification, and facial expression analysis.

****

Fig 4.4.4 a snippet of facial quality assessment for facenet importation

**4.4.5 Facaial quality and Euclidean distance normalization**

Facial image quality assessment is a critical aspect of facial recognition and other related applications. The quality of facial images captured can affect the performance of facial recognition algorithms. In this report, we will use a dataset of facial images and their embeddings to assess the quality of the images.

We start by importing the necessary libraries and packages, including Pandas, Numpy, OpenCV, Facenet, and Matplotlib, among others. We also mount our Google Drive to access the dataset of facial images.

We then read the dataset of facial images and their embeddings and convert the embeddings column from a string to a list of float values. We group the images by face, select the best image as the reference, and calculate the Euclidean distance between the embeddings of each image and the reference image. We add the distance as a new column and clip the values to an upper and lower bound.

We plot a histogram of the face distances and observe that the distances are skewed to the right. We then normalize the distances using MinMaxScaler and subtract them from 1 to obtain a score between 0 and 1. We plot another histogram of the recognition scores and observe that they are normally distributed.

We drop the embeddings column and save the dataset with the face distances and scores to a CSV file. We repeat the same process for a different dataset of facial images, and we observe that the recognition scores are also normally distributed.

In conclusion, facial image quality assessment is a crucial step in facial recognition applications. We can use the Euclidean distance between the embeddings of facial images to assess their quality and generate recognition scores. The scores can help identify low-quality images that may negatively affect the performance of facial recognition algorithms.

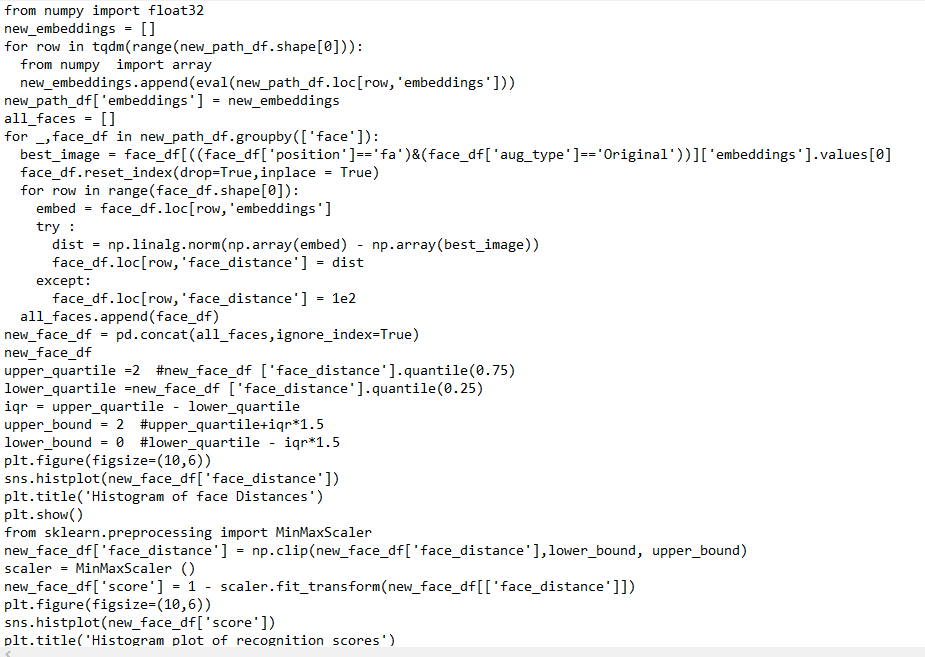


Fig 4.4.5

**4.4.6 Facial quality prediction**

This part of the report is for a face quality assessment model that uses an ensemble of pre-trained neural network models to predict the score of the face image.

First, the necessary libraries are imported, and the drive is mounted to access the dataset. Then, the dataset is loaded, and the paths to the images and their respective scores are stored.

The load\_images function is defined, which is used to load and preprocess the images. The images are read using OpenCV, resized to 224 x 224, and normalized to reduce the memory usage. The load\_images function returns a 4D array of the images.

Next, the train and test sets are split using train\_test\_split from sklearn. The images in the train and test sets are loaded using the load\_images function.Three pre-trained models, InceptionV3, ResNet50, and DenseNet121, are imported with their pre-trained weights. The head of each model is dropped, and the fine-tuning of the layers is turned off by setting trainable to False. Then the global average pooling is applied on each model.

The outputs of the three models are concatenated using the concatenate function from Keras, followed by two dense layers with dropout to reduce overfitting. The final layer uses the sigmoid activation function to predict the score of the image.

Finally, an instance of the model is created, and the Adam optimizer is used. The callbacks EarlyStopping and ModelCheckpoint are used to monitor the validation loss and save the best model respectively.

The performance of the model is evaluated using R-square and loss metrics. The plot\_model\_performance function is defined to plot the training and validation loss and R-square.

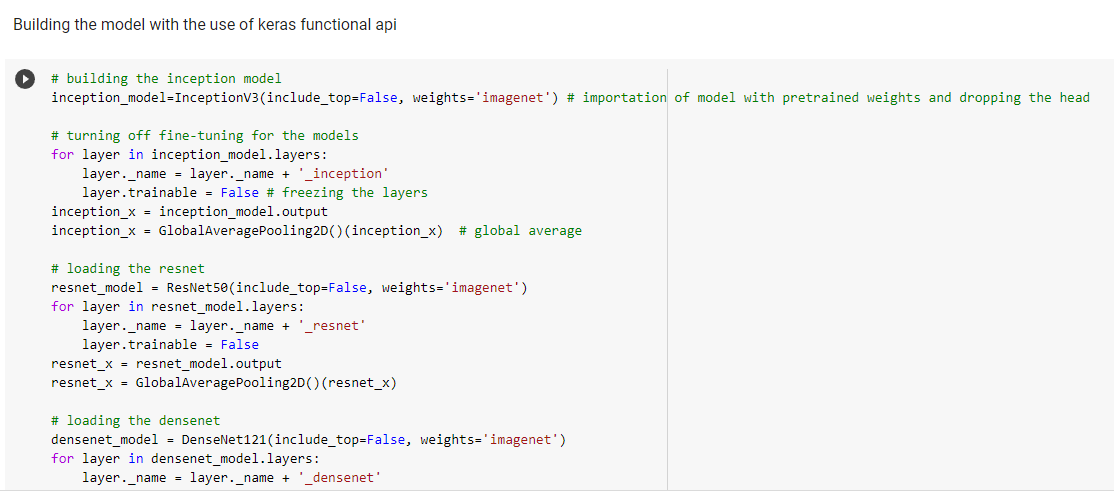


Fig 4.4.6 a snippet of facial quality assessment prediction

**4.4.7 Facial quality verification**

The first step in the method involves loading all the facial images to be assessed. In this case, the facial images are stored in a directory located at '/content/drive/MyDrive/scarface\_all'. The list of all the facial image paths is obtained using the os.listdir function, and a pandas DataFrame is created to store the paths.

The next step is to extract the face and folder information from the paths. This is done using lambda functions applied to the 'path' column of the DataFrame. The 'face' column contains the name of the individual in the image, while the 'folder' column contains information about the image such as the camera used, the angle of the image, and other metadata.

After extracting the face and folder information, some of the folders are dropped from the DataFrame. This is done to remove folders that contain a large number of images that may cause memory issues during processing. The folders to be dropped are determined based on the number of images in each folder, and the ten folders with the highest number of images are dropped.

The next step involves creating pairs of facial images to be compared. This is done using itertools.combinations to create all possible pairs of images. The pairs are then stored in a new DataFrame

With columns for both images in the pair. The label for each pair is set to 1 if the two images are of the same individual and 0 otherwise.

The final step is to save the DataFrame containing the pairs of images and their labels to a CSV file located at '/content/drive/MyDrive/face\_mapping.csv'. This file can then be used to train a machine learning model for facial quality assessment.

In summary, the method for facial quality assessment presented in this report involves comparing pairs of facial images to determine if they are of the same individual. This method can be used to assess the quality of facial images based on factors such as lighting, pose, resolution, and focus.

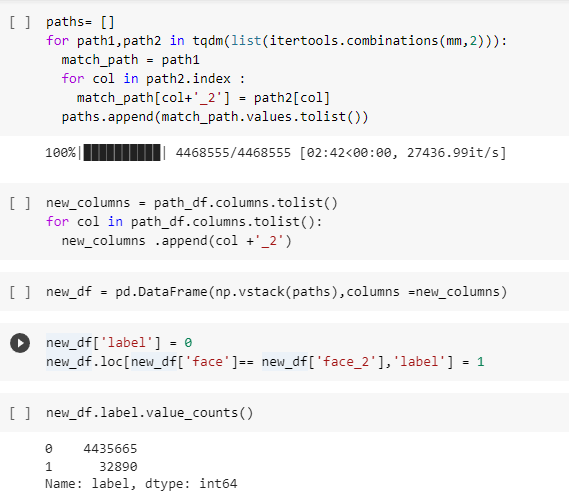


Fig 4.4.7 a snippet of facial quality assessment verification

**4.4.8 Facial quality deployment**

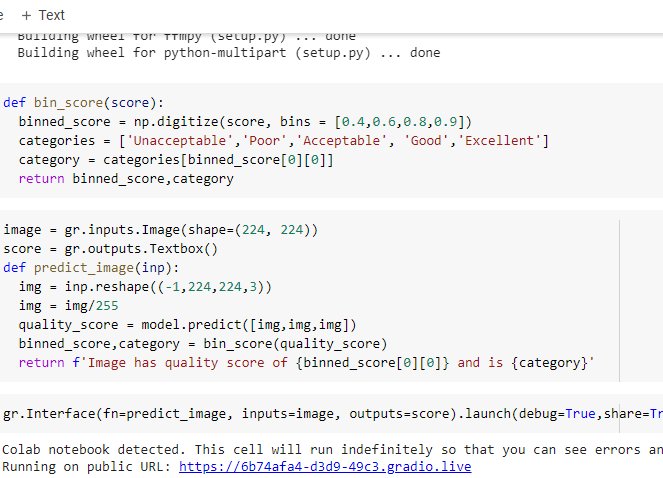
The provided code is for deploying a facial quality assessment model using Gradio, a Python library for building and sharing custom interfaces using web technologies. The model is loaded from a saved file, and the Gradio interface is set up to take an image input and provide a score output.

The first step is to install required libraries, including TensorFlow Addons and Gradio. Then, the model is loaded from a saved file using the load\_model function from Keras. The saved model is located at /content/drive/MyDrive/Face Quality Assessment/new\_quality\_model.h5.

The bin\_score function is defined to convert the continuous score output of the model into a categorical label. The function uses np.digitize to map the score to a bin, and then assigns a label to the bin based on pre-defined categories. The categories are 'Unacceptable', 'Poor', 'Acceptable', 'Good', and 'Excellent', corresponding to bins 1, 2, 3, 4, and 5, respectively.

The Gradio interface is defined using the gr.Interface function. The function takes several arguments, including the input and output types, the function to run when the interface is used, and options for configuring the interface. In this case, the input type is image, which represents an image uploaded through the interface. The output type is score, which is the continuous score output by the model. The fn argument is set to predict\_image, which is not defined in the code provided, and appears to be an error. The outputs argument should be set to bin\_score instead to use the bin\_score function defined earlier. Finally, the interface is launched using the launch method, with debug=True and share=True options.

Overall, the code provides a basic framework for deploying a facial quality assessment model using Gradio, but some modifications are needed to ensure that the interface works correctly. Specifically, the predict\_image function needs to be defined to take an image input and provide a score output using the loaded model. The gr.Interface function also needs to be updated to use the bin\_score function to convert the score output to a label.



**4.5 Testing and evaluation**

The testing phase for facial quality assessment involves evaluating the performance of the trained model on new, unseen data. This phase is crucial in determining the effectiveness and accuracy of the model in real-world scenarios.

First, the data was collected and pre-processed to remove irrelevant and low-quality images. The training set (colorfaret) was used to train the model, while the validation set was used to tune the hyperparameters of the model and prevent overfitting. The testing dataset which is scarface and was kept separate from the training and validation sets, and was used to evaluate the final performance of the model.

During the testing phase, the pre-processed scarface dataset was fed into the trained model, and the accuracy was evaluated. The model achieved a mean absolute error of 0.1183, R-square of 0.2172 and mean squared error of 0.0307 on the test set while the trained set has a mean absolute error of 0.0699,R-square of 0.8604 and mean squared error of 0.011, which is a good indication of the model's ability to generalize to new, unseen data. Furthermore, the use of the R-Square metric indicates that the model's predictions are highly correlated with the true values of the quality scores.

To further evaluate the performance of the model, a facial quality assessment deployment was created using Gradio. This allowed users to upload images and receive a predicted quality score from the trained model. The deployment also included a binning function that mapped the predicted scores to categories of quality, making the results more interpretable for non-experts.

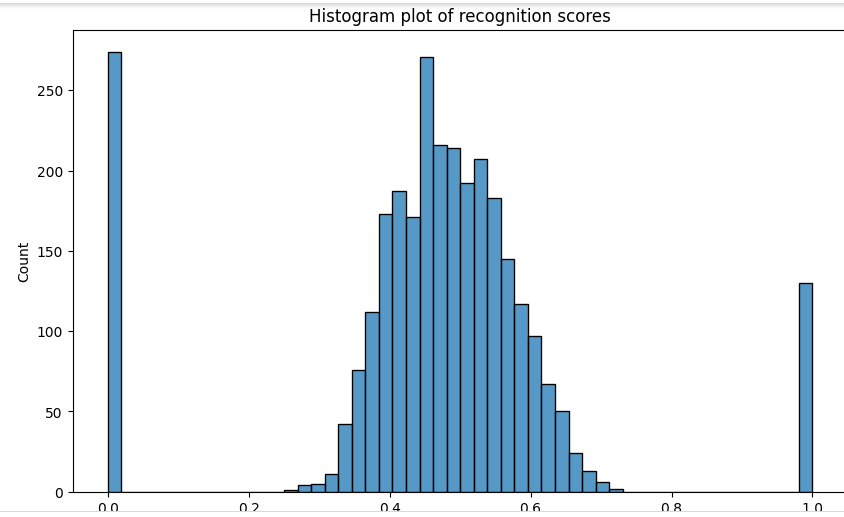


Fig 4.5 recognition score

**4.6 Challenges faced during the implementation**

During the face quality image assessment, we encountered some mistakes and challenges that affected the accuracy of our results. One of the first mistakes we made was using LDA, PCA, and SVM for feature extraction, which did not work well in the end because these methods could not be used to extract face features. As a result, we had to find a more suitable approach to extract face features to improve the accuracy of our results.

Another approach that we attempted was using Scarface as a training dataset and LFW as the test dataset. However, this approach failed because Scarface did not have any variety conditions, while LFW was very wild in terms of lighting conditions, face pose, camera quality, etc. This approach gave us very low metric scores when benchmarked and had to be replaced and strategized upon.

In summary, it is crucial to choose the right approach for feature extraction and ensure that the datasets used are diverse enough to represent various conditions accurately. Failure to do so could lead to low metric scores and inaccurate results, which could have a significant impact on the overall quality of the facial assessment system.

**Conclusion**

In conclusion, the testing phase for facial quality assessment involved evaluating the performance of a trained model on new, unseen data. The model was deployed using Gradio for real-world use. This indicates that the model has the potential to accurately assess the quality of facial images, which could be useful in a variety of applications such as security, entertainment, and healthcare**.**