**Umist\_cropped Analysis Report**

**Project Overview:**

The project involves an in-depth analysis of facial image data using various machine learning and deep learning techniques. The code is organized into multiple files, each serving a specific purpose in the data preprocessing, clustering, and model training pipeline. The main files are named Project.py, Project\_Four.py, and project final.py.

**File name- Project.py**

The Project.py script primarily focuses on exploring and preprocessing a facial image dataset, using tools like data visualization, manipulation, and normalization. Additionally, it employs machine learning techniques such as Principal Component Analysis (PCA) to analyze and reduce dimensionality. The goal is to prepare the data for further model training, specifically for an autoencoder.

1. **Data Loading and Exploration:**

**Objective**:

* Load facial image data from a MAT file and investigates the dimensions and characteristics of the data.

**Findings:**

* The data contains face images (facedat) and corresponding directories (dirnames).
* The script uses Pandas DataFrames to store the loaded data.

1. **Data Preprocessing:**

**Objective**:

* Preprocess the facial images.

**Steps**:

* Converts data to DataFrames and saves as CSV files.
* Normalizes grayscale images and saves the normalized data to a CSV file.
* Applies Gaussian noise to the normalized data.

It imports custom functions for image manipulation (e.g., flipping, rotating, darkening, zooming, shifting) and demonstrates these operations on selected images.

**Insights:**

* Grayscale conversion is essential for uniformity.
* Normalization and noise addition contribute to data robustness.

1. **Image Augmentation:**

**Objective**:

* Generate augmented images to enhance dataset diversity.

**Techniques Applied:**

* Augments the dataset through various image manipulations such as flipping, rotating, darkening, lightening, zooming, and random shifting.

**Benefits:**

* Augmentation increases the variety of data, potentially improving model generalization.

Data augmentation techniques were employed to enhance model generalization by diversifying the dataset. Transformations like darkening, rotating, shifting, flipping, and zooming images increased data variability, enabling the model to recognize patterns across diverse scenarios.

1. **Data Concatenation and Labeling:**

**Objective**:

* Original and augmented images are concatenated along with their labels.

**Steps**:

* Stacking images alongside new axes.
* Labels are created based on the index of the person in the original data.
* The total number of images for each person is calculated.

**Significance:**

* Augmented and original images are unified for model training.

1. **Principal Component Analysis (PCA):**

**Objective**:

* Dimensionality reduction while retaining 95% explained variance.

**Procedure**:

* Applying PCA to the reshaped data.
* Transforming and approximating data.

**Benefits**:

* Reducing dimensionality while preserving essential features.
* Efficient representation for potential machine learning applications.

PCA was pivotal for handling high-dimensional image datasets, reducing complexity while retaining essential features. PCA is more suited for linear dimensionality reduction and might not capture non-linear relationships present in image data, which deep learning models can achieve.

1. **Data Saving:**

**Objective**:

* Save various data arrays into numpy files.

**Saved Arrays:**

* Original and augmented images.
* PCA-transformed data.
* Labels.

**Purpose**:

* Preserving augmented data for future use in model training.

1. **Custom Functions:**

* Several custom functions are defined and imported (plot\_all\_images, print\_image, etc.).
* These functions facilitate image visualization, manipulation, and printing.

1. **Overall Insights:**

The script demonstrates a comprehensive pipeline for facial image data processing, augmentation, and dimensionality reduction.

Augmented data is stored for potential use in training machine learning models.

1. **Conclusion:**

The overall goal was to create a stable, diverse, and manageable dataset for effective machine learning model training. Each step addressed challenges in face image datasets, ensuring readiness for models to generalize effectively to new, unseen data.

**Filename: CNN\_Auto.py**

* The code trains the autoencoder on noisy data to learn denoising patterns and then evaluates its performance by visualizing the reconstructed images. This helps assess how effectively the model can remove noise from images.
* We opted not to proceed with the autoencoder due to its inability to learn in an ideal manner. While the model successfully learned, it exhibited a tendency to introduce noise into images. Moreover, during the denoising process, the reconstructed images appeared blurred even after multiple attempts with various hyperparameter adjustments and alterations to the model architecture. As a result, despite efforts to refine the model's performance, it failed to achieve the desired level of clarity and accuracy in denoising images.

**File name- Project\_Four.py**

The Project\_Four.py script builds upon the previous script (Project.py) by implementing an autoencoder and clustering techniques. It utilizes K-Means, Agglomerative Clustering, and DBSCAN for clustering the data and visualizing the results. The script concludes with the training of a Convolutional Neural Network (CNN) and a ResNet model.

1. **Data Loading and Preprocessing (Autoencoder Model):**

**Objective:**

* Load preprocessed data and labels.
* Compiles and fits the model on noisy and original data.

**Actions:**

* Load saved numpy arrays containing data, PCA-transformed data, and label.
* Reshaping and shuffling data for clustering.

1. **K-Means Clustering:**

**Objective:**

* Apply K-Means clustering to the reshaped data.

**Actions:**

* Use the KMeans class from scikit-learn.
* Print and visualize K-Means results.

K-Means exhibited better performance by effectively diversifying images across different clusters. Considering our dataset, which consists of 20 unique images per person, initializing 20 clusters allowed K-Means to allocate images more appropriately into these clusters. This strategy resulted in a more accurate representation of the unique images within each cluster, aligning well with the dataset's structure and facilitating clearer distinctions between individual images.

**3. Agglomerative Clustering:**

**Objective:**

* Apply Agglomerative Clustering to the reshaped data.

**Actions:**

* Use the **AgglomerativeClustering** class from scikit-learn.
* Print and visualize Agglomerative Clustering results.

Agglomerative clustering forms hierarchical clusters (according to importance/ based on rank), which might not be optimal for image data representation, especially when images don't exhibit a clear hierarchical structure.

**4. DBSCAN Clustering:**

**Objective:**

* Apply DBSCAN clustering to the reshaped data.

**Actions:**

* Use the **DBSCAN** class from scikit-learn.
* Print and visualize DBSCAN clustering results.

The presence of noise in image data can impact the performance of DBSCAN due to its sensitivity to varying densities within the image space. This sensitivity may result in misclassifying noise or an inability to effectively capture subtle clusters within the complex, high-dimensional image spaces. In our case, it has considered all images as a noise and didn’t assign a single image to a cluster.

**5. Data Visualization and Labeling:**

**Objective:**

* Display images with corresponding cluster labels.

**Actions:**

* Visualize data points in a 2D space.
* Display images with respective labels.
* Save the labeled data.

**6. Model Training - CNN:**

**Objective:**

* Define and train a Convolutional Neural Network.

**Actions:**

* Split data for training and validation
* Build a CNN using Keras.
* Compile, train, evaluate, and save the model.
* Visualize training history and predictions.

In our case, we trained CNN and ResNet, two distinct models, after using the clustering algorithms. Surprisingly, CNN fared much better in terms of accuracy than ResNet. ResNet only attained around 9% accuracy, whereas CNN attained almost 92% accuracy. The variation in performance showed that the CNN architecture was far more effective at learning and producing precise predictions than ResNet for our dataset or task.

CNNs can learn and extract information at many levels of abstraction because their convolutional layers are adapted at capturing hierarchical representations of picture data.

**7. Model Training - ResNet:**

**Objective:**

* Fine-tune a pre-trained ResNet50 model for image classification.

**Actions:**

* Load ResNet50 without classifier layers.
* Add custom layers for the new task.
* Compile, train, evaluate, and save the ResNet model.
* Visualize training history and predictions.

ResNets add residual connections to deeper architectures to solve vanishing gradient problems, their greater depth and complexity might require a bigger and more varied dataset for them to train and generalize well.

**8. Visualization of CNN and ResNet Models:**

**Objective:**

* Plot training and validation accuracies and losses for both CNN and ResNet models.

**Actions:**

* Predict and display labels for a subset of validation images.
* Utilize Matplotlib for visualization.
* Display images with predicted and actual labels.

**9. Conclusion:**

The objective now is to apply clustering algorithms (K-Means, Agglomerative Clustering, and DBSCAN) directly to the data without utilizing an autoencoder, followed by training two distinct models: Convolutional Neural Network (CNN) and ResNet.

Among the clustering techniques, K-Means outperformed Agglomerative Clustering and DBSCAN, effectively diversifying images into different clusters by initiating 20 clusters aligned with our dataset's structure.

Overall, CNN tends to perform better with smaller datasets or less varied data because they are shallower and more flexible, which helps them avoid overfitting and efficiently learn representations from the available picture data.

**File name- Project final.py**

The project\_final.py script concludes the project by loading preprocessed data, models, and labels. It visualizes images with respective labels, evaluates two pre-trained models (CNN (Convolutional Neural Network) and ResNet (Residual Neural Network)) on a validation set, and displays predicted and actual labels.

1. **Loading Necessary Data and Models**

**Objective:**

Load preprocessed data, clustering model, and final labeled data.

* **Numpy Loading**: Loading various data and models from Numpy files and joblib.
* **model\_data**: Original image data.
* **model\_data\_pca**: PCA-transformed image data.
* **label**: Labels corresponding to the original data.
* **kmeans\_model**: K-Means clustering model loaded from file.
* **final\_data**: Final normalized and shuffled image data.
* **final\_labels**: Labels corresponding to the final data.

A screen shot of a computer program

Description automatically generated

1. **Displaying Images with Labels**

**Objective:**

* Display images with their respective cluster labels.

**Actions:**

* Loop through the final labeled data.
* Display images along with their cluster labels.
* Show the images whenever the label changes.

1. **Splitting Data for Validation**

**Objective:**

* Prepare the data for validation.

**Actions:**

* Use train\_test\_split to split the final data into training and validation sets.

1. L**oading and predicting with the First Model (CNN Model)**

**Objective:**

* Loads and evaluates a pre-trained CNN model (model\_1.keras) on the validation set.

**Actions:**

* Load the CNN model.
* Displays predicted and actual labels.
* Visualizes images with predicted and actual labels.

1. **Loading and Predicting with the Second Model (ResNet)**

**Objective:**

* Loads and evaluates a pre-trained ResNet model (model\_resnet.keras) on the validation set.

**Actions:**

* Load the ResNet model.
* Repeat grayscale images to create RGB images.
* Displays predicted and actual labels.
* Visualizes images with predicted and actual labels.

1. **Conclusion:**

The file encapsulates the entire process from loading preprocessed data to training, evaluating models, and generating predictions. Utilizing NumPy and joblib for efficient data handling, it loads crucial components including preprocessed image data, the K-Means clustering model, and finalized labeled data. The script displays images alongside corresponding cluster names, aiding comprehension of clustering outcomes from prior project phases. Subsequently, it splits the data into an 80:20 ratio for training and validation. It proceeds by loading pre-existing CNN and ResNet models from earlier files and subsequently prints and displays both actual and predicted labels, facilitating an understanding of model performance and predictions on the dataset.

**Overall Assessment**

The project demonstrates a thorough exploration of facial image data, encompassing data loading, preprocessing, clustering analysis, and model training and evaluation. The inclusion of both traditional clustering techniques and deep learning models provides a comprehensive analysis of the dataset.

**K Means Clustering**

A diagram of a cluster plot

Description automatically generated

**Agglomerative Clustering**

A diagram of a clustering graph

Description automatically generated

**DBSCAN Clustering**

A graph showing a number of dots

Description automatically generated with medium confidence

**Autoencoder**

A collage of images of two people

Description automatically generated

**A collage of images of a person

Description automatically generated**

A close-up of a person's face

Description automatically generated

**CNN (Convolutional Neural Network)**

A collage of different people's faces

Description automatically generated

**Model Accuracy- CNN**

A graph showing the results of a model accuracy

Description automatically generated

**Loss Accuracy – CNN**

Loss Accuracy - CNN


**ResNet (Residual Neural Network)**

A collage of different people's faces

Description automatically generated

**Model Accuracy- ResNet**

**A graph showing the results of a model accuracy

Description automatically generated**

**Loss Accuracy – ResNet**

A graph showing a loss of a train and validation

Description automatically generated