|  |  |
| --- | --- |
|  |  |
| ANALISYS REPORT |  |
| **Autoencoders** |  |
|  | Unsupervised & Reinforcement learning (SEC 003) |
|  | **OMEGA Group**Ashish Gupta |

|  |  |  |
| --- | --- | --- |
|  | Team Omega  * Andy Bandela 301282674 * Juan Arevalo 301323792 * Rasheed Adeyemi 301308878 * Mahfuzur Rahman 301336576 * Anthony Mariadas 301234143 * Mariela Ramos Vila 301324510 * Dinh Hoang Viet Phuong 301123263   **Introduction**:  In this project, we're like detectives organizing a mystery album full of face photos. Using a computer program, we start by gathering all the face images from a special set called the Olivetti faces dataset. It's similar to spreading out all the photos on a table for a good look.  Then, we sort the photos into three piles. One pile is for learning, the second is for checking our work, and the third is for the final test to see if we can match the faces to the names correctly. It's a bit like doing a practice quiz before a big test.  Next, we teach our computer how to match names to faces. We use a method known as "support vector machine." It's like showing the computer different facial features so it can learn who's who.  We also group the faces by similarity, which we call "clustering." We try different ways to decide how similar the faces are and choose the best method.  To make this easier, we use a technique called "principal component analysis," or PCA. This helps us focus on the most important parts of the faces, like only remembering the key things that help you recognize your friends.  Lastly, we create a "neural network," a type of computer brain, to draw a simple version of each face. This helps our computer remember each person even better.  Our goal is to make the computer as good at recognizing faces as we are. The steps we take are common in computer programs that learn from images. |  |

|  |  |  |
| --- | --- | --- |
|  | **In this assignment, you will implement an autoencoder that encodes and decodes data using the Olivetti faces dataset.** |  |
|  | 1. **Import all necessary libraries.**     In our project, we begin by collecting a set of face images from the Olivetti dataset, which provides a varied collection of human facial photos. The dataset is shuffled to ensure randomness, aiding in unbiased analysis. To visualize what we're working with, we display the first 20 faces, labeling each with a number corresponding to the person's identity in the dataset. This step is crucial for understanding the data before any complex processing.       1. **Split the training set, a validation set and test a set using stratified sampling**   Our project takes a careful approach to preparing data by splitting the images into training, validation, and test groups. We use a method called stratified sampling to make sure that there are equal numbers of images for each person across these groups. This way, our computer model gets a fair chance to learn and prove its ability to recognize faces. By doing this, we ensure our model is tested on a balanced and fair selection of images.    The output indicates that we've organized the face images into three distinct sets with the following counts: 240 images for training, 80 for validation, and 80 for testing. This distribution allows us to train the model with a majority of the data, fine-tune it using the validation set, and finally, check its performance with the test set to ensure it accurately recognizes faces across all samples. The balanced allocation helps in achieving a robust model that is well-versed with the varied features of the dataset.  Output:     1. **Determine most suitable covariance type Apply PCA to reduce dimensionality.**   We're simplifying our collection of facial images by using a technique called PCA, which reduces the complexity from 4096 distinctive details down to just 260, without losing the essence of what makes each face unique. It's like distilling a book into a summary while keeping all the key points intact. This streamlined version is easier for our computer models to work with and helps in identifying the most important features that distinguish one person's face from another. By doing so, we ensure our facial recognition system is efficient and focused on what truly matters.       1. **Define an autoencoder with the following architecture:**   autoencoder with the specific architecture  We've created a special kind of neural network called an autoencoder, which is designed to learn a compact representation of our face data. The autoencoder has layers that first compress the data down, finding the essence of the faces, and then try to reconstruct it back to the original. It's like summarizing a detailed story into a few key sentences and then trying to retell it with as much detail as possible. This process helps the autoencoder learn the most important features of the faces without all the extra noise. We add some rules to keep the summary from straying too far from the actual story, which helps in learning a clean, concise representation.     1. **K-Fold Cross-Validation for Autoencoder Model Tuning**   To fine-tune our autoencoder, we're using a technique called K-Fold Cross-Validation. It's like a series of practice tests for our model to make sure it learns well. We divide our data into 5 sets, train our model on 4 of them, and then test it on the 5th one, repeating this process 5 times so each set gets a turn to be the test. This helps us understand how well our model can learn and generalize to new data. After each test, we record how well it did, and in the end, we calculate the average performance. This gives us confidence that our autoencoder will be good at summarizing faces it hasn't seen before.    To get the best out of our autoencoder, we play a matching game called Grid Search. We test different settings like learning rates and sizes of the model's layers to see which combination teaches our autoencoder most effectively. It's like trying different recipes to see which one makes the tastiest cake. We change one ingredient at a time—how quickly the model learns, how strict we are on its learning, and how complex the model is. After trying all the mixes, we found the sweet spot: a learning rate of 0.001, a pinch of regularization at 0.00001, and a balanced layer size of 512, 256, and 128. This recipe gives us an autoencoder that's just right for summarizing faces.      ReLU is chosen for the hidden layers because it introduces non-linearity to the model without affecting the scale of the input, which is particularly beneficial for deep networks. It helps with gradient propagation since the gradient is either 0 (for negative inputs) or 1 (for positive inputs), reducing the risk of vanishing gradients during backpropagation. ReLU is computationally efficient as it involves simpler mathematical operations compared to other functions like sigmoid or tanh.  The sigmoid function is used in the output layer because it squashes the output to be within the range of 0 to 1. This is particularly useful if the autoencoder is meant to reconstruct input data that has been normalized to be within this range.  Mean Squared Error (MSE) Loss Function: MSE is a common choice for regression problems, and an autoencoder can be considered a regression problem since it aims to output a value (the reconstructed input) that is as close as possible to the input value. It is suitable for models where the output activation function is linear or sigmoid, and the range of values is continuous. MSE penalizes larger errors more than smaller ones, which can lead to better model performance when it is important to minimize large reconstruction errors. For autoencoders, especially when dealing with normalized inputs, it ensures that the model is penalized based on the squared difference between each pixel of the input and its reconstruction, emphasizing a precise reconstruction of the input data.    The image shows a comparison between original face images and their reconstructed versions after being processed by our autoencoder. In the top row, we have the original images, which are clear and detailed. The bottom row shows the reconstructed images, which the autoencoder has attempted to recreate from a more compact representation. While the reconstructed images might look a bit blurry or less detailed, they still capture the essential features of the faces such as the eyes, nose, and mouth. This demonstrates that our autoencoder has learned to grasp the core aspects of the data, allowing for a decent reconstruction of the original input even after significant data compression. |  |

|  |  |
| --- | --- |
|  |  |