CHAPTER 3
DATASET AND ALGORITHM

3.1. Dataset

The dataset used in this project is the Olivetti faces dataset acquired by AT&T Laboratories Cambridge between April 1992 and April 1994. This dataset contains 40 classes and have different facial details such as the lighting, facial expressions, and accessories like glasses or no glasses. All the images were using a dark and homogeneous background, with the subject in an upright front position and able to withstand for some lateral movements. The image is quantized to 256 grey levels and stored as an unsigned 8-bit integer. The target of this database is an integer from 0 to 39, indicating the person's identity pictured.

The `sklearn.datasets` package helps the machine learning researchers to fetch large datasets that they use to benchmark algorithms. This module also includes some utilities such as load datasets, fetch reference datasets, and generates artificial data. There are three main functions of `sklearn`: general dataset API, dataset loaders, and dataset fetchers. This project uses `sklearn.datasets` as datasets that download and load larger datasets from the real world. Further detailed information about the dataset is explained later. The `sklearn.datasets` functions return an object called Bunch. The Bunch object is a dictionary that has key as an attribute.

Figure 3.1: SKLearn

As stated above, the dataset is managed as the training and testing data. Because of that, the dataset must be split into two parts. To do this process, `sklearn.model_selection.train_test_split` is used. The train-test split is a technique

for evaluating the performance of a machine learning algorithm. There are some parameters in this quick utility, such as `test_size`, `train_size`, `random_state`, `shuffle`, and `stratify`. The `test_size` parameter represents the proportion of the dataset to be set as test subsets, likewise, with the `train_size` parameter. These parameters should have some variation in the experiment to measure the algorithm's accuracy, and there is no absolute value. It all depends on the dataset and the project's objective found from experimental trials. The `random_state` parameter controls the shuffling applied to the data before splitting. This parameter intends to control the random number generation used so that consistency is ensured. Integer random seeds 42 is used to fill this parameter. The `shuffle` parameter is to determine whether or not to shuffle the data before splitting. The last parameter is `stratify`, which does a split. Our dataset has a various number of data for each class. It is advisable to splits the dataset into the train sets and test sets in the same proportions of data in each class. As shown in the Figure 3.2 below, the splitting proportion is equal for each class. In this way, the train and test sets contain all of the classes on the dataset.

Figure 3.2: Stratify Illustration

Meanwhile, the ORL dataset used for the PCA algorithm is processed from the image data type. This data was obtained from the kaggle by Marlon Tavarez with 400 images that will be manually split into two parts: the training and the test sets. The images data are already labeled with the image id and the person id, separated by the underscore symbol (_).

For example, label 10_1 means the photo with id ten (10) belongs to person number 1. Later, these images are switched into the ndarray shape.

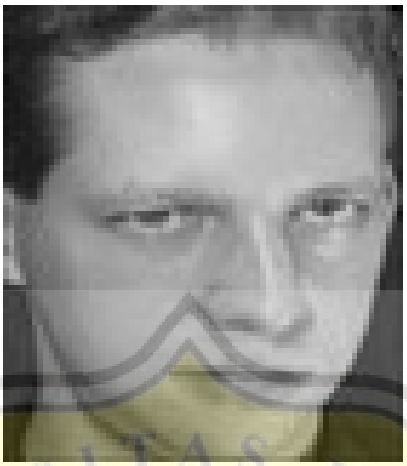

Figure 3.3: ORL
Image 10_1.jpg

3.2. Deep Neural Network

Artificial Intelligence (AI) is a technology developed to 'think' like the works of the human brain. Inside AI, there are other subfields: machine learning, expert systems, and natural language processing, to name a few**.** However, machine learning is the most popular field at this cultural moment [16], and Deep Neural Networks (DNNs) are currently the state-of-the-art ML algorithms [17]. The DNN is a machine learning member with multiple layers between the input and output layers. Each layer of DNN uses the output from the previous layer as input which is very similar to how the human brain transmits information from one neuron to another. Figure 3.4 [18] shows the typical architecture of DNN consists of an input layer, some hidden layers, and the output layer.

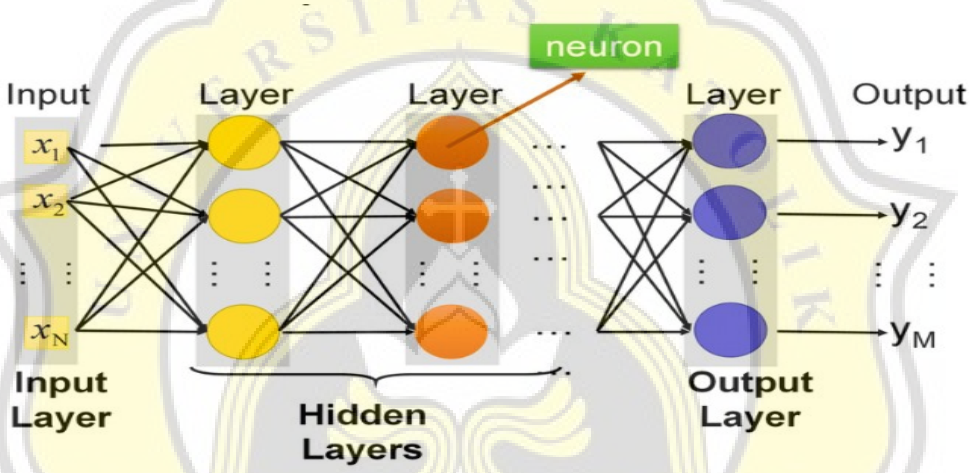

Figure 3.4: Typical Architecture of DNN

The model used in this project is a sequential model. The sequential model is suitable for a plain stack of layers with only one input tensor and one output tensor. This model allows us to build a model layer by layer. Each layer has weights that correspond to the layer that follows it. Those multiple layers are non-linear processing units, often called activation functions, used for feature extraction and transformation. There are some activation functions such as Rectified Linear Activation (ReLU), Logistic (Sigmoid), and Hyperbolic Tangent (Tanh). Meanwhile, this project uses the ReLU and softmax activation function. More details are explained below.

ReLU is an activation function that has a biological and mathematical base. It works by thresholding values at 0, i.e., as shown in Figure 3.5 ReLU activation function [19], when the output value is less than 0, convert the value to 0. Conversely, it outputs a linear function when the output value is more than 0, unlike the tanh and sigmoid activation functions, which

approximate a zero output, e.g., a value very close to zero but not an actual zero value. The ReLU also did not require exponential calculation, so the computations are cheaper [20]. Emphasized by Krizhevsky et al. as cited in Wu and Deng, 2016, the ReLU function's advantages include faster training speed, decreased saturation problems, a smaller number of epochs, and usually fewer samples.

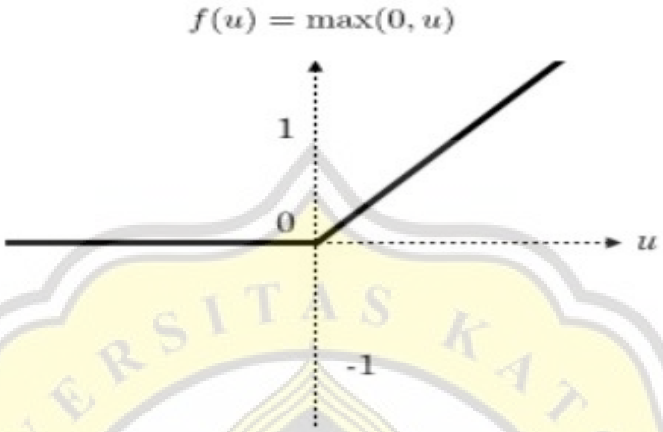

Figure 3.5: ReLU Activation Function

The softmax function is used to calculates the probability of each target class from all probable target classes. It is used as the activation function of the output layer of a neural network model that predicts a polynomial probability distribution. The probability is used to determine the target class for the specified inputs. These probability values are ranged from 0 to 1, and if we sum all of the probabilities, it will equal to 1. Later, the calculated probabilities will be used for determining the target class of the inputs. If we use the softmax function for the multi-classification model, it will returns the probabilities of each class. Here in Figure 3.6 [21] show a graphical representation of the softmax activation function.

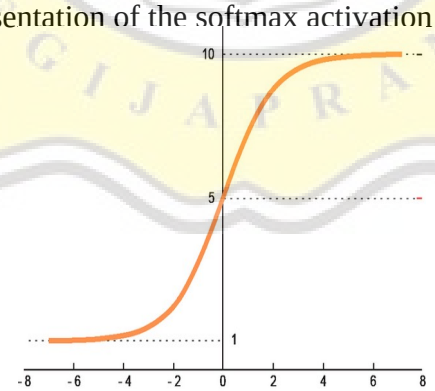

Figure 3.6: Graphic Representation of Softmax Function

Figure 3.7 below shows the softmax function formula that calculates the ratio of the exponential of the input value and the sum of the exponential values.

$$\sigma(\vec{z})i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 3.7: Softmax Formula

For example, we have an array with three values $\begin{bmatrix}2\\9\\7\end{bmatrix}$. These values could be the output of the machine learning model, but with the softmax function, we convert those values into a probability distribution.

$e^{z1} = e^2 = 7.39$

$e^{z2} = e^9 = 8130.08$

$e^{z3} = e^7 = 1096.63$

Then, sum all three exponentials to obtain the normalization term.

$\sum_{j=1}^{K} e^{zj} = e^{z1} + e^{z2} + e^{z3} = 7.39 + 8130.08 + 1096.63 = 9234.1$

From the value, we can see that z2 dominates the normalization term. The next step is to divide each of the array values to z2.

$\sigma(\vec{z})_1 = \frac{7.39}{9234.1} = 0.0008$

$\sigma(\vec{z})_2 = \frac{8130.08}{9234.1} = 0.8804$

$\sigma(\vec{z})_1 = \frac{1096.63}{9234.1} = 0.1188$

Three output values lie between 0 and 1, and also they sum to 1.

3.3. PCA

In 1991, *Turk and Pentland s*uggested an approach to face recognition that uses dimensionality reduction and linear algebra concepts. This approach is commonly used as it is computationally less expensive and easy to implement. Principal component analysis (PCA) is a technique used to reduce the dimensions of a dataset. This method works by transforming a large set of variables into smaller ones containing most of the information while minimizing information loss. Smaller datasets make data analyzing easier and faster for machine learning. PCA is one of the oldest and most widely used algorithms. The PCA can be divided into five main steps.

1. Normalize the dataset

   Subtract the mean of each variable from the dataset to normalizing them. This step means that we are removing the "common features" we got from the mean of the dataset.

2. Compute the covariance matrix

   To know if there is a correlation in the input dataset's variables, compute the covariance matrix of normalized data. It's the value of covariance that matters. If the value is positive, then the two variables are correlated. If negative, then it is uncorrelated

3. Compute the eigenvectors and eigenvalues from the covariance matrix

   From the covariance matrix, compute the eigenvectors and eigenvalues. Eigenvector is a nonzero vector that does not change direction when a transformation is applied. Meanwhile, the eigenvalue is a scalar associated with eigenvectors. Let us assume A is an "n × n" matrix, λ is an eigenvalue of matrix A, and x (a nonzero vector) is called an eigenvector if it satisfies $Ax = \lambda x$ expression.

4. Sort the eigenvalues in descending order and select a subset from the rearranged Eigenvalues matrix

Each column in the eigenvector matrix corresponds to a principal component. The principal components are new variables constructed as a linear combination that uncorrelated, and most of the information within the initial variables is squeezed or compressed. Arranging the eigenvalues in descending order of their eigenvalue will also set the principal component by their variability. Then, select a subset or from the arranged eigenvalue that captures the highest variability.

5. Transform the data

Compute a dot product between the transpose of the eigenvector subset and the transpose of the normalized data. The outcome of this step is data that is reduced to lower dimensions.