Machine Learning Assignment 4

Abhijay Singh
Roll Number: 2021226

Section A (Theoretical)

(a) Suppose you are given an input image with the dimensions of 15 × 15 × 4, where 4 denotes the number of channels. The same is passed to a CNN shown below:- The kernels are of shape h × w × I × O, representing height, width, number of input channels, and number of output channels, respectively.

(a) What is the output image size?

For the first Conv2D layer:

- Input image dimensions: $15 \times 15 \times 4$ (height × width × channels)
- Kernel size: 5×5
- Padding: 1
- Stride: 1

The formula for the output size of one dimension for convolutional layers is:

$$O = \frac{W - K + 2P}{S} + 1$$

where O is the output height/width, W is the input height/width, K is the filter size, P is the padding, and S is the stride.

For the first Conv2D layer:

$$O_{\text{conv1}} = \frac{15 - 5 + 2 \cdot 1}{1} + 1 = 12$$

So the output dimension after the first Conv2D layer is 12×12 per channel.

For the Maxpooling layer with a stride of 2, the size will reduce by half:

$$O_{\text{pool}} = \frac{12}{2} = 6$$

So the output dimension after the Maxpooling layer is 6×6 per channel.

For the second Conv2D layer:

- Kernel size: 3×3
- Padding: 2
- Stride: 2

$$O_{\text{conv2}} = \frac{6 - 3 + 2 \cdot 2}{2} + 1 = 4$$

So the final output dimension is 4×4 per channel.

(b) What is the significance of pooling in CNN?

The significance of pooling in CNNs is manifold:

- Pooling reduces the spatial dimensions (width and height) of the input volume for the next convolutional layer, which decreases the computational load and the number of parameters.
- It helps to make the detection of features somewhat invariant to scale and orientation changes.
- Pooling also contributes to the reduction of the sensitivity of the output to noise and small variations.
- Max pooling, a common pooling function, assists in extracting the most significant features like edges, which are crucial for feature learning in deep networks.

(c) Compute the total number of learnable parameters for the above CNN architecture (ignore bias)

For the first Conv2D layer:

- Kernel size: 5×5
- Input channels: 4
- Number of kernels: 1 (assuming the number of output channels is 1)

$$\text{Parameters}_{\text{conv1}} = 5 \times 5 \times 4 \times 1 = 100$$

For the second Conv2D layer:

- Kernel size: 3×3
- Input channels: 4 (assuming the output channels from the first Conv2D layer are 4)
- Number of kernels: 1 (assuming the number of output channels is 1)

$$\text{Parameters}_{\text{conv2}} = 3 \times 3 \times 4 \times 1 = 36$$

The total number of learnable parameters (ignoring bias terms) is the sum from both layers:

$$\text{Total Parameters} = \text{Parameters}_{\text{conv1}} + \text{Parameters}_{\text{conv2}}$$

$$\text{Total Parameters} = 100 + 36 = 136$$

Thus, the CNN architecture has a total of 136 learnable parameters (ignoring bias).

(b) Let a configuration of the k means algorithm correspond to the k-way partition (on the set of instances to be clustered) generated by the clustering at the end of each iteration. Is it possible for the k-means algorithm to revisit a configuration? Justify how your answer proves that the k means algorithm converges in a finite number of steps.

The k-means algorithm is an iterative process used for clustering a dataset into k distinct, non-overlapping subgroups, or clusters. Each iteration of the k-means algorithm consists of two steps: assignment of data points to the nearest centroid and recalculation of centroids as the mean of the assigned points. The algorithm converges when there is no change in the assignment of data points to centroids.

Revisitation of Configurations in k-means

A configuration in the k-means algorithm is defined by the assignment of data points to specific clusters and the location of centroids. The possibility of revisiting a configuration during the algorithm's execution can be assessed as follows:

- The dataset is finite, and hence there is a finite number of possible configurations.
- The Within-Cluster Sum of Squares (WCSS) is a monotonic non-increasing function in the k-means algorithm. Each iteration aims to minimize the WCSS, meaning it cannot increase in subsequent iterations.
- The algorithm converges when successive iterations no longer yield a different assignment of data points, implying that the WCSS has reached its minimum for the given configuration.

Given these properties, the k-means algorithm cannot revisit a configuration. Revisiting a configuration would imply a cyclic pattern which contradicts the monotonic behavior of the WCSS function.

Proof of Convergence

The convergence of the k-means algorithm can be summarized as follows:

- The algorithm operates in a finite state space, defined by the number of distinct ways to partition the dataset into k clusters.
- Each iteration either maintains or reduces the WCSS, never increasing it.
- Consequently, the algorithm will reach a point where the WCSS cannot be reduced any further, and the configuration of clusters and centroids will remain unchanged.

Therefore, the k-means algorithm is guaranteed to converge to a local minimum of the WCSS in a finite number of steps. This property assures that the k-means algorithm is suitable for practical clustering applications, where an exact global minimum is not required, but an approximate solution found in a finite amount of time is sufficient.

(c) Can a neural network be used to model K-Nearest Neighbours algorithms? If yes, state the neural network structure (how many hidden layers are required) and the activation function(s) used at the internal and output nodes. If not, describe why not.

K-Nearest Neighbors (K-NN) is a non-parametric, instance-based learning algorithm commonly used for classification and regression tasks. It operates by identifying the k closest data points to a given input and making predictions based on the majority label or average outcome of these neighbors. Unlike K-NN, neural networks are a set of parametric models that learn a generalizable mapping from inputs to outputs, optimizing the parameters based on a loss function.

Challenges in Modeling K-NN with Neural Networks

Given the operational mechanisms of K-NN, using a neural network to model it presents several challenges:

- Neural networks do not retain the entire training dataset in memory, a requirement for K-NN to function.
- The non-parametric nature of K-NN contrasts with the parametric learning approach used by neural networks.
- Neural networks learn from errors and update their parameters, whereas K-NN is a lazy learner that does not adapt in this manner.
- K-NN can have highly irregular decision boundaries based on the local distribution of data, which does not align well with the smoother decision boundaries typically learned by neural networks.

Conclusion

While specialized neural network architectures, such as Siamese Networks or networks with a differentiable k-nearest neighbor layer, can be designed to simulate aspects of K-NN, standard feedforward neural networks are not suitable for modeling the K-NN algorithm. The intrinsic differences in learning processes and data retention requirements prevent standard neural networks from replicating the behavior of K-NN algorithms effectively.

(d) Explain the difference between linear and non-linear kernel or filters in CNN.

Difference between Linear and Non-linear Filters in CNN

Linear Filters	Non-linear Filters
Used in the convolution operation	Applied after convolution
Output is a linear combination of inputs	Output is transformed by a non-linear function
Preserves proportionality and additivity	Introduces non-linearity to the model
Typically used for edge detection, blurring, etc.	Used to introduce activation functions like ReLU
Cannot model complex patterns alone	Can capture complex patterns in data

Table 1: Comparison of Linear and Non-linear Filters in CNNs

Section B : Implementation of Convolutional and Pooling Functions

Convolution Function

The convolution function is implemented to perform a forward pass with zero-padding and specified stride. It computes the output of applying filters to the input data. The backward pass computes the gradients with respect to the input data and the filter weights.

Forward Pass

The forward pass takes an input volume (data), a set of filters (weights), and biases. It applies the filters to the input using the convolution operation, adding the bias at the end. Zero-padding is applied to the input to preserve the spatial dimensions, and the stride determines the step size for the sliding window of the filter over the input.

Backward Pass

The backward pass receives the gradient of the loss function with respect to the output of the forward pass. It calculates the gradients with respect to the input volume and the filter weights to enable the gradient descent optimization.

Pooling Function

The pooling function is designed to perform downsampling operations to reduce the spatial dimensions of the input volume. It computes a mask to identify the max or average values during the forward pass, and the backward pass uses this mask to distribute gradients.

Forward Pass

During the forward pass, the pooling function slides a window over the input volume and applies a downsampling operation such as max or average pooling within that window.

Backward Pass

In the backward pass, the pooling function backpropagates the gradients to the locations identified by the mask created during the forward pass.

Input/Output Example

An example input is processed through the convolution function to demonstrate its operation. The output shows the transformed input after applying the convolution operation with the specified filters and biases.

Operation	Input	Output
Convolution Forward	$10 \times 10 \times 3$ array	$8 \times 8 \times 3$ array with values such as $[6.23, 5.97, 8.27]$
Convolution Backward	$8 \times 8 \times 3$ array	$10 \times 10 \times 3$ array with values such as $[0.91, 0.69, 1.04]$
Pooling Forward	$8 \times 8 \times 3$ array	$4 \times 4 \times 3$ array with values such as $[0.70, 0.64, 0.99]$
Pooling Backward	$4 \times 4 \times 3$ array	$8 \times 8 \times 3$ array with values such as $[0, 0.011, 0.176]$

Table 2: Examples of inputs and outputs for the implemented convolution and pooling functions.

(e) Code Implementation

The Python code for the convolution and pooling operations, including both forward and backward passes, is detailed in the python notebook file. The code is extensively commented to explain each step and the purpose of the operations.

Section C: Clustering Analysis using PCA and K-Means

Exploratory Data Analysis (EDA)

Upon initial examination of the dataset, several key observations were made:

- No missing values were found in the dataset, eliminating the need for any imputation techniques.
- Summary statistics indicated a wide range of values for each feature, suggesting significant variability among the countries. Notably:
  - Child mortality (`child_mort`) varied from 2.6 to 208 deaths per 1000 live births.
  - GDP per capita (`gdpp`) spanned a broad range from $231 to $105,000 international dollars.
- Given the extensive range of values, standardization of the dataset was required. This step is crucial before applying algorithms like PCA and K-Means clustering, which are sensitive to the scale of data.

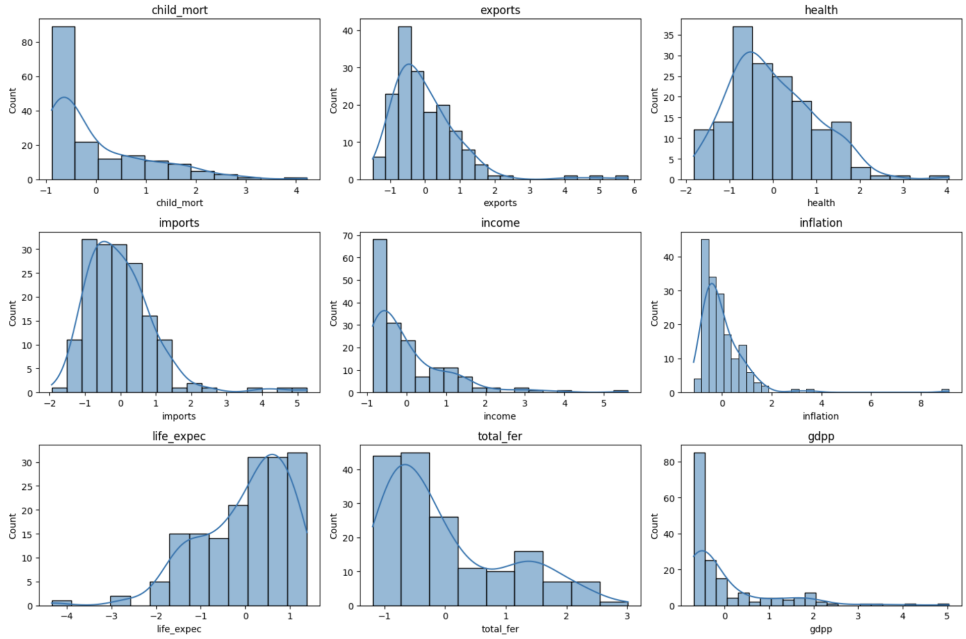

Figure 1: Histograms for the standardized features of the dataset.

- The histograms of the standardized features revealed that most features are right-skewed. This skewness implies the existence of a number of countries with high values for these socio-economic and health indicators.

Principal Component Analysis (PCA)

The PCA was implemented to reduce the dimensionality of the data while retaining significant variance. The analysis suggests the following:

- Six principal components were chosen as optimal to retain over 95% of the data variance.
- The explained variance by the principal components is as follows:
  - PC1 accounts for approximately 45.95% of the variance.
  - PC2 contributes an additional 17.18%, cumulatively explaining about 63.13%.
  - PC3 raises the cumulative explained variance to 76.14%.
  - PC4 further increases this to 87.19%.
  - PC5 adds to a cumulative 94.53%.
  - PC6 finally surpasses the 95% threshold, reaching a cumulative explained variance of approximately 97.02%.

PCA Scatter Plot

The scatter plot for the first two principal components is shown below, useful for visual identification of potential clusters.

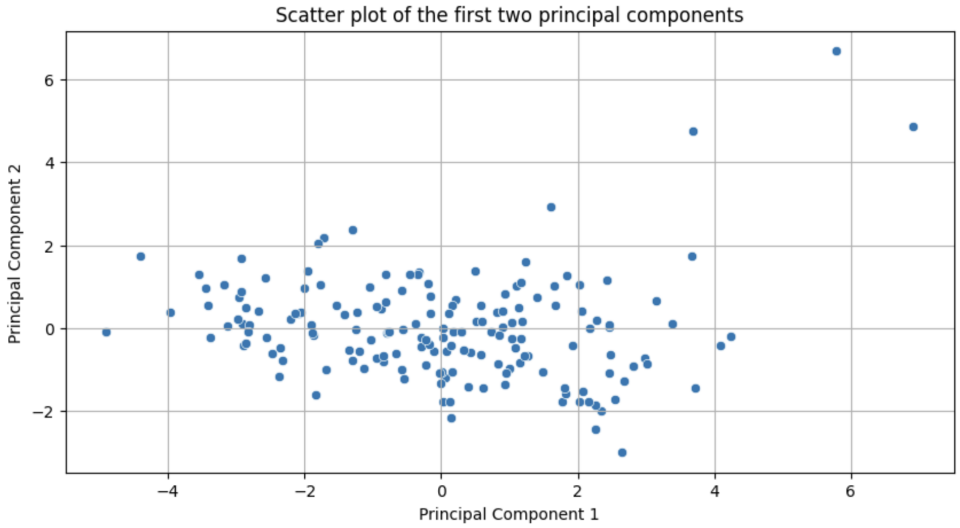

Figure 2: Scatter plot of the first two principal components.

PCA Correlation Heatmap

A heatmap of the correlation matrix for the PCA components, which shows very low correlations between components, indicating the effectiveness of PCA in removing multicollinearity.

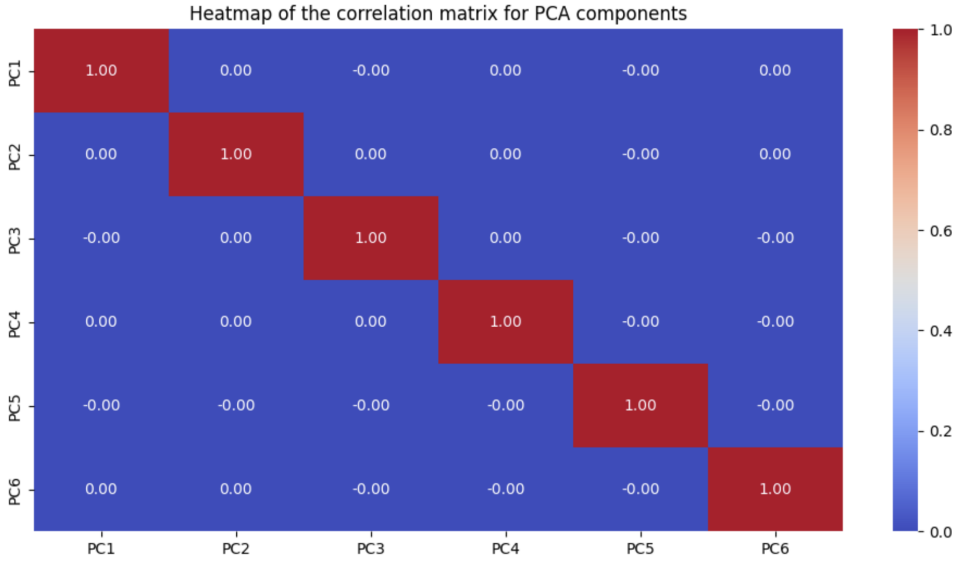

Figure 3: Heatmap of the correlation matrix for PCA components.

K-Means Clustering

The K-Means clustering algorithm was applied to the PCA-transformed data to identify groups of countries with similar socio-economic and health characteristics.

Determining the Optimal Number of Clusters

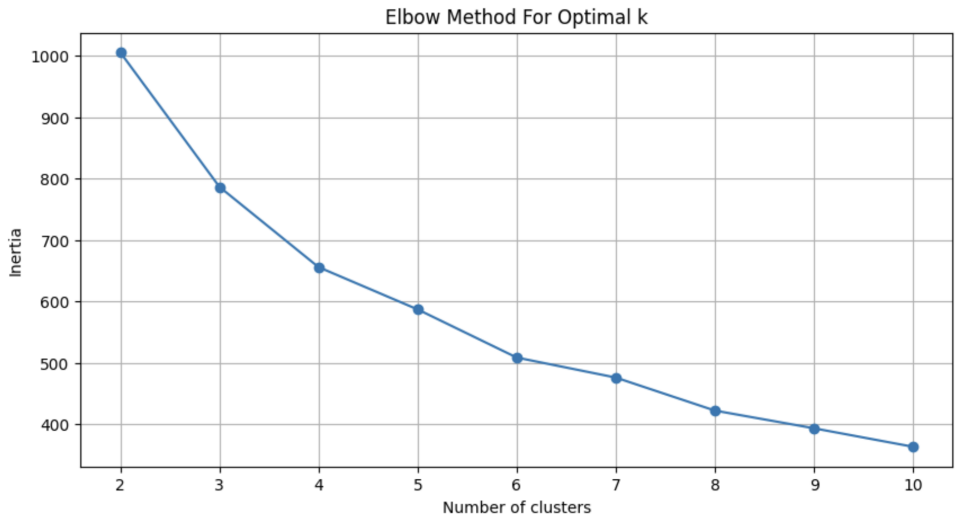

Figure 4: Silhouette Score indicating the optimal number of clusters.

The elbow method graph shows a bend around 3 to 5 clusters, suggesting that the within-cluster sum of squares (inertia) does not significantly decrease after 5 clusters.

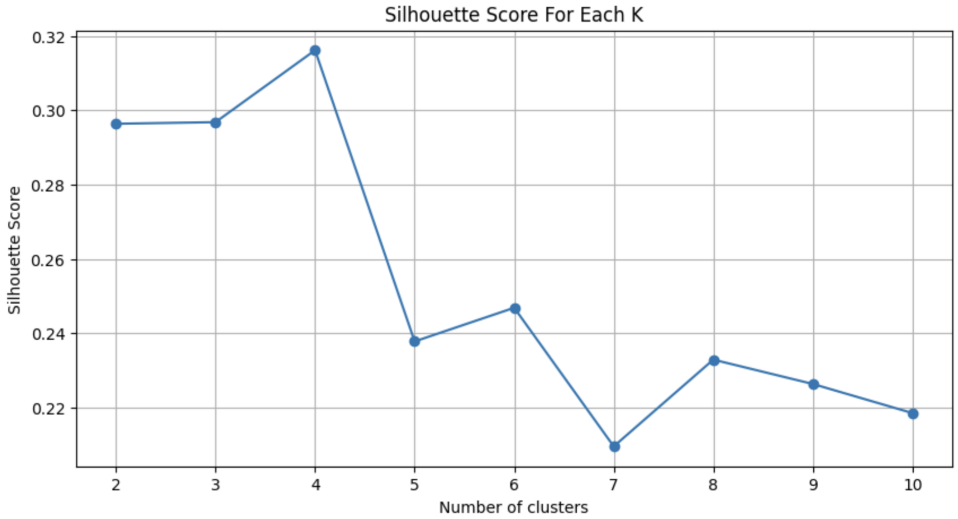

Figure 5: Silhouette Score indicating the optimal number of clusters.

The silhouette scores are used to measure how similar an object is to its own cluster compared to other clusters. The silhouette score plot shows that the highest silhouette score is obtained when the number of clusters is 4, indicating the best-defined clusters.

The optimal number of clusters was determined using both the Elbow Method and the Silhouette Score. The Silhouette Score plot suggested that the highest score and thus the best-defined clusters were obtained with four clusters.

K-Means Clusters Scatter Plot

The scatter plot with the optimal number of clusters is shown below, with each cluster represented by a different color.

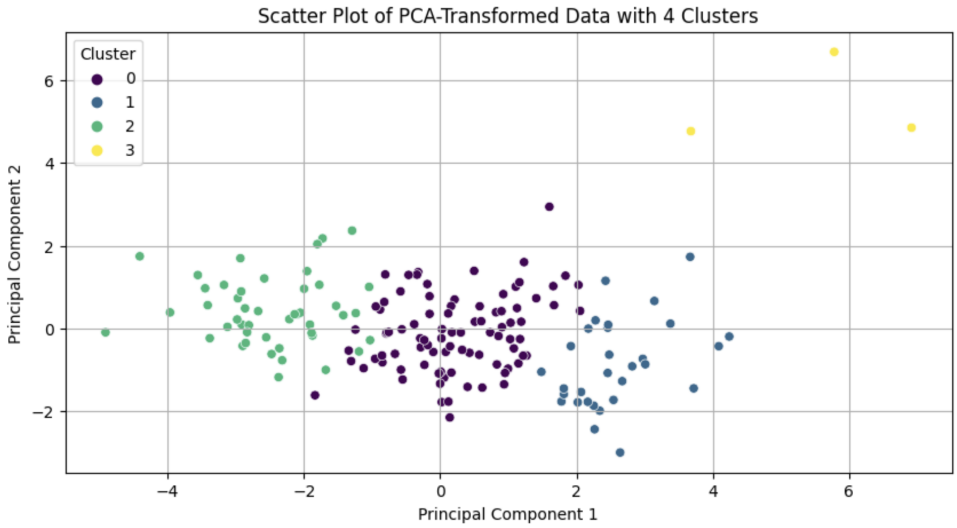

Figure 6: 2D scatter plot of the PCA-transformed data with 4 clusters.

Cluster Characteristics Analysis

The analysis of the K-Means clusters with four clusters revealed the following insights:

- **Cluster 0**: Likely developed countries with strong economies and advanced health systems, characterized by very low child mortality, high trade levels, high income, low inflation, high life expectancy, low fertility, and high GDP per capita.
- **Cluster 1**: Likely developing countries with economic and health challenges, indicated by high child mortality, lower trade levels, lower income, higher inflation, low life expectancy, high fertility, and low GDP per capita.
- **Cluster 2**: Emerging economies with improving health and economic indicators, with moderate values across the features.
- **Cluster 3**: Very developed countries with excellent health systems and economies, with low child mortality, high health spending, high income, low inflation, high life expectancy, and very high GDP per capita.
- **Cluster 4**: Countries possibly in economic crisis, with the highest child mortality, very low trade, low income, extremely high inflation, low life expectancy, high fertility, and low GDP per capita.

These clusters provide a framework to understand the socio-economic and health-related similarities and differences among groups of countries, which can be instrumental in informing policy decisions and international development efforts.