CSE343/ECE343: Machine Learning
Assignment-4 CNN, PCA, K-means clustering
Max Marks: 25 (Programming: 15, Theory: 10) Due Date: 26/11/2023, 11:59 PM Instructions

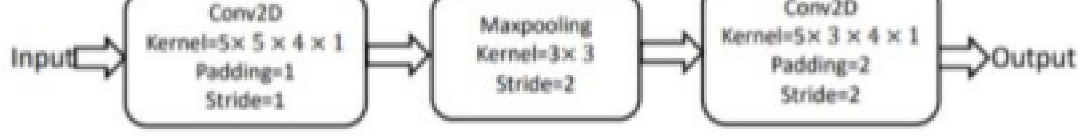
- Keep collaborations at high-level discussions. Copying/Plagiarism will be dealt with strictly.
- Late submission penalty: As per course policy.
- Your submission should be a single zip file 2020xxx_HW1.zip (Where *2020xxx* is your roll number). Include all the files (code and report with theory questions) arranged with proper names. A single .pdf report explaining your codes with results, relevant graphs, visualization, and solutions to theory questions should be there. The structure of submission should follow:

  2020xxx_HW1
  |− code_rollno.py/.ipynb
  |− report_rollno.pdf
  |− (All other files for submission)
- Anything not in the report will not be graded.
- Remember to turn it in after uploading it on Google Classroom. No excuses or issues will be taken regarding this after the deadline.
- Start the assignment early. Resolve all your doubts from TAs during their office hours at least two days before the deadline.
- Your code should be neat and well-commented.
- You have to do either Section B or C.
- Section A is mandatory.

1. (10 points) Section A (Theoretical)

   (a) (5 marks) 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? [2]
   (b) What is the significance of pooling in CNN? [1]



   (c) Compute the total number of learnable parameters for the above CNN architecture (ignore bias) [2]

(b) (2 marks) 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.

(c) (2 marks) 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.

(d) (1 mark) Explain the difference between linear and non-linear kernels or filters in CNN.

2. (15 points) Section B (Scratch Implementation)

(a) For this problem, you have to implement a Convolutional neural network from scratch. To implement the CNN from scratch, you have to implement the following required functions by yourself: You are allowed to use NumPy, matplotlib, and random libraries.

(a) (6 marks) Convolution Function: Develop forward and backward passes, encompassing windowing and zero-padding with parameters [3,3,3,3].

(b) (6 marks) Pooling Functions: Design a pooling process involving mask creation, value distribution, and both forward and backward pooling with [2,3,3,2].

For each function, use appropriate inputs for testing the functions and report their outputs. For all the functions, you need to include their working and use in the report along with an input/output example. All the function code must be clean and well-commented (3 marks).

OR

3. (15 points) Section C (Algorithm implementation using packages)

Clustering Analysis using PCA and K-Means
Dataset: Country Dataset
The task is to perform a clustering analysis to categorize countries based on socio-economic and health factors. You are provided with a dataset containing socio-economic and health indicators for various countries. The columns include features such as child mortality, exports, health spending, imports, income, inflation, life expectancy, total fertility rate, and GDP per capita.

(a) (3 marks) VLoad the dataset and perform exploratory data analysis (EDA) to gain insights into the data’s structure and distribution. Preprocess the data by standardizing the features using a suitable scaling technique.

(b) (6 marks) Implement PCA to reduce the dimensionality of the data while retaining a significant portion of the variance. Identify the optimal number of principal components based on the explained variance ratio. Create scatter plots and heat maps for visualization.

(c) (6 marks) Apply the K-Means clustering algorithm to the PCA-transformed data. Determine the optimal number of clusters using techniques like both the elbow method and silhouette score. Plot the clusters on a 2D scatter plot. Perform an analysis of the clusters to identify the characteristics of each cluster.

SOLUTION A:

A)-

DL:
Pg.: Delta

(d) $P(F=7 \mid \text{Good mood}) = 0.7$

$P(F=7 \mid \text{Bad mood}) = 0.45$

$P(F=7) = P(F=7 \mid G) \times P(G) + P(F=7 \mid B) \times P(B)$

$= 0.7 \times 0.6 + 0.45 \times 0.4 = 0.6$

$$P(G \mid F=7) = \frac{P(F=7 \mid G) \times P(G)}{P(F=7)}$$

$$= \frac{0.7 \times 0.6}{0.6} = 0.7$$

$$P(B \mid F=7) = \frac{P(F=7 \mid B) \times P(B)}{P(F=7)}$$

$$= \frac{0.45 \times 0.4}{0.6} = 0.3$$

hence most likely that Rahul is in good with 70 + probability when he had 7 hours to sleep.

(b) Channel impulse response $h(\tau, t)$,

it can be obtained by taking inverse fourier transform of the transfer function.

where $H(f,t) = \alpha(t) e^{j\phi(t)}$

$$h(\tau, t) = \mathcal{F}^{-1}\{H(f,t)\}$$

$$= \int_{-\infty}^{\infty} H(f,t) e^{j2\pi f t} . df$$

(c) received signal $y(t)$ is obtained by convolving the transmitted signal $x(t)$ with channel impulse response, $h(\tau, t)$

$$\therefore \quad y(t) = x(t) \underset{\hookrightarrow \text{ convolution}}{*} h(\tau, t)$$

In absence of noise,

mean delay,

$$\bar{\tau} = \int_{-\infty}^{\infty} \tau \left| h(\tau, t) \right|^2 . d\tau$$

delay spread,

$$\sigma_\tau^2 = \sqrt{\int_{-\infty}^{\infty} (\tau - \bar{\tau})^2 \left| h(\tau, t) \right|^2 . d\tau}$$

whe σ_τ is delay spread

B)-

The k-means algorithm converges in a finite number of steps due to the limited nature of data and centroid spaces. It minimizes an objective function, typically the sum of squared distances between instances and cluster centroids. As this function is bounded and instances are finite, the algorithm must converge to a configuration where further improvement is impossible. The convergence ensures stability in cluster assignments over iterations.

K-means converge to a stable state in a finite number of steps, but revisiting configurations is possible. Its convergence is guaranteed by the limited space of instances and centroids, while the revisitation highlights the algorithm's sensitivity to initial conditions and local minima. Multiple runs with diverse initializations help obtain a more robust final solution.

C)-

Yes, we can model K-Nearest Neighbours (KNN) algorithms using Neural Networks one way to do this is by creating a neural network that learns the underlying patterns in the data to perform a similar task as KNN.

An easy way to create a neural network that emulates the KNN algorithm is to use a simple architecture with one hidden layer. The input layer should have neurons representing the features of your data, and the output layer should have neurons expressing the classes or labels.

Input Layer: Neurons representing the features of your data.

Hidden Layer: A single hidden layer with a moderate number of neurons. This layer helps the network learn the relationships between features.

Output Layer: Neurons representing the classes or labels of our data. The number of neurons in this layer depends on the number of classes you have.

For the activation function, we can use the rectified linear unit (ReLU) or the hyperbolic tangent (tanh) for the hidden layer.

The choice of activation function for the output layer :

Binary Classification: Use a sigmoid activation function in the output layer.

Multiclass Classification: Use a softmax activation function in the output layer.

D)-

Linear kernels in CNNs perform linear transformations on input data, generating outputs that are simple linear combinations of input values and kernel weights. This linear operation cannot capture complex, non-linear patterns in data. On the contrary, non-linear kernels introduce non-linear activation functions, such as ReLU, after the convolution operation. This enables the network to model and understand intricate relationships in the data, enhancing its capacity to learn from more complex patterns.

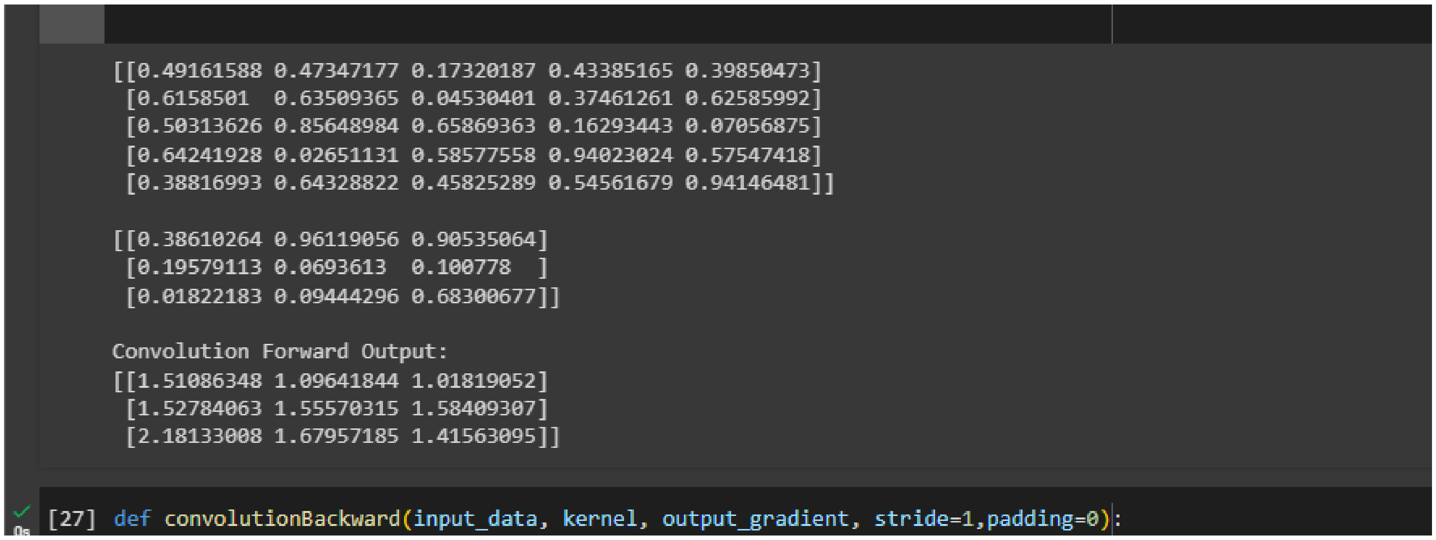
The non-linearities introduced by activation functions are essential for tasks like image recognition, where the relationships between pixels are often non-linear. By allowing the network to learn and represent non-linear patterns, non-linear kernels contribute to the CNN's capability to extract and understand intricate features, improving its performance on tasks requiring nuanced pattern recognition.

SECTION B

CODE::

```python
def convolutionForward(input_data, kernel, stride=1, padding=0):
    input_height, input_width = input_data.shape
    kernel_height, kernel_width = kernel.shape

    # Calculate output dimensions
    output_height = (input_height - kernel_height + 2 * padding) // stride + 1
    output_width = (input_width - kernel_width + 2 * padding) // stride + 1

    # Apply zero-padding
    padded_input = np.pad(input_data, padding, mode='constant')

    # Initialize output
    output = np.zeros((output_height, output_width))

    # Performing  convolution
    for i in range(0, output_height, stride):
        for j in range(0, output_width, stride):
```

```
                        output[i, j] = np.sum(padded_input[i:i+kernel_height,
j:j+kernel_width] * kernel)

    return output




input_data = np.random.rand(5, 5)  #  input data
print(input_data)
kernel = np.random.rand(3, 3)  # input kernel
print()
print(kernel)
output = convolutionForward(input_data, kernel)
print()
print("Convolution Forward Output:")
print(output)
```



```
[[0.49161588 0.47347177 0.17320187 0.43385165 0.39850473]
 [0.6158501  0.63509365 0.04530401 0.37461261 0.62585992]
 [0.50313626 0.85648984 0.65869363 0.16293443 0.07056875]
 [0.64241928 0.02651131 0.58577558 0.94023024 0.57547418]
 [0.38816993 0.64328822 0.45825289 0.54561679 0.94146481]]

[[0.38610264 0.96119056 0.90535064]
 [0.19579113 0.0693613  0.100778  ]
 [0.01822183 0.09444296 0.68300677]]

Convolution Forward Output:
[[1.51086348 1.09641844 1.01819052]
 [1.52784063 1.55570315 1.58409307]
 [2.18133008 1.67957185 1.41563095]]
```

```
[27] def convolutionBackward(input_data, kernel, output_gradient, stride=1,padding=0):
```

The function takes input data (input_data), a convolutional kernel (kernel), and optional parameters for stride and padding.

It calculates the output dimensions based on input size, kernel size, stride, and padding.

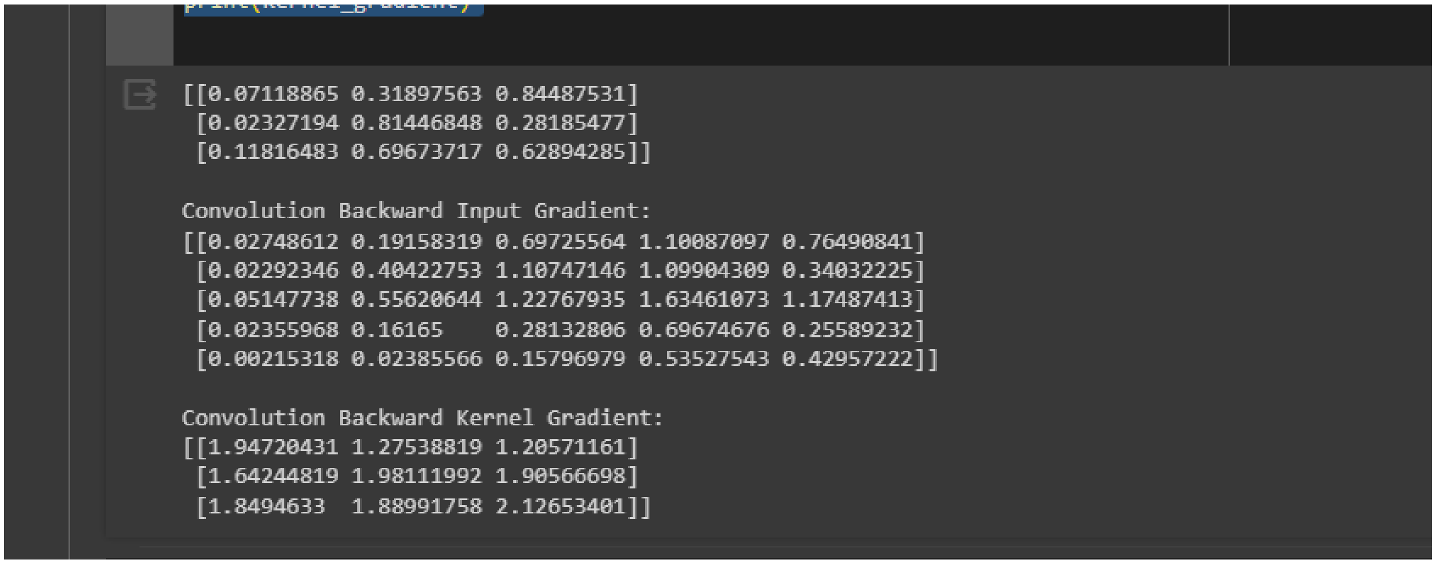
It applies zero-padding to the input data.

It initializes an output matrix.

It performs the convolution operation by sliding the kernel over the input data, computing the element-wise product, and summing the results.

```
def        convolutionBackward(input_data,        kernel,        output_gradient,
stride=1,padding=0):
    input_height, input_width = input_data.shape
    kernel_height, kernel_width = kernel.shape

    # Calculate output dimensions
    output_height = (input_height - kernel_height + 2 * padding) // stride + 1
    output_width = (input_width - kernel_width + 2 * padding) // stride + 1

    # Apply zero-padding
    padded_input = np.pad(input_data, padding, mode='constant')

    # Initialize gradients
    input_gradient = np.zeros_like(input_data)
    kernel_gradient = np.zeros_like(kernel)

    # Performing backward pass
    for i in range(0, output_height, stride):
        for j in range(0, output_width, stride):
                input_gradient[i:i+kernel_height,  j:j+kernel_width]  +=  kernel  *
output_gradient[i, j]
            kernel_gradient += padded_input[i:i+kernel_height, j:j+kernel_width]
* output_gradient[i, j]

    return input_gradient, kernel_gradient

# Example usage
output_gradient = np.random.rand(3, 3)  # Example gradient from the next layer
```

```
input_gradient,   kernel_gradient   =   convolutionBackward(input_data,   kernel,
output_gradient)


print(output_gradient)

print("\nConvolution Backward Input Gradient:")

print(input_gradient)

print("\nConvolution Backward Kernel Gradient:")

print(kernel_gradient)
```



```
[[0.07118865 0.31897563 0.84487531]
 [0.02327194 0.81446848 0.28185477]
 [0.11816483 0.69673717 0.62894285]]

Convolution Backward Input Gradient:
[[0.02748612 0.19158319 0.69725564 1.10087097 0.76490841]
 [0.02292346 0.40422753 1.10747146 1.09904309 0.34032225]
 [0.05147738 0.55620644 1.22767935 1.63461073 1.17487413]
 [0.02355968 0.16165    0.28132806 0.69674676 0.25589232]
 [0.00215318 0.02385566 0.15796979 0.53527543 0.42957222]]

Convolution Backward Kernel Gradient:
[[1.94720431 1.27538819 1.20571161]
 [1.64244819 1.98111992 1.90566698]
 [1.8494633  1.88991758 2.12653401]]
```

The function takes input data (input_data), a convolutional kernel (kernel), the gradient of the output (output_gradient), and optional parameters for stride and padding.

It calculates the output dimensions and applies zero-padding to the input data.

It initializes gradients for both input data and the kernel.

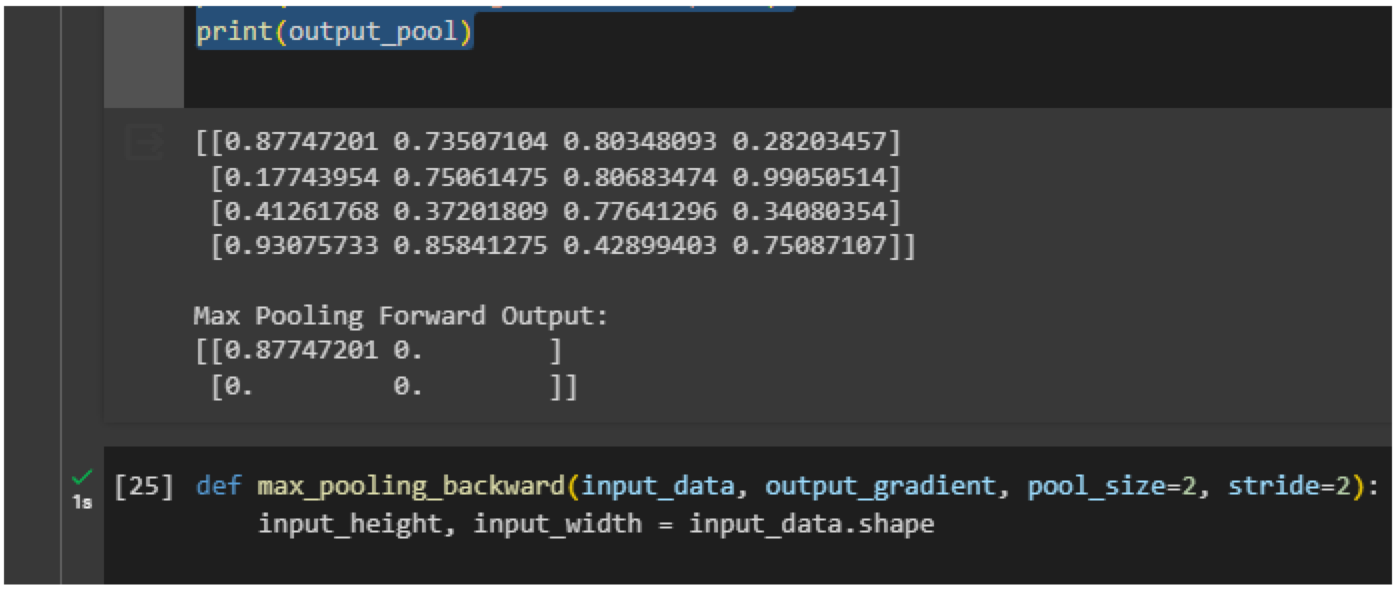
It performs the backward pass to compute the input and kernel gradients.

A sliding window is used to iterate over the input data in this forward pass. For each window position, np.max() is used to find the maximum value within that window. The output matrix is then filled with these maximum values. Finding the maximum value inherently involves creating a mask (to identify the position of the maximum value) and distributing the maximum value to the corresponding position in the output.

In the backward pass, the same sliding window is used, and the mask variable is created by comparing each element of the pool_slice to the max_value. The gradient is then propagated to the maximum value position in the original input using this mask.

So, the mask creation and value distribution are inherent in the nature of the max pooling operation itself. If your goal is a different type of pooling (e.g., average pooling), the implementation would involve different operations, but the basic concept of creating a mask and distributing values would still be present.

```
def max_pooling_forward(input_data, pool_size=2, stride=2):
    input_height, input_width = input_data.shape

    # Calculate output dimensions
    output_height = (input_height - pool_size) // stride + 1
    output_width = (input_width - pool_size) // stride + 1

    # Initialize output
    output = np.zeros((output_height, output_width))

    # Perform max pooling
    for i in range(0, output_height, stride):
        for j in range(0, output_width, stride):
            output[i, j] = np.max(input_data[i:i+pool_size, j:j+pool_size])

    return output



input_data_pool = np.random.rand(4, 4)  # input
print(input_data_pool)
output_pool = max_pooling_forward(input_data_pool)
print("\nMax Pooling Forward Output:")
print(output_pool)
```


```
print(output_pool)
```

```
[[0.87747201 0.73507104 0.80348093 0.28203457]
 [0.17743954 0.75061475 0.80683474 0.99050514]
 [0.41261768 0.37201809 0.77641296 0.34080354]
 [0.93075733 0.85841275 0.42899403 0.75087107]]

Max Pooling Forward Output:
[[0.87747201 0.        ]
 [0.         0.        ]]
```

```
[25] def max_pooling_backward(input_data, output_gradient, pool_size=2, stride=2):
         input_height, input_width = input_data.shape
```

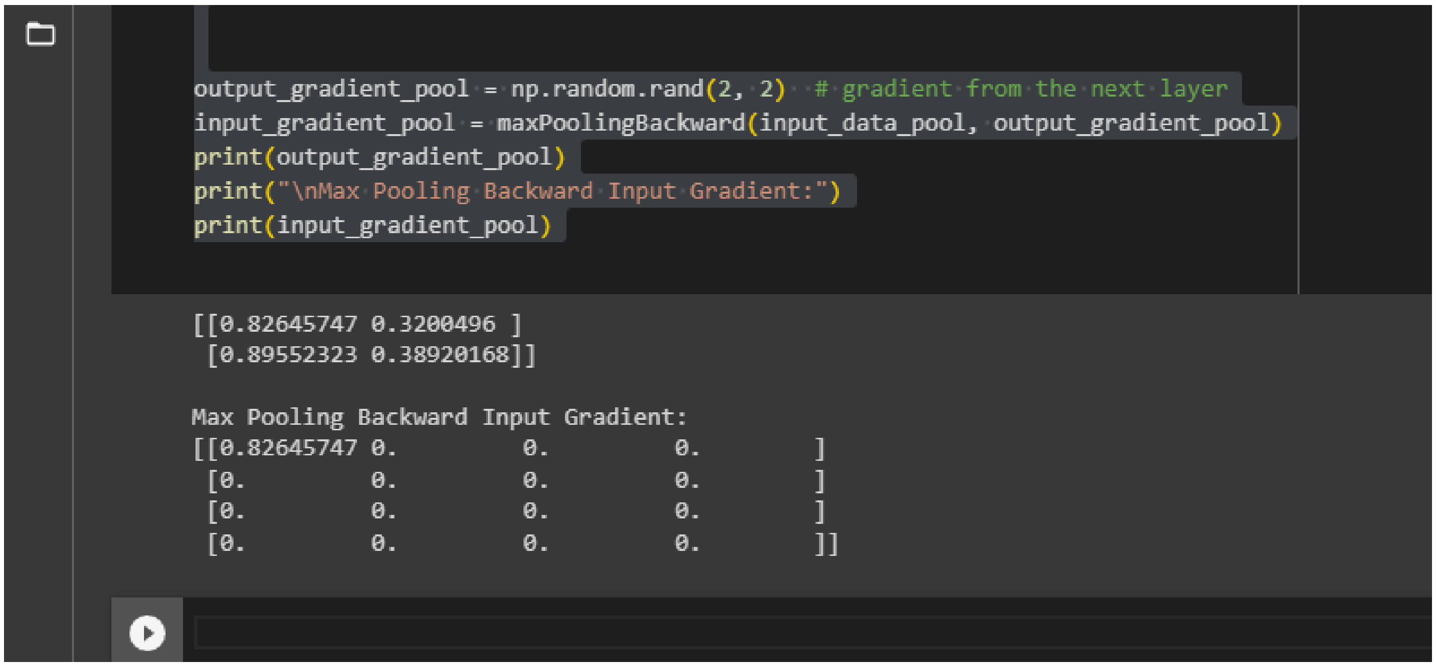
The function takes input data (input_data), and optional parameters for pool size and stride.

It calculates the output dimensions.

It initializes the output matrix.

It performs max-pooling by sliding a window over the input data and taking the maximum value in each window.

```
def maxPoolingBackward(input_data, output_gradient, pool_size=2, stride=2):

    input_height, input_width = input_data.shape


    # Calculate output dimensions

    output_height = (input_height - pool_size) // stride + 1

    output_width = (input_width - pool_size) // stride + 1


    # Initialize gradient

    input_gradient = np.zeros_like(input_data)


    # Perform backward pass

    for i in range(0, output_height, stride):

        for j in range(0, output_width, stride):

            pool_slice = input_data[i:i+pool_size, j:j+pool_size]

            max_value = np.max(pool_slice)
            mask = pool_slice == max_value
                        input_gradient[i:i+pool_size,  j:j+pool_size]  +=  mask  *
output_gradient[i, j]

    return input_gradient



output_gradient_pool = np.random.rand(2, 2)  # gradient from the next layer
input_gradient_pool = maxPoolingBackward(input_data_pool, output_gradient_pool)
print(output_gradient_pool)
print("\nMax Pooling Backward Input Gradient:")
print(input_gradient_pool)
```



```
output_gradient_pool = np.random.rand(2, 2)  # gradient from the next layer
input_gradient_pool = maxPoolingBackward(input_data_pool, output_gradient_pool)
print(output_gradient_pool)
print("\nMax Pooling Backward Input Gradient:")
print(input_gradient_pool)
```

```
[[0.82645747 0.3200496 ]
 [0.89552323 0.38920168]]

Max Pooling Backward Input Gradient:
[[0.82645747 0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]]
```

The function takes input data (input_data), the gradient of the output (output_gradient), and optional parameters for pool size and stride.

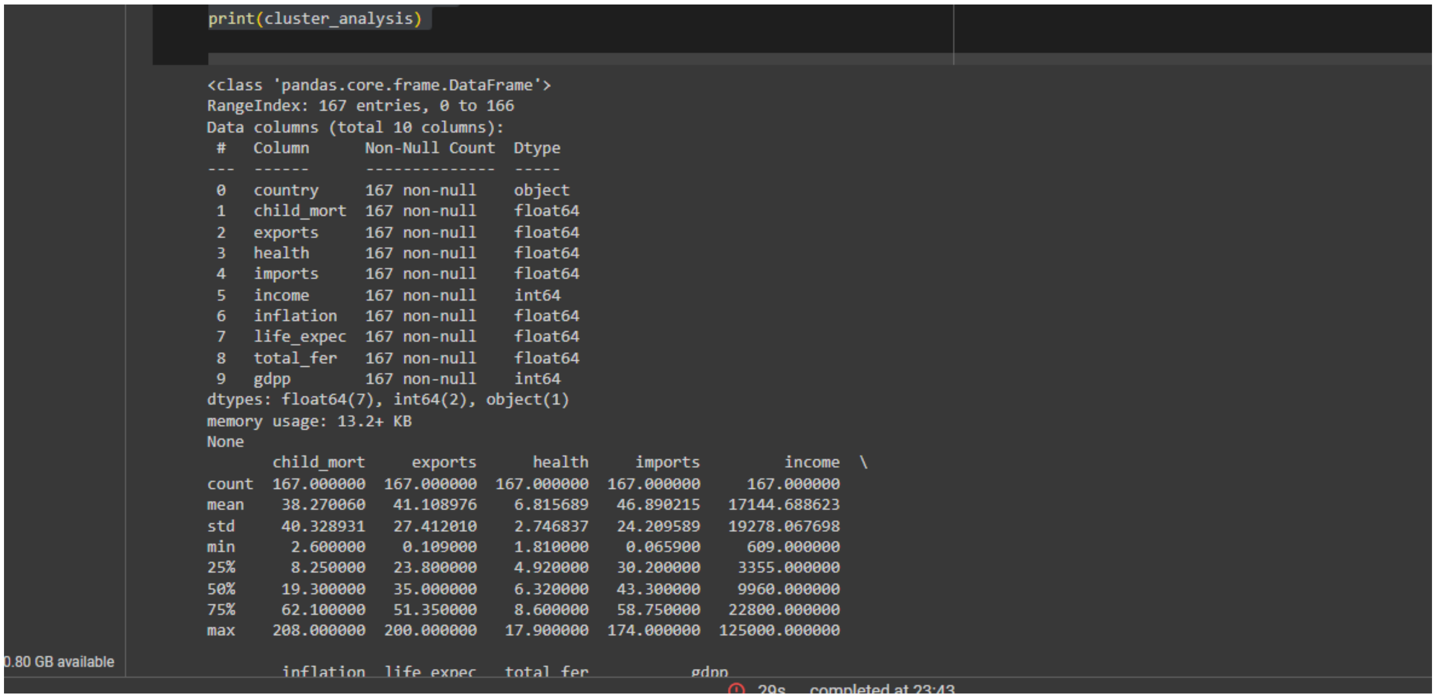
It calculates the output dimensions.

It initializes the gradient for the input data.

It performs the backward pass for max pooling, propagating the gradient to the maximum value in each window.

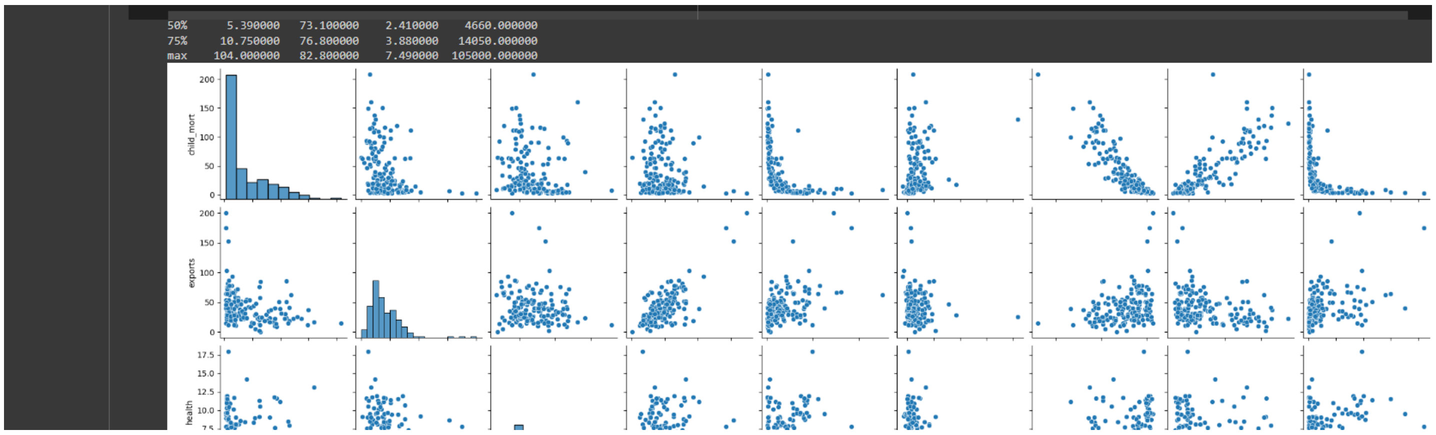
SECTION C

```
def load_data_file(file_path):
        names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']
    data = pd.read_csv(file_path, names=names)
    return data


def load_data():
                                                              url              =
'https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/process
ed.cleveland.data'
        names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']
    data = pd.read_csv(url, names=names)
    return data


    # Modify the handle_missing_values function
def handle_missing_values(data):
    data = data.replace('?', pd.NA)
    data = data.apply(pd.to_numeric, errors='coerce')
    data.fillna(data.mean(), inplace=True)   # Filling missing values with mean
    return data



def preprocess_data(data):
    data = pd.get_dummies(data, columns=['cp', 'restecg', 'slope', 'thal'])
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data.drop('num', axis=1))

    data = pd.DataFrame(scaled_data, columns=data.columns[:-1])

    data['num'] = data['num'].astype(int)

    return data


def visualize_data(data):

    sns.pairplot(data, hue='num')

    plt.title('Pairplot for the Dataset')

    plt.show()
```

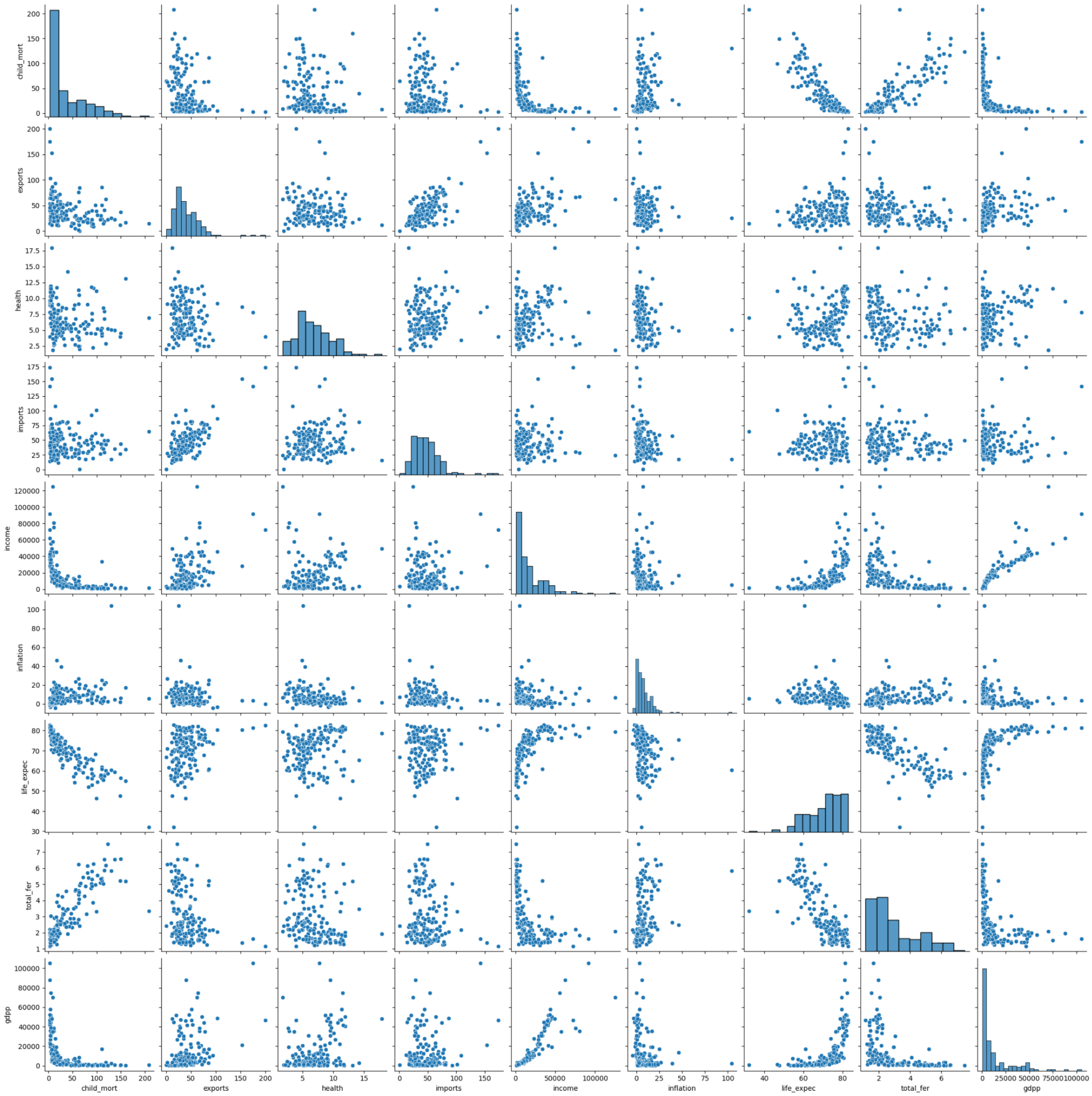


```
print(cluster_analysis)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   country     167 non-null    object
 1   child_mort  167 non-null    float64
 2   exports     167 non-null    float64
 3   health      167 non-null    float64
 4   imports     167 non-null    float64
 5   income      167 non-null    int64
 6   inflation   167 non-null    float64
 7   life_expec  167 non-null    float64
 8   total_fer   167 non-null    float64
 9   gdpp        167 non-null    int64
dtypes: float64(7), int64(2), object(1)
memory usage: 13.2+ KB
None
       child_mort     exports      health     imports         income  \
count  167.000000  167.000000  167.000000  167.000000     167.000000
mean    38.270060   41.108976    6.815689   46.890215   17144.688623
std     40.328931   27.412010    2.746837   24.209589   19278.067698
min      2.600000    0.109000    1.810000    0.065900     609.000000
25%      8.250000   23.800000    4.920000   30.200000    3355.000000
50%     19.300000   35.000000    6.320000   43.300000    9960.000000
75%     62.100000   51.350000    8.600000   58.750000   22800.000000
max    208.000000  200.000000   17.900000  174.000000  125000.000000

        inflation  life_expec   total_fer          gdpp
```



```
50%       5.390000   73.100000    2.410000    4660.000000
75%      10.750000   76.800000    3.880000   14050.000000
max     104.000000   82.800000    7.490000  105000.000000
```


child_mort
exports
health
imports
income
inflation
life_expec
total_fer
gdpp