



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY

HCIA-AI Huawei

Lecture 11

AI Important Modules with Practice

Prepared by
Eng. Eman Elgallad

www.midocean.ae

Lecture

11

Content:

Python important Modules

- Pandas
- Matplotlib
- Seaborn

Deep Learning Practical Exercise

The background of the slide features a complex network diagram. It consists of numerous small, light blue circular nodes connected by thin, white lines. These connections form a web-like structure that spans the right side of the image. The overall color scheme is dark blue, with the network elements providing a high-contrast, futuristic aesthetic.

Pandas

Pandas First Steps: **install** and **import**

- Pandas is an easy package to install. Open up your terminal program (shell or cmd) and install it using either of the following commands:

Installation: https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html

```
$ conda install pandas  
OR  
! pip install pandas
```

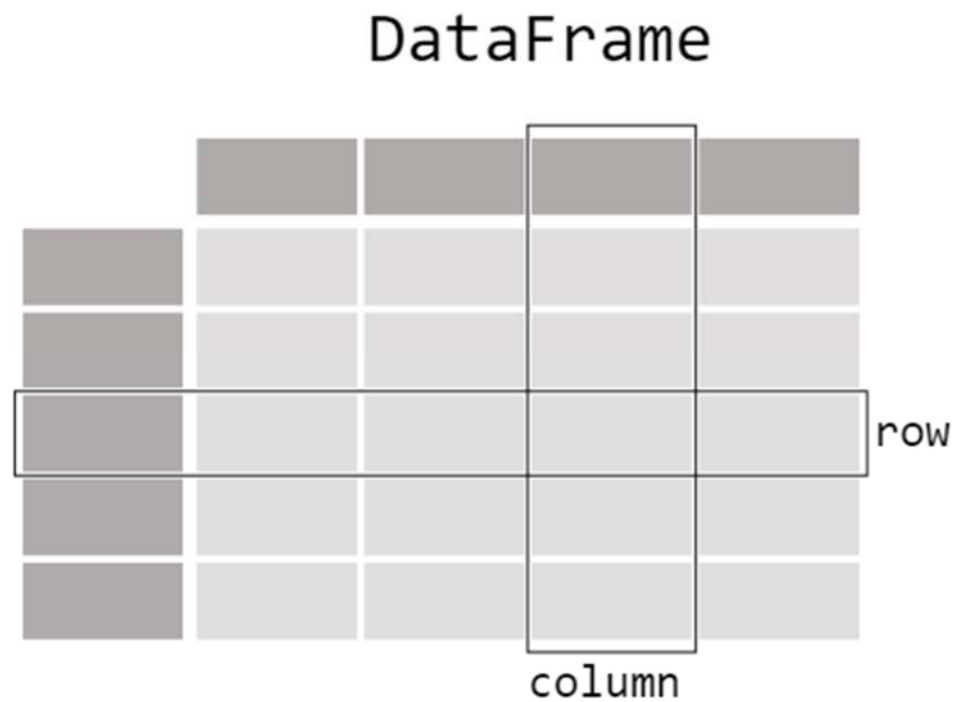
- For **jupyter notebook** users, you can run this cell:
 - The **!** at the beginning runs cells as if they were in a terminal.

```
!pip install pandas
```

- To import pandas we usually import it with a shorter name since it's used so much:

```
import pandas as pd
```

pandas: Data Table Representation



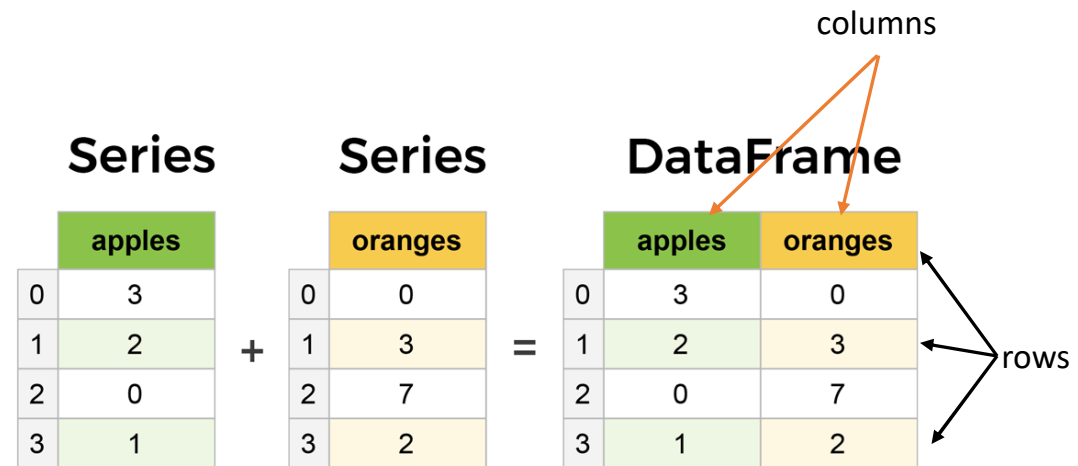
Core components of pandas: Series & DataFrames

The primary two components of pandas are the Series and DataFrame.

- **Series** is essentially a **column**, and
- **DataFrame** is a multi-dimensional table made up of a **collection of Series**.
- **DataFrames** and **Series** are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.
 - A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Features of DataFrame

- Potentially columns are of different types
- Size – Mutable
- Labeled axes (*rows* and *columns*)
- Can Perform Arithmetic operations on rows and columns



Types of Data Structure in Pandas

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array with immutable size
Data Frames	2	General 2D labeled, size mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size mutable array.

- **Series & Data Frame**

- [Series](#) is a one-dimensional array (1D Array) like structure with homogeneous data.
- [Data Frame](#) is a two-dimensional array (2D Array) with [heterogeneous](#) data.

- **Panel**

- Panel is a three-dimensional data structure (3D Array) with [heterogeneous](#) data.
- It is hard to represent the panel in graphical representation.
- But a panel can be illustrated as a container of Data Frame

Creating a DataFrame from scratch

- There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dict. But first you must import pandas.

```
import pandas as pd
```

- Let's say we have a fruit stand that sells apples and oranges. We want to have a column for each fruit and a row for each customer purchase. To organize this as a dictionary for pandas we could do something like:

```
data = { 'apples': [3, 2, 0, 1] , 'oranges': [0, 3, 7, 2] }
```

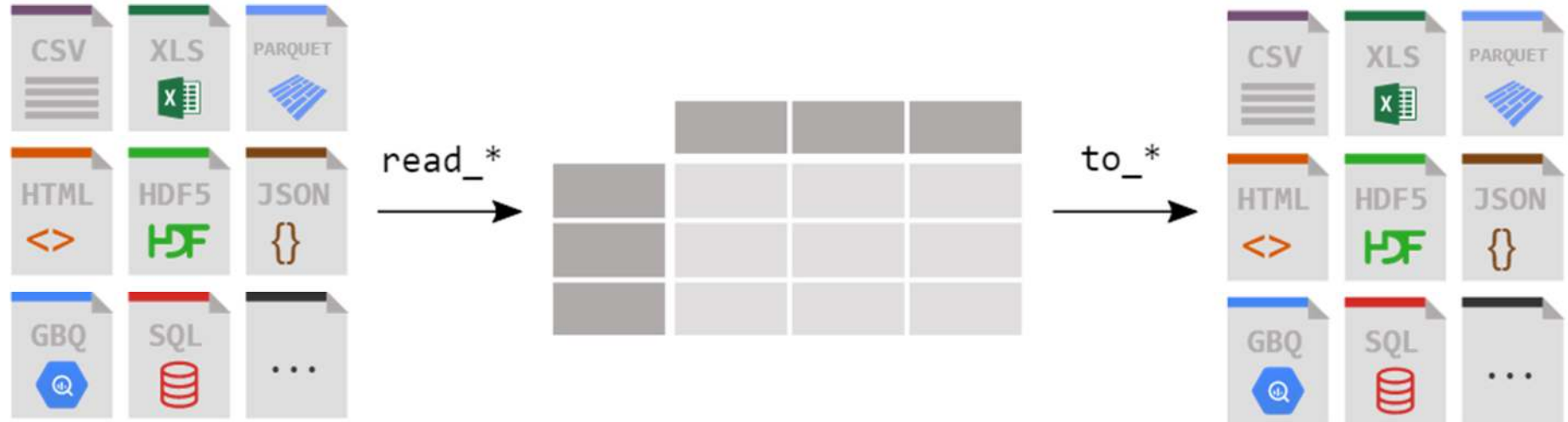
- And then pass it to the pandas Data Frame constructor:

```
df = pd.DataFrame(data)
```

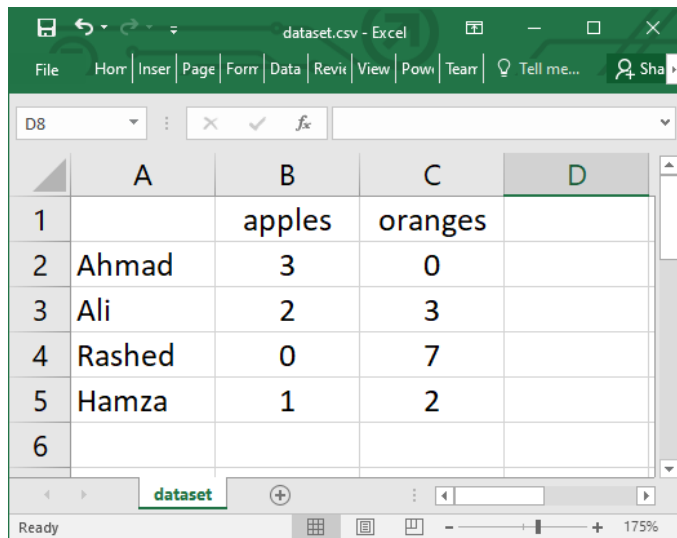


	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Loading a DataFrame from files



Reading data from a CSV file



	A	B	C	D
1		apples	oranges	
2	Ahmad	3	0	
3	Ali	2	3	
4	Rashed	0	7	
5	Hamza	1	2	
6				

```
File Edit Format Run Options Window Help
1 import pandas as pd
2
3 df = pd.read_csv('dataset.csv')
4 print(df)
5
6 # OR
7
8 df = pd.read_csv('dataset.csv', index_col=0)
9 print(df)
10
```

Ln: 6 Col: 0

Reading data from CSVs

- With CSV files, all you need is a single line to load in the data:

```
df = pd.read_csv('dataset.csv')
```

	Unnamed: 0	apples	oranges
0	Ahmad	3	0
1	Ali	2	3
2	Rashed	0	7
3	Hamza	1	2

- CSVs don't have indexes like our DataFrames, so all we need to do is just designate the `index_col` when reading:

```
df = pd.read_csv('dataset.csv', index_col=0)
```

	apples	oranges
Ahmad	3	0
Ali	2	3
Rashed	0	7
Hamza	1	2

- Note: here we're setting the index to be column zero.*

Reading data from JSON

- If you have a JSON file — which is essentially a stored Python dict — pandas can read this just as easily:

```
df = pd.read_json('dataset.json')
```

- Notice this time our index came with us correctly since using JSON allowed indexes to work through nesting.
- Pandas will try to figure out how to create a DataFrame by analyzing structure of your JSON, and sometimes it doesn't get it right.
- Often you'll need to set the **orient** keyword argument depending on the structure

Example1: Reading data from JSON

```
{  
  "apples" : {"Ahmad":3, "Ali":2, "Rashed":0, "Hamza":1},  
  "oranges": {"Ahmad":0, "Ali":3, "Rashed":7, "Hamza":2}  
}
```



```
File Edit Format Run Options Window Help  
1 import pandas as pd  
2  
3 df = pd.read_json('dataset.json')  
4 print(df)  
Ln: 1 Col: 0
```



	apples	oranges
Ahmad	3	0
Ali	2	3
Rashed	0	7
Hamza	1	2

Example2: Reading data from JSON

```
{  
  "Ahmad" : {"apples":3,"oranges":0},  
  "Ali" : {"apples":2,"oranges":3},  
  "Rashed" : {"apples":0,"oranges":7},  
  "Hamza" : {"apples":1,"oranges":2}  
}
```



```
File Edit Format Run Options Window Help  
1 import pandas as pd  
2  
3 df = pd.read_json('dataset.json')  
4 print(df)  
Ln: 1 Col: 0
```



	Ahmad	Ali	Rashed	Hamza
apples	3	2	0	1
oranges	0	3	7	2

Viewing your data

- The first thing to do when opening a new dataset is print out a few rows to keep as a visual reference. We accomplish this with `.head()`:

```
movies_df.head()
```

- `.head()` outputs the first five rows of your DataFrame by default, but we could also pass a number as well: `movies_df.head(10)` would output the top ten rows, for example.

- To see the last five rows use `.tail()` that also accepts a number, and in this case we printing the bottom two rows.:

```
movies_df.tail(2)
```

Getting info about your data

- `.info()` should be one of the very first commands you run after loading your data
- `.info()` provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.

```
movies_df.info()
```

```
movies_df.shape
```

OUT:

```
<class 'pandas.core.frame.DataFrame'>  
Index: 1000 entries, Guardians of the Galaxy to Nine Lives  
Data columns (total 11 columns):  
Rank                1000 non-null int64  
Genre               1000 non-null object  
Description          1000 non-null object  
Director            1000 non-null object  
Actors              1000 non-null object  
Year               1000 non-null int64  
Runtime (Minutes)   1000 non-null int64  
Rating              1000 non-null float64  
Votes               1000 non-null int64  
Revenue (Millions)  872 non-null float64  
Metascore           936 non-null float64  
dtypes: float64(3), int64(4), object(4)  
memory usage: 93.8+ KB
```

OUT:

```
(1000, 11)
```

Understanding your variables

- Using `.describe()` on an entire DataFrame we can get a summary of the distribution of continuous variables:

```
movies_df.describe()
```

OUT:

	rank	year	runtime	rating	
count	1000.000000	1000.000000	1000.000000	1000.000000	1.00
mean	500.500000	2012.783000	113.172000	6.723200	1.69
std	288.819436	3.205962	18.810908	0.945429	1.88
min	1.000000	2006.000000	66.000000	1.900000	6.10
25%	250.750000	2010.000000	100.000000	6.200000	3.60
50%	500.500000	2014.000000	111.000000	6.800000	1.10
75%	750.250000	2016.000000	123.000000	7.400000	2.30
max	1000.000000	2016.000000	191.000000	9.000000	1.79

- `.describe()` can also be used on a categorical variable to get the count of rows, unique count of categories, top category, and freq of top category:

```
movies_df['genre'].describe()
```

OUT:

```
count          1000
unique          207
top    Action,Adventure,Sci-Fi
freq           50
Name: genre, dtype: object
```

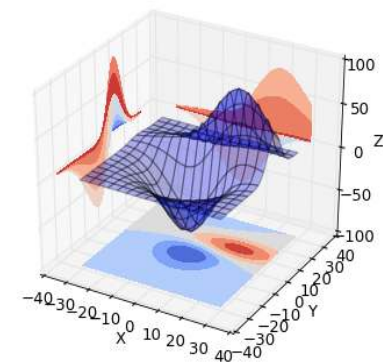
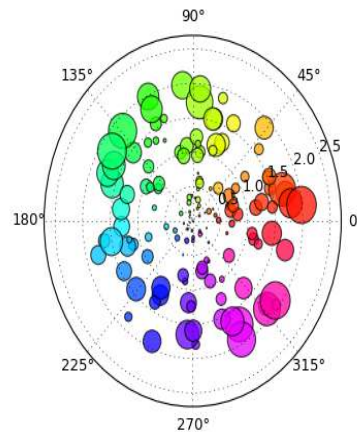
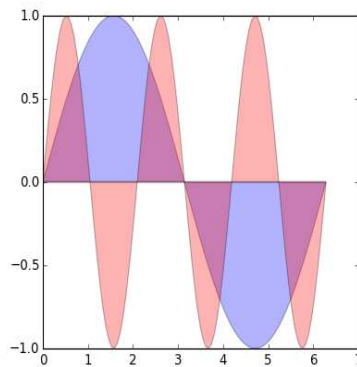
- This tells us that the genre column has 207 unique values, the top value is Action/Adventure/Sci-Fi, which shows up 50 times (freq).

The background of the slide features a complex network diagram. It consists of numerous small, light blue circular nodes connected by thin, white lines. These lines form a web-like structure that spans the right side of the image. The overall color scheme is dark blue, with the network elements providing a high-contrast, technical aesthetic.

Matplotlib and Seaborn

What is data visualization?

- Data visualization is the graphical representation of information and data.
 - Can be achieved using visual elements like figures, charts, graphs, maps, and more.
- Data visualization tools provide a way to present these figures and graphs.
- Often, it is essential to analyze massive amounts of information and make data-driven decisions.
 - converting complex data into an easy-to-understand representation.



Matplotlib

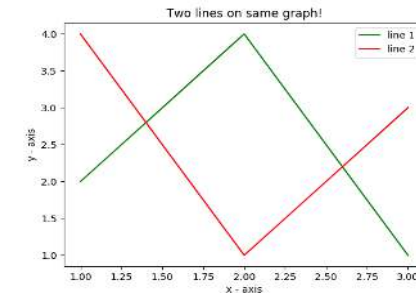
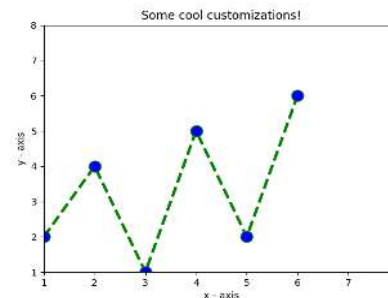
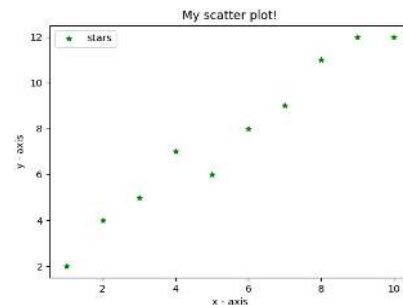
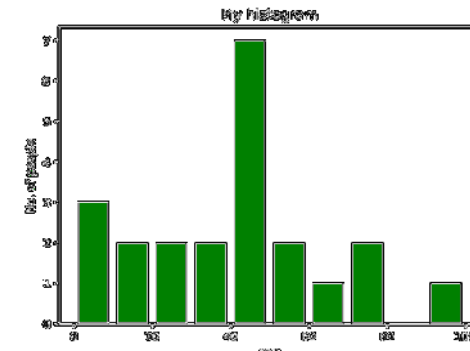
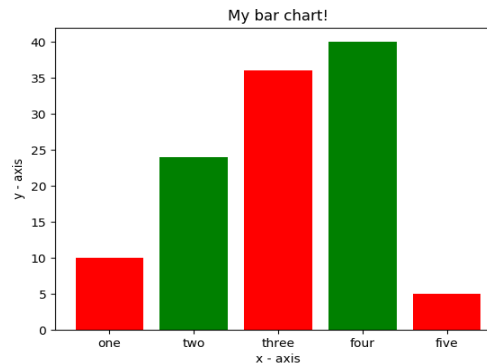
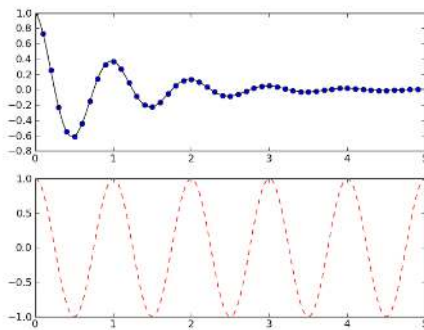
- **Matplotlib** is one of the most powerful tools for data visualization in Python.
- **Matplotlib** is an incredibly powerful (and beautiful!) 2-D plotting library.
 - It is easy to use and provides a huge number of examples for tackling unique problems
- In order to get **matplotlib** into your script,
 - first you need to import it, for example:

```
import matplotlib.pyplot as plt
```
- However, if it is not installed, you may need to install it:
 - Easiest way to install **matplotlib** is using **pip**.
 - Type the following command in the command prompt (cmd) or your Linux shell;
 - **pip install matplotlib**
 - *Note that you may need to run the above cmd as an administrator*

matplotlib

- Strives to emulate MATLAB
 - `matplotlib.pyplot` is a collection of command style functions that make `matplotlib` work like **MATLAB**.
- Each `pyplot` function makes some change to the figure:
 - e.g.,
 - creates a figure,
 - creates a plotting area in the figure,
 - plots some lines in the plotting area,
 - decorates the plot with labels, etc.
- **Note** that various states are preserved across function calls
- Whenever you plot with `matplotlib`, the two main code lines should be considered:
 - Type of graph
 - this is where you **define** a **bar** chart, **line** chart, **etc.**
 - Show the graph
 - this is to **display** the graph

- **Matplotlib** allows you to make easy things
- You can generate **plots, histograms, power spectra, bar charts, errorcharts, scatterplots**, etc., with just a few lines of code.



Seaborn :

It is also a Python library used for plotting graphs with the help of Matplotlib, Pandas, and Numpy. It is built on the roof of Matplotlib and is considered as a superset of the Matplotlib library. It helps in visualizing univariate and bivariate data. It uses beautiful themes for decorating Matplotlib graphics. It acts as an important tool in picturing Linear Regression Models. It serves in making graphs of statical Time-Series data. It eliminates the overlapping of graphs and also aids in their beautification.

Seaborn vs Matplotlib?

Features	Matplotlib	Seaborn
Functionality	It is utilized for making basic graphs. Datasets are visualised with the help of bargraphs, histograms, piecharts, scatter plots, lines and so on.	Seaborn contains a number of patterns and plots for data visualization. It uses fascinating themes. It helps in compiling whole data into a single plot. It also provides distribution of data.
Syntax	It uses comparatively complex and lengthy syntax. Example: Syntax for bargraph- <code>matplotlib.pyplot.bar(x_axis, y_axis).</code>	It uses comparatively simple syntax which is easier to learn and understand. Example: Syntax for bargraph- <code>seaborn.barplot(x_axis, y_axis).</code>

Activate Wind

Seaborn vs Matplotlib?

Dealing Multiple Figures	We can open and use multiple figures simultaneously. However, they are closed distinctly. Syntax to close one figure at a time: <code>matplotlib.pyplot.close()</code> . Syntax to close all the figures: <code>matplotlib.pyplot.close("all")</code>	Seaborn sets time for the creation of each figure. However, it may lead to (OOM) out of memory issues
Visualization	Matplotlib is well connected with Numpy and Pandas and acts as a graphics package for data visualization in python. Pyplot provides similar features and syntax as in MATLAB. Therefore, MATLAB users can easily study it.	Seaborn is more comfortable in handling Pandas data frames. It uses basic sets of methods to provide beautiful graphics in python.

Lecture

11

Pandas, Matplotlib, and Seaborn Code

<https://drive.google.com/file/d/1a2F8SZU7n79VTCYfXL8hW9L2i7TXTwaG/view?usp=sharing>

<https://drive.google.com/file/d/1bpn5k-uw7nPPy3uDBchg9ez6aXmUEFXE/view?usp=sharing>

https://drive.google.com/file/d/1i_Q0Gwmse4mgjtw7RI6g1lZzTpbw5w3/view?usp=sharing

<https://drive.google.com/file/d/11bG05RrQpjmd5BSL8XXvA3OTndPdb4nl/view?usp=sharing>



DL Practical Exercise

Implementation of CNN :

Creating the Sequential model

This initializes a Sequential model. In Keras, the Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

```
model = Sequential([
```

Adding Convolutional Layers

These lines add two convolutional layers with 32 filters each, a filter size of (3,3), 'same' padding, and ReLU activation. The first layer also specifies the input shape as (32, 32, 3), indicating that the input images are 32x32 pixels with three color channels.

```
Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(32, 32, 3)),  
Conv2D(32, (3,3), padding='same', activation='relu'),
```

Adding MaxPooling and Dropout

```
MaxPooling2D((2,2)),  
Dropout(0.25),
```

After each set of convolutional layers, there's a max-pooling layer to downsample the spatial dimensions, and a dropout layer to introduce regularization, preventing overfitting by randomly setting a fraction of input units to 0 at each update during training.

Implementation of CNN Cont. :

Flatten Layer

This layer is used to flatten the input, transforming it into a one-dimensional array, before passing it to the dense layers.

`Flatten()`

Adding Dense (Fully Connected) Layers

Two fully connected layers (dense layers) are added with 256 and 10 units, respectively. ReLU activation is used for the first dense layer, and softmax activation for the output layer, which is common for multi-class classification problems.

`Dense(256, activation='relu'),`
`Dropout(0.5),`
`Dense(10, activation='softmax')`

Model Summary

This line prints a summary of the model architecture, including the number of parameters in each layer and the total number of parameters in the model.

`model.summary()`

Lecture

11

CNN Code

https://drive.google.com/file/d/15M76gn5Lh-xdz1aBUAN_ZgYRLLQsTYc9/view?usp=sharing



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

www.midocean.ae