

RC11 SKILLS FINAL REPORT

INFO

Supporting folders are available on OneDrive via the following link:

https://liveuclac-my.sharepoint.com/:f/g/personal/ucbvmmk_ucl_ac_uk/Ev4Bouh6C5FFmJ5eFS9eXxMBWz7CiNbFmqUhqtVEyNdaBw?e=XfU3IV

CONTENT

Vectorial Encodings Of Qualitative Domains	I. Data Collection	Text Images Videos Text 02	6 7 9 12
	II. Self-organizing maps	Text 01 Images Videos Text 02	13 23 29 31
	III. Search engine	TF-IDF Doc2Vec Text search Text to videos Videos to image Image to videos Videos to images Images to text	33 34 36 37 38 39 40 41
Houdini	IV. Houdini Fundamentals		43
	V. Visualizing GPS and Image Metadata		46
	VI. Video to 3D Model		50
	VII. Visualizing JSON in Houdini		54
Complexity	VIII. Exercise One		58
	IX. Exercise Two		60
	X. Exercise Three		63
Bibliography			65

VECTORIZATION ENCODINGS OF QUALITATIVE DOMAINS

I. Data Collection

II. Self-organizing maps

III. Search engine

I. DATA COLLECTION

INTRODUCTION

Classism

What is a classic design? The inquiry about the “Classic” begins with defining it using adequate terminology. As part of the development of the Search Engine, this introduction presents a nuanced outlining for directing later steps and decisions in the project. Looking at the “classic” from a critical point of view, Demetri Porphyrios presents in his paper (Classism is not a style) a critical rethinking of what is the classic. He argues that classicism should not be reduced to merely a stylistic preference or a set of visual characteristics (Amundson, 2015). Instead, he states that classicism is a comprehensive philosophical and cultural framework that encompasses principles of proportion, harmony, order, and continuity. This framework is transformed into a physical entity by the constructional logic of vernacular. In conclusion, he emphasizes the importance of recognizing classicism as a holistic approach to architecture and design, rather than merely as a superficial style.

Columns Encyclopedia

In the context of columns, based on the earlier introduction it would be reductive to state that there are three main orders of columns: Doric, Ionic, and Corinthian. “Heterodoxia Architectonica” is a research by Andrés Duany that emphasizes the departure from the classical conventional understanding of columns (Congress for the New Urbanism, 2012). Each column is worthy of documentation and presents a style by itself. He criticizes the perception of classicism, although he aligns with Porphyrios but both of them did not build their arguments based on each other’s work.

Data Collection

The collected data presents the reflection of the discussed logic. The datasets form the ground to conduct an analytical exercise whereby an array of sources are consulted, critically reinterpreted, and reformulated to begin to arrive at the essence of each column. Given the nature of the columns, the datasets consist of text, images, videos, and text 02. These datasets resonate with the earlier introduction. Each dataset is explained in detail in the following pages.

DATASET 01: TEXT

The first text dataset consists of 983 webscraped architectural books that contain 39,779 sentences associated with columns. The architectural books include books such as A Pattern Language by Christopher Alexander, Elements of Architecture by Rem Koolhaas, and Ten Books on Architecture by Vitruvius. These books will provide a dictionary of definitions and descriptions of any hypothetical column. They represent the existing architectural knowledge on columns that can be used to sculpt new interpretations.

Folder_Path = “Datasets/ Text/ Books”

Abulafia_The_Boundless_Sea.txt	Asimov_I_Robot.txt	Bergdoll_Oechslin_Fragments_Architecture_and_the_Unfinished.txt	Brillouin_Science_and_Information_Theory.txt
Ackerman_Origins_Imitation_Conventions_Representation_in_the_Visual_Arts.txt	Askin_Narrative_and_Becoming.txt	Berkenhout_The_Ruins_of_Poestum_or_Posidonia_a_City_of_Magna_Graecia_in_the_Kingdom_of_Naples.txt	Brook_A_History_of_Future_Cities.txt
Ackerman_The_Architecture_of_Michelangelo.txt	Aureli_The_Possibility_of_an_Absolute_Architecture.txt	Bernini_The_Life_of_Gian_Lorenzo_Bernini.txt	Brown_The_Genius_of_Rome_1592_1623.txt
Acocella_Stone_Architecture_Ancient_and_Modern_Construction_Skills.txt	Avanessian_Hennig_Present_Tense_A_Poetics.txt	Bernoulli_The_Art_of_Conjecturing.txt	Bruno_On_the_Composition_of_Images_Signs_and_Ideas.txt
Adams_Empire_and_Communications.txt	Avery_Finn_Bernini.txt	Betsky_Architecture_Matters.txt	Buehlmann_Hovestadt_Coding_as_Literacy.txt
Adams_Mont_Saint_Michel_and_Chartres.txt	Ayache_The_Blank_Swan.txt	Bichat_General_Anatomy_Applied_to_Physiology_and_Medicine_Vol_1.txt	Buehlmann_Hovestadt_Domesticating_Symbols.txt
Adam_Ruins_of_the_Palace_of_the_Emperor_Diocletian_at_Spalatro_in_Dalmatia.txt	Bacchi_Bernini_and_the_Birth_of_Baroque_Portrait_sculpture.txt	Bichat_General_Anatomy_Applied_to_Physiology_and_Medicine_Vol_2.txt	Buehlmann_Hovestadt_Symbolizing_Existence.txt
Adorno_The_Stars_Down_to_Earth_and_Other_Essays_on_the_Irrational_in_Culture.txt	Bacon_Selected_Philosophical_Works.txt	Bichat_General_Anatomy_Applied_to_Physiology_and_Medicine_Vol_3.txt	Buehlmann_Mathematics_and_Information_in_the_Philosophy_of_Michel_Serres.txt
Aeschylus_The_Persians.txt	Bacon_The_Advancement_of_Learning.txt	Biebricher_The_Political_Theory_of_Neoliberalism.txt	Buffon_Natural_History_Quadrupeds.txt
Agrest_Conwy_Weisman_The_Sex_of_Architecture.txt	Bacon_The_New_Atlantis.txt	Blacklock_The_Emergence_of_the_Fourth_Dimension.txt	Buffon_Natural_History_Vol_1.txt
Agricola_De_Re_Metallica.txt	Baldinucci_The_Life_of_Bernini.txt	Blackwell_Nineteenth_Century_Architecture.txt	Buffon_Natural_History_Vol_2.txt
Agrippa_The_Fourth_Book_of_Occult_Philosophy.txt	Ball_The_Domus_Aurea_and_the_Roman_Architectural_Revolution.txt	Black_Experiments_upon_Magnesia_Alba_Quicklime_and_some_other_Alcaline_Substances.txt	Buffon_Natural_History_Vol_4.txt
Agrippa_Three_Books_of_Occult_Philosophy.txt	Ball_The_Selfmade_Tapestry_Pattern_Formation_in_Nature.txt	Blonde_City_and_Society_in_the_Low_Countries_1100_1600.txt	Buffon_Natural_History_Vol_6.txt
Aharoni_Mathematics_Poetry_and_Beauty.txt	Balzac_The_Unknown_Masterpiece.txt	Blunt_Artistic_Theory_in_Italy_1450_1600.txt	Buffon_Natural_History_Vol_7.txt
Alberti_10_books_of_architecture.txt	Banham_Critic_Writes.txt	Blunt_Art_and_Architecture_in_France_1500_1700.txt	Burckhardt_The_Architecture_of_the_Italian_Renaissance.txt
Alberti_Delineation_of_the_City_of_Rome.txt	Banham_Theory_and_Design_in_the_First_Machine_Age.txt	Blunt_Baroque_and_Rococo_Architecture_and_Decoration.txt	Burke_Art_And_Identity_In_Early_Modern_Rome.txt
Alberti_Momus.txt	Banham_The_Architecture_of_the_Well_Tempered_Environment.txt	Blunt_Guide_to_Baroque_Rome.txt	Burke_Hybrid_Renaissance.txt
Alberti_On_Painting.txt	Barn_The_Clothing_of_Clio.txt	Blunt_Philibert_de_l'Orme.txt	Burke_Rethinking_the_High_Renaissance.txt
Alberti_On_the_Art_of_Building_in_Ten_Books.txt	Barasch_Theories_of_Art.txt	BoBardi_Stones_Against_Diamonds.txt	Burke_The_Fabrication_of_Louis_XIV.txt
Alberti_The_Mathematical_Works_of_Leon_Battista_Alberti.txt	Barber_A_Companion_To_World_Mythology.txt	Bohnet_What_Works.txt	Burrows_Fictioning.txt
Alder_Engineering_the_Revolution.txt	Barnes_Michelangelo_in_Print.txt	Bolzoni_The_Gallery_of_Memory.txt	Bussells_Spectacle_Rhetoric_and_Power.txt
Alexander_A_Pattern_Language.txt	Barry_The_Rome_of_Piranesi.txt	Bonne_maison_Macy_Festival_Architecture.txt	Butler_Synaesthesia_and_the_Ancient_Senses.txt
Alexander_From_Renaissance_to_Counter_Reformation.txt	Barthes_Empire_of_Signs.txt	Bonnet_A_Compendium_of_Natural_Philosophy.txt	Cache_Projectiles.txt
Alexander_The_Timeless_Way_of_Building.txt	Barthes_Mythologies.txt	Boole_The_Laws_of Thought.txt	Cahan_They_All_Fall_Down.txt
Allatios_The_Newer_Temples_of_the_Greeks.txt	Barthes_The_Fashion_System.txt	Borges_Collected_Fictions.txt	Calasso_Ardor.txt
Anderson_Antiquities_What_Everyone_Needs_to_Know.txt	Barthes_The_Language_of_Fashion.txt	Bork_Late_Gothic_Architecture.txt	Calasso_Ka_Stories_of_the_Mind_and_Gods_of_India.txt
An_Atlas_of_Fantastic_Infrastructures.txt	Barthes_Writing_Degree_Zero.txt	Borromeo_Sacred_Painting.txt	Calasso_The_Marriage_of_Cadmus_and_Harmony.txt
Aquinas_Summa_Theologica.txt	Bataille_%20Death_and_Sensuality.txt	Bosker_Original_Copies.txt	Callan_Dictionary_of_Fashion_and_Fashion_Designers.txt
Archimedes_On_the_Sphere_and_the_Cylinder.txt	Bataille_Death_and_Sensuality_A_Study_of_Eroticism_and_the_Taboo.txt	Bottazzi_Digital_Architecture_Beyond_Computers_Fragments_of_a_Cultural_History_of_Computational_Design.txt	Calvino_Invisible_Cities.txt
Arendt_Eichmann_in_Jerusalem.txt	Baudrillard_Nouvel_The_Singular_Objects_of_Architecture.txt	Boucher_Italian_Baroque_Sculpture.txt	Calvin_Harmony_of_the_Law_Vol_1.txt
Aristotle_A_Treatise_on_Government.txt	Baudrillard_Simulacra_and_Simulation.txt	Boullee_Architecture_Essay_on_Art.txt	Calvin_Harmony_of_the_Law_Vol_2.txt
Aristotle_Ethics.txt	Baxandall_Giotto_and_the_Orators.txt	Bourdieu_Distinction.txt	Calvin_Harmony_of_the_Law_Vol_3.txt
Aristotle_Metaphysics.txt	Bayle_Bartlett_Various_Thoughts_on_the_Occasion_of_a_Comet.txt	Boyd_The_Australian_Ugliness.txt	Calvin_Harmony_of_the_Law_Vol_4.txt
Aristotle_Physics.txt	Beauty_Poems_From_Plenty.txt	Bradbury_Romantic_Theories_of_Architecture_of_the_19th_century.txt	Cameron_The_Baths_of_the_Romans_Explained_and_Illustrated.txt
Aristotle_Politics.txt	Beckett_Stories_and_Texts_for_Nothing.txt	Bradley_Smell_and_the_Ancient_Senses.txt	Campanella_The_Book_and_the_Body_of_Nature.txt
Aristotle_The_Art_of_Rhetoric.txt	Belier_Bergdoll_Lecoeur_Henri_Labrouste_Structure_Brought_to_Light.txt	Braidaotti_Dolphijn_Philosophy_After_Nature.txt	Campanella_The_City_of_the_Sun.txt
Arnold_Literature_and_Dogma.txt	Bell_Art_History_in_the_Age_of_Bellori.txt	Braidaotti_Patterns_of_Dissonance.txt	Camus_The_Myth_of_Sisyphus_And_Other_Essays.txt
ArtBasel_Catalogue.txt	Bell_Men_of_Mathematics.txt	Brennan_The_Sacred_Made_Real.txt	Carpo_Furlan_Leon_Battista_Albertis_Delineation_of_the_City_of_Rome.txt
Ascher_The_Works_Anatomy_of_a_City.txt	Belting_Florence_and_Baghdad.txt	Bredekamp_The_Lure_of_Antiquity_and_the_Cult_of_the_Machine.txt	Carpo_Architecture_in_the_Age_of_Printing.txt
Asimov_Complete_Robot_Anthology.txt	Belting_Likeness_and_Presence.txt		Carpo_The_Alphabet_and_the_Algorithm.txt

I. Data Collection

DATASET 02: IMAGES

The second dataset consists of 4411 webscraped images of columns. The locations of the images are diverse including the UK, Italy, Greece, USA, France, etc. The information of each image is stored in a dictionary with the keys of ‘name’, ‘Country’, ‘City’, ‘Style’, ‘Subject Date’, and ‘View’.

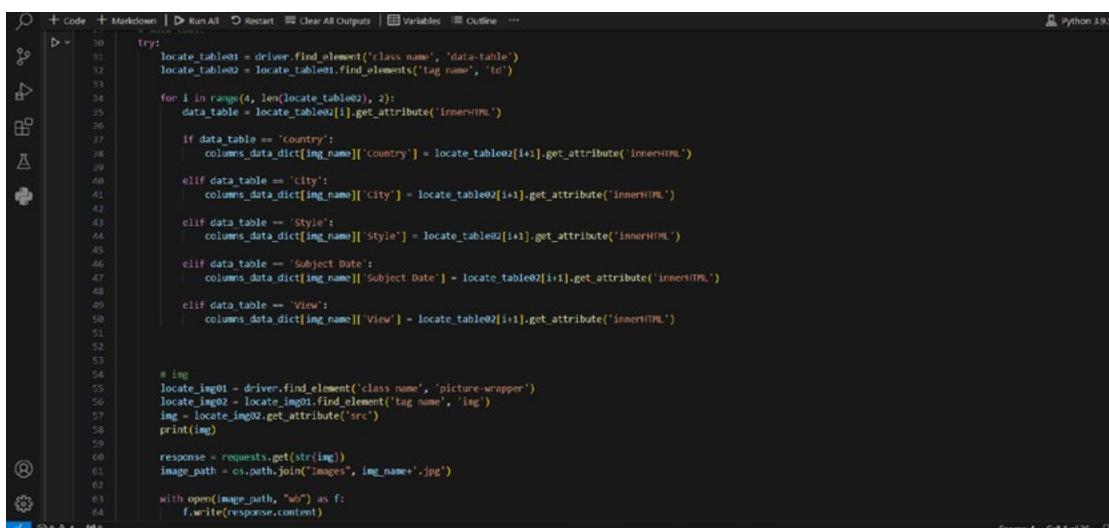
Sample from the dictionary:

```
{“Great-Bath-Bath”: {“Country”: “UK: England”, “City”: “Bath”, “Subject Date”: “0001”, “View”: “Exterior”, “Style”: “Roman”}, “Rotunda-Wentworth-Castle-South-Yorkshire-detail-of-entablature”: {“Country”: “UK: England”, “Subject Date”: “1742”, “View”: “Exterior”, “Style”: “Palladian”}, “Scarborough-North-Yorkshire-aerial-perspective-of-The-Crescent-and-surrounding-buildings-arranged-around-a-circular-green-and-column”: {“Country”: “UK: England”, “City”: “Scarborough”, “Subject Date”: “1832”, “View”: “Aerial”, “Style”: “Classical Revival”}}
```

Then all of this information all combined in a new key named ‘info’ to be used later.

Sample of the list:

```
[  
  {  
    “Subject Date”: “0001”,  
    “name”: “0_Great-Bath-Bath.jpg”,  
    “info”: “UK: England, Bath, Roman, Exterior, Great Bath Bath”  
  },  
  {  
    “Subject Date”: “1742”,  
    “name”: “1_Rotunda-Wentworth-Castle-South-Yorkshire-detail-of-entablature.jpg”,  
    “info”: “UK: England, Palladian, Exterior, Rotunda Wentworth Castle South Yorkshire detail of entablature”  
  },  
]
```



A screenshot of a Jupyter Notebook interface. The code cell contains Python 3.9.5 code for webscraping. The code uses Selenium to find elements by class name ('picture-wrapper') and tag name ('img'). It then extracts the 'src' attribute of the img element and saves it as a file in the 'Images' directory. The code is annotated with comments explaining the logic for extracting data from tables and saving images.

```
# Code + Markdown | ▶ Run All ⌘ Restart ⌘ Clear All Outputs | Variables | Outline ...  
Python 3.9.5  
try:  
    locate_table01 = driver.find_element('class name', 'data-table')  
    locate_table02 = locate_table01.find_elements('tag name', 'td')  
  
    for i in range(4, len(locate_table02), 2):  
        data_table = locate_table02[i].get_attribute('innerHTML')  
  
        if data_table == 'Country':  
            columns_data_dict[img_name][‘Country’] = locate_table02[i+1].get_attribute('innerHTML')  
  
        elif data_table == 'City':  
            columns_data_dict[img_name][‘City’] = locate_table02[i+1].get_attribute('innerHTML')  
  
        elif data_table == 'Style':  
            columns_data_dict[img_name][‘Style’] = locate_table02[i+1].get_attribute('innerHTML')  
  
        elif data_table == 'Subject Date':  
            columns_data_dict[img_name][‘Subject Date’] = locate_table02[i+1].get_attribute('innerHTML')  
  
        elif data_table == 'View':  
            columns_data_dict[img_name][‘View’] = locate_table02[i+1].get_attribute('innerHTML')  
  
    # img  
    locate_img01 = driver.find_element('class name', 'picture-wrapper')  
    locate_img02 = locate_img01.find_element('tag name', 'img')  
    img = locate_img02.get_attribute('src')  
    print(img)  
  
    response = requests.get(str(img))  
    image_path = os.path.join("Images", img_name+'.jpg')  
  
    with open(image_path, "wb") as f:  
        f.write(response.content)  
Spaces: 4 Cell 1 of 26
```

* Webscraping code snippet

I. Data Collection

Data Sample

Folder_Path = “Datasets/ Images/ Images”



{"name": "0_Great-Bath-Bath.jpg", "info": "Exterior, Roman, Bath, UK: England, 0001, Great Bath Bath"}



{"name": "1_Rotunda-Wentworth-Castle-South-Yorkshire-detail-of-entablature.jpg", "info": "Exterior, Palladian, UK: England, 1742, Rotunda Wentworth Castle South Yorkshire detail of entablature"}



{"name": "2_Scarborough-North-Yorkshire-aerial-perspective-of-The-Crescent-and-surrounding-buildings-arranged-around-a-circular-green-and-column.jpg", "info": "Aerial, Classical Revival, Scarborough, North Yorkshire aerial perspective of The Crescent and surrounding buildings arranged around a circular green and column"}



{"name": "3_Royal-Brunswick-Theatre-Wellclose-Square-Stepney-London-street-elevation.jpg", "info": "Exterior, Greek Revival, London, UK: England, 1828, Royal Brunswick Theatre Wellclose Square Stepney London street elevation"}



{"name": "4_St-Philips-Chapel-Regent-Street-London-plan-and-west-elevation.jpg", "info": "Greek Revival, London, UK: England, 1820, St Philips Chapel Regent Street London plan and west elevation"}



{"name": "5_Heathcote-Ilkley-West-Yorkshire-the-main-hall.jpg", "info": "Interior, Edwardian, 1906, Heathcote Ilkley West Yorkshire the main hall"}



{"name": "6_Heathcote-Ilkley-West-Yorkshire-the-lobby-off-the-main-hall.jpg", "info": "Interior, Edwardian, Ilkley, UK: England, 1906, Heathcote Ilkley West Yorkshire the lobby off the main hall"}



{"name": "7_Plaza-de-Espana-from-the-Ibero-American-Exhibition-of-1929-Maria-Luisa-Park-Seville-the-tiled-alcoves.jpg", "info": "Exterior, Seville, Spain, 1929, Plaza de Espana from the Ibero American Exhibition of 1929 Maria Luisa Park Seville the tiled alcoves"}



{"name": "8_National-Building-Museum-4th-and-F-Streets-Washington-DC-the-main-hall.jpg", "info": "Interior, Washington, D.C., United States, 1887, National Building Museum 4th and F Streets Washington DC the main hall"}



{"name": "9_St-Leonards-on-Sea-East-Sussex-view-of-the-west-end-of-the-marina-Victoria-House-and-St-Leonards-Church.jpg", "info": "Exterior, St Leonards on Sea, UK: England, 1830, St Leonards on Sea East Sussex view of the west end of the marina Victoria House and St Leonards Church"}



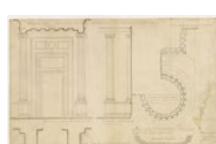
{"name": "10_St-Leonards-on-Sea-East-Sussex-view-of-the-west-end-of-the-marina-and-Victoria-House.jpg", "info": "Exterior, St Leonards on Sea, UK: England, 1830, St Leonards on Sea East Sussex view of the west end of the marina and Victoria House"}



{"name": "11_Coilsfield-or-Montgomerie-House-near-Tarbolton-Ayrshire-perspective-from-the-park.jpg", "info": "Exterior, Classical Revival, Tarbolton, UK: Scotland, 1798, Coilsfield or Montgomerie House near Tarbolton Ayrshire perspective from the park"}



{"name": "12_Detail-of-capital-possibly-from-Palazzo-Strozzi-Bevilacqua-or-Palazzo-Rondinelli-piazza-Ariostea-Ferrara.jpg", "info": "Worm's-eye, Renaissance, Ferrara, Italy, 1400, Detail of capital possibly from Palazzo Strozzi Bevilacqua or Palazzo Rondinelli piazza Ariostea Ferrara"}



{"name": "13_Designs-for-the-Reform-Club-Pall-Mall-London-elevation-and-details-of-columns-and-pilasters-in-alterations-to-coffee-room.jpg", "info": "Interior, Italianate, London, UK: England, 1853, Designs for the Reform Club Pall Mall London elevation and details of columns and pilasters in alterations to coffee room"}



{"name": "14_The-Circus-Bath-detail-of-the-facade.jpg", "info": "Exterior, Georgian, Bath, UK: England, 1766, The Circus Bath detail of the facade"}



{"name": "15_Eastern-Dispensary-Cleveland-Place-East-Bath.jpg", "info": "Exterior, Classical Revival, Bath, UK: England, 1845, Eastern Dispensary Cleveland Place East Bath"}

DATASET 03: VIDEOS

The third dataset consists of 135 web scraped videos showcasing different places in London. London was chosen as it is the most frequent location in the image dataset. The videos document various locations within Greater London. Information for each video is stored in a dictionary with keys of ‘name’, ‘summary’, ‘description’, and ‘style’.

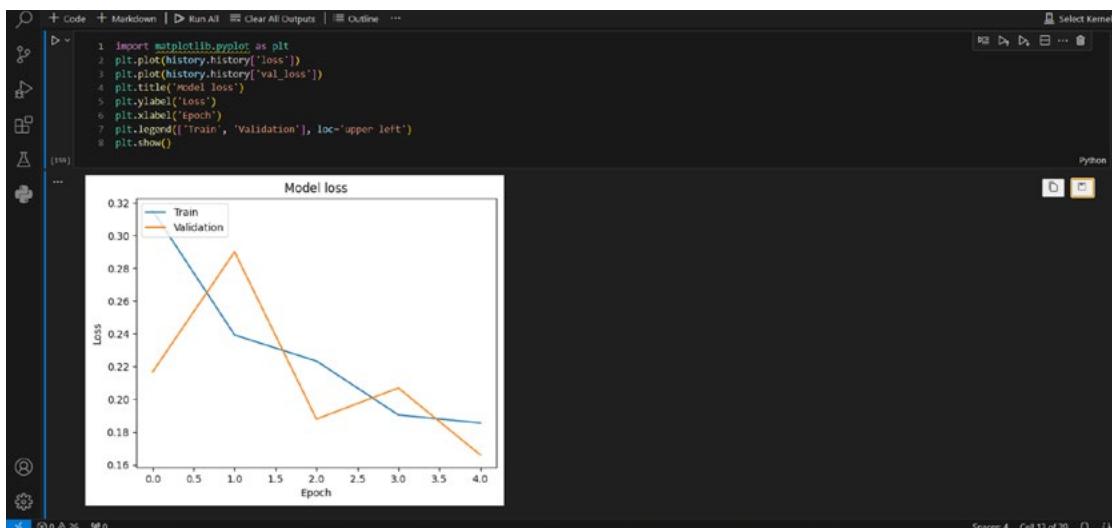
* datasets used for training the columns detection model are available on:
File_Path =
Datasets/Videos/ Columns Detection model/ Images

Columns Detection Model

As part of the video processing pipeline, an image classification model was trained specifically to identify column scenes within the video dataset. The model was trained using five distinct types of images: buttresses, columns, domes, vaults, and random images. Subsequently, I used the column detection model to analyze all the videos and generate a list of the ten-second that most accurately correspond to column scenes. This list is then merged with the existing dictionary.

The process of training the model starts by inputting the 5 different categories of images:

1. Train the model and save it.
2. Extract the features of the videos.
3. Sort the features based on proximity to the category “column”.
4. Extract the seconds of the closest clips.



* Column detection model code snippets

```

1 def load_image(img_file, target_size=(224,224)):
2     x = np.zeros((1, *target_size, 3))
3     x[0, :] = np.asarray(tf.keras.preprocessing.image.load_img(
4         img_file,
5         target_size=target_size)
6     )
7     x = tf.keras.applications.mobilenet.preprocess_input(x)
8     return x
9
10 def ensure_folder_exists(folder):
11     if not os.path.exists(folder):
12         os.makedirs(folder)
13
14 def processFilmByFrames(filmName, filmPath, interval, model):
15     features = []
16
17     file = VideoFileClip(filmPath)
18     nframes = int(file.duration/interval)
19     for i in range(nframes):
20         s = i*interval
21         frame = file.get_frame(s)
22         frame = Image.fromarray(frame,'RGB')
23         # no need for the .. anymore because you're not using the
24         # action classification model
25         # frame.save('../Frames/tmp.jpg'
26
27

```

I. Data Collection

Data Sample

Folder_Path = "Datasets/ Videos/ columns_videos_data.json"

Final list strucutre

{"name": "Lights of London",
"summary": " Night footage including Christmas lights, stores and tourist sights taken of Central London by amateur filmmaker Bernard Chaplin.",
"description": " The film shows the Christmas lights around Oxford Street and Regent Street and the colourful advertising around Piccadilly Circus. Red and green garlands of lights hang across the street under which vehicles pass. There are glimpses of several stores – Hamleys, Galeries Lafayette, Burberrys and Swan and Edgar. Illuminated advertising hoardings light up Piccadilly Circus. Adverts for alcohol, cigarettes and chewing gum dominate the signs. An illuminated sign for the Regent Place Hotel can be seen. The film finishes with a Christmas tree decorated with coloured lights.",
"second": [54, 0, 132, 36, 48, 108, 126, 114, 42, 12]},

{"name": "Local views of Friern Barnet",
"summary": " Local footage from the Friern Barnet area",
"description": " The film starts with the street sign Lawrence Campe Close, N20, on a white picket fence. It then shows a new building, possibly new almshouses around a lawned courtyard, behind a terrace of old buildings. The gravel driveway to the new building has then got a crowd of people, clergy and a bishop all in formal dress opening the building. The clergy give a speech regarding the building and the film cuts to a shot of a plaque, this reads \"These Almshouses blessed by the Bishop of London on 2nd Oct 1984\".\n",
"second": [42, 36, 156, 570, 636, 210, 198, 162, 30, 270] },

{"name": "Cyril Flat",
"summary": " Different views from and of a flat in Sara House, Larner Road, Erith.",
"description": " Outside Flats outside flats overlooking pub on corner. Sara House taken from road. More shots of Sara House. Now up in flat looking down. More views of urban planning. Older lady sitting under parasol in garden - underexposed. Garden views. Possible shots of grandchildren.",
"second": [72, 42, 12, 36, 192, 180, 126, 0, 96, 84] },

I. Data Collection

Data Sample

Folder_Path = “Datasets/ Videos/Clips”

A Good Name The Story of One Neighbourhood.mp4
A Journey Through Time.mp4
About Hampstead.mp4
Albany Mission and construction of Islington Central Methodist Church.mp4
Alfred Sansom and Family.mp4
Alice and Andrew Orpington.mp4
All On A Winters Day.mp4
Around amp About Bexley.mp4
Around Kingston 50s amp 60s.mp4
Around Richmond 73 79 1980.mp4
Around Stanwell.mp4
Barbican Regained.mp4
Beauty in the Borough.mp4
Bexleyheath Broadway Past and Present. mp4
Bookfair 89 Peckham Bookplace Bookfair Part 1.mp4
Brent A State of Mind.mp4
Bruce Shawcroft Films FebruaryMarchApril 1970 June 1971.mp4
Bruce Shawcroft Films MarchApril 1970. mp4
Camberwell Road.mp4
Christmas in Newham.mp4
Clubland Activities of the 1950s and 60s. mp4
Cranford Park and St Dunstans Church. mp4
Croydon in the 1970s Part 1.mp4
Croydon in the 1980s Part 2.mp4
Cyril Flat.mp4
Cyril Visits Lesney Farm Allotment and Pubs in Erith.mp4
Danson Park.mp4
Development of Romford in the Sixties. mp4
Discovering Newham.mp4
Dogs Life 3.mp4
Dogs Life Issue One.mp4
Dublin and District.mp4
Duckers n Divas.mp4
East Wickham and Welling.mp4
Erith Town Centre.mp4
Erith Town.mp4
Family together including Tottenham carnival 1964.mp4
Filming the Unusual.mp4
For What We Are About To Lose Carlton Mansions.mp4
Frequency 16.mp4
Georges Square Hoxton.mp4
Growing Up in Southwark Community Life in Southwark and Passport to the Future. mp4
Hackney Housing Estates.mp4
Hampstead Old amp New.mp4
Harrow Heritage.mp4
Hayes and Harlington.mp4
Here and There 1.mp4
High Hopes Voices from the Streets of Lewisham.mp4
Hounslow High St and Staines Road.mp4
If Walls Could Speak.mp4
In the Counties Around amp About Kent. mp4
Its Our Park a film about Southwark Park in the Summer of 2014.mp4
Kew Gardens during the War.mp4
Kingston in Decay.mp4
Kingston Movie Makers History of Walton. mp4
Lambeth c1972.mp4
Life of the Bicycle.mp4
Lights of London.mp4
Local Lore Just around the Corner Around amp About Bexley.mp4
Local Lore Now and Then.mp4
Local Lore Out for a Walk Around amp About Bexley.mp4
Local views of Friern Barnet.mp4
London Albert amp George Docks Chusan Wharf.mp4
London Albert Dock Dry Dock.mp4
London Borough.mp4
London Bridge.mp4
London Looking at London London No 6 Into the 70s.mp4
London Petticoat Lane Barrels Tower Bridge Docks Pros amp Whit Wapping. mp4
London Scenes c 1960.mp4
London Scenes Tower.mp4
London Sid amp Dave Hello amp Goodbye Guards at Bank Piccadilly Trafalgar Square Mall.mp4
London Sid Buck House Embankment. mp4
London Something Old Something New. mp4
London St Katherines Dock Tower Bridge Cable St.mp4
Looking for Sierra Leone.mp4
Loyalty amp Service Jimmy Butterworth amp Clubland.mp4
Manor Park.mp4
My Elephant.mp4
Newham in Widescreen.mp4
Notting Hill reel 1.mp4
Notting Hill reel 2.mp4
Notting Hill reel 3.mp4
Our First Film .mp4
Oxford Street.mp4
Piccadilly Circus Timelapse.mp4
Potter Allotment Lesney Farm.mp4
Pullenites.mp4
Richmond 1 19551968.mp4
Rickmansworth Week 1956.mp4
Royal Borough.mp4
Sample
Save The Rose.mp4
SE18 Impressions of a London Suburb. mp4
South Gate California.mp4
Southgate Tech From the Air.mp4
Southwark Cities series People and Planners.mp4
Southwarks Pride.mp4
Spring in Hackney.mp4
St Olaves on Horsleydown.mp4
Stately Homes of Newham.mp4
Street Scene Muswell Hill II.mp4
Street Scene.mp4
Sum of Us All Bede House Bermondsey and Rotherhithe 19382008.mp4
Swan Corner.mp4
Taken from East Lane Bridge.mp4
Thamesmead.mp4
The Castle.mp4
The Changing Face of Camberwell.mp4
The Development of Marks Gate Housing Estate.mp4
The Early History of the Isle of Dogs.mp4
The Elephant Never Forgets.mp4
The Great West Road and Bath Road Near Hounslow.mp4
The New Croydon.mp4
The New North Heath Library Snows of 1991.mp4
The Spike a place of peace for restless minds.mp4
The Tarn Eltham.mp4
The Thames Down River.mp4
The Thames Londons River.mp4
The Thames Somethings Going On.mp4
The Tower.mp4
The Urban District of Hayes and Harlington. mp4
The Villages of Stepney.mp4
This is London.mp4
Thompson Home Movie Grove Hotel Ickenham and the Hillingdon Show.mp4
Tour of RuislipNorthwood UDC.mp4
Upminster's Heritage.mp4
Uxbridge Redevelopment.mp4
Uxbridge.mp4
Various Scenes including Metropolitan Borough of Hackney Housing Estates.mp4
Views Outside Hackney Town Hall with Warwick Grove Flats.mp4
Ville de Sceaux.mp4
Welcome to Thamesmead.mp4
WELLBELOVED.mp4
What Future for Haringey.mp4
Where and What Is Paradise.mp4
Worthy Inheritance.mp4

DATASET 04: TEXT 02

This study is not merely a technical study on columns, thus, to add a layer to the understanding from a different dimension, 94 fiction books were collected. This dataset will expand the understanding of columns beyond architectural interpretations. It will initiate a poetic conversation that is linked to columns in a nuanced approach. It adds a layer to understanding a column that is not merely technical

Folder_Path = “Datasets/ Text 02/ Books02”

pg10002.epub	pg17134.epub	pg55-images.epub
pg10041.epub	pg1751.epub	pg5660.epub
pg10148-images.epub	pg17973.epub	pg5713-images.epub
pg10662.epub	pg18328.epub	pg5988-images.epub
pg10745-images.epub	pg19393.epub	pg604.epub
pg1076-images.epub	pg19959-images.epub	pg6050.epub
pg1077-images.epub	pg19973-images.epub	pg6582-images.epub
pg10806.epub	pg19976-images.epub	pg7477.epub
pg10882-images.epub	pg2060-images.epub	pg7838.epub
pg11097.epub	pg22566-images.epub	pg8129.epub
pg11283.epub	pg234-images.epub	pg8183-images.epub
pg11440.epub	pg2414.epub	pg831.epub
pg1152.epub	pg2565.epub	pg832.epub
pg11639-images.epub	pg2865.epub	pg8395-images.epub
pg11752.epub	pg288.epub	pg8715.epub
pg1251.epub	pg2885.epub	pg8771.epub
pg1252.epub	pg2892-images.epub	pg8778.epub
pg1267-images.epub	pg3055.epub	pg9488.epub
pg12747.epub	pg3261.epub	pg955.epub
pg12753-images.epub	pg3687.epub	pg956.epub
pg1311.epub	pg419.epub	pg957.epub
pg13486.epub	pg420.epub	pg958.epub
pg13664.epub	pg4282.epub	pg959-images.epub
pg13820.epub	pg4356.epub	pg960.epub
pg13821.epub	pg4358.epub	pg9608.epub
pg15551-images.epub	pg436.epub	pg961.epub
pg1557.epub	pg485-images.epub	pg964.epub
pg15948.epub	pg486.epub	pg9663.epub
pg1605-images.epub	pg5160.epub	pg9829.epub
pg16259-images.epub	pg517.epub	
pg16435.epub	pg518.epub	
pg169.epub	pg54-images.epub	

II. SELF-ORGANIZING MAPS

TEXT

1. The process of developing a self-organizing map for text data starts by defining the functions. A key note is that when processing text cosine_similarity should be instead of Euclidean distance.

```
def normalise(train, p):
    min_d = np.min(train)
    max_d = np.max(train)
    normalised_p = (p-min_d)/(max_d - min_d)
    return normalised_p

def denormalise(train, p):
    min_d = np.min(train)
    max_d = np.max(train)
    denormalised_p = p * (max_d - min_d) + min_d
    return denormalised_p

# Return the (g,h) index of the BMU in the grid
def find_BMU(SOM, x):
    somshape = SOM.shape
    simSom = SOM.reshape(-1, len(x))
    cos_sim = cosine_similarity([x], simSom).reshape(somShape[2])
    return np.unravel_index(np.argmax(cos_sim, axis=None), cos_sim.shape)

# Update the weights of the SOM cells when given a single training example
# and the model parameters along with BMU coordinates as a tuple
def update_weights(SOM, train_ex, learn_rate, radius_sq,
                    BMU_coord, step=3):
    g, h = BMU_coord
    if radius_sq < 1e-3:
        return SOM
    for i in range(max(0, g-step), min(SOM.shape[0], g+step)):
        for j in range(max(0, h-step), min(SOM.shape[1], h+step)):
            dist_sq = np.square(i-g) + np.square(j-h)
            dist_func = np.exp(-dist_sq / 2 / radius_sq)
            SOM[i,j,:] += learn_rate * dist_func * (train_ex - SOM[i,j,:])
```

Cell 13 of 34

* Defining necessary functions

2. Train doc2vec with a vector size of 300 and vectorize the text

```
paragraphs = [item['paragraph'] for item in columns_paragraphs]

with open('..../Datasets/Text/preprocessed2.json', "r") as file:
    # Preprocess the data
    tagged_data = [TaggedDocument(doc, [1]) for 1, doc in enumerate(preprocessed_paragraphs)]

# Train Doc2Vec model
max_epochs = 80
vec_size = 300
alpha = 0.025

model = Doc2Vec(vector_size=vec_size,
                 alpha=alpha,
                 min_alpha=0.05,
                 min_count=1,
                 dm=1)

model.build_vocab(tagged_data)
model.train(tagged_data,
            total_examples=model.corpus_count,
            epochs=model.epochs)

vectors = [model.infer_vector(text) for text in preprocessed_paragraphs]
```

Cell 41 of 48

* Training Doc2Vec and vectorize the text

3. Define a function to fit the PCA on all cells of the SOM.

```
def plot_som_pca(SOM):
    # Reshape the SOM grid into a 2D array
    som_flat = SOM.reshape(-1, SOM.shape[-1])

    # Perform PCA with 3 components
    pca = PCA(n_components=3)
    som_pca = pca.fit_transform(som_flat)

    # Normalize PCA components to [0, 1]
    som_pca_norm = (som_pca - som_pca.min(axis=0)) / (som_pca.max(axis=0) - som_pca.min(axis=0))

    # Create a U-matrix with PCA components as RGB values
    u_matrix_pca = som_pca_norm.reshape(SOM.shape[-1] + (3,))

    # Plot the U-matrix with PCA colors
    plt.figure(figsize=(4, 4))
    plt.imshow(u_matrix_pca)
    plt.title('PCA U-Matrix')
    plt.colorbar()
    plt.show()

def plot_epoch_results(epoch_number, QE_values, TE_values, SOM):
    # Plot QE and TE values
    plt.figure(figsize=(10, 4))
    plt.subplot(231)
    plt.plot(epoch_number, QE_values, label='QE')
    plt.plot(epoch_number, TE_values, label='TE')
    plt.xlabel('Epoch')
    plt.ylabel('Value')
    plt.title('QE and TE Values')
```

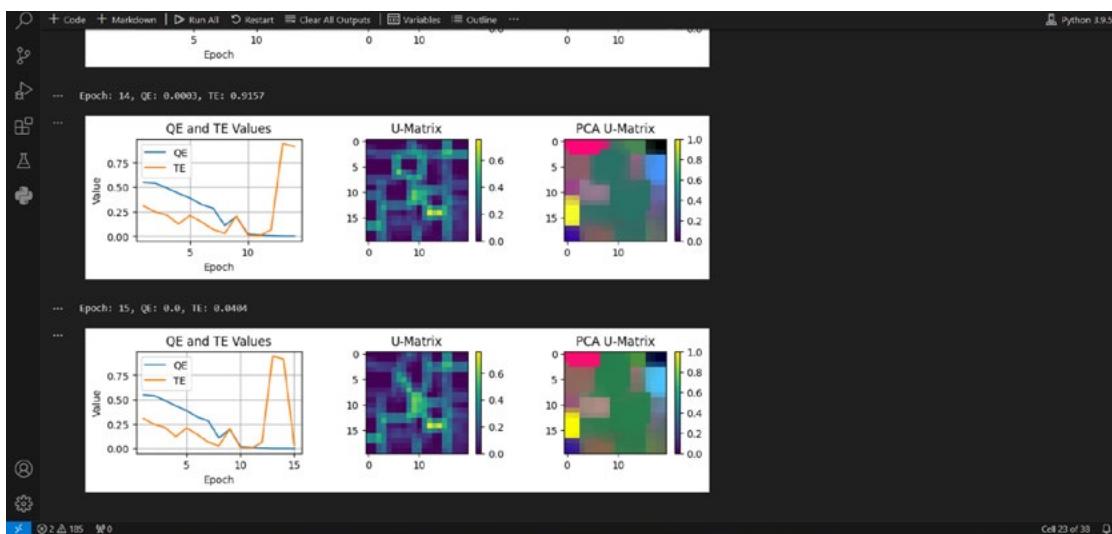
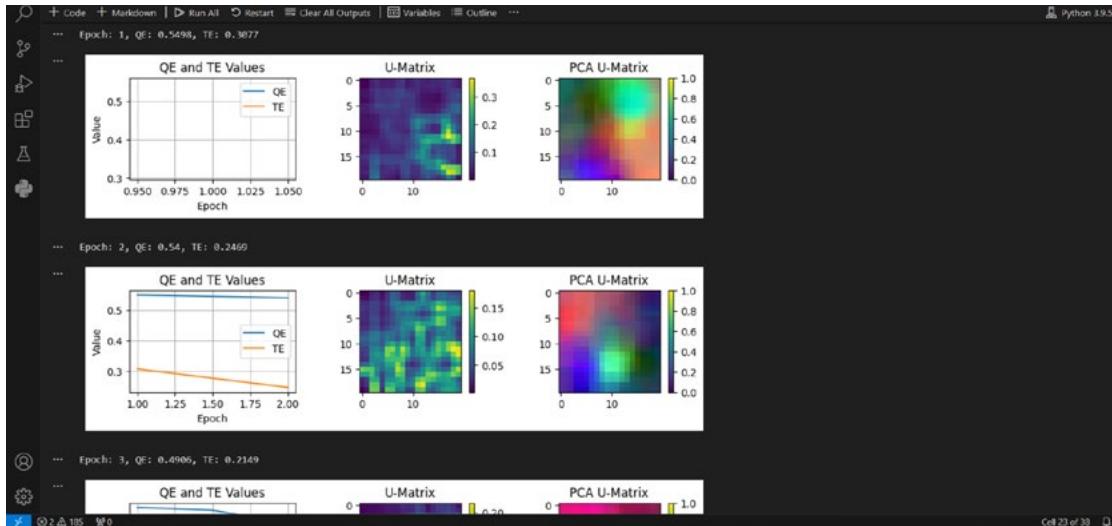
II. Self-organizing maps

4. Set the size of the training data for each epoch to 10,000 (line 24) and number of training epochs (line 21). The training code saves the SOM, QE, and TE after each epoch each a list to be recalled later. While the code is running, it prints the epoch number, QE and TE values. Also, it plots a graphs of the QE, TE, U-Matrix, and PCA U-Matrix after each epoch.

```

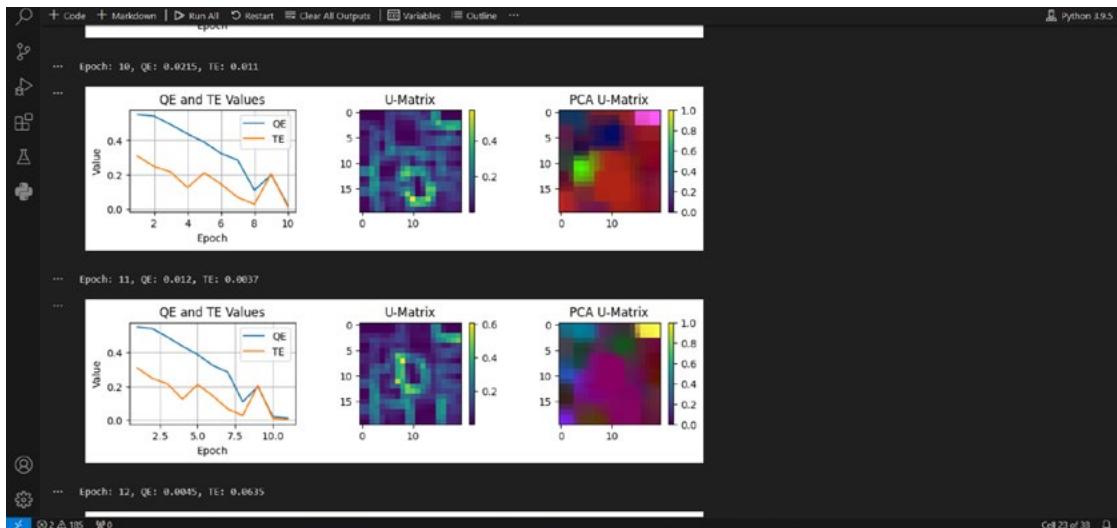
1 # Initialize SOM grid dimensions
2 m = 20
3 n = 20
4
5 # Initialize training data
6 train_data = vectors
7
8 # Initialize SOM grid with random weights
9 SOM = np.random.rand(m, n, train_data[0].shape[0])
10
11 total_epochs = 0
12 SOMs = []
13 QE_values = []
14 TE_values = []
15 epoch_numbers = []
16
17 # Convert train_data to numpy array
18 train_data_array = np.array(train_data)
19
20 # Training loop
21 for epochs_per_iter, i in zip([10], range(1)):
22     total_epochs += epochs_per_iter
23     num_sentences = len(train_data_array) # Number of sentences in the dataset
24     random_indices = np.random.choice(num_sentences, size=10000, replace=False)
25     train_data_subset = train_data_array[random_indices]
26
27     # Train the SOM for the specified number of epochs_per_iter
28     for epoch in range(1, epochs_per_iter + 1):
29         SOM = train(SOM, train_data_subset, learn_rate=0.1, radius_sq=12, epochs=1) # Train for 1 epoch
30
31     # Calculate QE and TE
32     QE = round(calculateQE(SOM, train_data_subset), 4)
33     TE = round(calculateTE(SOM, train_data_subset), 4)
34
35

```

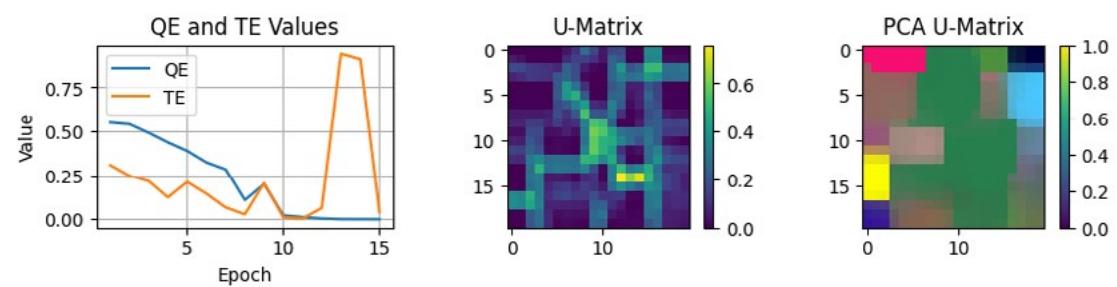


II. Self-organizing maps

As illustrated in the last printed graph, the QE and TE values were decaying till reaching epoch 11. Hence, epoch 10 is selected as it has the closest QE and TE values to zero but not zero. Although epoch 11 has values closer to zero, however these values are almost zero which impacts the quality of the SOM.

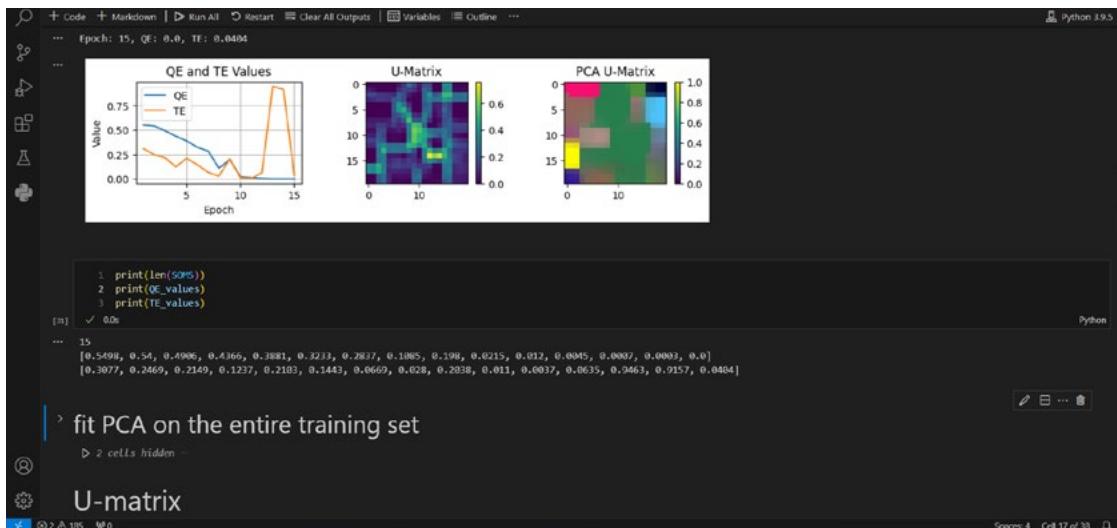


* Epoch 10 data



* Last epoch data graph

5. Print the length of the list “SOMS” and the content of “QE_values” and “TE_values” to ensure that the data of each epoch was saved during the training.



II. Self-organizing maps

6. Define a function to fit PCA on the entire training set.

The screenshot shows a Jupyter Notebook interface with a Python 3.9.5 kernel. The code cell contains a function definition for visualizing a Self-Organizing Map (SOM) as a 3D terrain. The code uses NumPy arrays for the lattice and Matplotlib for plotting. The background is black, and the plot features white axis labels and a gray color map for the terrain surface.

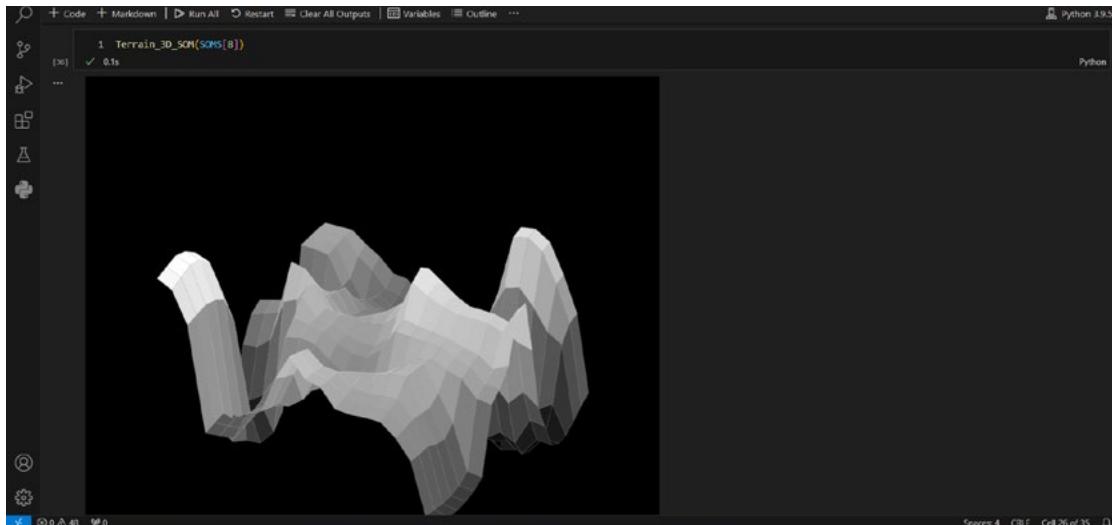
```
1 def Terrain_3D_SOM(lattice):
2     """
3         visualizes a self-organizing map (SOM) as a 3D terrain.
4     Parameters:
5         lattice (numpy.ndarray): the SOM lattice with shape (X, Y, z), where X and Y are the dimensions of the grid, and z is the dimensionality of the input space.
6     Returns:
7         None (displays the visualization).
8     """
9
10    X, Y, Z = lattice.shape
11
12    # Create a Figure with a black background
13    fig = plt.figure(figsize=(10, 10), facecolor='black')
14
15    ax = fig.add_subplot(111, projection='3d')
16
17    # Create meshgrid for the terrain surface
18    xx, yy = np.meshgrid(range(X), range(Y))
19
20    # Plot the terrain surface using the values of the weight vectors
21    ax.plot_surface(xx, yy, lattice.sum(axis=2), cmap='gray', edgecolor='white', linewidth=0.1)
22
23    # Set the background color to black
24    ax.set_facecolor('black')
25
26    # Set axis labels color to white
27    ax.xaxis.label.set_color('white')
28    ax.yaxis.label.set_color('white')
29    ax.zaxis.label.set_color('white')
30
31    # Set axis ticks color to white
32    ax.xaxis._axinfo['label'].set_color('white')
33    ax.yaxis._axinfo['label'].set_color('white')
34    ax.zaxis._axinfo['label'].set_color('white')
```

7. Define a function to plot the SOM as a 3D terrain

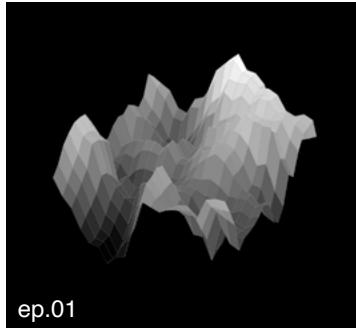
This screenshot is identical to the one above, showing the same code for visualizing a SOM as a 3D terrain. The code defines a function `Terrain_3D_SOM` that takes a NumPy array `lattice` as input and plots it as a 3D surface with a black background and white axis labels.

```
1 def Terrain_3D_SOM(lattice):
2     """
3         visualizes a self-organizing map (SOM) as a 3D terrain.
4     Parameters:
5         lattice (numpy.ndarray): the SOM lattice with shape (X, Y, z), where X and Y are the dimensions of the grid, and z is the dimensionality of the input space.
6     Returns:
7         None (displays the visualization).
8     """
9
10    X, Y, Z = lattice.shape
11
12    # Create a Figure with a black background
13    fig = plt.figure(figsize=(10, 10), facecolor='black')
14
15    ax = fig.add_subplot(111, projection='3d')
16
17    # Create meshgrid for the terrain surface
18    xx, yy = np.meshgrid(range(X), range(Y))
19
20    # Plot the terrain surface using the values of the weight vectors
21    ax.plot_surface(xx, yy, lattice.sum(axis=2), cmap='gray', edgecolor='white', linewidth=0.1)
22
23    # Set the background color to black
24    ax.set_facecolor('black')
25
26    # Set axis labels color to white
27    ax.xaxis.label.set_color('white')
28    ax.yaxis.label.set_color('white')
29    ax.zaxis.label.set_color('white')
30
31    # Set axis ticks color to white
32    ax.xaxis._axinfo['label'].set_color('white')
33    ax.yaxis._axinfo['label'].set_color('white')
34    ax.zaxis._axinfo['label'].set_color('white')
```

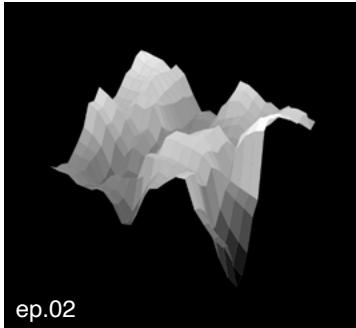
8. Plot the SOM as 3D terrain



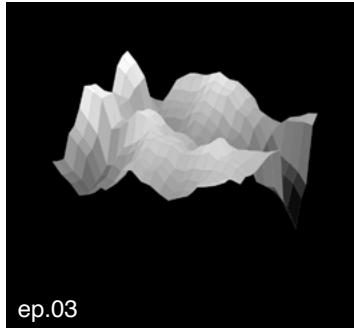
II. Self-organizing maps



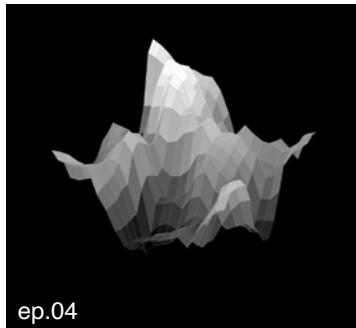
ep.01



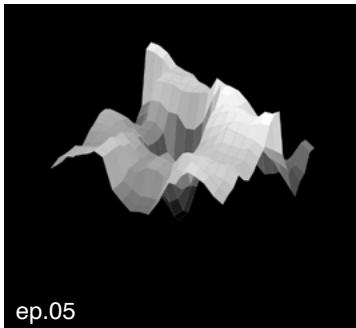
ep.02



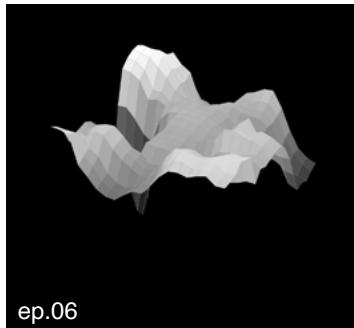
ep.03



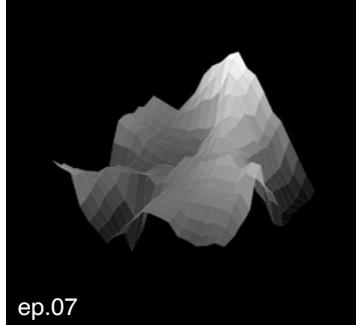
ep.04



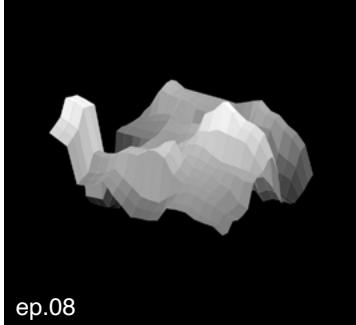
ep.05



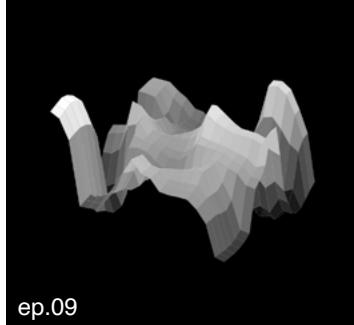
ep.06



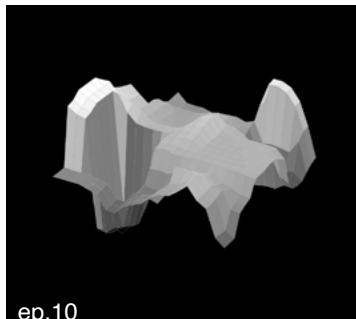
ep.07



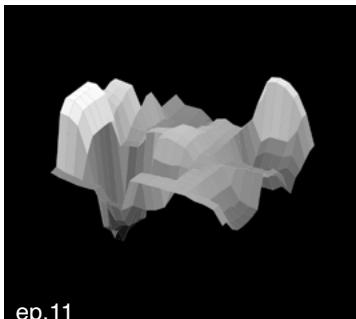
ep.08



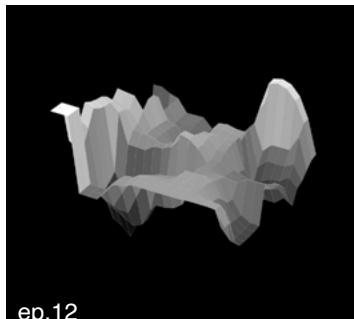
ep.09



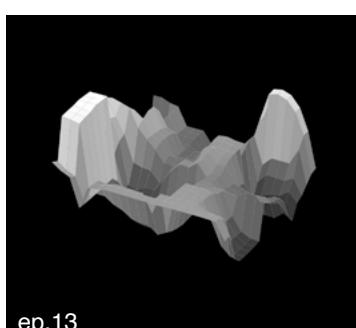
ep.10



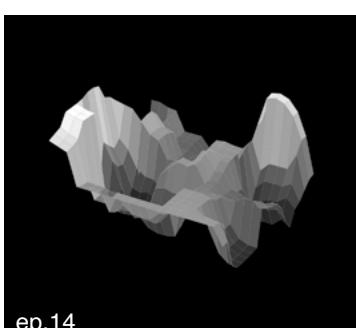
ep.11



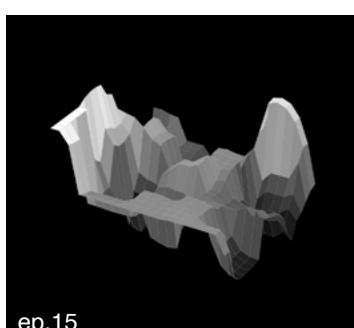
ep.12



ep.13



ep.14



ep.15

* One of the provocative comments we received during term 02 final crit is the uncovered spatial potential of SOMs. Based on that, I tried to translate the SOM into a form that can be translated physically. This does not mean this physical form can be turned into architecture. Rather, it has the potential to represent and visualize SOMs in physical models that can be 3D printed to explore a new attribute of SOMs further than 2D representations.

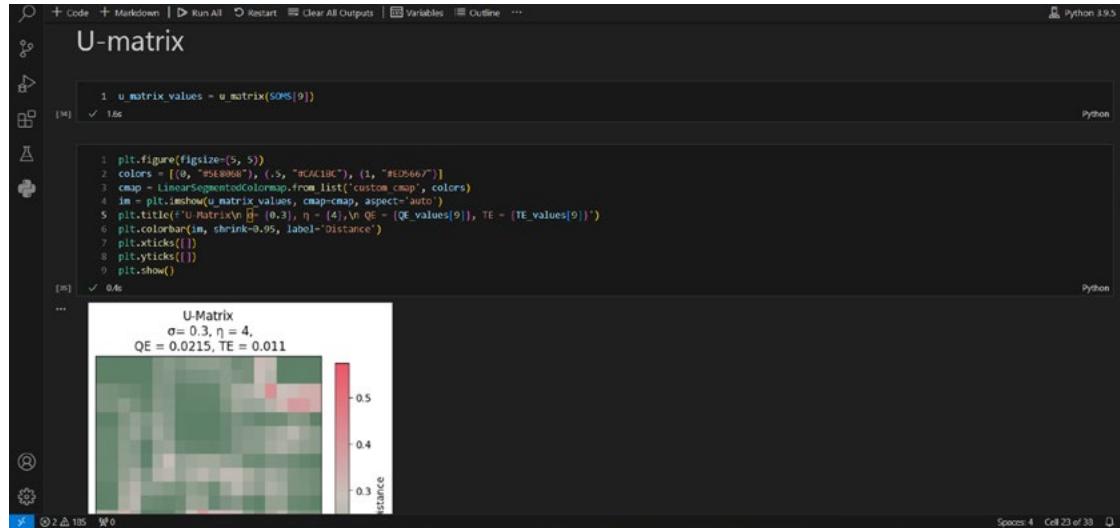
II. Self-organizing maps

9. Plot the U-Matrix of the SOM at epoch 10

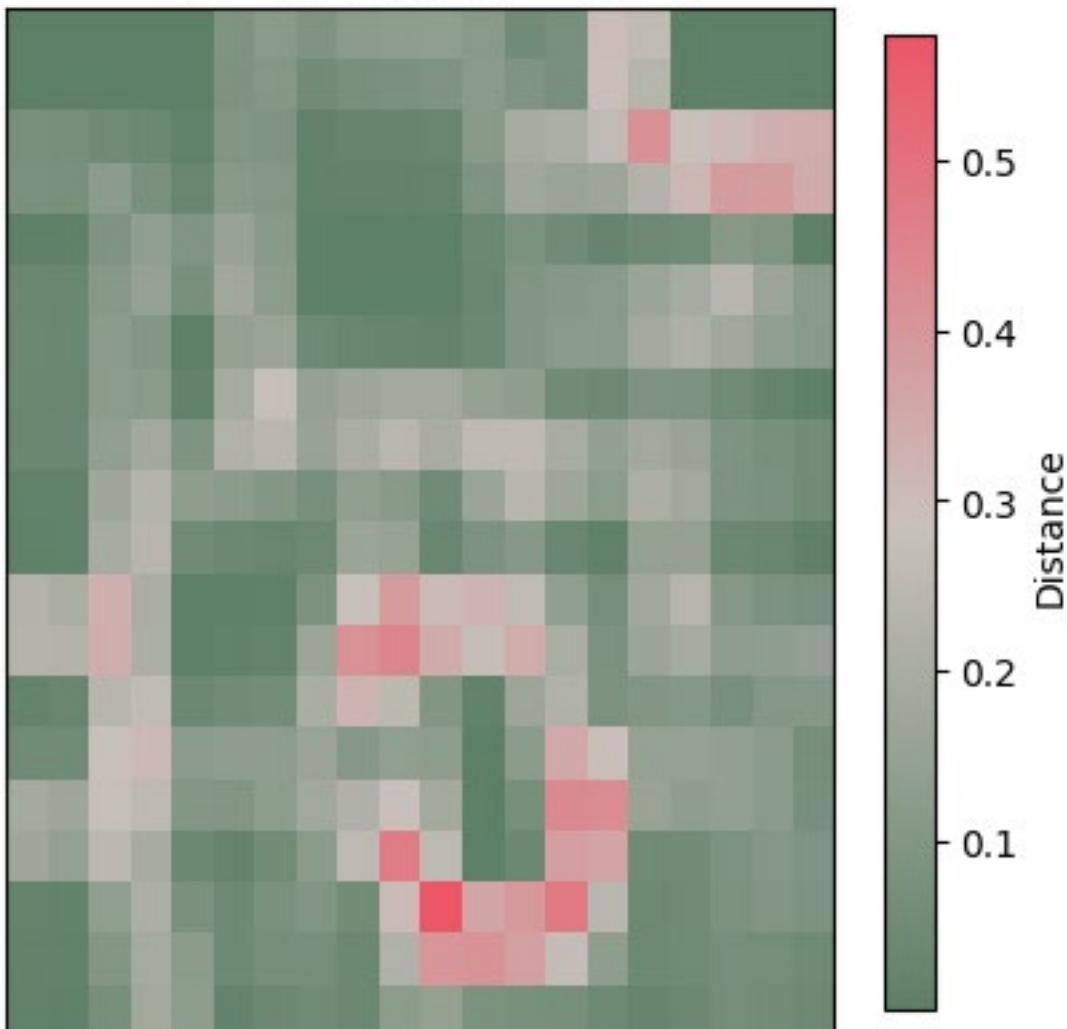
U-matrix

```
1 u_matrix_values = u_matrix(SOMs[9])
2
3 plt.figure(figsize=(5, 5))
4 colors = [(0, "#5E8B0B"), (.5, "#C1E9C"), (1, "#D5D667")]
5 cmap = LinearSegmentedColormap.from_list('custom_cmap', colors)
6 im = plt.imshow(u_matrix_values, cmap=cmap, aspect='auto')
7 plt.title(f'U-Matrix\n\sigma= {0.3}, \eta = {4},\nQE = {QE_values[9]}, TE = {TE_values[9]}')
8 plt.colorbar(im, shrink=0.95, label='Distance')
9 plt.xticks([])
10 plt.yticks([])
11 plt.show()
```

U-Matrix
 $\sigma = 0.3, \eta = 4,$
 $QE = 0.0215, TE = 0.011$



U-Matrix
 $\sigma = 0.3, \eta = 4,$
 $QE = 0.0215, TE = 0.011$



II. Self-organizing maps

10. Print the sentences of each cell on the U-Matrix

The screenshot shows a Jupyter Notebook interface with two code cells and a resulting visualization.

Code Cell 1:

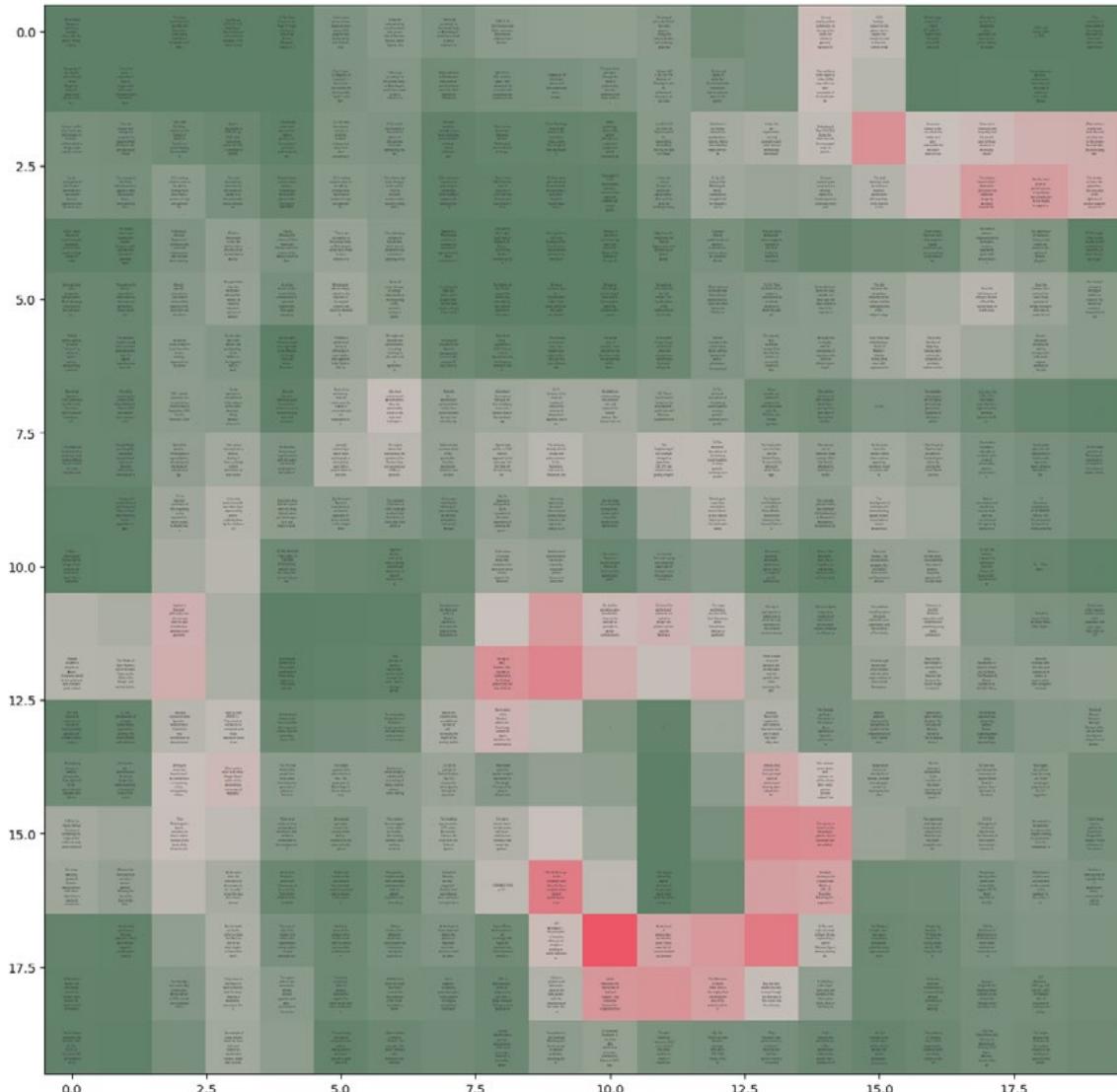
```
1 # Initialize an empty dictionary to store sentences associated with each neuron
2 neuron_sentence_mapping = {(i, j): [] for i in range(som.shape[0]) for j in range(som.shape[1])}
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Code Cell 2:

```
1 import textwrap
2
3 # Iterate through each sentence and find its idx, then append it to the corresponding neuron's list
4 for idx, sentence in enumerate(paragraphs):
5     vector = vectors[idx]
6     lns = find_idx(SOMs[0], vector)
7     neuron_sentence_mapping[lns].append(sentence)
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Result:

A 18x18 grid U-Matrix plot where each cell contains a wrapped sentence. The plot uses a color scale from green (lower distance) to red (higher distance). The sentences are printed in black font within their respective grid cells.



II. Self-organizing maps

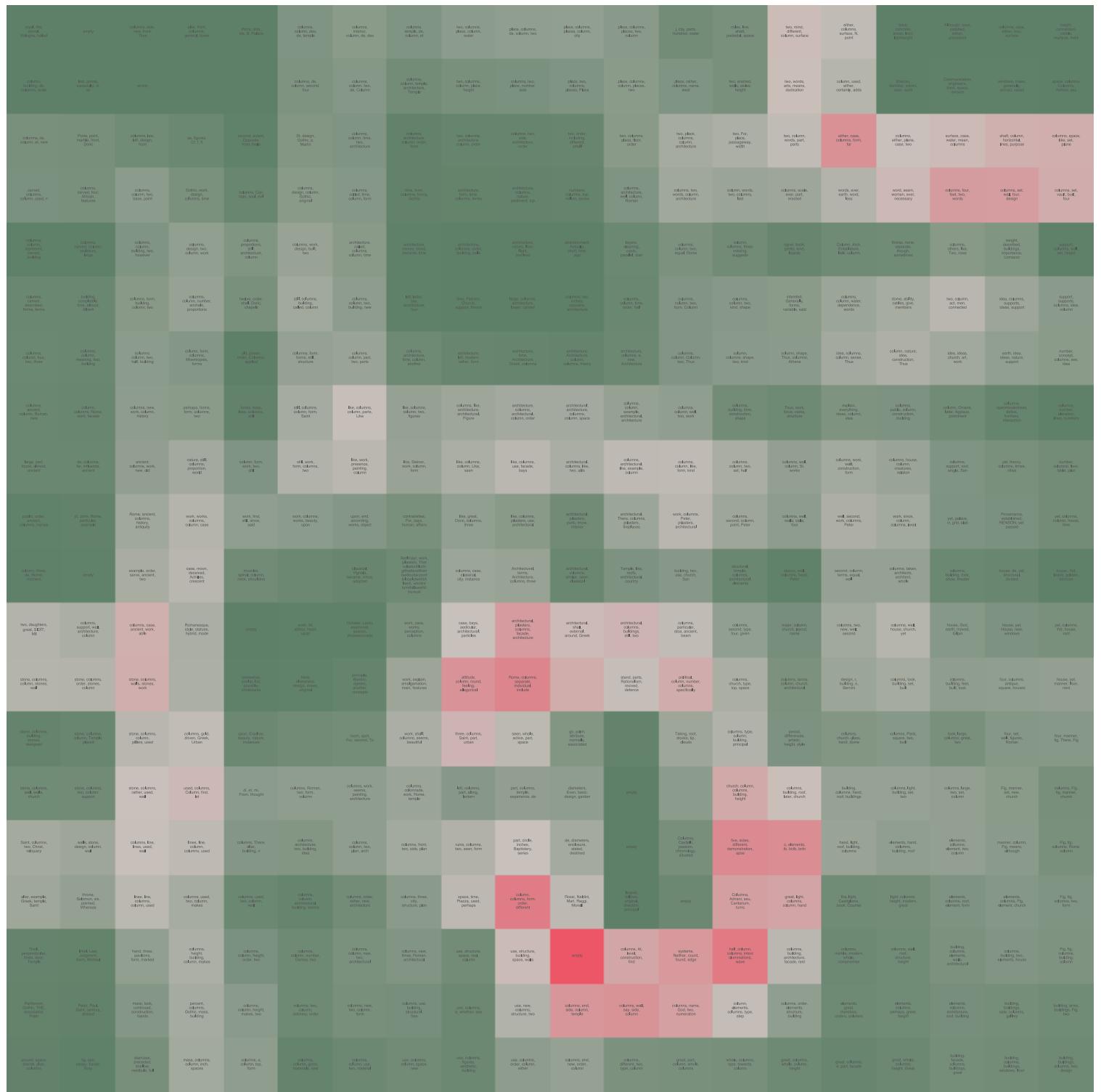
11. Define a function to find the common concepts of sentences in each cell

```
1 the_common_concepts = []
2 The first 6 common.concepts = []
3 The_wrapped_text = []
4
5 # Display first 6 common concepts for each grid cell
6 for (i, j), neuron in neuron_sentence_mapping.items():
7     # Reduce font size
8     fontsize = 2
9     # Initialize an empty list to store common concepts
10    common_concepts = []
11
12    # Check if neuron is empty
13    if not neuron:
14        common_concepts.append('empty')
15    else:
16        concepts = find_common_concepts(neuron)
17        common_concepts.extend(concepts)
18
19    print(i,j)
20    print(common_concepts)
21
22 The_common_concepts.append(common_concepts.copy())
Python
... 0.0
['royal', 'His', 'arrival', 'Bologna', 'hailed', 'triumphal', 'entry', 'altar', 'throne', 'glory', 'second', 'phase', 'officially', 'began', 'Abel', 'Francois', 'Poisson', 'marquis', 'empty']
0.1
0.2
['columns', 'side', 'new', 'front', 'thus', 'design', 'German', 'found', 'work', 'say', 'number', 'around', 'different', 'column', 'st', 'two', 'seen', 'art', 'Borromini', 'Column', 'is', 'altar', 'front', 'columns', 'general', 'tower', 'die', 'ca', 'de', 'chancel', 'choir', 'Maria', 'church', 'floor', 'louis', 'words', 'charge', 'means', 'actual', 'figures', 'average', 'Porta', 'side', 'die', 'B', 'Palace', 'Aurea', 'wider', 'overall', 'together', 'Ferrea', 'iron', 'Gate', 'court', 'Juftice', 'One', 'Ancient', 'Octagonal', 'Towers', 'Piazza', 'Market'
Spaces 4 Cell 30 of 37
```

12. Print the common concept of each cell on the SOM

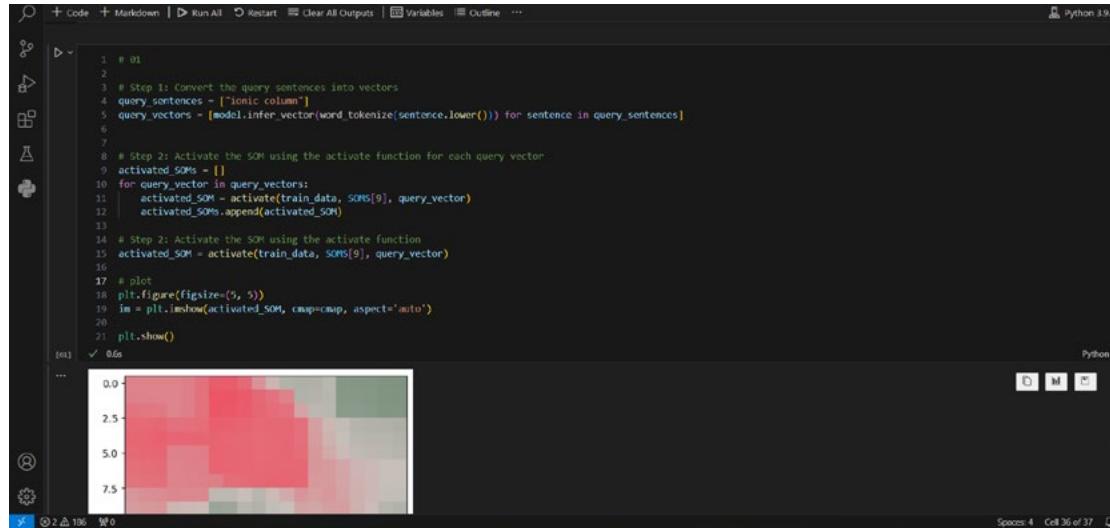
```
1 import textwrap
2
3 # Plot the u-Matrix and display the mapped sentences for each neuron
4 plt.figure(figsize=(10, 10))
5 im = plt.imshow(u_matrix_values, cmap='cmap', aspect='auto')
6 # plt.title('U-Matrix\no = (0,3), n = (4),\nDE = (QE), TE = (TE)')
7 plt.colorbar(im, orientation='horizontal', label='Distance', pad=0.01)
8
9 # Display only the first 3 words of each list in each grid cell
10 for (i, j), words_list in neuron_common.concepts.items():
11     if words_list: # check if there are words associated with the neuron
12         # Reduce font size
13         fontsize = 3
14         # Select the words associated with the neuron
15         first_3_words = ", ".join(words_list[:3]) # take the first three words
16         # Wrap the words to fit within the grid cell
17         wrapped_words = "\n".join(textwrap.wrap(first_3_words, width=15))
18         plt.text(j, i, wrapped_words, ha='center', va='center', color='mssssss', fontsize=fontsize, fontname='Helvetica')
19
20 # Force aspect ratio to be equal
21 plt.gca().set_aspect('equal', adjustable='box')
22
23 # plt.savefig('SOM_commonconcepts_9.pdf', format='pdf')
24
25 # Show the plot
26 plt.tight_layout()
27 plt.show()
Python
... 0.0
Spaces 4 Cell 30 of 37
```

II. Self-organizing maps



II. Self-organizing maps

13. Activate the SOM

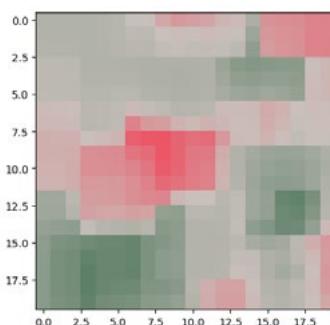
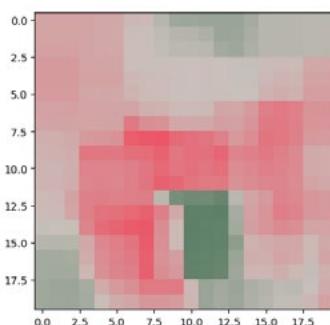
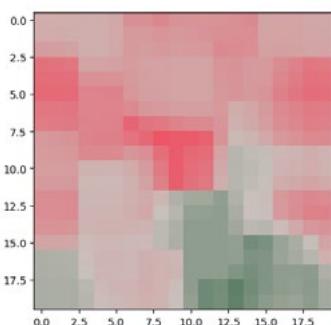
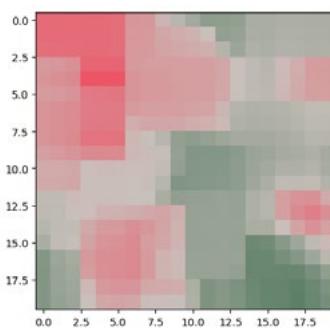
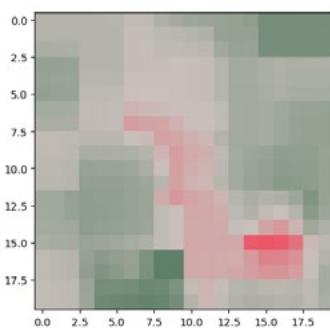
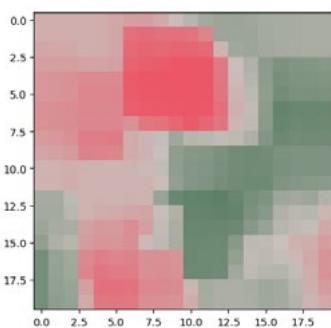
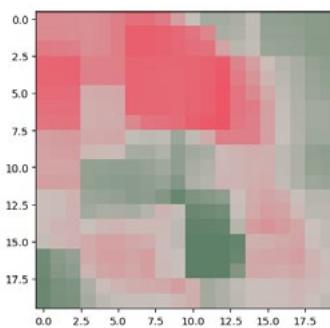
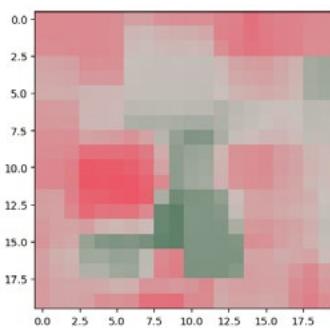
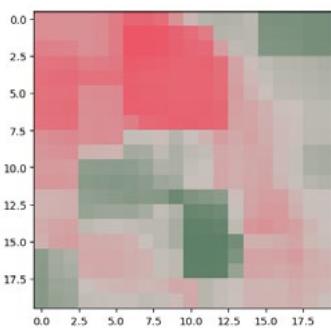


The screenshot shows a Jupyter Notebook interface with a Python code cell and its output. The code activates a SOM using a query vector and plots the results. The output is a heatmap titled 'SOM activations'.

```
1 # 01
2
3 # Step 1: Convert the query sentences into vectors
4 query_sentences = ['Ionic column']
5 query_vectors = [model.infer_vector(word_tokenize(sentence.lower())) for sentence in query_sentences]
6
7
8 # Step 2: Activate the SOM using the activate function for each query vector
9 activated_SOMs = []
10 for query_vector in query_vectors:
11     activated_SOM = activate(train_data, SOMs[9], query_vector)
12     activated_SOMs.append(activated_SOM)
13
14 # Step 2: Activate the SOM using the activate function
15 activated_SOM = activate(train_data, SOMs[9], query_vector)
16
17 # plot
18 plt.figure(figsize=(5, 5))
19 im = plt.imshow(activated_SOM, cmap=cmap, aspect='auto')
20
21 plt.show()
```

(in) ✓ 0.6s

...
SOM activations



II. Self-organizing maps

IMAGES

1. Define the functions, with processing images Euclidean distance is used instead of cosine_similarity.

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code. The code defines two functions: `euclidean` and `u_matrix`. The `euclidean` function calculates the Euclidean distance between two vectors `a` and `b`. The `u_matrix` function builds a U-matrix on top of a trained lattice. It iterates through each cell in the lattice, calculating the Euclidean distance to its neighbors. The code uses `np.linalg.norm` for vector norms and `np.empty` to initialize the U-matrix. The notebook interface includes tabs for Code, Markdown, Run All, and Restart, along with output sections for variables and outline.

```
def euclidean(a, b):
    return np.linalg.norm(a-b)

def u_matrix(lattice):
    """Builds a U-matrix on top of the trained lattice.

    Parameters
    ---
    lattice : list
        The SOM generated lattice.

    Returns
    ---
    the lattice of the shape (n,c):
    R - number of rows; C - number of columns;
    """
    X, Y, Z = lattice.shape
    u_values = np.empty((X,Y), dtype=np.float64)

    for y in range(Y):
        for x in range(X):
            current = lattice[x,y]
            dist = 0
            num_neigh = 0
            # left
            if x > 0:
                middle
                vec = lattice[x-1,y]
                dist += euclidean(current, vec)
                num_neigh += 1
            if y > 0:
                up
                vec = lattice[x-1, y-1]
```

* Defining necessary functions

2. Load mobilenet as a model and extract the features of the 4411 images

The screenshot shows a Jupyter Notebook interface with three code cells. The first cell imports TensorFlow and loads the MobileNet model. The second cell lists all images in a directory and processes them using the model to extract features. The third cell shows the progress of feature extraction. The notebook interface includes tabs for Code, Markdown, Run All, and Restart, along with output sections for variables and outline.

```
model = tf.keras.applications.MobileNet(
    # The 3 is the three dimensions of the input: r,g,b,
    input_shape=(224, 224, 3),
    include_top=False,
    pooling='avg'
)
✓ 2Ds
```

```
Outputs are collapsed --
```

```
1 imgs = os.listdir('../..//datasets/Images/Images')
✓ 0Ds
```

```
1 features = []
2 for n in imgs:
3     path = os.path.join('../..//datasets/Images/Images', n)
4     f = processImage(path, model)
5     features.append(f)
...
1/1 [=====] - is 1s/step
1/1 [=====] - is 2ms/step
1/1 [=====] - is 7ms/step
1/1 [=====] - is 66ms/step
1/1 [=====] - is 64ms/step
1/1 [=====] - is 59ms/step
1/1 [=====] - is 65ms/step
```

3. Define a function to fit the PCA on all cells of the SOM.

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code. The code defines two functions: `plot_som_pca` and `plot_epoch_results`. The `plot_som_pca` function flattens the training data, performs PCA with 3 components, reshapes the SOM grid into a 2D array, and then plots the U-matrix with PCA colors. The `plot_epoch_results` function plots QE and TE values for a specific epoch. The notebook interface includes tabs for Code, Markdown, Run All, and Restart, along with output sections for variables and outline.

```
def plot_som_pca(SOM, train_data):
    # Flatten the training data
    train_data_flat = train_data.reshape(train_data.shape[0], -1)

    # Perform PCA with 3 components on the Flattened training data
    pca = PCA(n_components=3)
    pca.fit(train_data_flat)

    # Reshape the SOM grid into a 2D array
    som_flat = SOM.reshape(-1, SOM.shape[-1])

    # Transform SOM cells to PCA space
    som_pca = pca.transform(som_flat)

    # Normalize PCA components to [0, 1]
    som_pca_norm = (som_pca - som_pca.min(axis=0)) / (som_pca.max(axis=0) - som_pca.min(axis=0))

    # Create a U-matrix with PCA components as RGB values
    u_matrix_pca = som_pca_norm.reshape(SOM.shape[:-1] + (3,))

    # Plot the U-matrix with PCA colors
    plt.imshow(u_matrix_pca)
    plt.title('PCA U-Matrix')
    plt.colorbar()
    plt.show()

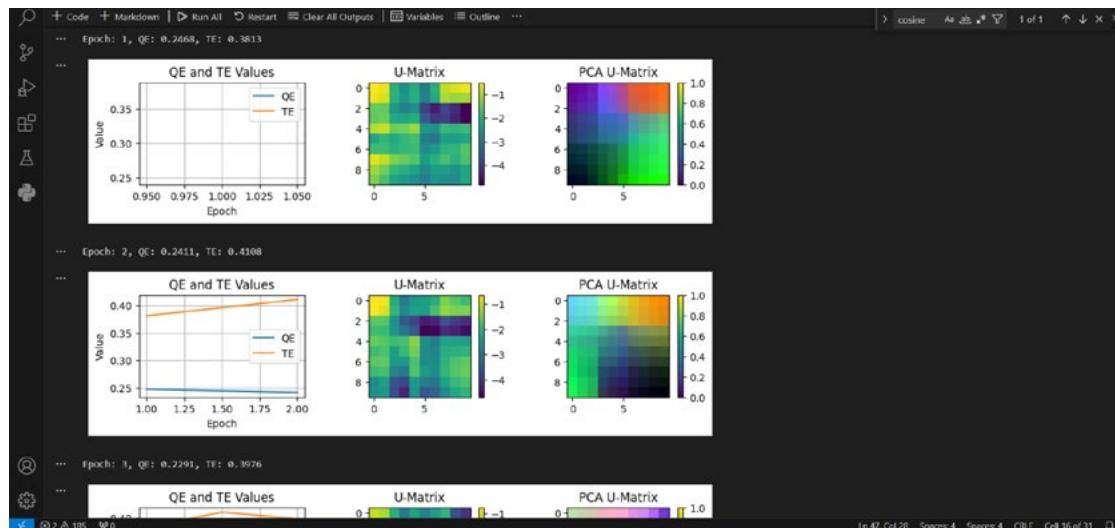
def plot_epoch_results(epoch_number, QE_values, TE_values, SOM):
    # Plot QE and TE values
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2)
```

II. Self-organizing maps

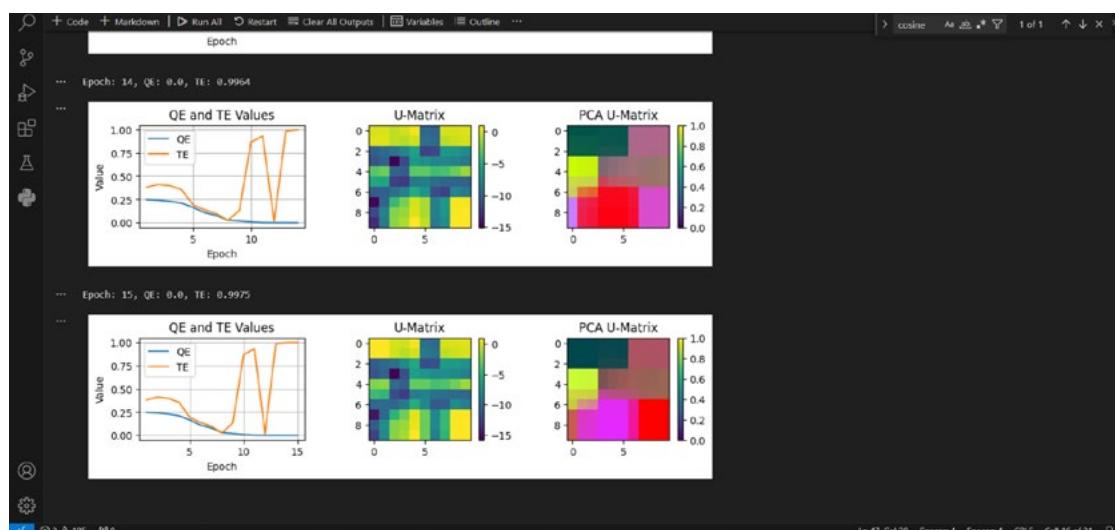
4. Train a 10x10 SOM using the extracted features for 15 epochs.

```
1 # Initialize SOM grid dimensions
2 m = 30
3 n = 20
4
5 # Initialize training data
6 train_data = features
7
8 # Initialize SOM grid with random weights
9 SOM = np.random.rand(m, n, train_data[0].shape[0])
10
11 epochs = 100
12 total_epochs = 0
13 SOWs = []
14 QEs = []
15 TEs = []
16 epoch_numbers = []
17
18 # Convert train data to numpy array
19 train_data_array = np.array(train_data)
20
21 # Training loop
22 for epochs_per_iter, i in zip([10], range(1)):
23     total_epochs += epochs_per_iter
24
25     # Train the SOM for the specified number of epochs_per_iter
26     for epoch in range(i, epochs_per_iter + i):
27         SOM = train_SOM(SOM, train_data_array, learn_rate=0.1, radius_sq=12, epochs=1) # Train for 1 epoch
28
29         # calculate QE and TE
30         QE = round(calculateQE(SOM, train_data_array), 4)
31         TE = round(calculateTE(SOM, train_data_array), 4)
32
33         # Print epoch number and QE, TE values
34         print(f"Epoch: {total_epochs - epochs_per_iter + epoch}, QE: {QE}, TE: {TE}")
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
```

* Plots and prints after each epoch

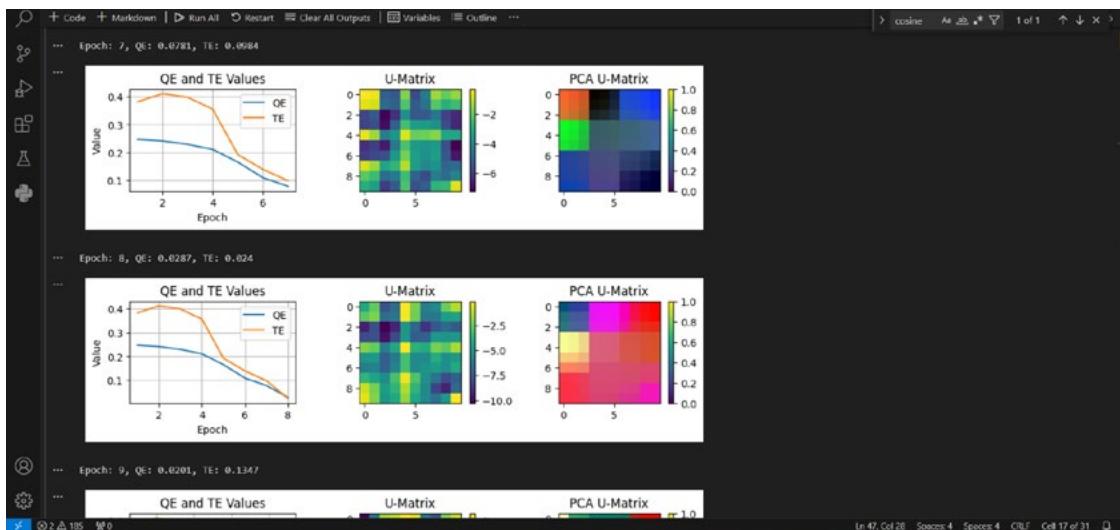


* Last epoch graph

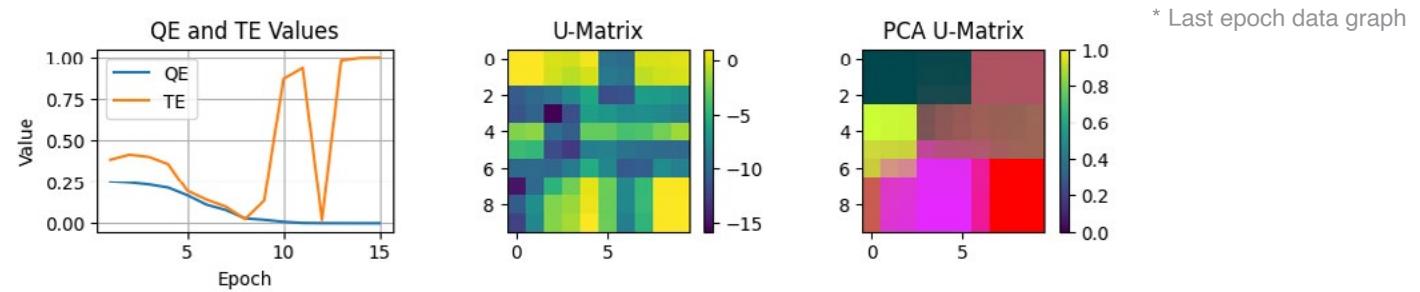


II. Self-organizing maps

5. Epoch 8 is selected as it has the QE and TE values close to zero. Also, it is noteworthy to mention that the QE and TE values spiked after this epoch as illustrated by the graph.



* Epoch 8 data



* Last epoch data graph

6. Print the length of the list “SOMS” and the content of “QE_values” and “TE_values” to ensure that the data of each epoch was saved during the training.

```
1 print(len(SOMS))
2 print(QE_values)
3 print(TE_values)
```

The output of the code execution is:

```
15
[0.2668, 0.2411, 0.2291, 0.2187, 0.1652, 0.1093, 0.0781, 0.0287, 0.0281, 0.0085, 0.0816, 0.0005, 0.0001, 0.0, 0.0]
[0.3813, 0.4108, 0.3976, 0.3552, 0.192, 0.1392, 0.0984, 0.026, 0.1347, 0.8651, 0.9302, 0.0193, 0.9812, 0.9964, 0.9975]
```

Below the code cell, there are two collapsed sections:

- U-Matrix**: Shows a heatmap of the U-Matrix.
- Activate**: Shows a heatmap of the activation matrix.

II. Self-organizing maps

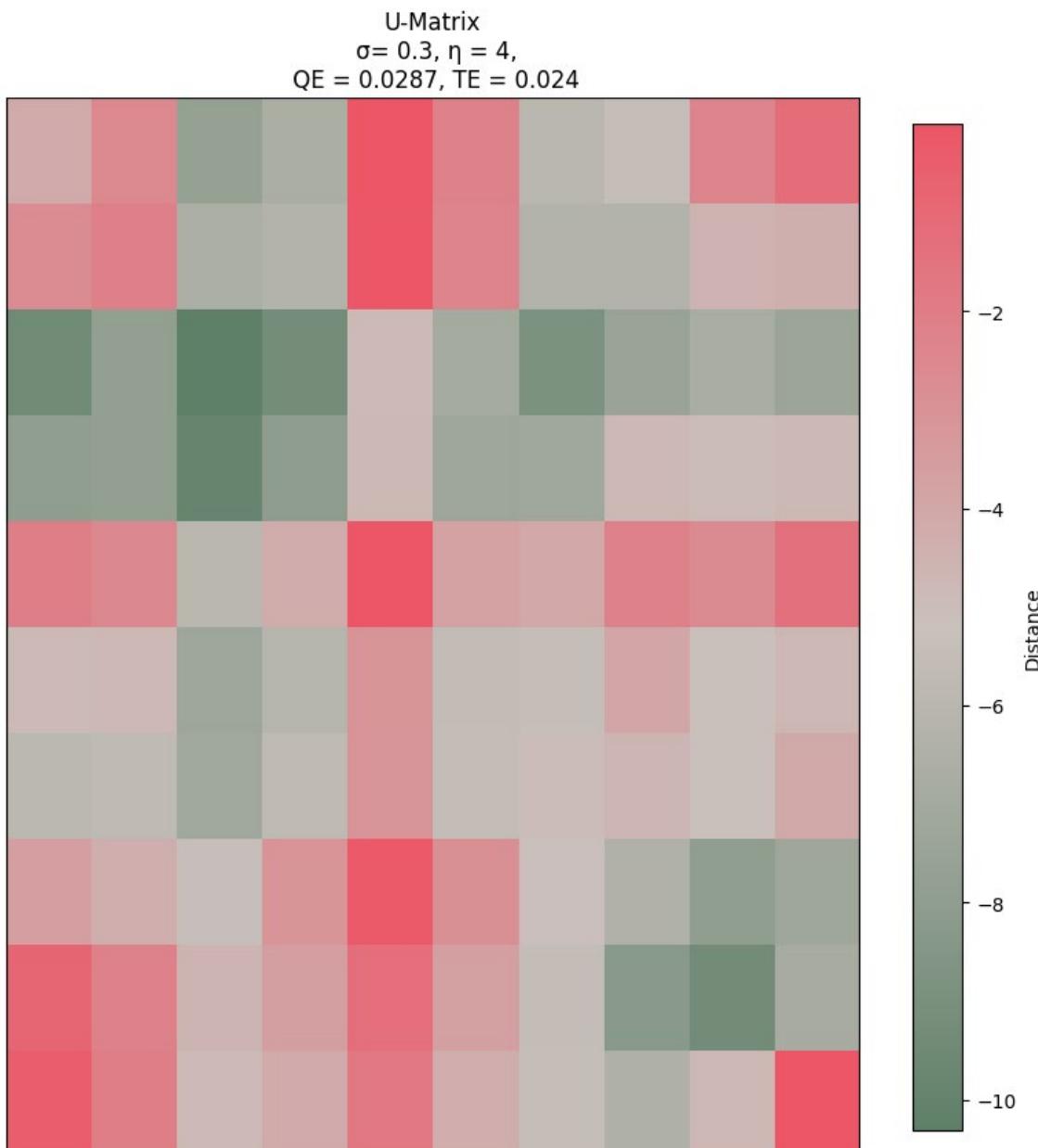
7. Plot the U-Matrix of the SOM at epoch 8

U-Matrix

```
1 u_matrix_values = u_matrix(som[7])
2
3 plt.figure(figsize=(10, 10))
4 colors = [(0, "#E80008"), (.5, "#CAC1BC"), (1, "#ED5667"), (.5, "#CAC1BC"), (1, "#E80008"), (.5, "#5E8008"), (1, "#5E8008"), (.5, "#5E8008"), (1, "#5E8008"), (.5, "#5E8008"), (1, "#5E8008")]
5 cmap = LinearSegmentedColormap.from_list('custom_cmap', colors)
6 im = plt.imshow(u_matrix_values, cmap=cmap, aspect='auto')
7 plt.title("U-Matrix\nσ= 0.3, η = 4,\nQE = 0.0287, TE = 0.024")
8 plt.colorbar(im, shrink=0.95, label="Distance")
9 plt.xticks([])
10 plt.yticks([])
11 plt.show()
```

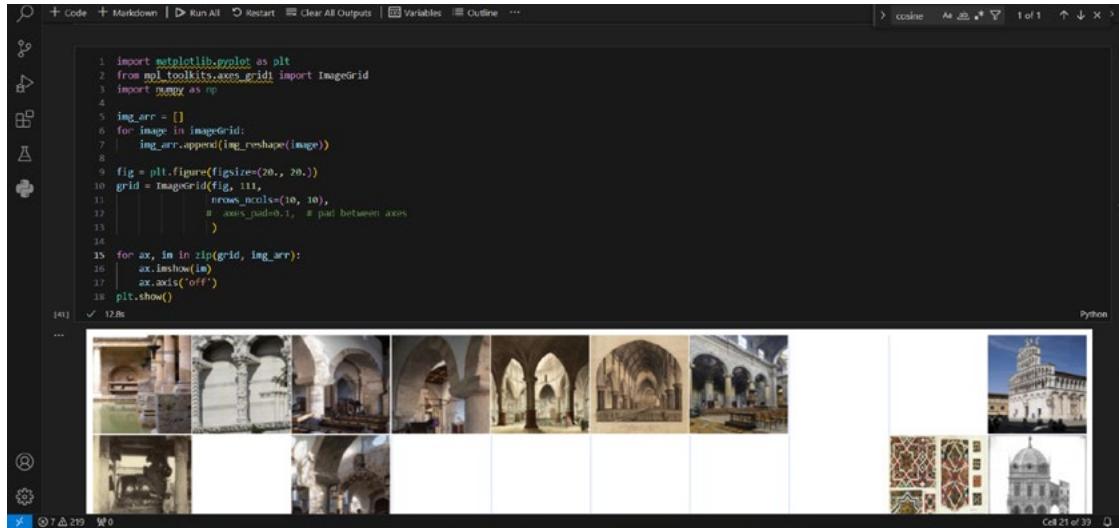
0.0s

U-Matrix
σ= 0.3, η = 4,
QE = 0.0287, TE = 0.024



II. Self-organizing maps

8. Plot the images on the SOM



A screenshot of a Jupyter Notebook interface. The top bar shows tabs for Code, Markdown, Run All, Restart, Clear All Outputs, Variables, and Outline. The code cell contains Python code for plotting images on a grid:

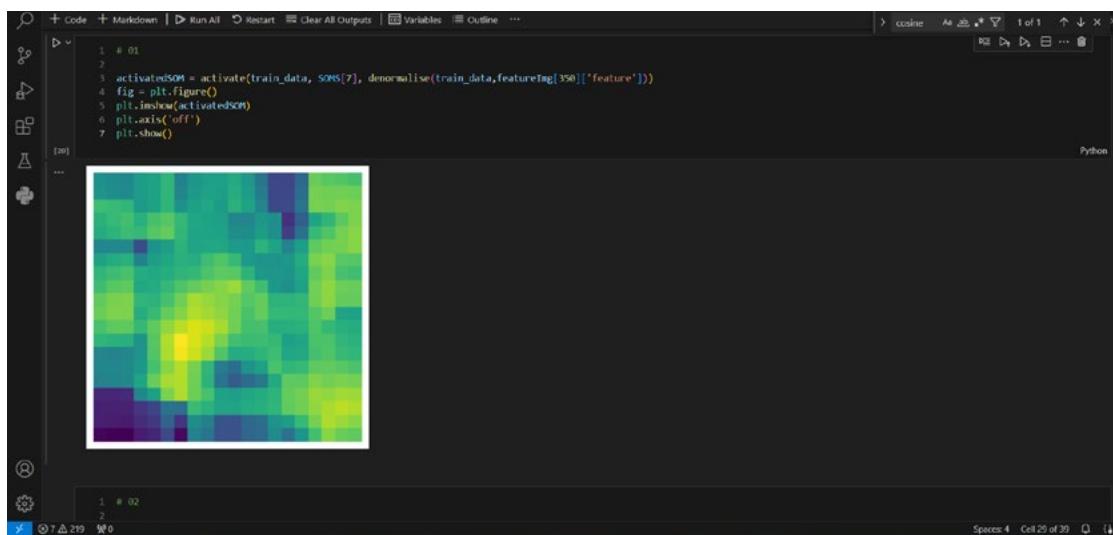
```
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.axes_grid import ImageGrid
3 import numpy as np
4
5 img_arr = []
6 for image in imageGrid:
7     img_arr.append(img_reshape(image))
8
9 fig = plt.figure(figsize=(20., 20.))
10 grid = ImageGrid(fig, 111,
11                  nrows_ncols=(10, 10),
12                  axes_pad=0.1, # pad between axes
13                  )
14
15 for ax, im in zip(grid, img_arr):
16     ax.imshow(im)
17     ax.axis('off')
18 plt.show()
```

The bottom right corner of the code cell shows "Cell 21 of 39".



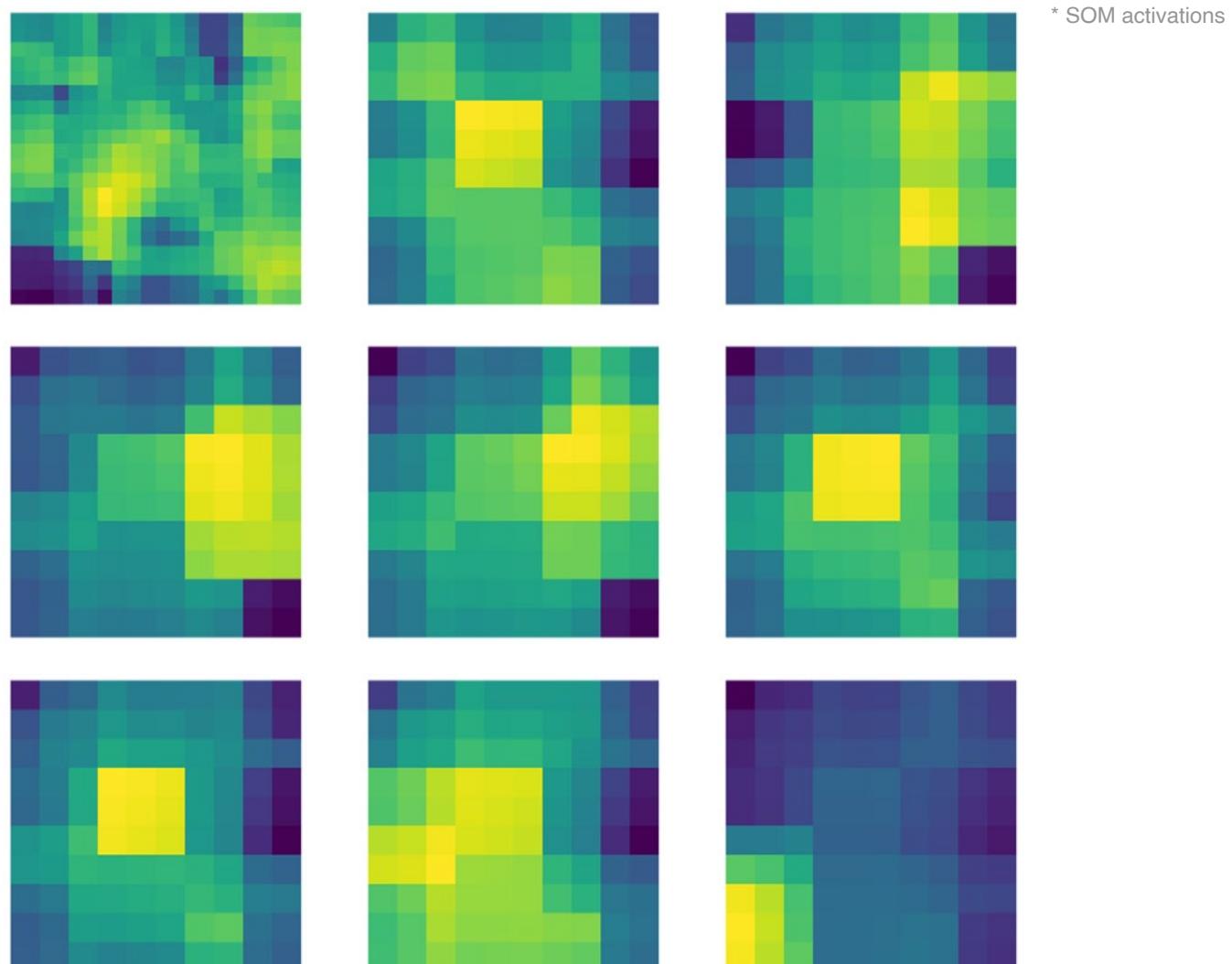
II. Self-organizing maps

9. Activate the SOM



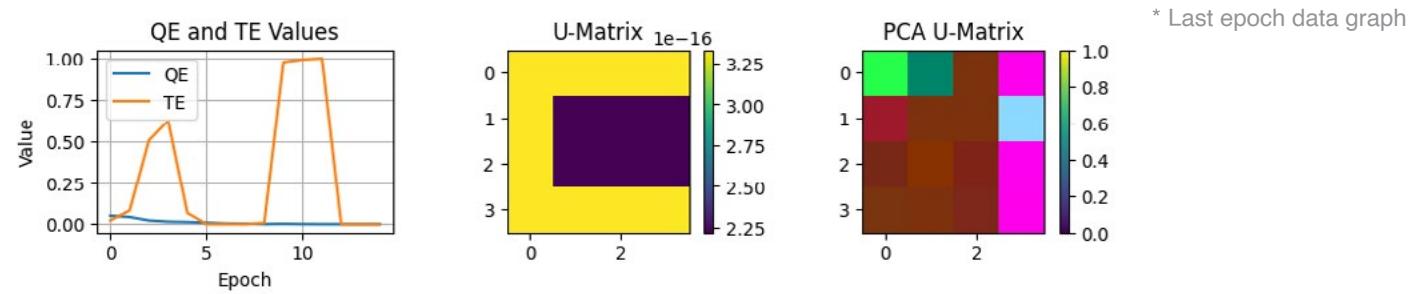
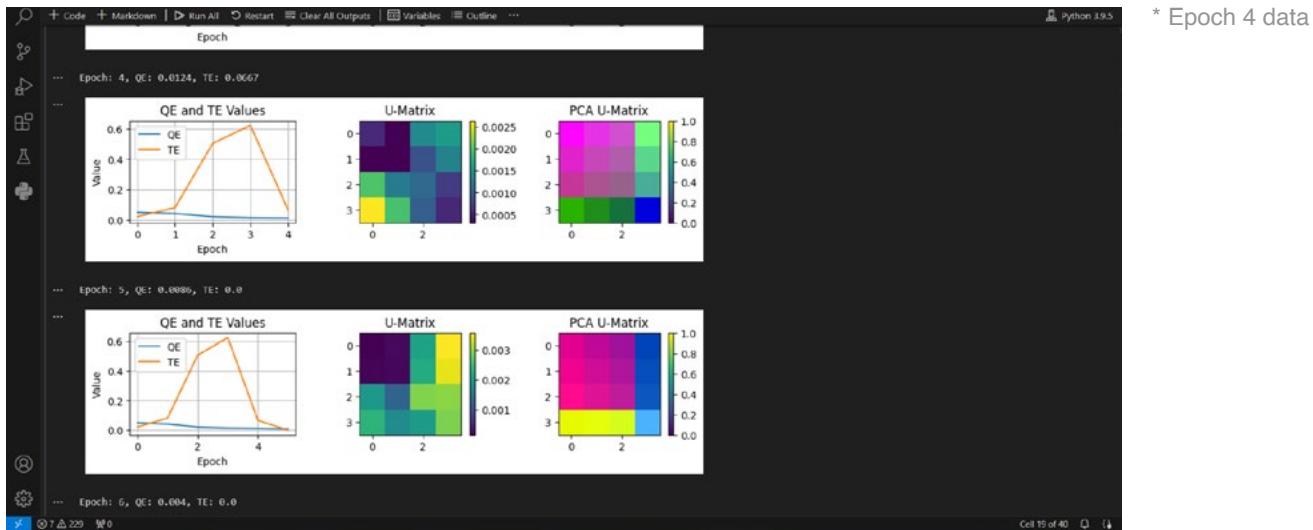
```
# 01
2
3 activatedSOM = activate(train_data, soms[7], denormalise(train_data,featureImg[350][['feature']]))
4 fig = plt.figure()
5 plt.imshow(activatedSOM)
6 plt.axis('off')
7 plt.show()
```

The screenshot shows a Jupyter Notebook interface with a single code cell containing Python code. The code uses the `activate` function from a module to process training data and a specific SOM model (index 7). It then creates a figure, displays the activated SOM matrix, and removes the axes. The resulting heatmap is shown in the main area.

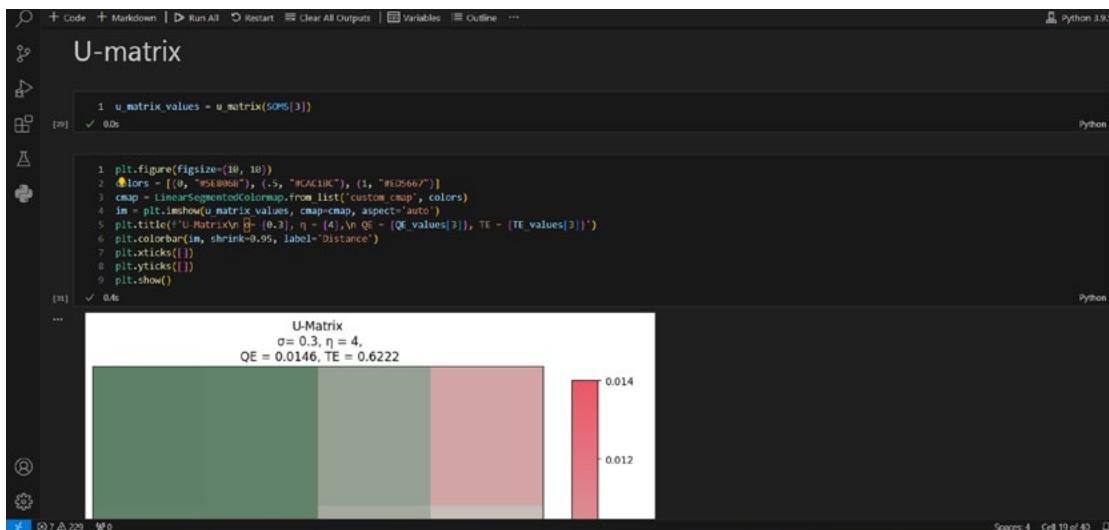


VIDEOS

1. Proceed with the same process explained earlier. After training the SOM, analyze the data and select the best epoch. In this case, epoch 4 is selected as it has the closest QE and TE values to zero.



2. Plot the U-Matrix of the SOM at epoch 4



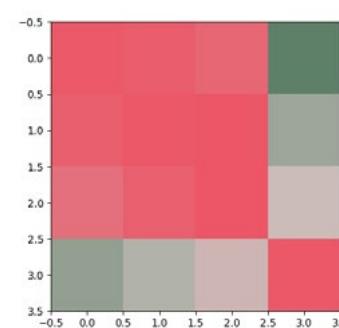
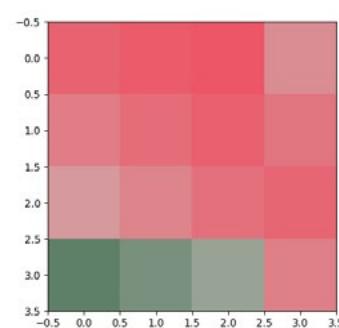
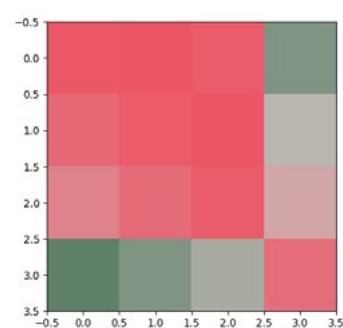
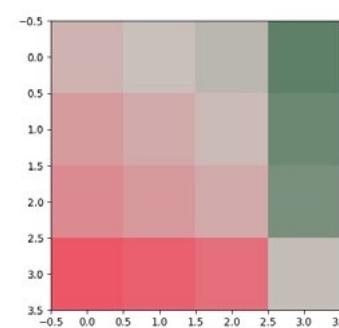
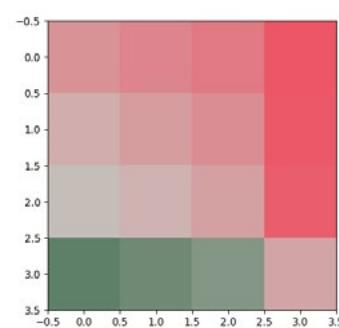
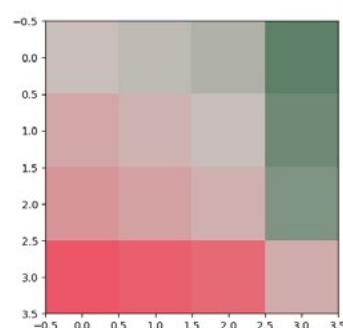
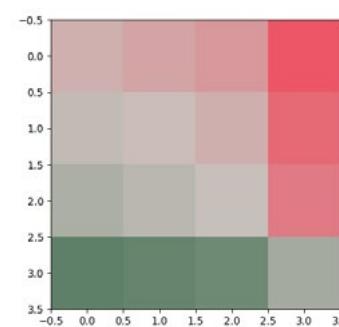
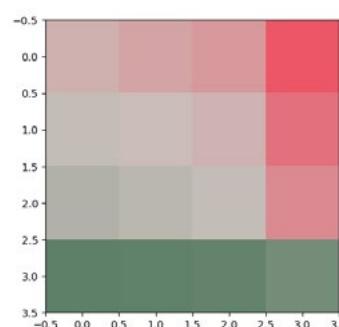
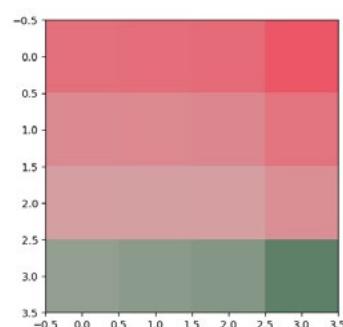
II. Self-organizing maps

9. Activate the SOM

Activate the SOM

```
1 train_data = vectors
2
3 # Step 1: Convert the query sentences into vectors
4 query_sentences = ["streets"]
5 query_vectors = [model.infer_vector(word_tokenize(sentence.lower())) for sentence in query_sentences]
6
7
8 # Step 2: Activate the SOM using the activate function for each query vector
9 activated_SOMs = []
10 for query_vector in query_vectors:
11     activated_SOM = activate(train_data, SOMs[3], query_vector)
12     activated_SOMs.append(activated_SOM)
13
14 activated_SOM = activate(train_data, SOMs[3], query_vector)
15
16 # plot
17 plt.figure(figsize=(5, 5))
18 im = plt.imshow(activated_SOM, cmap=cmap)
19
20 plt.show()
```

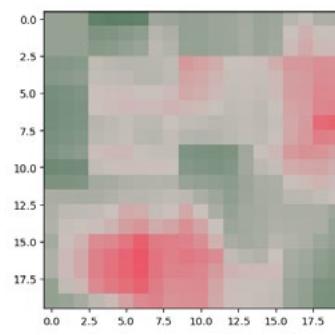
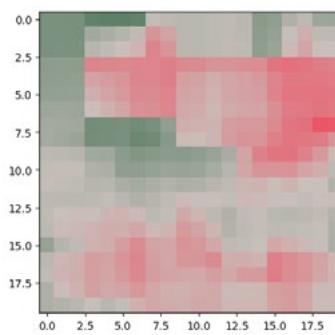
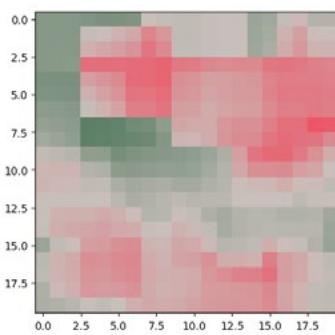
[14]: ✓ 0.2s



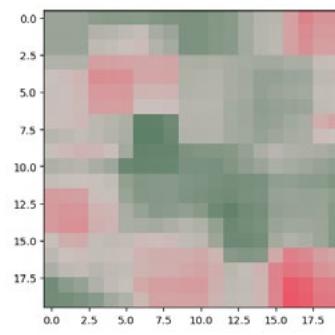
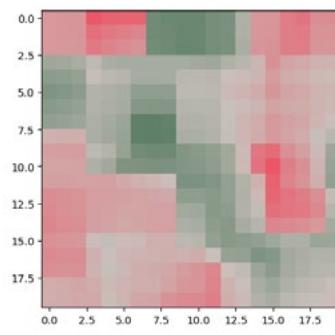
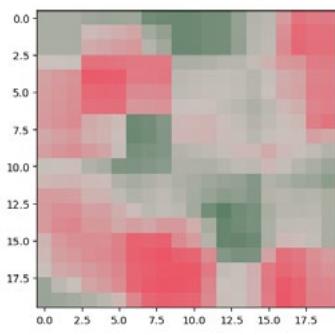
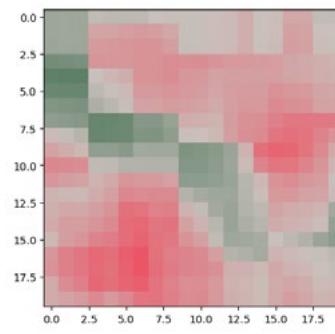
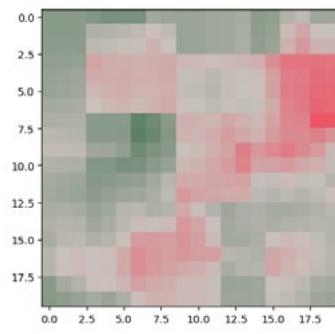
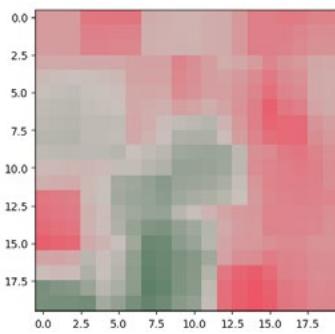
* SOM activations

TEXT 20

1. Proceed with the process explained earlier and activate the SOM



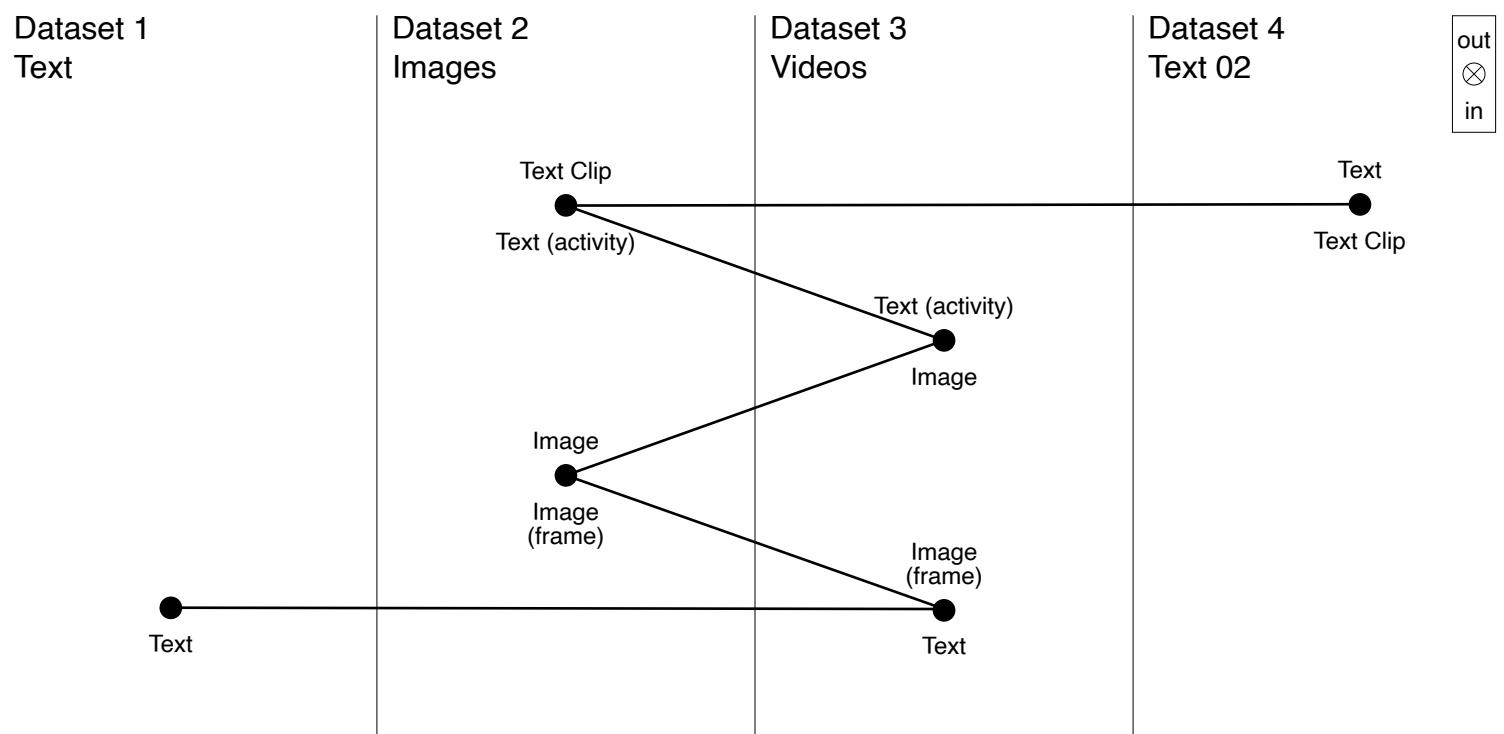
* SOM activations



III. SEARCH ENGINE

COLUMNS ATLAS

Ebitiust quidundio. Ut apietur? Quiat aspici digendundit eatur ad et moditat inctet omniatem quunda comnien dantibus, offici oremporum, que et esenimp orest, volo tem ullabor aut re preprem harciatem suntur aut evelles et atempor sim et et lam quid quoditate si non perit, tem eici ut in pellescidel ipsumqu identor simagna musandu cipsanda inveliquatur modisin veliam que dem fugia sequos sus con pro vel mod quid eate ate dolupturist earibus ducilisquis ut andandero molo doluptae quasped molorissus qui cus prest volligendel exerissimet as explignim di dolupta veri cum valorateni dio tet maximpos rem consequod qui acere nim re sim quas que net et voluptatur? Officae venihil lessequo etur alia endam sitatur?



TF-IDF VECTORISATION

Import The Model & Process The Text.

The imported model reads all the files and extracts the paragraphs. Then, the text is preprocessed which includes removing the stopwords and lemmatizing. Finally, the model vectorizes the text.

Text: TF-IDF

```

1 from TextSearch_TFIDF_Model import TextModel_TFIDF
[1] ✓ 0s

2 Books_DIR = 'Books'
3 Books_FILES = []
4 Books_FILES.extend(glob.glob(os.path.join(Books_DIR, '**.txt')))

[10] ✓ 0s

1 novel_model = TextModel_TFIDF(Books_FILES, min_df=2)
[12] ✓ 1m 07s

1 novel_model.search('In that essay Rowe wants to say that modern architecture', n=5)
[18] ✓ 0s
...
[[{"paragraph": "In that essay Rowe wants to say that modern architecture in Chicago is not really architecture because it is not really disciplined. And his project is an answer to the", "nm": 0,
  "ID": "Books\\Wieser_Koolhaas_Supercritical",
  "type": "txt"}, {"paragraph": " In reading the table any letter, say C, is taken from the left hand column, and any letter, say D, from the top row, and the entry, here A, where the corresponding row", "nm": 8,
  "ID": "Books\\Bell_Men_of_Mathematics",
  "type": "txt"}, {"paragraph": " You also wanted to have rows of columns and surrounded the forecourt of St.", "nm": 36,
  "ID": "Books\\Violett_Literary_Sources_of_Art_History",
  "type": "txt"}]
[25] ✓ 251 90

```

Spaces 4 CRLF Cell 7 of 44

* TF-IDF

```

105 class TextModel_TFIDF:
106     def __init__(self, files, min_df=2):
107         self.paragraphs = []
108
109         IDs = [f.split('/')[1].split('.')[0] for f in files]
110         for f, ID in zip(files, IDs):
111             filetype = f.split('.')[1]
112             if filetype == 'epub':
113                 paragraph = read_epub_paragraphs(f, ID, 'epub')
114                 self.paragraphs.append(paragraph)
115             elif filetype == 'txt':
116                 paragraph = read_txt_paragraphs(f, ID, 'txt')
117                 self.paragraphs.append(paragraph)
118
119         self.preprocessed_paragraphs = self.preprocess([p['paragraph'] for p in self.paragraphs])
120
121         # Initialize TF-IDF vectorizer
122         self.vectorizer_tfIdf = TfidfVectorizer(min_df=min_df)
123         self.vector_matrix = self.vectorizer_tfIdf.fit_transform(self.preprocessed_paragraphs)
124
125         self.nModel = NearestNeighbors(n_neighbors=10, metric='cosine', algorithm='brute', n_jobs=-1)
126         self.nModel.fit(self.vector_matrix)
127
128     def preprocess(self, paragraph):
129         lemmatizer = WordNetLemmatizer()
130         stemmer = PorterStemmer()
131         stop_words = stopwords.words("english")
132         processed_docs = []
133         for paragraph in paragraphs:
134             tokens = paragraph.split()
135             # Remove stopwords
136             tokens = [token for token in tokens if token not in stop_words]
137             # Lemmatize tokens
138             tokens = [lemmatizer.lemmatize(token) for token in tokens]
139             # Stem tokens
140             tokens = [stemmer.stem(token) for token in tokens]

```

Spaces 4 CRLF Cell 7 of 44

* Snippet of TextSearch_TFIDF_Model.py

III. Search engine

DOC2VEC VECTORISATION

Import The Model & Process The Text.

The imported model reads all the files and extracts the paragraphs. Then, the text is preprocessed which includes removing the stopwords and lemmatizing. Finally, the model vectorizes the text.

Text: Doc2Vec

```
1 from TextSearch.Doc2Vec_Model import TextModel_Doc2Vec
[1] ✓ 2026 Python
```

```
1 doc2Vec_Model = TextModel_Doc2Vec(BOOKS_FILES)
[1] ✓ 2m 51.3s Python
```

```
1 Doc2Vec_Model.search_doc2vec('In that essay Rose wants to say that modern architecture', k=5)
[1] ✓ 0.4s Python
```

```
... [{"paragraph": "One thought alone remained in our mind and spirit amid our worries when we could not find any, namely there were marvelous columns at Rome in the palace of Diocletian :",
  "nr": 26,
  "ID": "Books\\Suger_Selected_Works_of_Abbot_Suger_of_Saint-Denis",
  "type": "txt"}, {"paragraph": "It's like, in fancy of column, I certainly do not know; there being hardly two correspondent, and the architect having been ready, as it seems, to adopt ideas and reuse",
  "nr": 73,
  "ID": "Books\\Ruskin_The_Seven_Lamps_of_Architecture",
  "type": "txt"}, {"paragraph": "And instant'nt in tryingles, that all approved an equilater, and disliked a scalene; and so the comon demensions of colunes, which were agreeable to all. I thought ever",
  "nr": 33,
  "ID": "Books\\Wm_Fck_British_Architectural_Theory_1568_1758",
  "type": "txt"}, {"paragraph": "Yes, his essays on the picturesque are very interesting. His approach is of course quite different from both of yours, as you point out. But, I know that he visited Hag",
  "nr": 15,
  "ID": "Books\\Girillner_Rambleinger_and_Gaze",
  "type": "txt"}, {"paragraph": "387 when the largest of these columns, pillars of luculent marble, as much as eight and thirty feet in height, were erected in the atrium of scaurus? a thing, too, that",
  "nr": 2,
```

CRLF - Cell 10 of 44

* Doc2Vec

```
121 class TextModel_Doc2Vec:
122     def __init__(self, files, vectorization='l1a', dimension=200, min_df=2):
123         self.vectorization = vectorization
124         self.dimension = dimension
125         self.min_df = min_df
126         self.paragraphs = []
127
128         IDs = [f.split('/')[-1].split('.')[0] for f in files]
129         for f, ID in zip(files, IDs):
130             filetype = f.split('.')[1]
131             if filetype == 'epub':
132                 paragraph = read_epub.paragraphs(f, ID, 'epub')
133                 self.paragraphs.extend(paragraph)
134             elif filetype == 'txt':
135                 paragraph = read_txt.paragraphs(f, ID, 'txt')
136                 self.paragraphs.extend(paragraph)
137
138
139         self.preprocessed_paragraphs2 = preprocess2([p['paragraph'] for p in self.paragraphs])
140
141     # Initialize Doc2Vec model
142     self.Doc2Vec_model = Doc2Vec(vector_size=self.dimension, min_count=self.min_df)
143
144     # Tag documents
145     self.Doc2Vec_Documents = [(taggedDocument(doc, [i]) for i, doc in enumerate(self.preprocessed_paragraphs2))]
146
147     # Build vocabulary and train model
148     self.Doc2Vec_model.build_vocab(self.Doc2Vec_Documents)
149     self.Doc2Vec_model.train(self.Doc2Vec_Documents, total_examples=self.Doc2Vec_model.corpus_count, epochs=self.Doc2Vec_model.epochs)
150
151     def get_doc2vec_vector(self, query):
152         tokens = preprocess2([query][0])
153         vector = self.Doc2Vec_model.infer_vector(tokens)
154         return vector
```

* Snippet of
TextSearch_Doc2Vec_Model.py

CONCLUSION

To examine both models, I am using the same query to compare the outputs. The used query is a segment extracted from a sentence from the data processed by both models. The TF-IDF model was able to respond with the full sentence of the query. However, the full sentence was not present in the best five matches of the Doc2Vec model. Also, throughout the tests, Doc2Vec model responses were not consistent. This is because each time the model runs, it selects different segments from the data for training, resulting in inconsistent responses for the same query. Yet, whether this is a disadvantage or not depends on the context.

```

1 from TextSearch_Tfidf.Model import TextModel_Tfidf
2 Books_DIR='./Datasets/Text/books'
3 Books_FILES = []
4 Books_FILES.extend(glob.glob(os.path.join(Books_DIR, '*.txt')))

5 novel_model = TextModel_Tfidf(Books_FILES, min_df=2)

6 novel_model.search('In that essay Rowe wants to say that modern architecture', n=5)

```

* TF-IDF response

```

1 from TextSearch_Doc2vec_Model import TextModel_Doc2vec
2 books_FILES
3 
4 Doc2vec_Model = TextModel_Doc2vec(books_FILES)
5 
6 Doc2vec_Model.search_doc2vec('In that essay Rowe wants to say that modern architecture', k=5)

```

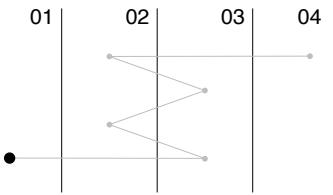
* Doc2Vec response

TEXT SEARCH

Use dataset 1 (text) as an input for the TF-IDF vectorizing model. Then search in dataset 1 using the query “a column for birds to hang out”.

Output:

```
{'paragraph': '.', 'nr': 18, 'ID': 'Books\\Darwin__The_Variation_of_Animals_and_Plants_under_Domestication', 'type': 'txt'}, {'paragraph': 'In these two tables we see in the first column the actual length of the feet in thirty six birds belonging to various breeds, and in the two other columns we see by how much the feet are too shore or too long, according to the size of bird, in comparison with the rock pigeon.', 'nr': 32, 'ID': 'Books\\Darwin__The_Variation_of_Animals_and_Plants_under_Domestication', 'type': 'txt'}]
```



Text: TF-IDF

```
1 from TextSearch_TFIDF_Model import TextModel_TFIDF
2
3 Books_DIR = './Datasets/Text/Books'
4 Books_FILES = []
5 Books_FILES.extend(glob.glob(os.path.join(Books_DIR, '*.txt')))
```

```
1 novel_model = TextModel_TFIDF(Books_FILES, min_df=2)
2
3 novel_model.search('a column for birds to hang out', n=2)
```

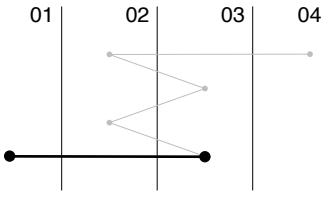
```
1 ['paragraph': 'The bird's eye view differs materially..',
2 'nr': 68,
3 'ID': 'Books\\louis_melibert_de_lorme',
4 'type': 'txt'},
5 {'paragraph': '. | 8 In these two tables we see in the first column the actual length of the feet in thirty six birds belonging to various breeds, and in the two other columns we see I',
6 'nr': 12,
7 'ID': 'Books\\Darwin__The_Variation_of_Animals_and_Plants_under_Domestication',
8 'type': 'txt'}]
```

* Text search

III. Search engine

TEXT TO VIDEO

Select the second response of the previous step. Input the response into the Text_Video model which searches in the text data linked to dataset 3 (videos). The response is a list that contains the film name that is closest to the query, a summary of the video, a description, and the seconds that were extracted by the columns detection model. Then use Display_Clip to display the clip using the output of the previous step. Eventually, extract the frame for the exact second.



* Text search

```
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ...
```

Text to video

```
1 from Text_Video_Model import Text_Video
[1] ✓ 0.5s
```

```
1 import json
2 with open('../datasets/videos/columns_videos_data.json', 'r') as json_file:
3     paragraphs = json.load(json_file)
[2] ✓ 0.1s
```

```
1 TextVid_Model = Text_Video(paragraphs)
[3] ✓ 0.7s
```

```
1 query = 'in these two tables we see in the first column the actual length of the feet in thirty six birds belonging to various breeds, and in the two other columns we see by how
2 TextVid_Model.search(query, n=2)
[4] ✓ 0.1s
```

```
[{"name": "Croydon in the 1980s Part 2",
"summary": "Fairfield Gardens, Croydon Carnival 1983; Boano's Music Store and Grants Department Store, Fire at Christ Church 1985",
"description": "In the opening shots we see cars driving by (the shot is blurry) and flowers in a park. We then see a large fountain behind Fairfield Halls, in Fairfield Gardens. A fire truck is shown at Christ Church. In the next shot we see a train passing through Croydon. The final shot shows a building under construction with scaffolding.", "second": [280, 336, 276, 320, 342, 426, 648, 636, 360]},
{"name": "Kingston upon Thames Film Makers document Kingston upon Thames, providing unique coverage of key locations such as the market place and train station.",
"summary": "Kingston upon Thames Film Makers document Kingston upon Thames, providing unique coverage of key locations, though with a less than positive attitude towards the town.", "description": "Filmmakers from Kingston and District Movie Makers document Kingston upon Thames, providing unique informal coverage of key locations, though with a less than positive attitude towards the town.", "second": [36, 0, 42, 54, 66, 10, 126, 40, 60, 102]})
```

```
In 3, Col 38 | Spaces: 4 | Spacing: 4 | CRLF | Cell 16 of 45
```

* Display the clip by inputting the clip name and specify the second

```
+ code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ...
```

```
1 from Text_Video_Model import Extract_Frame
[1] ✓ 0.1s
```

```
1 video_displayer = Extract_Frame("../Datasets/Videos/Clips/croydon in the 1980s Part 2.mp4")
2 # Display a frame at a specific time
3 start_time_seconds = 288
4 video_displayer.display_frame(start_time_seconds)
[2] ✓ 1.6s
```

```
Spaces 4 | CRLF | Cell 21 of 45
```

* Extract the frame

```
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ...
```

```
1 from Text_Video_Model import Display_Clip
[1] ✓ 0.2s
```

```
1 file_path = "../Datasets/Videos/Clips/croydon in the 1980s Part 2.mp4"
2 start_time_seconds = 288 # Change this to the desired start time in seconds
3
4 video_display = Display_Clip(file_path, start_time_seconds)
5 video_display.display_segment(duration=8)
[2] ✓ 3.4s
```

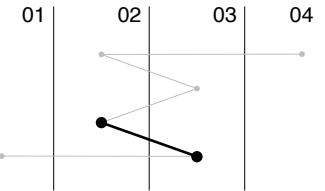
```
Movipy - Building video __temp__.mp4.
Movipy - Writing audio in __temp__TEMP_MP4_wav_snd.mp3
Movipy - Done.
Movipy - Writing video __temp__.mp4

Movipy - Done!
Movipy - video ready __temp__.mp4
```

```
In 1, Col 42 | Spaces: 4 | Spacing: 4 | CRLF | Cell 19 of 45
```

VIDEO TO IMAGE

Use the frame extracted from the previous step to search in dataset 02 (images) and get the best matching image. The model mainly extracts features of the query image and dataset 02 using mobilenet model. Then it compares the features to find the best matches. Eventually, it prints the path of the three best matching images which is displayed in the next block.



Video to image

```
from Video_Image_Model import Image_Search
query_image_path = 'Attachments/frame.jpg'
dataset_folder = '../Datasets/Images/Images'
image_search = Image_Search(dataset_folder)
image_search.search_similar_images(query_image_path)
```

1/1 [=====] - 1s 1s/step
1/1 [=====] - 0s 199ms/step
1/1 [=====] - 0s 60ms/step
1/1 [=====] - 0s 67ms/step
1/1 [=====] - 0s 60ms/step
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 67ms/step
1/1 [=====] - 0s 59ms/step
1/1 [=====] - 0s 67ms/step
1/1 [=====] - 0s 59ms/step
1/1 [=====] - 0s 66ms/step
1/1 [=====] - 0s 57ms/step
1/1 [=====] - 0s 56ms/step
1/1 [=====] - 0s 62ms/step
1/1 [=====] - 0s 62ms/step
1/1 [=====] - 0s 64ms/step
1/1 [=====] - 0s 59ms/step
1/1 [=====] - 0s 60ms/step

* Images search

```
... ['../Datasets/Images/Images\0_Great-Bath-Bath.jpg',
... '../Datasets/Images/Images\1000_Temple Church the Temple City of London the Round.jpg',
... '../Datasets/Images/Images\1001_Syon House Isleworth-London the State Dining Room.jpg']

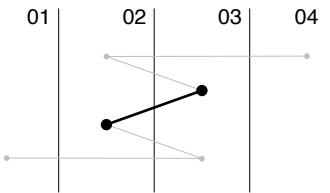
image_path = '../Datasets/Images/Images\0_Great-Bath-Bath.jpg'
image_search.display_image(image_path)
```

* The path of the best matching image and displaying it

III. Search engine

IMAGE TO VIDEO

At this step, the output image of the previous step is used as a query to search in dataset 03 (videos). The output of this step is a film name and the best-matching second in the film. Then, these two outputs are used to save the clip.



The screenshot shows a Jupyter Notebook interface with three code cells. The first cell contains imports for TensorFlow and Image_Video, and a checkmark indicating successful execution (0.0s). The second cell instantiates a MobileNet model and an Image_Video processor, also marked as successful (0.0s). The third cell finds the best matching frame in a folder of clips, prints the result, and marks it as successful (7m 0.0s). The bottom status bar shows progress bars for two tasks: one at 1/1 (~ 1s 95ms/step) and another at 1/1 (~ 0s 58ms/step).

```
1 import tensorflow as tf
2 from Image_Video.Model import Image_Video
3
4 [1] ✓ 0.0s
5
6
7
8 # Instantiate MobileNet model
9 model = tf.keras.applications.mobilenet.MobileNet(
10     input_shape=(224, 224, 3),
11     include_top=False,
12     pooling='avg'
13 )
14
15
16 # Instantiate Image_Video class
17 image_video_processor = Image_Video(model)
18
19 [2] ✓ 0.0s
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670]
```

* Using the extracted frame to search in dataset 3

The screenshot shows a Jupyter Notebook interface with the following details:

- Toolbar:** Includes icons for Code, Markdown, Run All, Restart, Clear All Outputs, Variables, Outline, and Python 3.9.
- Code Cell 1:** Displays the command `!ffprobe -v error -select_streams v:0 -show_entries stream=duration -of default=noprinter` and its output, which lists various video streams with their duration.
- Code Cell 2:** Displays the command `!ffprobe -v error -select_streams a:0 -show_entries stream=duration -of default=noprinter` and its output, which lists various audio streams with their duration.
- Text Cell:** Shows the message "Best matching frame found in file: ../../dataset/videos/clips/SampleA good Name The Story of the Neighbourhood.mp4".
- Text Cell:** Shows the message "Second in the file: 891".
- Text Cell:** Shows the message "Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings..."
- Code Cell 3:** Displays the following Python code:

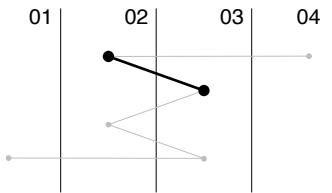
```
1 # Get the frame clip around the identified frame
2 frame_clip = image_video_processor.get_file_around_frame(best_frame_info, 2)
3
4 # display the clip
5 frame_clip.ipython.display()
6 frame_clip.write_videofile("Attachments/output_video/output_clip.mp4", codec="libx264")
```
- Output of Cell 3:** Shows the status "✓ 2.8s" and the command "Moviepy - Building video __temp__.mp4".
- Output of Cell 3:** Shows the command "MoviePy - Writing audio in __temp__TEMP_MPV_wvf_snd.mp3".
- Output of Cell 3:** Shows the command "MoviePy - Done."
- Output of Cell 3:** Shows the command "MoviePy - Writing video __temp__.mp4".
- Output of Cell 3:** Shows the command "MoviePy - done!".
- Output of Cell 3:** Shows the command "MoviePy - video ready __temp__.mpd".
- Output of Cell 3:** Shows the command "MoviePy - building video Attachments/output_video/output_clip.mp4".
- Output of Cell 3:** Shows the command "MoviePy - writing audio in output_clipTEMP_MPV_wvf_snd.mp3".
- Output of Cell 3:** Shows the command "MoviePy - done."
- Output of Cell 3:** Shows the command "MoviePy - Writing video Attachments/output_video/output_clip.mpd".

* The output of the film search is the best matching film path and best matching second of the film. Then save the clip around the best matching second.

III. Search engine

VIDEO TO IMAGE: ACTIVITY

The features (activity) of the output clip from the previous step are extracted and then used as a query to search in the text data linked to dataset 02 (images). The output of this step is an image.



Video to img: Activity

```

1 # extract feature
2 from Video_Text_Model import Video_Text
[0] ✓ 0.1s

1 Xcd Attachments\video-classification-3d-cnn-pytorch
[1] ✓ 0.0s
Outputs are collapsed ...

1 Video_Text.load_videos_to_infile('../output_video')
[0] ✓ 0.0s

1 thon main.py --input input --video_root ../output_video --output ../output.json --model resnet-34-kinetics-cpu.pth --model_depth 34 --mode score --resnet_shortcut A --no_cuda
[0] ✓ 15.1s
Outputs are collapsed ...

1 dicts = Video_Text.read_all_output('../')
2 dicts.keys()
3 keyClips = dicts['Output_Clip.mp4']['clips']
4 Labels = [d['label'] for d in keyClips]
5 print(Labels)
[0] ✓ 0.0s
... ['busking', 'throwing axe', 'throwing axe', 'skateboarding', 'playing tennis', 'playing basketball']

Spacer 4 - CRLF - Cell 32 of 40

```

* Extract the activity

Search in images text

```

1 # search in images text
2 from Video_Image_viaText_Model import Image_Video_viaText
[0] ✓ 0.1s

1 import json
2 with open('../datasets/Images/columns_images_data.json', 'r') as json_file:
3     images_text = json.load(json_file)
[0] ✓ 0.1s

1 TextVid_Model = Image_Video_viaText(images_Text)
[0] ✓ 1.4s

1 query = 'busking, throwing axe, throwing axe, skateboarding, playing tennis, playing basketball'
2 TextVid_Model.search(query, n=3)
[0] ✓ 0.1s
... [{"name": "0_Great_Bath_Bath.jpg",
  "info": "Exterior, Roman, Bath, UK; England, 0001, Great Bath Bath"},
 {"name": "1_Rotunda_Wentworth_Castle-South_Yorkshire_detail-of-entablature.jpg",
  "info": "Exterior, Palladian, UK; England, 1742, Rotunda Wentworth Castle South Yorkshire detail of entablature"},
 {"name": "2_Scarborough-North_Yorkshire-aerial-perspective-of-The-Crescent-and-surrounding-buildings-arranged-around-a-circular-green-and-column.jpg",
  "info": "Aerial, Classical Revival, Scarborough, UK; England, 1832, Scarborough North Yorkshire aerial perspective of The Crescent and surrounding buildings arranged around a circular green and column"}]

1 # display image
2 from Video_Image_viaText_Model import display_image
[0] ✓ 0.1s

```

* Use the activities to search in the text data linked to dataset 02 (images)

Best matching image

```

1 # display image
2 from Video_Image_viaText_Model import display_image
[0] ✓ 0.1s

1 image_viewer = display_image()
2 image_path = '../datasets/Images/Images/0_Great_Bath-Bath.jpg'
3 image_viewer.display_image(image_path)
[0] ✓ 0.9s

```

* Best matching image

III. Search engine

IMAGE TO TEXT 02: CLIP MODEL

Use Clip model to describe the image using the 3 best-matching words from the dictionary. Then use TextModel_TFIDF to vectorize and process dataset 04 (text). Use the output of clip model as a query in dataset 04.

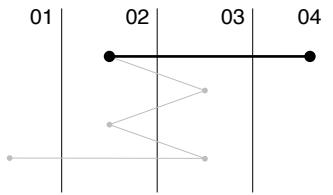


Image to text 02: Clip model

```

1. from clip_model import ClipModel
[1] ✓ 0.0s

2. clip_model = ClipModel()
3. image_path = './datasets/Images/Images/0/Great-Bath-Bath.jpg'
4. best_sentence, confidence = clip_model.find_best_sentences(image_path)
5. print("Best matching sentence:", best_sentence)

... Best matching sentence: ['imagery', 'courtesy', 'aspiration']

1. from TextSearch_TFIDF_Model import TextModel_TFIDF
[1] ✓ 14.3s

1. Books_DIR='./Datasets/Text_02/Books02'
2. Books_FILES = []
3. Books_FILES.extend(glob.glob(os.path.join(Books_DIR, '*.epub')))
[1] ✓ 0.3s

1. novel_model = TextModel_TFIDF(Books_FILES, min_df=2)
[1] ✓ 2m 1.1s

Outputs are collapsed ...

```

* Labeling the image using Clip model

```

1. from TextSearch_TFIDF_Model import TextModel_TFIDF
[1] ✓ 14.3s

1. Books_DIR='./Datasets/Text_02/Books02'
2. Books_FILES = []
3. Books_FILES.extend(glob.glob(os.path.join(Books_DIR, '*.epub')))
[1] ✓ 0.3s

2. novel_model = TextModel_TFIDF(Books_FILES, min_df=2)
[1] ✓ 2m 1.1s

Outputs are collapsed ...

1. novel_model.search('imagery, courtesy, aspiration', n=5)
[1] ✓ 0.2s
... [{"paragraph": " So an hour before sunset he saw something white and gay gleaming through the boles of the oak-trees, and presently there was clear before him a most goodly house build",
  "n": 222,
  "ID": "Books02\\pg3055",
  "type": "epub"}, {"paragraph": " It was strange; was it not, that they should thus ignore that aspiration after complete equality which we now recognise as the bond of all happy human society? I did i",
  "n": 1059,
  "ID": "Books02\\pg3261",
  "type": "epub"}, {"paragraph": " And he deemed the hall fairer within than without; and especially over the shut-beds were many stories carved in the panelling, and Hallblithe beheld them gladly. But i",
  "n": 154,
  "ID": "Books02\\pg2565",
  "type": "epub"}, {"paragraph": " He joyed in his old home,—in the hipped roof of it, the mullioned casements, the wide window-seats, the high and spacious rooms, the geometrical gardens and broad lawn:",
  "n": 322,
  "ID": "Books02\\pg3262",
  "type": "epub"}]

```

* Search in dataset 04

Houdini

IV. Houdini Fundamentals

V. Visualizing GPS and Image Metadata

VI. Video to 3D Model

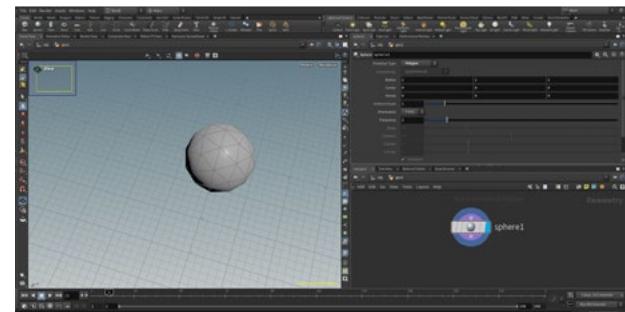
VII. Visualizing JSON in Houdini

IV. HOUDINI FUNDAMENTALS

PROCEDURAL ANIMATION

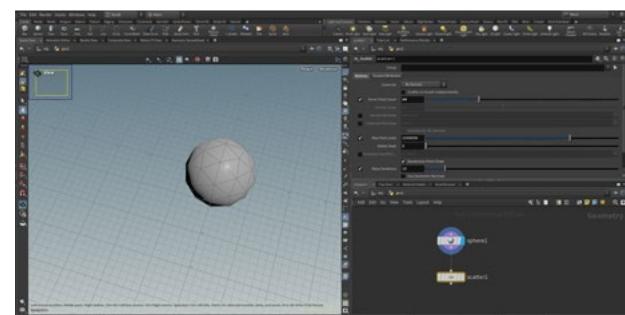
01 PYTHON NODE

Create a geomtry, then create a sphere and set to polygon with a frequency of 2



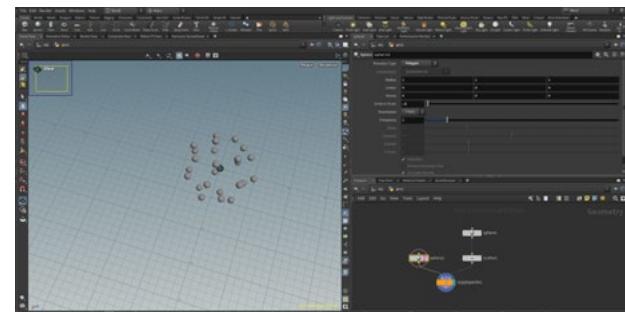
02 SCATTER

Add scatter to scatter some points on the sphere and set to 24



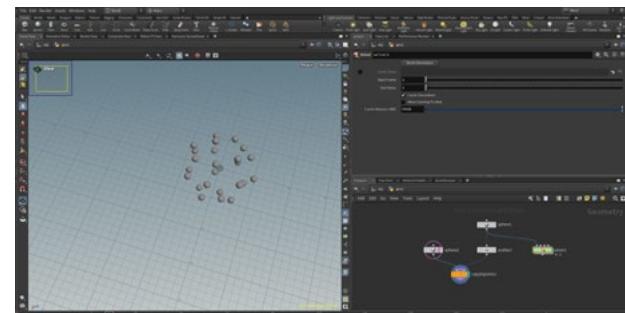
03 COPY TO POINTS, ADD A SPHERE

set the sphere to polygons with a frequency of 2, and scale it to 0.1. Then, connect the scatter and the sphere with the copy to points



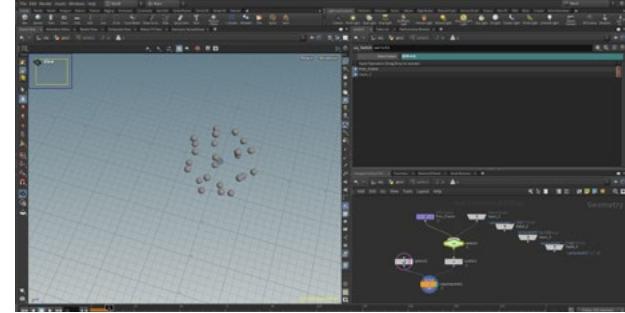
04 ADD A SOLVER

To create the animation and operate on the previous frame geometry



05 ADD A SWITCH

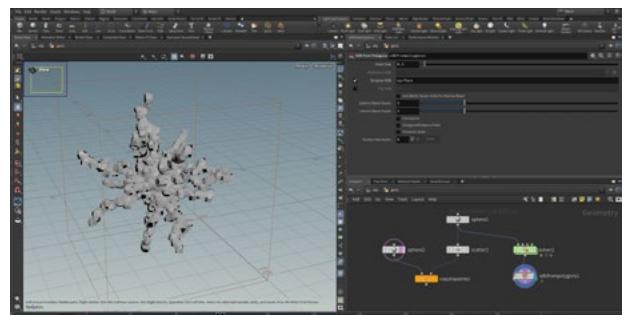
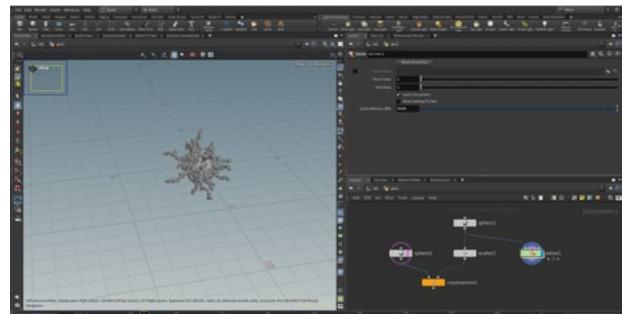
Inside the solver, add a switch and add a condition if the frame equals 1 switch to input_1, then paste the 3 nodes that were created in the previous steps and connect the switch to the scatter.



IV. Houdini Fundamentals

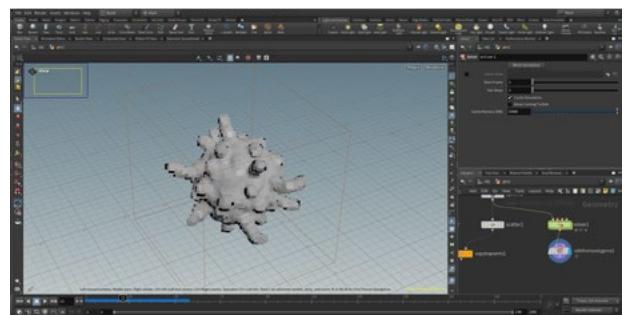
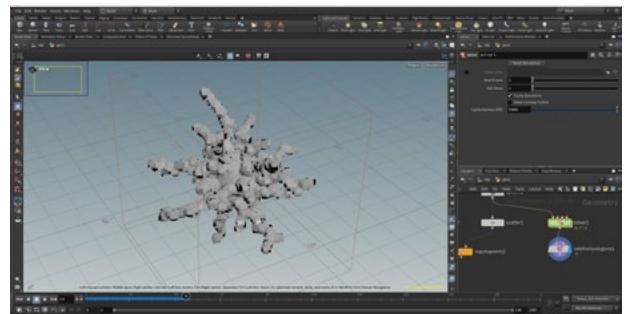
07 ADD A VDB FROM POLYGONS

Add a vdb from polygons to add a pixelated effect on the spheres and connect it to the solver

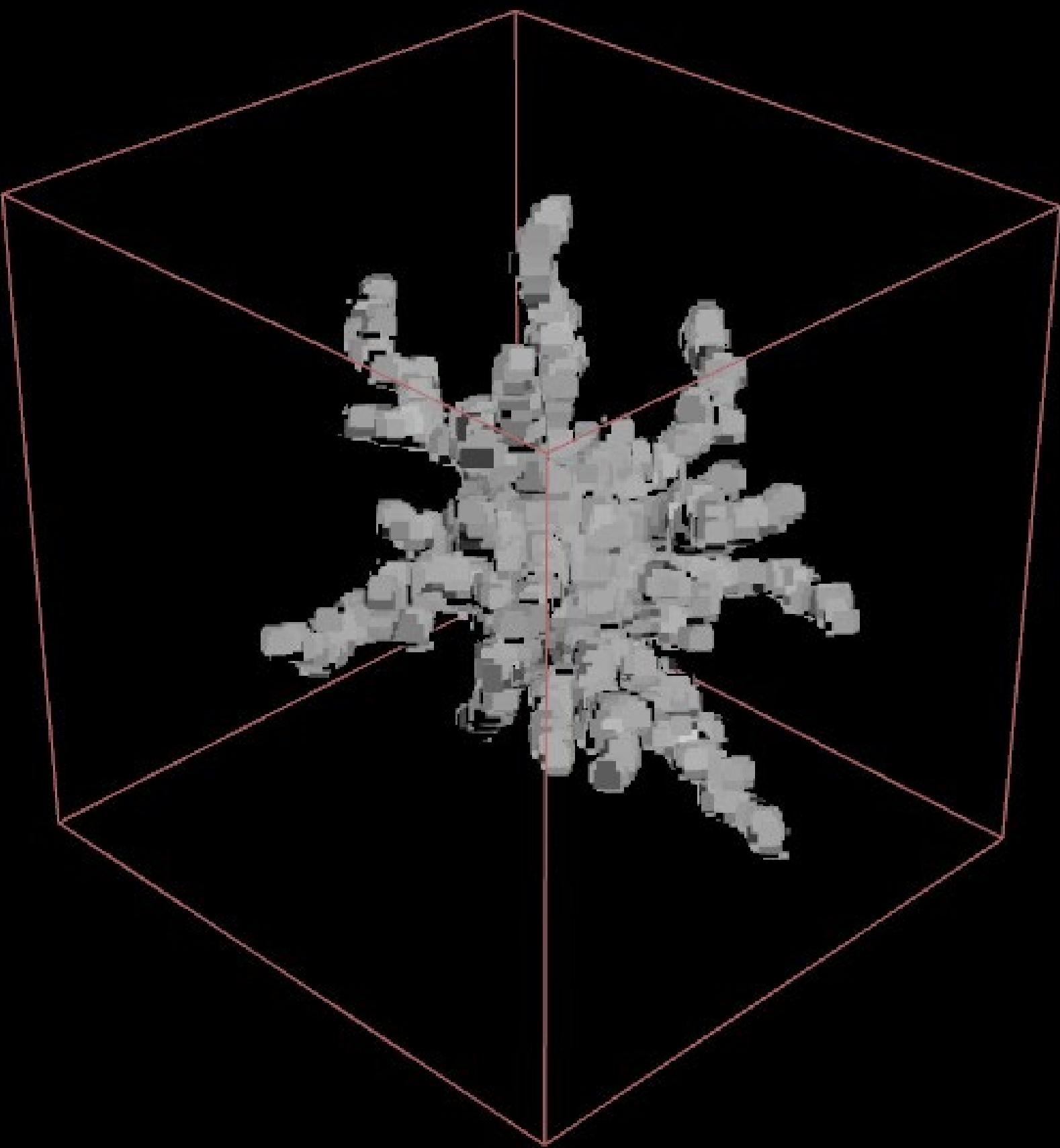


08 FINAL ANIMATION

2 different frames from the animation



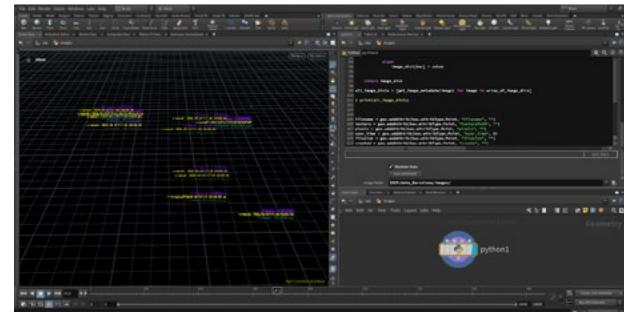
v 0.0



V. VISUALIZING GPS & IMAGE METADATA

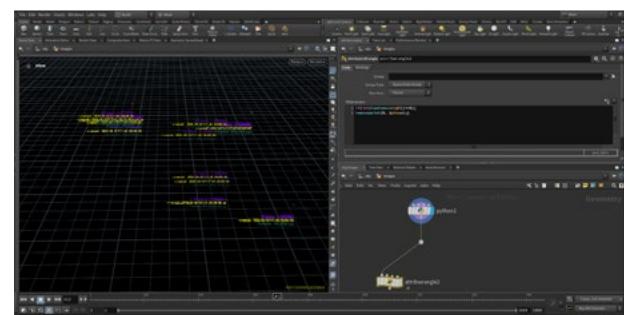
01 PYTHON NODE

Create a Python SOP and link it to the images folder



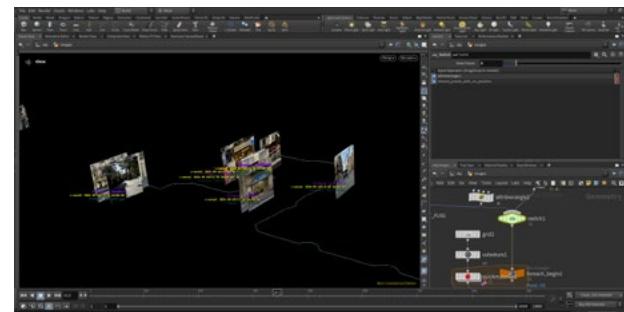
02 ATTRIBUTE WRANGLE

Remove unwanted images



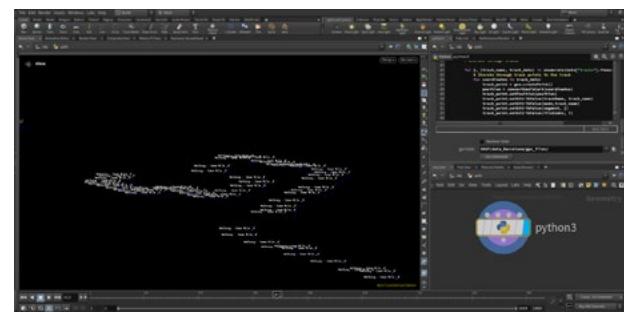
03 SWITCH SOP

Oriente images toward the camera



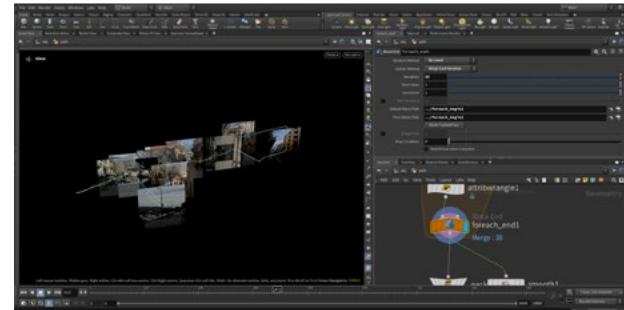
04 PYTHON NODE

Create a Python SOP and link it to the path gpx file



05 ADD A FOREACH

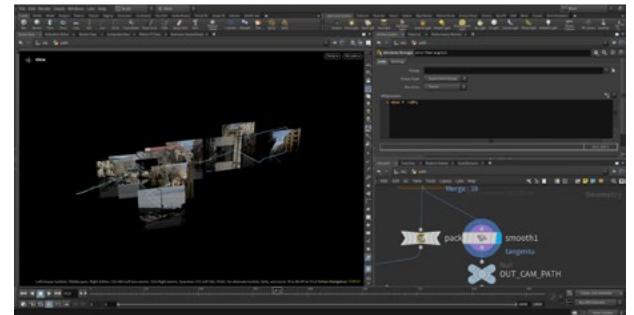
Create a line out of the points



V. Visualizing GPS & Image Metadata

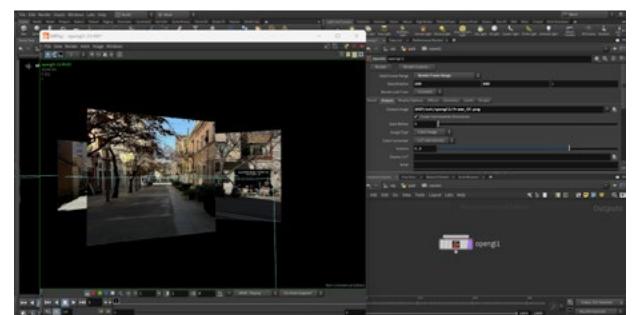
06 SMOOTH SOP

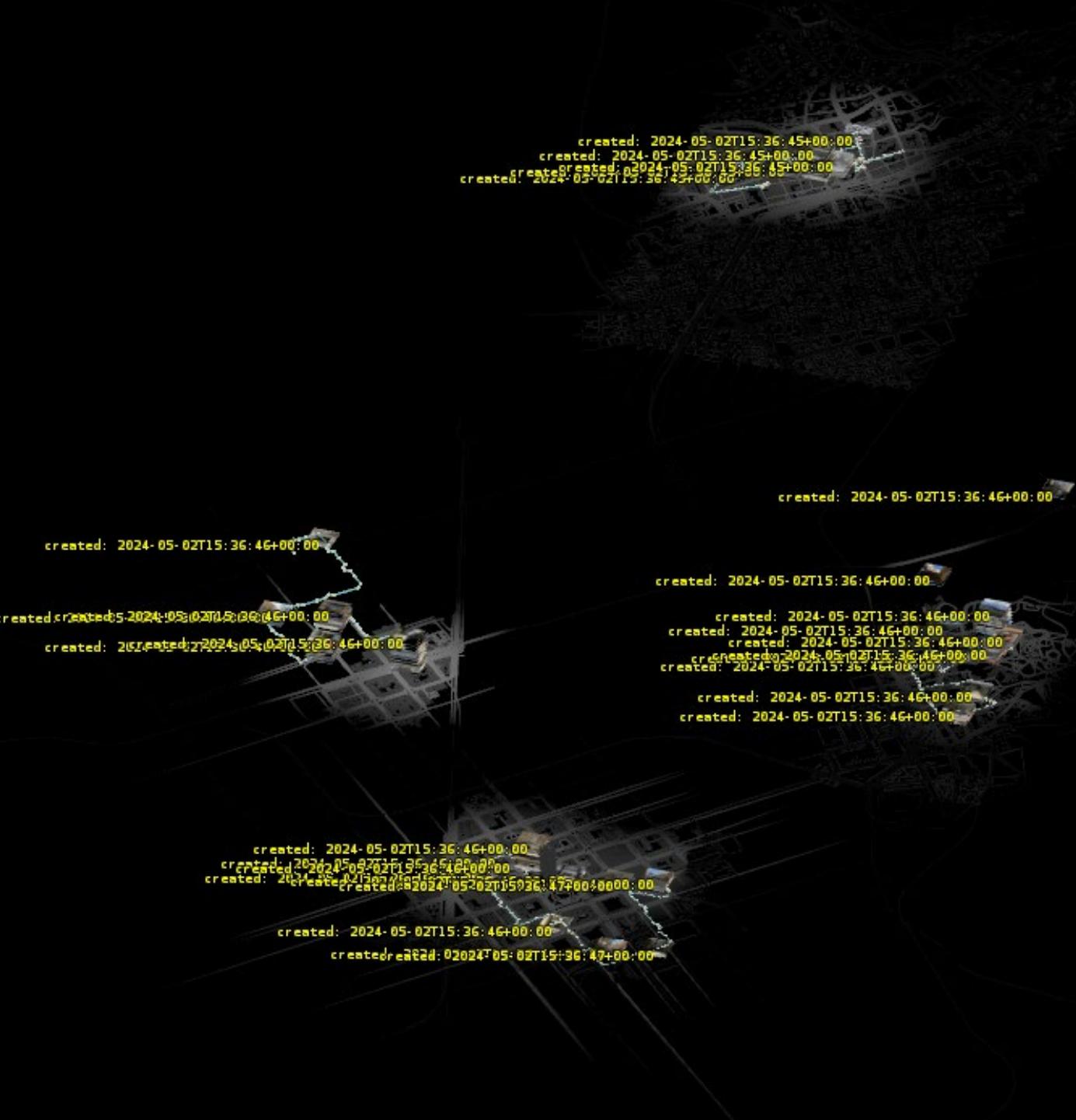
Create a geomtry, then create a sphere and set to polygon with a frequency of 2

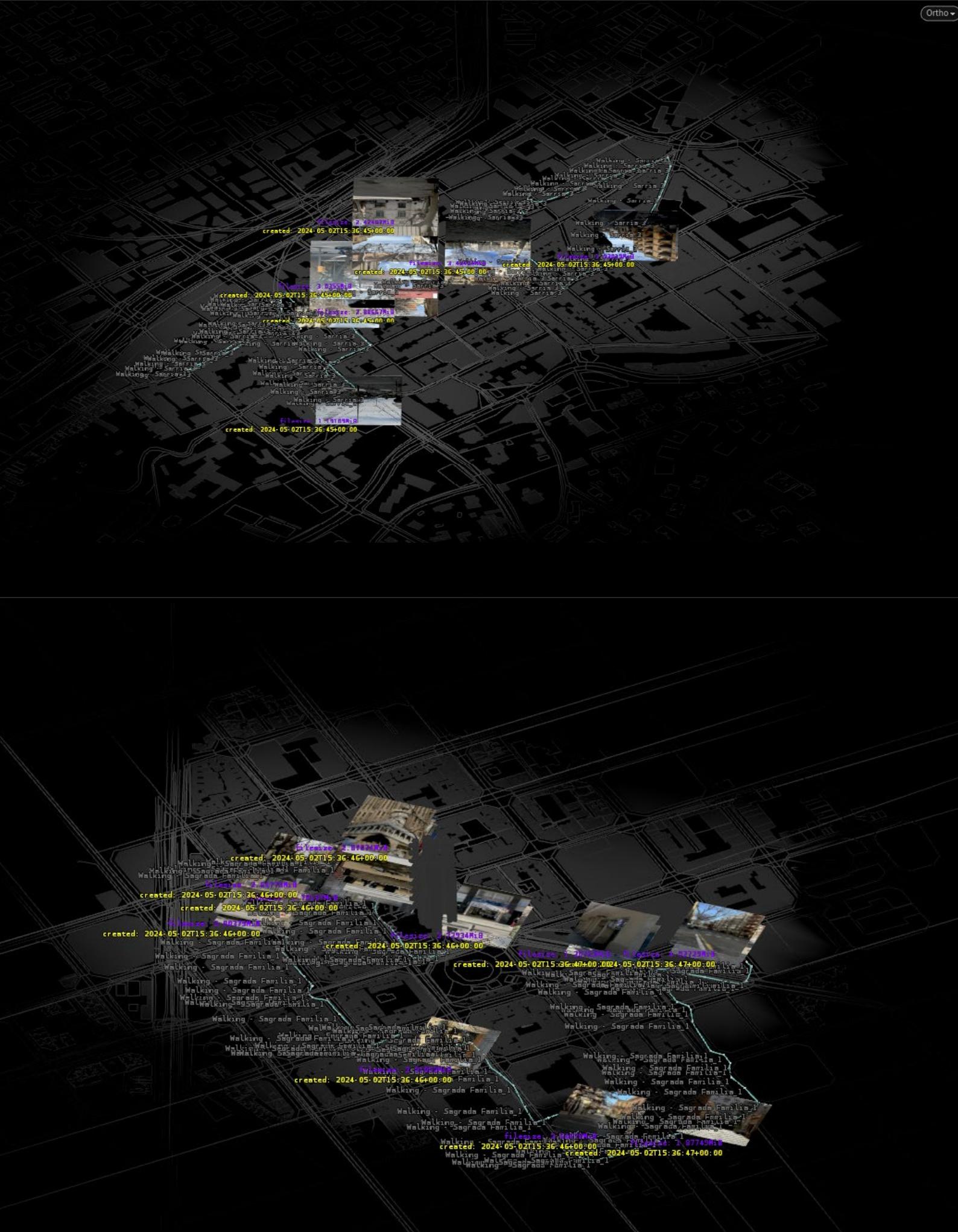


07 SCATTER

smooth the created line to set it as a reference for the camera path



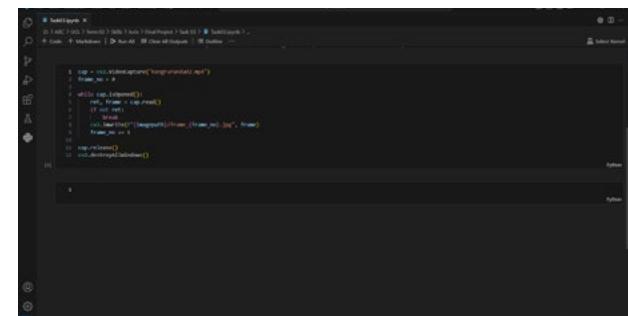




VI. VIDEO TO 3D MODEL

01 EXTRACT FRAMES

Use the Python code to extract the frames from the mp4 videos.



```
# Import required libraries
import cv2
import os

# Set the path to your mp4 video
video_path = "path_to_your_video.mp4"

# Create a folder to save the frames
frame_folder = "frames"
os.makedirs(frame_folder, exist_ok=True)

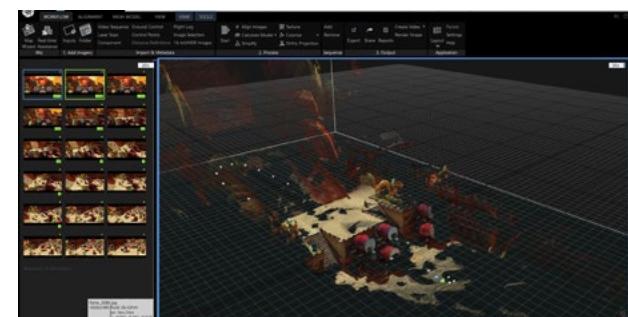
# Initialize the video capture
cap = cv2.VideoCapture(video_path)

# Loop through the frames
frame_id = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    frame_name = f"frame_{frame_id}.jpg"
    frame_path = os.path.join(frame_folder, frame_name)
    cv2.imwrite(frame_path, frame)
    frame_id += 1

# Release the video capture
cap.release()
```

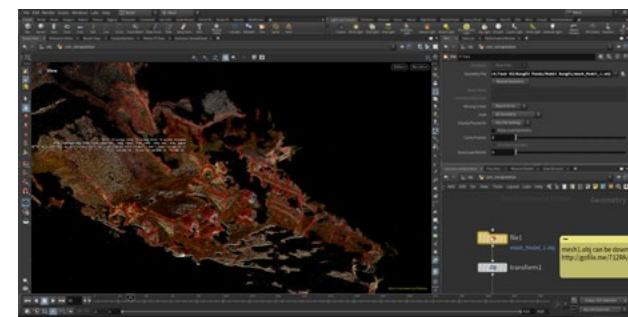
02 CONSTRUCT THE 3D MODE

Import the frames CaptureReality and reconstruct the scenes. Then export as obj.



03 IMPORT TO HOUDINI

Import the model to Houdini using a file SOP. Then add captions and annotations. Match the camera path and export a video.





FLIK: Princess Atta! Princess Atta! Princess Attaaa!

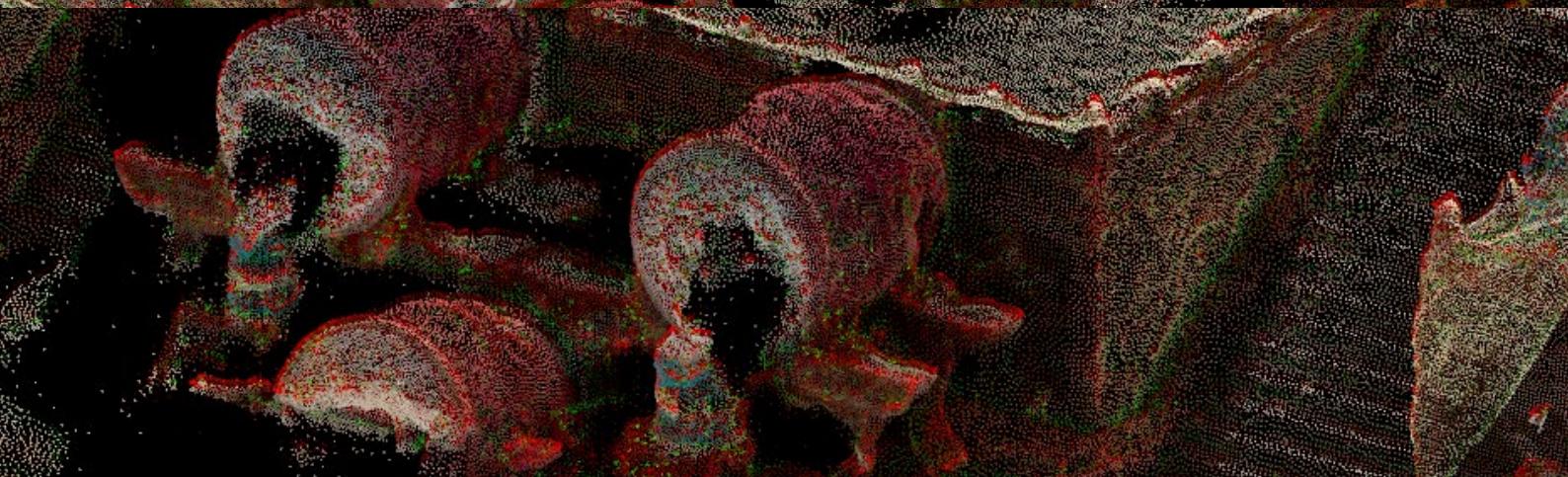
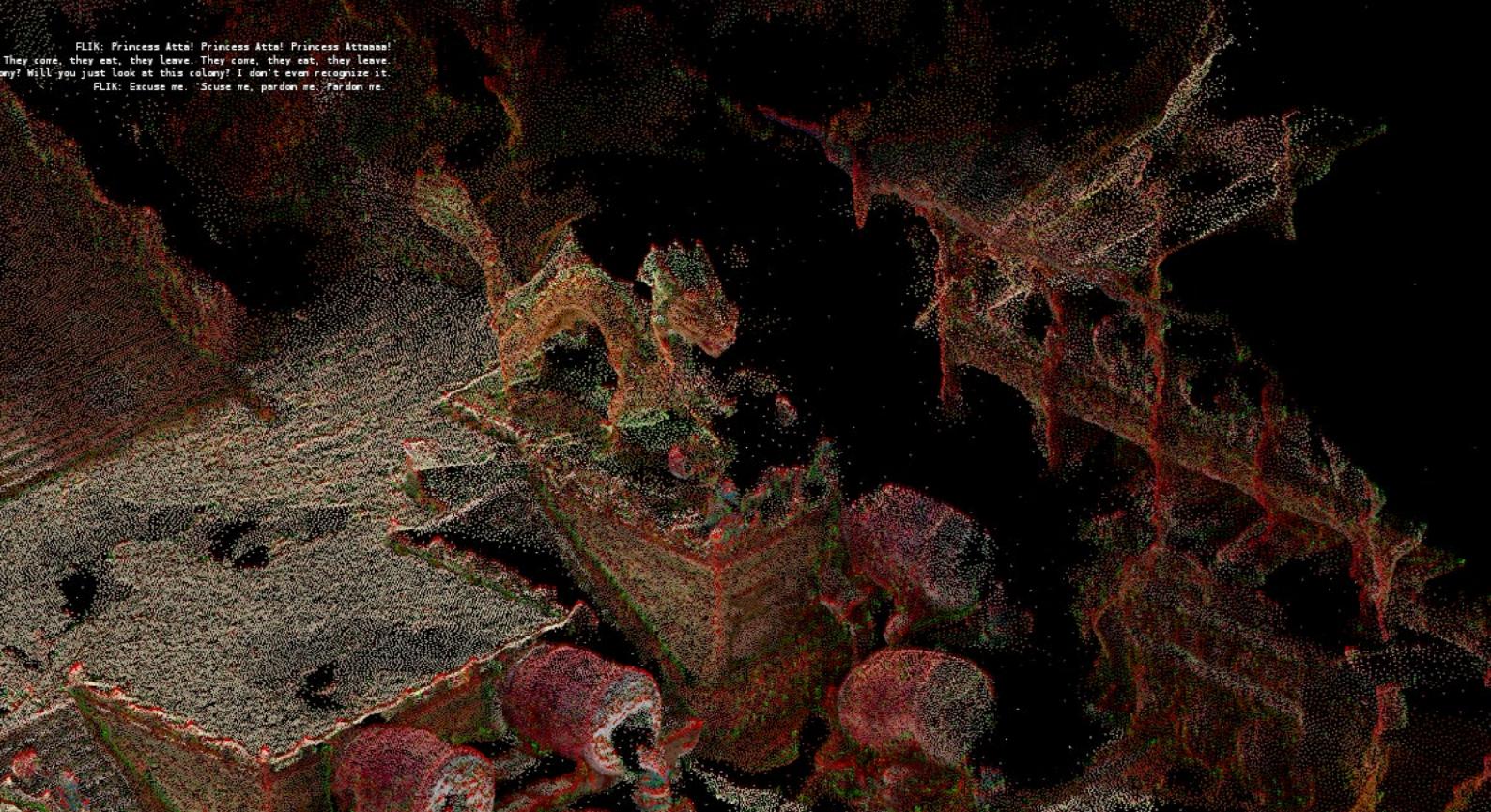
ATTA: (whispering) They come, they eat, they leave. They come, they eat, they leave.

FATTA: Will you look at this colony? Will you just look at this colony? I don't even recognize it.

FLIK: Excuse me. 'Scuse me, pardon me. Pardon me.



FLIK: Princess Atta! Princess Atta! Princess Attanna!
They come, they eat, they leave. They come, they eat, they leave.
any? Will you just look at this colony? I don't even recognize it.
FLIK: Excuse me. 'Scuse me, pardon me. Pardon me.



FLIK: Princess Atta! Princess Atta! Princess Attaaaa!

ATTA: (whispering) They come, they eat, they leave. They come, they eat, they leave.

FATTA: Will you look at this colony? Will you just look at this colony? I don't even recognize it.

FLIK: Excuse me. 'Scuse me, pardon me. Pardon me.

FLIK: Princess Atta! Princess Atta! Princess Attaaaa!

ATTA: (whispering) They come, they eat, they leave. They come, they eat, they leave.

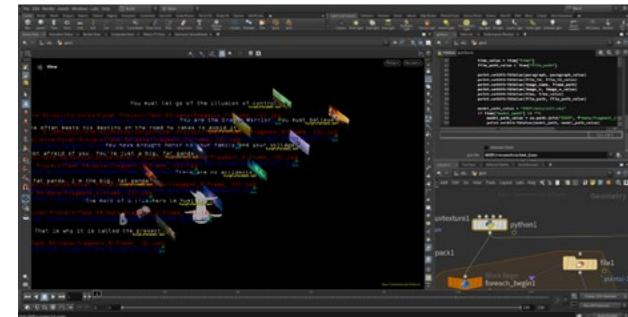
FATTA: Will you look at this colony? Will you just look at this colony? I don't even recognize it.

FLIK: Excuse me. 'Scuse me, pardon me. Pardon me.

VII. VISUALIZING JSON IN HOUDINI

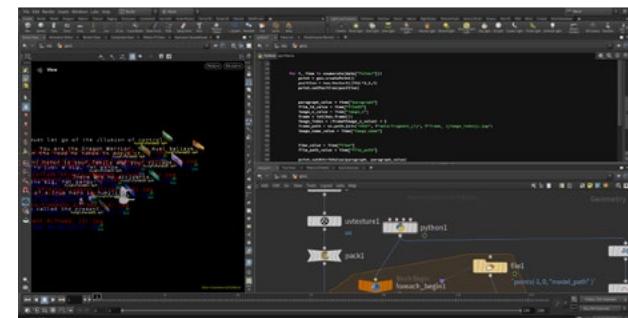
01 PYTHON SOP

Link the json file to the python SOP



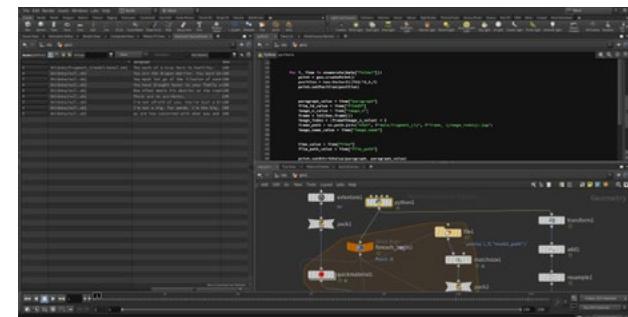
02 IMPORT FRAMES

Through the python SOP code, load the frames.



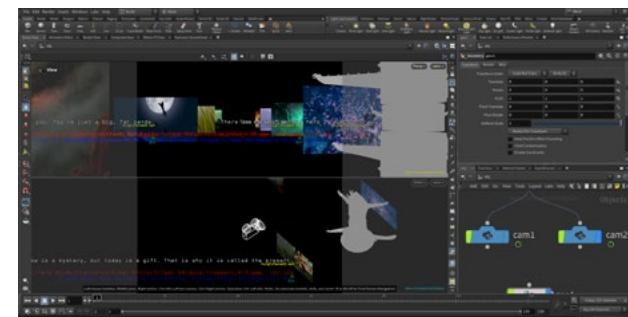
03 IMPORT A 3D MODEL

Based on the json data import the model.



03 EXPORT THE VIDEO

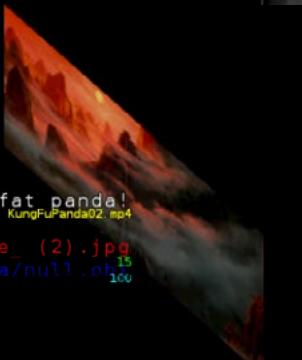
Set two cameras and render the video.





big, fat panda. I'm the big, fat panda!
KungFuPanda02.mp4

Task 04\data\fragment_1\frame_(14).jpg
ris\Final Project\Task 04\data\null.oh₁₅₁₆₀



, fat panda. I'm the big, fat panda!
KungFuPanda02.mp4

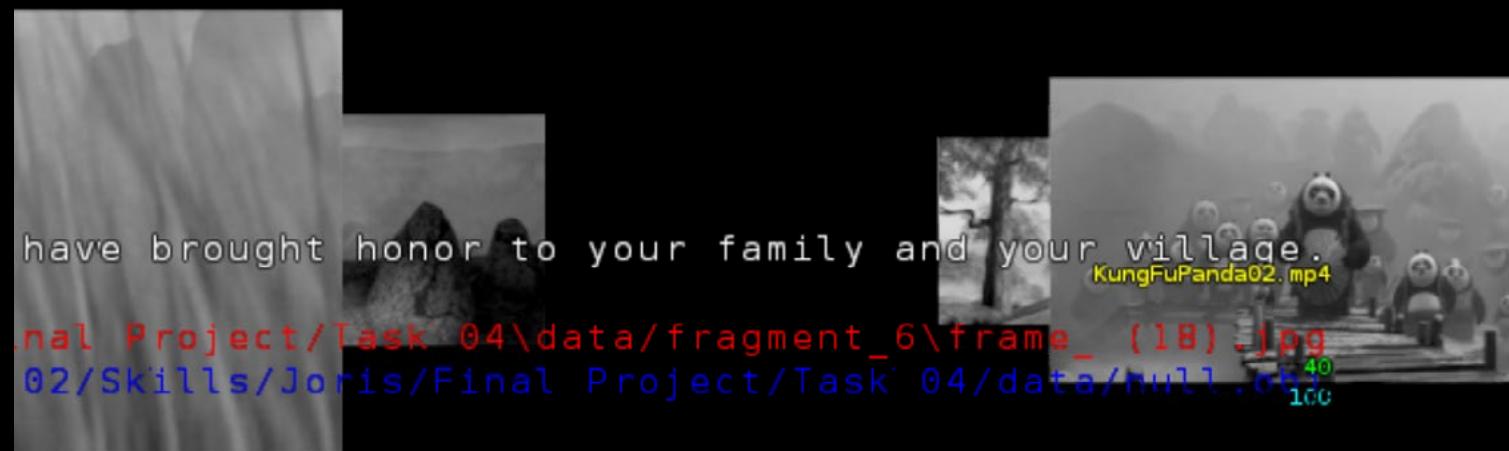
sk 04\data\fragment_1\frame_(2).jpg
/Final Project\Task 04\data\null.oh₁₅₁₆₀



t panda. I'm the big, fat panda!
KungFuPanda02.mp4

4\data\fragment_1\frame_(5).jpg
al Project\Task 04\data\null.oh₁₅₁₆₀





COMPLEXITY

VIII. Exercise One

IX. Exercise Two

X. Exercise Three

VIII. EXERCISE ONE

Question 1

Translate the pseudo code to the notebook.

The screenshot shows a Jupyter Notebook interface with a dark theme. The title bar includes icons for search, code, markdown, run all, clear all outputs, outline, and select kernel. Below the title bar, there's a toolbar with icons for file, edit, cell, and help. A sidebar on the left shows a tree view with a single cell expanded, labeled 'Ex1.' and '1'. The main area contains Python code for matrix multiplication:

```
1 import numpy as np
2
3 def SMW(A,B):
4     n = A.shape[0]
5     C = np.ndarray(shape=(n,n))
6     for i in range(n):
7         for j in range(n):
8             C[i][j]=0
9             for k in range(n):
10                C[i][j] = C[i][j] + A[i][k] * B[k][j]
11
12 return C
```

Question 2

The time complexity of the provided algorithm is $O(n^3)$, where 'n' is the size of the matrices A and B. This is because of the presence of three nested loops in the algorithm, each iterating 'n' times. Hence, the term ' n^3 ' accurately represents the number of operations performed by the algorithm.

The screenshot shows a Jupyter Notebook interface with a dark theme. The title bar includes icons for search, code, markdown, run all, clear all outputs, outline, and select kernel. Below the title bar, there's a toolbar with icons for file, edit, cell, and help. A sidebar on the left shows a tree view with a cell expanded, labeled '2'. The main area contains text explaining the time complexity:

The time complexity of the provided algorithm is $O(n^3)$, where 'n' is the size of the matrices A and B. This is because of the presence of three nested loops in the algorithm, each iterating 'n' times. Hence, the term ' n^3 ' accurately represents the number of operations performed by the algorithm.

Below this text, there are sections for 'Suggestions' (with 7 hidden cells) and examples 'Ex.2' and 'Ex.3' (each with 16 and 11 hidden cells respectively). The bottom status bar shows 'Cell 7 of 43'.

VIII. Exercise One

Suggestions 1,2

In the first block, I am defining three 2×2 matrices to use for testing. In the second block, I am inputting the defined matrices in the function.

Then, use “`np.matmul`” to use the predefined function from numpy model.

The screenshot shows a Jupyter Notebook interface with the following content:

- Suggestions**: A sidebar on the left lists one suggestion.
- Code Cell 1:** Contains code to create three 2x2 arrays: `arr`, `arr_02`, and `arr_03`.

```
1 arr = np.array([[1, 2],  
2 [3, 4]])  
3  
4 arr_02 = np.array([[1, 0],  
5 [0, 1]])  
6  
7 arr_03 = np.array([[3, 2],  
8 [0, 1]])
```
- Code Cell 2:** Contains code to calculate the matrix multiplication of `arr` and `arr_02`.

```
1 arr @ arr_02
```

Output: `array([[1., 2.],
... [4., 3.]])`
- Suggestions**: A sidebar on the left lists two suggestions.
- Code Cell 3:** Contains code to calculate the matrix multiplication of `arr` and `arr_03`.

```
1 np.matmul(arr, arr_03)
```

Output: `array([[1, 2],
... [4, 3]])`

Suggestions 3

I am using the three defined matrices as input for the functions.

The screenshot shows a Jupyter Notebook interface with three code cells. The first cell contains:

```
1 arr_01 = np.array([ [ 1, 2 ],  
... [ 0, 1 ] ])
```

The second cell contains:

```
1 arr_02 = np.array([ [ 3, 2 ],  
... [ 0, 1 ] ])
```

The third cell contains:

```
1 SMM(arr_01, arr_02)  
[2]  
... array([[1., 2.],  
... [4., 3.]])
```

• 2

```
1 np.matmul(arr, arr_02)  
[3]  
... array([[1., 2.],  
... [4., 3.]])
```

• 3

```
1 np.matmul(np.matmul(arr, arr_02), arr_03)  
[4]  
... array([[ 9.,  8.],  
... [16., 17.]])
```

IX. EXERCISE TWO

Question 1

Translate the pseudo code to the notebook.

Ex.2

```
1 def SMMRec(A,B):
2     n = A.shape[0]
3     C = np.ndarray(shape=(n,n))
4     if n == 1:
5         C[0][0] = A[0][0] * B[0][0]
6     else:
7         #quarter matrices A, B, and C
8         A11, A12 = A[:n//2, :n//2], A[n//2:, :n//2:]
9         A21, A22 = A[:n//2:, n//2:], A[n//2:, n//2:]
10        B11, B12 = B[:n//2, :n//2], B[n//2:, :n//2:]
11        B21, B22 = B[:n//2:, n//2:], B[n//2:, n//2:]
12
13        C11 = SMMRec(A11, B11) + SMMRec(A12, B21)
14        C12 = SMMRec(A11, B12) + SMMRec(A12, B22)
15        C21 = SMMRec(A21, B11) + SMMRec(A22, B21)
16        C22 = SMMRec(A21, B12) + SMMRec(A22, B22)
17
18        C[:n//2, :n//2] = C11
19        C[:n//2, n//2:] = C12
20        C[n//2:, :n//2] = C21
21        C[n//2:, n//2:] = C22
22
23
24    return C
25
```

- The base case is when the size of the matrices reduces to 1x1.
- The algorithm directly computes the product of the single elements of matrices A and B to obtain the corresponding element in matrix C.

Recursion Step:

- When the matrix size is greater than 1x1, the algorithm divides each input matrix A and B into four equal-sized submatrices.
- These submatrices are denoted as A11, A12, A21, A22, B11, B12, B21, and B22.

Recursive Calls:

- The algorithm recursively applies itself to each pair of submatrices until the base case is reached.

Combine Step:

- Once the base case is reached, the algorithm combines the results of the recursive calls to obtain the final result matrix C.

Return Result:

- Finally, the algorithm returns the resulting matrix C.

Division by Four:

- In the recursive step, each input matrix is partitioned into four equal-sized submatrices.

Conquer:

- Recursive calls are made to SMMRec to compute the multiplication of these smaller submatrices. Specifically, multiplication operations are performed on submatrices A11 with B11, A12 with B21, A21 with B12, and A22 with B22.

Combine:

- Once the base case is reached, the results of these multiplications are combined to obtain the final result matrix C.
- This process repeats recursively until the base case is reached, and then the results are combined back up the recursion tree to obtain the final result matrix C.

Question 2

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** + Code | + Markdown | ▶ Run All | Clear All Outputs | Outline ...
- Toolbar:** Select Kernel, Python
- Code Cell 1:** Contains the code `return c` and its output `24` and `25`.
- Code Cell 2:** Contains the code `2`.
- Code Cell 3:** Contains the code `1 SPPRec(arr,arr_02)` and its output:
 - array([[1., 2.,
 4., 3.]])
- Text:** 1. Quarturing was not clear
2. No combine step
3. In programming start by 0, however, the pseudo-code starts from 1
- Code Cell 4:** Contains the code `3` and its output `3 cell hidden`.
- Suggestions:** Suggestions for the next cell are listed as "Suggestions".

IX. Exercise two

Question 3

```
+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ... Select Kernel  
3  
Base Case Complexity:

- In the base case, where  $n=1$ , the algorithm simply performs a single multiplication operation. Thus, the complexity of the base case is  $O(1)$ .

Divide Step Complexity:

- The matrices  $A$  and  $B$  are divided into four submatrices each. Therefore, the complexity of the divide step is  $O(1)$ .

Conquer Step Complexity:

- The SMMRec function is recursively called on each pair of submatrices. Since the matrices are halved in size at each recursive call, the number of recursive calls made is  $\log \log_2(n)$ . Each recursive call involves smaller matrices, so the complexity of the conquer step can be expressed as  $T(n/2)$ .

Combine Step Complexity:

- The resulting submatrices are combined to form the final result matrix  $C$ . Therefore, the complexity of the combine step is  $O(n^2)$ .

The overall complexity of the algorithm can be expressed recursively as:  $T(n) = 7T(n/2) + O(n^2)$ 

This recurrence relation arises because in each step, the algorithm divides the problem into 7 subproblems of half the size, and each of these subproblems involves a matrix multiplication of size  $n/2 \times n/2$ , plus the linear time to combine the results. By applying the Master Theorem, we see that  $a = 7$ ,  $b=2$ , and  $f(n)=O(n^2)$ . Since  $\log b(a) = \log_2(7)$  is greater than  $f(n)$ , the complexity is  $O(n \log_2 7)$  which is approximately  $O(n^2.81)$ .



Suggestions



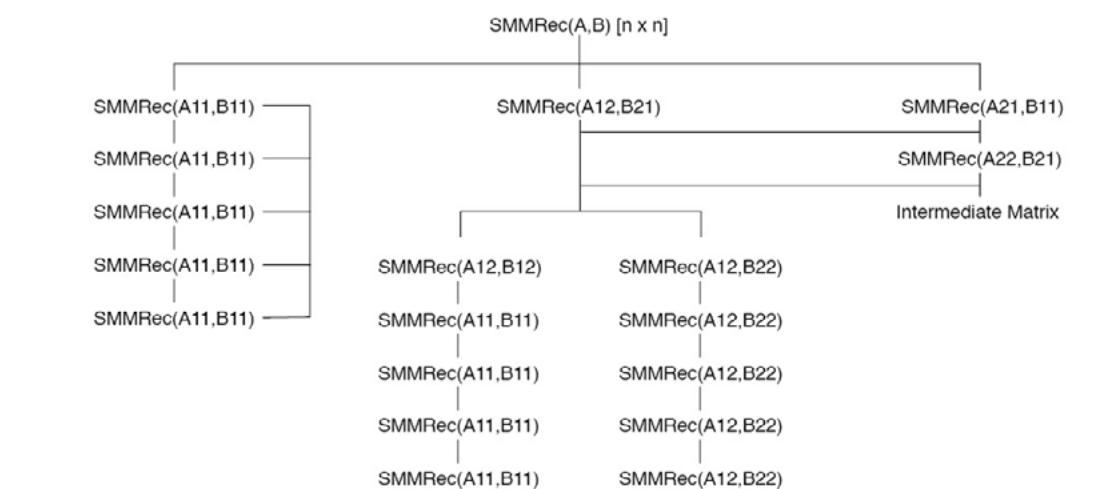
7 cells hidden



0 1 100%

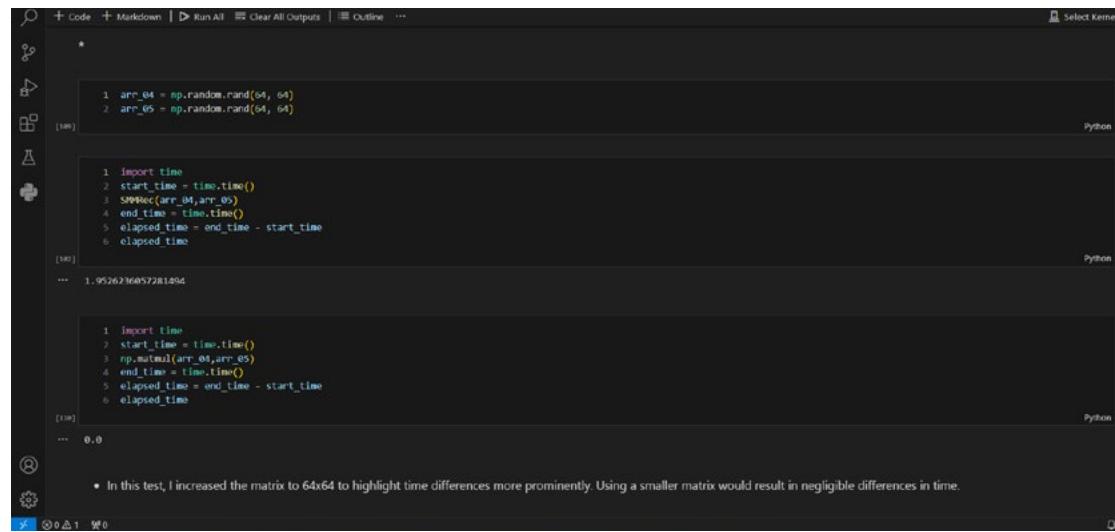

```

Suggestions 1



Suggestions 2

Generate 2 random 64x64 matrices. Then import time to count the running time for each function for comparison. In this test, I increased the matrix to 64x64 to highlight time differences more prominently. Using a smaller matrix would result in negligible differences in time. The results show that numpy function takes less time.



```
+ Code + Markdown | ▶ Run All | Clear All Outputs | ⌂ Outline ... Select Kernel
```

```
*  
1 arr_04 = np.random.rand(64, 64)  
2 arr_05 = np.random.rand(64, 64)
```

```
[100] Python  
1 import time  
2 start_time = time.time()  
3 SPARSEC(arr_04,arr_05)  
4 end_time = time.time()  
5 elapsed_time = end_time - start_time  
6 elapsed_time
```

```
[100] Python  
... 1.9526236057281094
```

```
[101] Python  
1 import time  
2 start_time = time.time()  
3 np.matmul(arr_04,arr_05)  
4 end_time = time.time()  
5 elapsed_time = end_time - start_time  
6 elapsed_time
```

```
[101] Python  
... 0.0
```

- In this test, I increased the matrix to 64x64 to highlight time differences more prominently. Using a smaller matrix would result in negligible differences in time.

X. EXERCISE THREE

Question 1

Ex.3

1

- Matrix addition and subtraction have a computational complexity of $O(n^2)$, meaning they require a number of operations proportional to the square of the size of the matrices.
- The standard matrix multiplication algorithm has a complexity of $O(n^3)$, meaning it requires a number of operations proportional to the cube of the size of the matrices.
- Strassen's algorithm aims to reduce the number of multiplication operations needed for matrix multiplication by recursively breaking down the matrices into smaller submatrices. While it reduces the number of multiplication operations to $O(n^{2.81})$, the overhead involved in splitting, combining, and additional addition/subtraction operations makes it less efficient for smaller matrices due to larger constant factors.
- In summary, while addition and subtraction are simpler and have a lower computational complexity, multiplication involves more operations and thus has a higher complexity, particularly for larger matrices.

> 2

1 cell hidden --

Suggestions

6 cells hidden --

+ Code + Markdown

Question 2

2

- In Strassen's algorithm, the main idea is to reduce the number of multiplication operations needed for matrix multiplication by recursively breaking down the matrices into smaller submatrices.

Divide:

- The matrices are divided into submatrices of size $n/2 \times n/2$ in each recursive step. This division step has a complexity of $O(1)$ because it involves only constant time operations.

Conquer:

- After dividing the matrices, seven recursive multiplications are performed on submatrices of size $n/2 \times n/2$. Each of these multiplications contributes $O((n/2)^3) = O(n^{3/2})$ operations. So, the total complexity for the seven recursive multiplications is $7 \cdot O(n^{3/2}) = O(n^{3/2})$.

Combine:

- Perform addition and subtraction operations to combine the results to get the final result. These operations have a complexity of $O(n^2)$ because they involve iterating over each element in the matrices.

Overall:

- The overall complexity of Strassen's algorithm can be expressed recursively as: $T(n) = 7T(n/2) + O(n^2)$

Use the Master theorem to determine the overall complexity of Strassen's algorithm. According to The Master theorem if a recursive algorithm splits its problem size into a subproblem of size n/b recursively and each subproblem takes $f(n)$ time, then the overall time complexity can be expressed as follows:

If $f(n) = O(n^{\epsilon} \log_b c)$ for some $\epsilon > 0$, then $T(n) = O(n^{\epsilon} \log_b c)$. If $f(n) = O(n^{\epsilon} \log_b b)$, then $T(n) = O(n^{\epsilon} \log_b b \log_b n)$. If $f(n) = O(n^{\epsilon} \log_b b(c+\epsilon))$ for some $\epsilon > 0$, and if $a \log_b b < c \log_b n$ for some constant $c < 1$ and sufficiently large n , then $T(n) = O(f(n))$. In Strassen's algorithm, $a = 7$ (because there are seven recursive multiplications), $b = 2$ (because each problem is divided into two subproblems), and $f(n) = O(n^{1/2})$ (because the combine step takes $O(n^2)$ time).

So, comparing $f(n) = O(n^{1/2})$ with $n^{\epsilon} \log_b b(a) = n^{\epsilon} \log_2 7 = n^{2.807}$, we see that $f(n)$ falls within the first case of the Master theorem. Therefore, the overall complexity of Strassen's algorithm is $O(n^{2.807}) = O(n^{2.807})$.

In summary, Strassen's algorithm reduces the number of multiplication operations required for matrix multiplication, achieving a slightly better complexity than the standard $O(n^3)$ algorithm, but with a larger constant factor due to the additional overhead involved in splitting and combining matrices.

Suggestions

6 cells hidden --

X. Exercise three

Suggestions 1

Translate the pseudo code to the notebook.

The screenshot shows the Visual Studio Code interface with the 'Suggestions' panel open. The panel displays several code snippets for matrix operations, such as addition, subtraction, splitting, combining, and multiplying matrices using the Strassen algorithm. The code is written in Python.

```
1 def matrix_add(A, B):
2     n = len(A)
3     return [[A[i][j] + B[i][j] for j in range(n)] for i in range(n)]
4
5 def matrix_sub(A, B):
6     n = len(A)
7     return [[A[i][j] - B[i][j] for j in range(n)] for i in range(n)]
8
9 def split_matrix(A, n):
10    A11 = [row[:n] for row in A[:n]]
11    A12 = [row[:n] for row in A[n:]]
12    A21 = [row[:n] for row in A[:n]]
13    A22 = [row[:n] for row in A[n:]]
14    return A11, A12, A21, A22
15
16 def combine_matrix(A, B, C, D, n):
17     top = [A[i] + B[i] for i in range(n)]
18     bottom = [C[i] + D[i] for i in range(n)]
19     return top + bottom
```

[1] Python

```
1 def strassen_matrix_multiply(A, B):
2     n = len(A)
3     if n == 1:
4         return [[A[0][0] * B[0][0]]]
5     else:
6         A11, A12, A21, A22 = split_matrix(A, n // 2)
7         B11, B12, B21, B22 = split_matrix(B, n // 2)
```

```
+ Code + Markdown | Run All | Clear All Outputs | Outline ... | Select Kernel
```

```
1 def strassen_matrix_multiply(A, B):
2     n = len(A)
3     if n == 1:
4         return [[A[0][0] * B[0][0]]]
5     else:
6         # Splitting and recombining matrices like in exercise 2
7         mid = n // 2
8         A11, A12, A21, A22 = split_matrix(A, mid)
9         B11, B12, B21, B22 = split_matrix(B, mid)
10
11         # Strassen's multiplication operations
12         M1 = strassen_matrix_multiply(matrix_add(A11, A22), matrix_add(B11, B22))
13         M2 = strassen_matrix_multiply(matrix_add(A21, A22), B11)
14         M3 = strassen_matrix_multiply(A11, matrix_sub(B12, B22))
15         M4 = strassen_matrix_multiply(A22, matrix_sub(B21, B11))
16         M5 = strassen_matrix_multiply(matrix_add(A11, A12), B22)
17         M6 = strassen_matrix_multiply(matrix_sub(A21, A11), matrix_add(B11, B12))
18         M7 = strassen_matrix_multiply(matrix_sub(A12, A22), matrix_add(B21, B22))
19
20         # Combining results
21         C11 = matrix_add(M1, M4)
22         C11 = matrix_sub(C11, M5)
23         C11 = matrix_add(C11, M7)
24         C12 = matrix_add(M3, M5)
25         C21 = matrix_add(M2, M4)
26         C22 = matrix_add(M1, M3)
27         C22 = matrix_sub(C22, M2)
28         C22 = matrix_sub(C22, M6)
29
30         C = combine_matrix(C11, C12, C21, C22, mid)
31     return C
```

Suggestions 2

Translate the pseudo code to the notebook.

- Approximate Algorithms:
 - For applications where exact precision is not critical, approximate algorithms can provide significant speedups. These algorithms sacrifice accuracy for speed by using techniques such as low-precision arithmetic or randomized algorithms. Fused multiply-add (FMA) operations can help achieve this optimization.
- Matrix Reordering:
 - Reordering the elements of matrices or changing the order of operations can sometimes improve cache utilization and reduce memory access patterns, leading to better performance. Techniques such as blocking, loop interchange, and loop fusion can be applied to achieve this optimization.

BIBLIOGRAPHY

Amundson, Jhennifer. (2015). 1983 Porphyrios - Jhennifer A. Amundson, ph.D.. jhenniferamundson. <http://jhenniferamundson.net/wp-content/uploads/2015/12/1983-Porphyrios.pdf>

Congress for the New Urbanism. (2012, May 30). CNU 20 - heterodoxia architectonica. YouTube. <https://www.youtube.com/watch?v=aEdQIYbr9hE>

Advanced search. RIBApix. (n.d.). <https://www.ribapix.com/>

London's Screen Archives. (n.d.). <https://www.londonsscreenarchives.org.uk/search-results?q=&filter=theme%2C26%2Chasvideo%2C1&sort=has-video#900> dv=false&cid=0&mid=0&vid=0&q=columns&sid=false&isc=true&orderBy=0&pageSize=24&pageNumber=1

**RC11
SKILLS
FINAL
REPORT**

Research Cluster 11
Student Number 23188508