SKILLS REPORT

Student Number :
22165920

Tutors :
Ceel Pierik, Julian Besems, Joris Putteneers

Research Cluster 11
MArch Urban Design

The Bartlett School of Architecture, UCL

CONTENTS

1 Computability and Complexity

2 Vectorisation and Algorithms

3 3D Reconstruction and Stable Diffusion

1 | COMPUTABILITY AND COMPLEXITY

ALGORITHM

An algorithm is a list of rules to follow in order to perform a task or solve a problem. It is essential that the steps that are to be followed are clear and have a logical sequence and the input and the output are defined precisely. An algorithm could be used for sorting sets of numbers or for more complicated tasks, like recommending user content on social media. They are also used as specifications for performing data processing and play a major role in automated systems.

For example, the Euclidean algorithm is a widely used algorithm for finding the greatest common divisor of two integers. It works by repeatedly dividing the larger number by the smaller number and taking the remainder until the remainder is zero. The last non-zero remainder is the greatest common divisor.

```
def greatestComDiv(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

Types of Algorithms

Algorithms are classified based on the concepts that they use to accomplish a task. The most fundamental types of computer science algorithms are:

1. **Divide and conquer algorithms:** Divide the problem into smaller subproblems of the same type; solve those smaller problems and combine those solutions to solve the original problem.

2. **Brute force algorithms:** Try all possible solutions until a satisfactory solution is found.
   Randomized algorithms: Use a random number at least once during the computation to find a solution to the problem.

3. **Greedy algorithms:** Find an optimal solution at the local level with the intent of finding an optimal solution for the whole problem.

4. **Recursive algorithms:** Solve the lowest and simplest version of a problem to then solve increasingly larger versions of the problem until the solution to the original problem is found.

5. **Backtracking algorithms:** Divide the problem into subproblems, each which can be attempted to be solved; however, if the desired solution is not reached, move backwards in the problem until a path is found that moves it forward.

6. **Dynamic programming algorithms:** Break a complex problem into a collection of simpler subproblems, then solve each of those subproblems only once, storing their solution for future use instead of re-computing their solutions.

COMPUTABLE FUNCTIONS

A computable function is a mathematical function that can be computed by an algorithm. A function is computable if there exists a finite sequence of steps that can be followed to compute the output of the function given its input.

The concept of computable functions was first introduced by Alan Turing in the 1930s as part of his work on the foundations of computation. There are several different models of computation that can be used to define computable functions, including the lambda calculus, the recursive functions, and Turing machines. All of these models are equivalent in terms of what functions they can compute, meaning that if a function is computable in one model, it is computable in all the other models.

THE HALTING PROBLEM

The Halting Problem is a famous problem in the theory of computation, first introduced by Alan Turing in 1936. It asks whether there exists an algorithm that can determine whether an arbitrary program will halt (i.e., terminate) or run forever. Turing showed that there is no general algorithm that can solve the halting problem for all possible programs and inputs.

This result has important implications for the theory of computation and the limits of what can be computed. It means that there are some problems that are fundamentally undecidable, meaning that no algorithm can solve them in general

GENERAL RECURSION

A function is said to be generally recursive if it can call itself with a modified argument that brings it closer to its base case.

Recursive functions have two key components: a **base case** and a **recursive case**.

The base case is the simplest possible case for which the function is defined. It provides a stopping condition for the recursion and ensures that the function terminates. The base case is usually defined explicitly in the function definition and does not involve any further recursive calls.

The recursive case is the more general case that involves calling the function again, but with a simpler or modified argument. In the recursive case, the function is defined in terms of itself, with the argument getting closer to the base case with each recursive call. The recursive case usually involves a conditional statement or a loop that checks whether the argument has reached the base case or not and determines the appropriate action to take in each case.

Recursive functions are often modular, meaning that they can be broken down into smaller subproblems that can be solved independently. This allows us to build larger programs and algorithms from smaller, reusable components, making our code more modular, maintainable, and flexible.

Recursive functions provide a formal and precise way of defining computable functions, and are an essential tool for studying the theory of computation.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

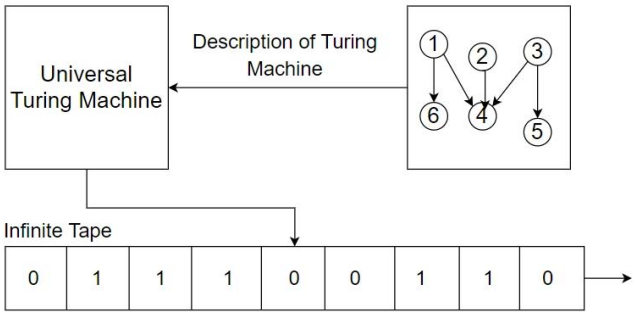
In this example, the factorial function takes an integer n as input and recursively calculates its factorial. The base case is when n is 0, in which case the function returns 1. Otherwise, it calls itself with n - 1 and multiplies the result by n. This process continues until the base case is reached.

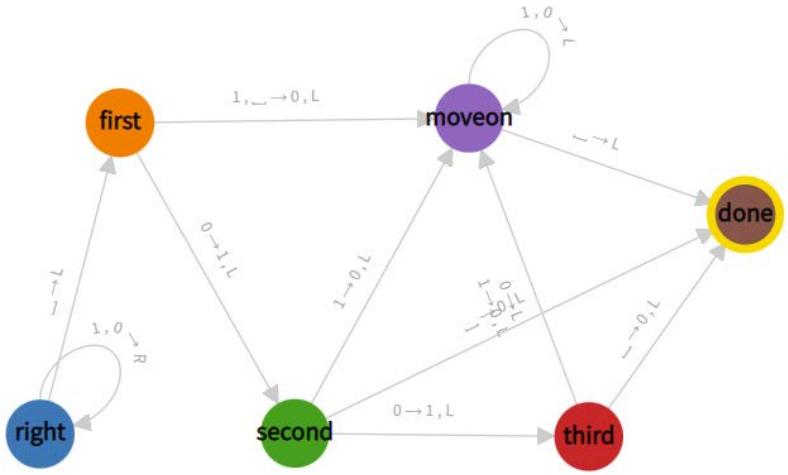
TURING MACHINE

A Turing machine is a theoretical model of a computational device that can simulate any computer algorithm or program. It was proposed by Alan Turing in the 1930s as a way to formalize the concept of algorithm and computability.

The Turing machine has several components:

- **Tape:** A long, one-dimensional tape divided into cells, each of which can contain a symbol from a finite alphabet.
- **Head:** A read/write head that can move back and forth along the tape and read or write symbols on the tape.
- **State register:** A finite set of states that the machine can be in at any given time.
- **Transition function:** A function that takes the current state and symbol on the tape, and produces a new state, symbol to write, and direction to move the head (left or right).






```
# Subtract 1 from a binary number.
input: '111'
blank: ' '
start state: right
table:
  # scan to the rightmost digit
  right:
    [1,0]: R
    ' '  : {L: first}
  # then carry the 1
  first:
    0 : {write: 1,L : second}
    [1, ' ']: {write: 0, L: moveon}
  second:
    1 : {write: 0, L: moveon}
    0 : {write: 1, L : third}
    ' ': {write: 0, L: done}
  third:
    1 : {write: 0, L: moveon}
    0 : {L : moveon}
    ' ': {write: 0, L: done}
  # to move on to  the leftmost digit
  moveon:
    [1,0] : L
    ' ' : {L : done}
  done:
```

Turing machine visualisation to show binary increment

The Turing machine can perform any computation that can be done by a digital computer, and can be used to simulate any algorithm or program. The machine operates on a finite alphabet of symbols and can manipulate an infinite tape, allowing it to represent and process data of arbitrary size.One of the key insights of the Turing machine is that any computation that can be done by a human or a digital computer can also be done by a Turing machine, as long as there is enough time and memory available.

The Turing machine is also important because it led to the development of the Church-Turing thesis, which states that any function that is computable by an algorithm can be computed by a Turing machine (or equivalently, any model of computation that is as powerful as a Turing machine).

LAMBDA CALCULUS

Lambda calculus is a formal system that provides a foundation for mathematical reasoning and computation and was first introduced by Alonzo Church in the 1930s, as a way of exploring the notion of computability and developing a theory of functions.

At its core, lambda calculus is a system for defining and manipulating functions using only a few simple rules. These rules allow us to define functions in terms of lambda expressions, which are essentially anonymous functions that take one or more arguments and return a value.

Lambda calculus provides a powerful framework for reasoning about functions and their properties. It allows us to reason about the behaviour of functions in a purely syntactic way, without reference to any particular programming language or implementation.

One of the most important concepts in lambda calculus is the notion of a reduction, which is a way of simplifying lambda expressions by applying the rules of the system. Reductions allow us to evaluate lambda expressions and compute their values, and they are a key part of the theory of computation.

Lambda terms could be used to express every function that could ever be computed. The order of the reductions is immaterial and makes it easier to manipulate them.

COMPLEXITY VS COMPUTABILITY

The concepts of complexity and computability are closely related but distinct. While computability refers to the ability to compute a function, complexity refers to the difficulty of computing a function. Some functions are computable but have high complexity, meaning that they require a large amount of resources (such as time or memory) to compute.

There are several ways to measure the complexity of an algorithm, including time complexity, space complexity, and computational complexity. Time complexity measures the running time of the algorithm as a function of its input size, while space complexity measures the amount of memory required by the algorithm.

FUNCTION GROWTH

Function growth refers to how the value of a function changes as its input increases in size. In the context of algorithms, the input size is typically the size of the data set being processed. Understanding the growth of a function is important because it helps us to determine the efficiency of an algorithm, and to compare different algorithms that solve the same problem.

TIME COMPLEXITY

Time complexity is a measure of the amount of time required to run an algorithm as a function of its input size. It is usually expressed in terms of the “big O” notation, which provides an upper bound on the running time of the algorithm. Time complexity is an important consideration in algorithm design and optimization, as algorithms with better time complexity are usually preferred over those with worse time complexity.

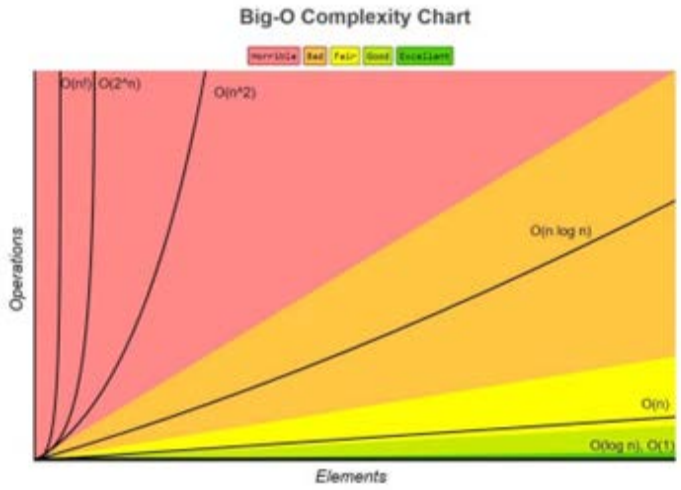
ASYMPTOTIC (‘BIG OH’) NOTATION

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

For example: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear i.e. the best case. But, when the input array is in reverse condition, the algorithm takes the maximum time (quadratic) to sort the elements i.e. the worst case.

When the input array is neither sorted nor in reverse order, then it takes average time. These durations are denoted using asymptotic notations.

Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.



SPACE COMPLEXITY

Space complexity is a measure of how much memory an algorithm requires to solve a problem. This is an important consideration for algorithms that work with large amounts of data, as memory can be a limiting factor. The space complexity of an algorithm is often expressed in terms of the amount of memory required as a function of the input size.

COMPLEXITY CLASSES

Complexity classes are a way of categorizing algorithms based on their efficiency. There are several different complexity classes, including P, NP, and NP-complete. P is the class of problems that can be solved in polynomial time, which means that the time required to solve the problem is proportional to a polynomial function of the input size. NP is the class of problems that can be verified in polynomial time, which means that the time required to check whether a solution is correct is proportional to a polynomial function of the input size. NP-complete is the class of problems that are at least as hard as the hardest problems in NP.

DIVIDE AND CONQUER PARADIGM

The divide-and-conquer paradigm is a common approach used in algorithm design. It involves breaking down a problem into smaller, more manageable sub-problems, solving each sub-problem independently, and then combining the results to solve the original problem. This approach is often used for problems that are too complex to solve using a single algorithm, or for problems that can be solved more efficiently by breaking them down into smaller parts.

SORTING ALGORITHMS

Sorting algorithms are a common type of algorithm that are used to arrange a list of items in a specific order. There are many different sorting algorithms, each with its own advantages and disadvantages. Some of the most popular sorting algorithms include bubble sort, insertion sort, quicksort, and merge sort.

Bubble Sort

Bubble sort is a simple comparison-based sorting algorithm. It repeatedly compares adjacent pairs of elements and swaps them if they are in the wrong order. The algorithm continues iterating over the list until the entire list is sorted. The **time complexity of bubble sort is O(n^2)** and the **space complexity is O(1)**.

Insertion Sort

Insertion sort is another simple comparison-based sorting algorithm. It iterates over the list and inserts each element into its correct position in a new, sorted list. The algorithm continues iterating over the original list until the entire list has been sorted. The time complexity of insertion sort is O(n^2) and the space complexity is O(1).

Selection Sort

Selection sort is another simple comparison-based sorting algorithm. It finds the minimum element in the list and swaps it with the first element. It then finds the minimum element in the remaining list and swaps it with the second element, and so on, until the list is sorted. The **time complexity of selection sort is O(n^2) and the space complexity is O(1)**.

Merge Sort

Merge sort is a divide-and-conquer sorting algorithm. It divides the list into two halves, recursively sorts each half, and then merges the two sorted halves back together. The **time complexity of merge sort is O(n log n) and the space complexity is O(n)**.

Quick Sort:

Quick sort is another divide-and-conquer sorting algorithm. It selects a pivot element and partitions the list around the pivot such that all elements less than the pivot are moved to its left and all elements greater than the pivot are moved to its right. The algorithm then recursively sorts the two partitions until the entire list is sorted. The **time complexity of quick sort is O(n log n) on average and O(n^2) in the worst case, and the space complexity is O(log n)**.

MATRICES AND MATRIX MULTIPLICATION

A matrix is a rectangular array of numbers, symbols, or expressions arranged in rows and columns. The numbers in the matrix are called entries or elements of the matrix. Matrices can be used to represent a wide range of mathematical objects and concepts, such as systems of linear equations, transformations, graphs, and vectors.

A matrix is usually denoted by a capital letter, such as A, and its entries are denoted by lowercase letters with subscripts, such as aij, where i represents the row number and j represents the column number.

Matrices can be added and subtracted element-wise, and can be multiplied by a scalar value. Matrix multiplication is a more complicated operation that involves multiplying rows of one matrix by columns of another matrix. The product of two matrices can only be found if the number of columns in the first matrix is equal to the number of rows in the second matrix.

Matrix multiplication is an important operation in algorithms that involve matrix manipulation and linear algebra. There are many algorithms that use matrix multiplication as a building block to solve more complex problems. Some examples are:

Linear regression

Linear regression is a statistical method used to model the relationship between two variables. It can be solved using matrix multiplication, where the coefficients of the regression model are calculated by finding the least squares solution to a system of linear equations. In linear regression, we are interested in finding the relationship between two variables, typically called the independent variable and the dependent variable. The goal is to create a linear model that can predict the value of the dependent variable based on the independent variable. To find the best fit line, we need to minimize the sum of squared errors between the predicted values and the actual values. This can be formulated as a system of linear equations and solved using matrix multiplication.

Image processing

Matrix multiplication is used in image processing algorithms such as convolution, which involves applying a filter to an image to enhance certain features or remove noise. Image processing involves manipulating digital images to enhance their visual quality or extract useful information. One common technique used in image processing is convolution, which involves applying a filter to an image. The filter is represented as a small matrix called a kernel, and it is applied to the image by sliding it over the image and multiplying each pixel in the kernel with the corresponding pixel in the image. This can be done efficiently using matrix multiplication, where the image is represented as a matrix of pixel values and the kernel is represented as a small matrix.

Example:

The convolutional layer used by MobileNet is the depthwise separable convolution, consisting of two parts: depthwise convolution and pointwise convolution. This means that first, the input tensor (matrix of a specific receptive field) is convolved (dot product obtained) with a set of filters (matrix with learnable parameters), where each filter operates on a single input channel independently (for colour images, one filter each for R, G and B). The result is a set of feature maps, one per input channel. In the second part, a 1x1 convolution is then applied to the feature maps obtained from the depthwise convolution. This helps to increase the number of channels and capture higher-level feature combinations. Convolution helps in capturing local patterns, such as edges, textures, and shapes, by leveraging the spatial relationships between adjacent pixels or elements in the input data.

The use of Matrix multiplication here allows for more learning and better results.

Machine learning

Machine learning involves training models on large amounts of data to make predictions or decisions. Many machine learning algorithms involve manipulating matrices and performing matrix operations,

including matrix multiplication. For example, principal component analysis is a technique used for dimensionality reduction, where the goal is to find a lower-dimensional representation of a dataset while preserving its structure. This can be done by performing singular value decomposition on the data matrix, which involves multiplying the matrix with its transpose. Support vector machines are another machine learning technique that rely on matrix multiplication to find the optimal hyperplane that separates the data points into different classes.

Graph algorithms

Graph algorithms involve manipulating graphs, which are mathematical structures that represent relationships between objects. Many graph algorithms rely on matrix multiplication to perform operations on adjacency matrices, which are matrices that represent the connections between nodes in a graph. For example, the Floyd-Warshall algorithm is a dynamic programming algorithm that finds the shortest path between all pairs of nodes in a graph. It uses matrix multiplication to compute the distance between nodes in the graph and update the shortest path matrix at each iteration.

ALGORITHM DESIGN AND OPTIMISATION

Algorithm design involves understanding the problem, selecting appropriate data structures and algorithms, and optimizing them to improve their efficiency. Optimization can be achieved by reducing the number of steps in an algorithm, reducing the amount of memory required, or improving the way that data is processed.

The Algorithm chosen to do the complexity analysis is the one where i want to process text before using it for analysis.

```
folder_path = "F:\\01_RC11\\Istanbul_Books\\
IstanbulBooks_txtFormat\\FinalBookList"

def break_sentences(text):
    text = re.sub(r'\n', ' ', text)

    sentences = re.split(r'[.?!]+', text)

    sentences = [s.strip() for s in sentences if
s.strip()]

    return sentences
```

The break_sentences function replaces newline characters with spaces and splits the text into sentences using regular expressions.

Time Complexity: The time complexity depends on the size of the text. Assuming it as O(t), where t is the size of the text.

```
def clean_string(s):
    s = re.sub(r' {2,}', ' ', s)
    return s.strip()
```

The clean_string function removes excessive spaces from a sentence using regular expressions.
Time Complexity: The time complexity depends on the length of the sentence. Assuming it as O(s), where s is the length of the sentence.

```
def read_books(folder_path):
    collection = []
    book_index = []
    file_count = 0

    # iterate over all files in the folder
    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path,
filename)
        if not os.path.isfile(file_path):
            continue

        # read the file contents
        with open(file_path, 'r', encoding='utf-8')
as f:
            text = f.read()

        # extract sentences and clean each
sentence
        sentences = break_sentences(text)
        sentences = [clean_string(s) for s in
sentences]

        # add cleaned sentences to collection
        collection.extend(sentences)
        book_index.extend([filename] *
len(sentences))

        file_count += 1

    return collection, book_index
```

The read_books function reads the contents of each file in the specified folder, extracts sentences, and cleans them.
Time Complexity: The time complexity depends on the number of files (m) and the size of each file (n). The overall time complexity is O(m * n).

Space Complexity: The space complexity depends on the size of the collection and book_index lists, which store the processed sentences and book indices. Assuming the total number of words in the collection is w, the space complexity is O(w).

```
myCorpus, book_index = read_books(folder_path)
```

Calls the read_books function to populate the myCorpus and book_index lists.

```
words_to_remove = 'and or but nor yet so been now
you your yours yourself me we us yourselves could
am was too is do will are were would should did
dont ever with was had have has make for as the
this from although because before again would even
though if in order that provided that since so
that than though unless until when whenever where
which whichever whereas wherever whether while'
stoplist = set(words_to_remove.split(' '))
```

Defines a set of words to remove, stored in stoplist.
Time Complexity: The time complexity is O(w), where w is the number of words in words_to_remove.

```
texts = [[word.replace(".","").replace(",","").
replace("'","").replace(":","").replace("","") for
word in document.lower().split()] for document in
myCorpus]
```

Processes each document in myCorpus by splitting it into words, converting them to lowercase, and removing specific characters.
Time Complexity: The time complexity depends on the number of documents in myCorpus (assuming there are d documents) and the average length of each document (let's assume it as a constant value l). Thus, the time complexity is O(d * l).

```
texts = [[word for word in text if (word not in
stoplist and len(word)>2)] for text in texts]
```

Filters out words from each document in texts that are present in the stoplist and have a length less than or equal to 2.
Time Complexity: The time complexity depends on the total number of words in texts. Assuming it as O(w').

```
to_delete = []
for i in range(len(texts)):
    t = texts[i]
    test = [w for w in t if w.isalpha()]
    if len(test) < 5:
        to_delete.append(i)
    else:
        texts[i] = test
```

Iterates over the texts list and checks the length of each document. If the document has less than 5 alphabetic words, its index is added to the to_delete list. Otherwise, non-alphabetic words are replaced in the document.
Time Complexity: The time complexity depends on the number of documents in texts. Assuming it as O(d').

```
for i in sorted(to_delete, reverse = True):
    del texts[i]
    del myCorpus[i]
```

Deletes the documents from texts and myCorpus based on the indices stored in the to_delete list.
Time Complexity: The time complexity depends on the number of documents to delete. Assuming it as O(d').

```
from collections import defaultdict
frequency = defaultdict(int)
for text in texts:
    for token in text:
        frequency[token] += 1
```

Calculates the frequency of each token in texts using a defaultdict.
Time Complexity: The time complexity depends on the total number of tokens in texts. Assuming it as O(p).

```
processed_corpus = [[token for token in text if
frequency[token] > 1] for text in texts]
```

Creates the processed_corpus list by filtering out tokens that have a frequency less than or equal to 1.
Time Complexity: The time complexity depends on the total number of tokens in texts. Assuming it as O(p).

```
pprint.pprint(processed_corpus)
```

Prints the processed_corpus.

The overall time complexity of the code can be approximated as O(m * n + t + s + d * l + w + d' + p), and the space complexity as O(w + p).

In order to optimise the algorithm, the word removal can be more efficient. Creating an instinctive method to put together general stop words could be a way. The cleaning of the sentences is done on a case basis. Efficient ways to get rid of more punctuation marks and other unnecessary characters can make the process more easier.

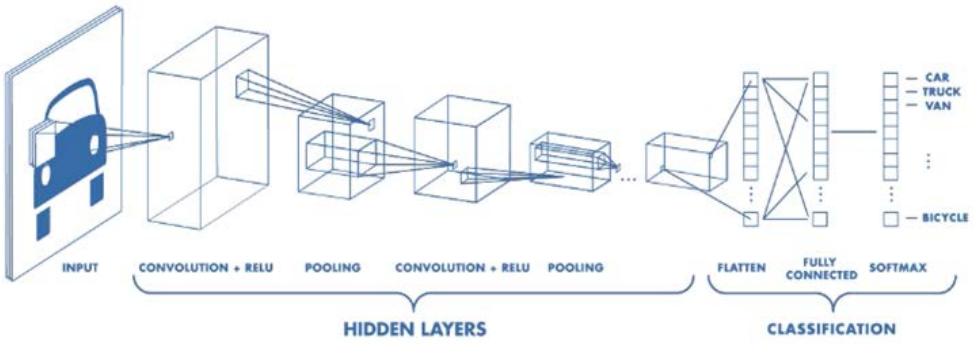
The code currently filters out short documents based on a fixed threshold of 5 words. However, different problems may require different criteria for document filtering. By allowing users to define their own filtering conditions, such as minimum or maximum word count, document length, or specific patterns, you can generalize the algorithm to handle a wider range of text processing tasks.

2 | VECTORISATION AND ALGORITHMS

VECTORISATION IMAGES AND FILMS

Convolutional neural networks (CNNs), a type of deep learning algorithm specifically designed for image and video processing allows to vectorise images and videos. They enable representing pixel data in the form of vectors, consisting of features that capture the important characteristic of the visual content.

CNN is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. Analogous to a biological vision system, where each neuron responds to stimuli only in the restricted region of the visual field called the receptive field, each neuron in a CNN processes data only in its receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along.



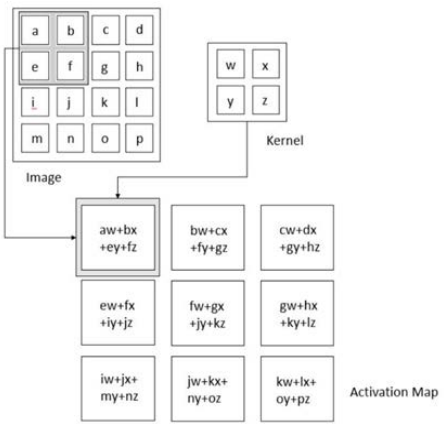
MobileNet

MobileNet is a class of CNN that provides a good starting point for training classifiers that are insanely small and insanely fast. They can can perform real-time image classification, object detection, and other computer vision tasks on mobile devices. This model is used as the basis for conducting Image Classification in the scheme of examining underlying connections in data. CNNs typically have three broad layers: a **convolutional layer**, a **pooling layer**, and a **fully connected layer**.

The key layers that make up the MobileNet model's overall structure are the input layer, convolutional layers, ReLU activation, pooling layers and a fully connected layer.

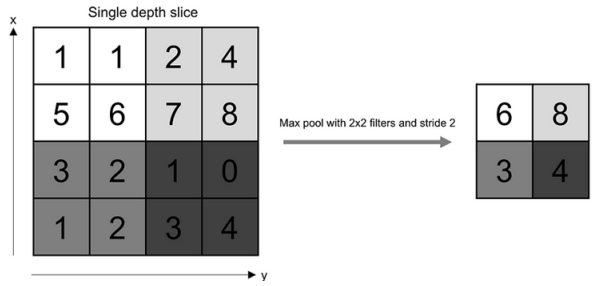
Input layer
The input layer defines the shape of the input data, that includes the height, width and the number of channels it represents. A colour image has 3 channels - R, G, B.

Convolutional layer
The convolutional layer used by MobileNet is the depthwise separable convolution, consisting of two parts: depthwise convolution and pointwise convolution. This means that first, the input tensor (matrix of a specific receptive field) is convolved (dot product obtained) with a set of filters (matrix with learnable parameters), where each filter operates on a single input channel independently (for colour images, one filter each for R, G and B). The result is a set of feature maps, one per input channel. In the second part, a 1x1 convolution is then applied to the feature maps obtained from the depthwise convolution. This helps to increase the number of channels and capture higher-level feature combinations. Convolution helps in capturing local patterns, such as edges, textures, and shapes, by leveraging the spatial relationships between adjacent pixels or elements in the input data.



Rectified Linear Unit (ReLU) activation
ReLU activation is applied element-wise to the outputs of the convolutional layers, the function rectifying negative values to 0. This introduces non-linearity to the network essential to capture intricate patterns more efficiently.

Pooling layer
The pooling layer helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually. MobileNet typically uses global average pooling, where spatial dimensions are reduced by taking the average of each feature map.



Fully connected layer
The fully connected layer connects every neuron in the previous layer to the neurons in the subsequent layer. It maps the learned features to the desired output classes and performs classification.

Softmax layer
The softmax layer is commonly used in the final layer of MobileNet for multi-class classification tasks. It applies the softmax function to the outputs of the fully connected layer, converting them into probability distributions across different classes. Each class probability represents the likelihood of the input belonging to that particular class.

The MobileNet model is fine-tuned using 26 custom classes. These are images of various objects found on the streets of Istanbul is order to achieve a more targeted Image Classification.

```
model = tf.keras.applications.mobilenet.MobileNet(
  input_shape=(224, 224, 3),
  include_top=False,
  pooling='avg'
)
```
Cell 4

The model is instantiated by defining the shape of the input data (**224x224x3**) and set **include_top** to **False** to exclude the fully connected layers. By setting pooling to '**avg**', the output of the model will be the global average pooling of the last convolutional layer.

```
x = Dropout(rate=0.4)(model.output)
x = Dense(26)(x)
x = Softmax()(x)
model = Model(model.inputs, x)
```
Cell 6

A dropout layer, followed by a dense layer with 26 units (26 classes are used to retrain) are added and softmax activation is applied. This modifies the original MobileNet model by appending these layers to the output of the base model.

```
for layer in model.layers[:-3]:
    layer.trainable = False
```
Cell 7

The convolutional and ReLU layers are usually not retrained. The pretrained weights are used in addition to retraining the last layers of the model to fine tune it based on specific requirements. Here, the only the last three layers are retrained.

https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingImagesAndVideos/ImageClassification.ipynb

```
test = load_image('/content/drive/MyDrive/
StreetViewImages/gsv-(40.8163164, 29.3019088)-475-270.
jpg')

print(f'''test:{np.round(image_model.
predict(test),2)}''')
```
Cell 7, 8

```
Output:    test: [[0.17 0.05 0.   0.01 0.01 0.01 0.
           0.07 0.01 0.02 0.03 0.01 0.01 0.02
             0.1  0.07 0.   0.01 0.01 0.   0.05 0.08
           0.03 0.   0.1  0.11]]
```

A Google streetView from Istanbul is used as a test to implement the newly trained model. The output is a list of 26 probabilities values in refernce to the 26 classes it was retrained with.

https://drive.google.com/file/d/171VjQIFuZqkE2qJOA15l3KQfn9EEV76w/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingImagesAndVideos/VideoExtractionWithImageClassification.docx

The final layer of a neural network is typically the output layer, which generates the final outputs or predictions based on the input data. The number of neurons in this layer is usually determined by the number of classes or regression targets in the problem being solved. However, the second-to-last layer is usually a hidden layer that has an intermediate representation of the input data. This layer, which could contain fewer neurons than the output layer, is in charge of formatting the input data so that the output layer can classify or regress it more quickly.

The primary distinction between the two layers is that although the **second-to-last layer aids in the extraction of important features** from the input data that are subsequently used by the output layer to create predictions, the **last layer produces the network's ultimate output**.

The custom trained MobileNet Image Classification model is used to process a video by its frames in order to determine which part of the video closely resembles to a specific image.

```
selectedFilmfeatures = processFilmByFrames('THE_
MOST_COLORFUL_PART_OF_ISTANBUL_BALAT.mp4','/con-
tent/drive/MyDrive/THE_MOST_COLORFUL_PART_OF_ISTAN-
BUL_BALAT.mp4', 1, image_model)
```

Cell 17

The features of a video is extracted by processesing frames of the video using the custom trained model.

```
matchingFrame = findFramesByImage(selectedFilm-
features, '/content/drive/MyDrive/ScrapedImages/
street_table_in_istanbul/streettableinistanbul107.
jpeg', image_model)
```

Cell 19

The features of the image, which is considered as the reference image, is extracted and used to find a frame that is most similar to the features of the image.

```
Output:    ['/content/drive/MyDrive/THE_MOST_
           COLORFUL_PART_OF_ISTANBUL_BALAT.mp4',
           129]
selectedFrameClip = getFilmAroundFrame(matching-
Frame, 5)
```

Cell 21

Using the moviepy library, a clip around the frame in the video is generated by giving the radius in terms no. of seconds.

https://drive.google.com/file/d/171VjQIFuZqkE2qJOA15l3KQfn9EEV76w/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingImagesAndVideos/VideoExtractionWithImageClassification.docx

Input image

Frames of clip around closest matching frame (frame 129, radius=5, duration of video=10s)

This enabled extracting a moving scene around the image. This was done to examine the activities that occured around an object.

Video Classification Using 3D ResNet

Video classification using 3D ResNet is a popular approach in deep learning for analyzing and categorizing videos. It operates on sequential data (video frames) by incorporating the temporal dimension alongside the spatial dimensions. 3D ResNet models use 3D convolutions that capture both spatial and temporal features simultaneously. This allows the model to analyze the motion and dynamics in videos, making it suitable for video classification tasks.

The frames of videos are extracted at fixed intervals and each frame is resized to ensure consistent input dimensions. The 3D ResNet model consists of convolutional layers, residual blocks, and fully connected layers. 3D convolutions are used to capture spatial and temporal information simultaneously. These convolutions have an additional dimension (time) compared to 2D convolutions. The 3D ResNet model is trained using a labeled video dataset. The training process involves forward propagation of video frames through the network, computing the loss with respect to the ground truth labels, and backpropagating the gradients to update the model's parameters. The output of the network can be either a single predicted class or a probability distribution over multiple classes, depending on the specific task.

To further extend on the examination of activities that occur around the objects in the image, the video classification model was employed and features were extracted. The videos chosen were 5 large walking tours of areas in Istanbul. The duration of these videos range between 30-60 mins. The features of the extracted video using the trained MobileNet model as was also extracted.

```
!python main.py --input input --video_root /content/drive/MyDrive/Skills/Julian/Videos_WalkingTours --output
/content/drive/MyDrive/Skills/Julian/output_features_20230508.json --model /content/drive/MyDrive/Skills/Ju-
lian/resnet-34-kinetics-cpu.pth --model_depth 34 --mode feature --resnet_shortcut A --no_cuda
```

Cell 7

https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingImagesAndVideos/VideoFeaturesExtraction.ipynb

The model is trained to give outputs (last layer) of labels of action classes (400). It also has an option where the features alone can be extracted (second last layer). This layer gives an output having a dimension of 512 for every 16 frames (sequential).

```
keyName = 'keyClip_streettableinistanbul107.mp4'
s = 0
d = 8

fragment = searchForMatch(keyName, allFeatures-
Dict, frame = [s,s+d], nBestMatches = 12, average =
False, mode = 'features')
```

Cell 26

```
fragmentFilms = []
for f in fragment:
    fragmentFilms.append(extractFragment(f, '/con-
tent/drive/MyDrive/Videos/Videos_WalkingTours'))
```

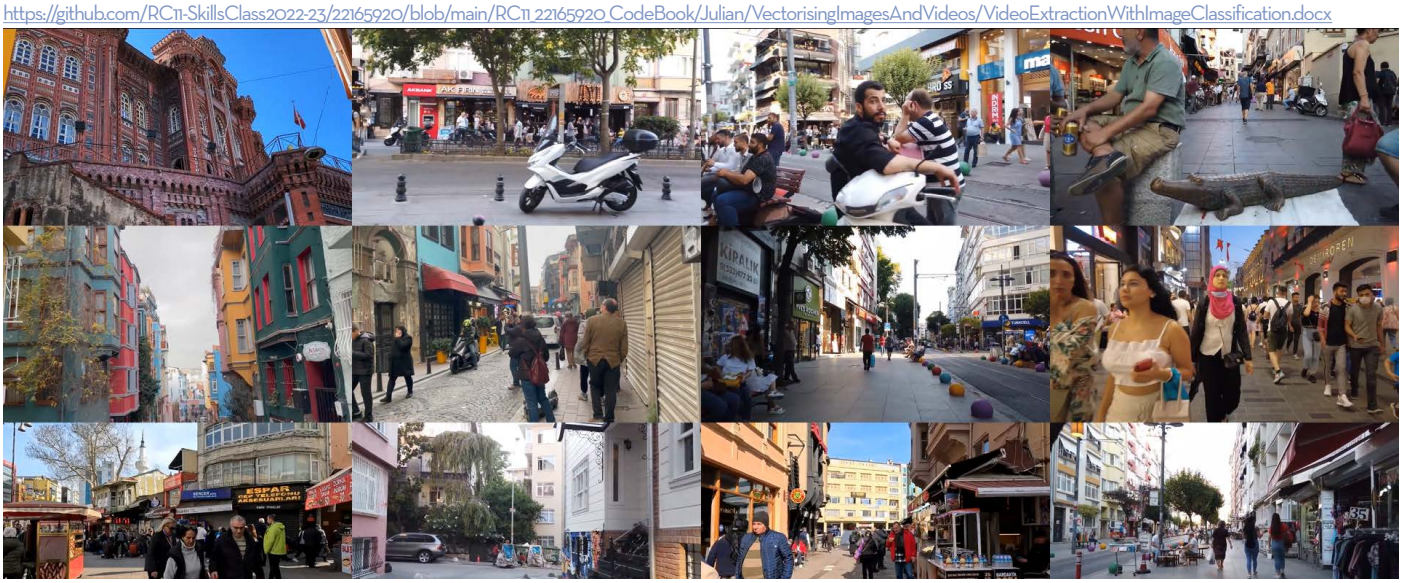
Cell 28

A funtion is defined where the extracted film, the feature dictionary of all videos are taken as input. The features of the extracted film are used to check the closest matching features in the entire dictionary. The best matching 12 clips are identified depending on the least distances. The output is a list of four values: [film, start, length, distance].

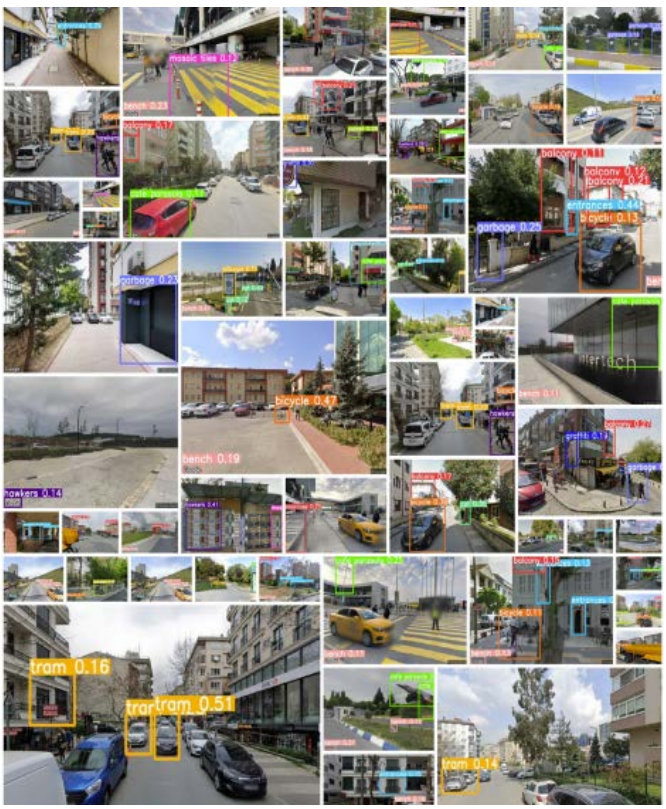
The best matching videos are then extracted from the larger videos and can be thus, used for further analysis.

https://drive.google.com/file/d/171VjQIFuZqkE2qJOA15l3KQfn9EEV76w/view?usp=share_link

https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingImagesAndVideos/VideoExtractionWithImageClassification.docx



Still of all 12 best matching clips most similar to the extracted film based on the input image



Results of Object Detection

Object Detection

The need to identify street objects for analysis was imminent. A custom trained Object Detection model was trained so that objects that were required to be focused on is identified.

26 objects specific to Istanbul were identified and images were scraped from Google in order to create a dataset for training. Each object had approximately 200 images.

Objects in images were annotated with bounding boxes using a labelling tool called labelImg. The annotations are saved in text files and converted to the YOLO format.

YOLOv5, a pretrained object detection model was used as the configuration of the custom model. The entire collection of images and txt files are divided to make a training set and a validation set and the YOLOv5 training script was used to conduct the training.

Custom Object Detection Model
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/ObjectDetection/ObjectDetectionCustomTrainedModel_IstanbulObjects.ipynb
Custom Object Detection Model - Training
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/ObjectDetection/ObjectDetection_training.ipynb

DESIGN OF ALGORITHM TO DISCERN STRUCTURES

Understanding the underlying characteristics of the data being analysed and selecting an appropriate technique that can efficiently identify and represent the underlying patterns are crucial when creating algorithms for identifying structures. Some common methods are

Clustering
Clustering algorithms group similar objects or data points together into clusters based on their features or characteristics. This can be useful for discerning structures in data that have natural groupings or patterns.

Graph theory
Graph theory algorithms analyze relationships between objects or data points by representing them as nodes in a graph and their relationships as edges. This can be useful for discerning structures in data that have complex relationships, such as social networks or chemical compounds.

Dimensionality reduction
Dimensionality reduction algorithms reduce the number of features or dimensions in a dataset while retaining as much of the original information as possible. This can be useful for discerning structures in high-dimensional data, such as images or text.

Pattern recognition
Pattern recognition algorithms analyze data to identify recurring patterns or motifs. This can be useful for discerning structures in data that have repeating elements, such as DNA sequences or music.

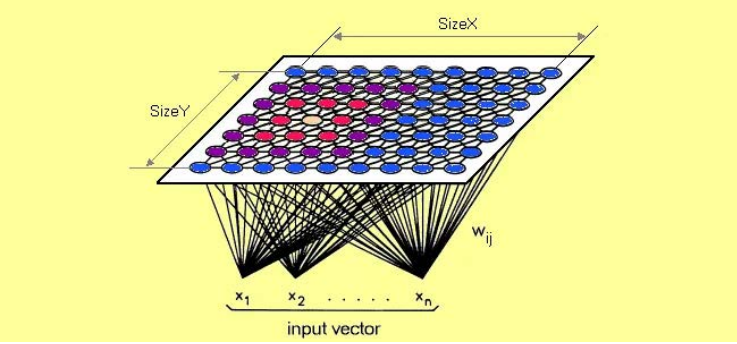
Neural networks
Neural networks are a type of machine learning algorithm that can be trained to identify patterns or structures in data. They consist of interconnected nodes or neurons that process information and can be trained to recognize specific features or patterns in the data.

SELF-ORGANISING MAPS (SOM)

A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction.

Each data point in the data set recognizes themselves by competeting for representation. SOM mapping steps starts from initializing the weight vectors. From there a sample vector is selected randomly and the map of weight vectors is searched to find which weight best represents that sample. Each weight vector has neighbouring weights that are close to it. The weight that is chosen is rewarded by being able to become more like that randomly selected sample vector. The neighbors of that weight are also rewarded by being able to become more like the chosen sample vector. This allows the map to grow and form different shapes. Most generally, they form square/ rectangular/hexagonal/L shapes in 2D feature space.

Best Matching Unit is a technique which calculates the distance from each weight to the sample vector, by running through all weight vectors. The weight with the shortest distance is the winner. There are numerous ways to determine the distance, however, the most commonly used method is the Euclidean Distance, and that's what is used in the following implementation.



```
featureImagePairs = []
for i in range(len(features)):
    featureImage = {}
    featureImage['image'] = pictures[i]
    featureImage['feature'] = features[i]
    featureImagePairs.append(featureImage)
```

Cell 14

The images to be used for the SOM are converted into vectors representing their features (dim=1024), through the image classification model. The features are then paired with the images themselves, in order to access them when the SOM grid is being filled with the images corresponding to the BMU.

The Algorithm:

```
# Dimensions of the SOM grid
m = 10
n = 10
# Number of training examples
n_x = 10000
rand = np.random.RandomState(0)
# Initialize the training data
train_data = features
# Initialize the SOM randomly
SOM = rand.uniform(0, 6, (m, n, 1024)).astype(float)
```

Cell 15

```
total_epochs = 10
for epochs, i in zip([1, 4, 5, 10], range(0,9)):
    total_epochs += epochs
    SOM = train_SOM(SOM, train_data, learn_rate =
.05, radius_sq = 0.5,
                lr_decay = .05, radius_decay = .05,
epochs = 50)
```

Cell 16

```
for fi in featureImagePairs:
    g,h = find_BMU(SOM,fi['feature'])
    SOMimages[g][h].append(fi)
```

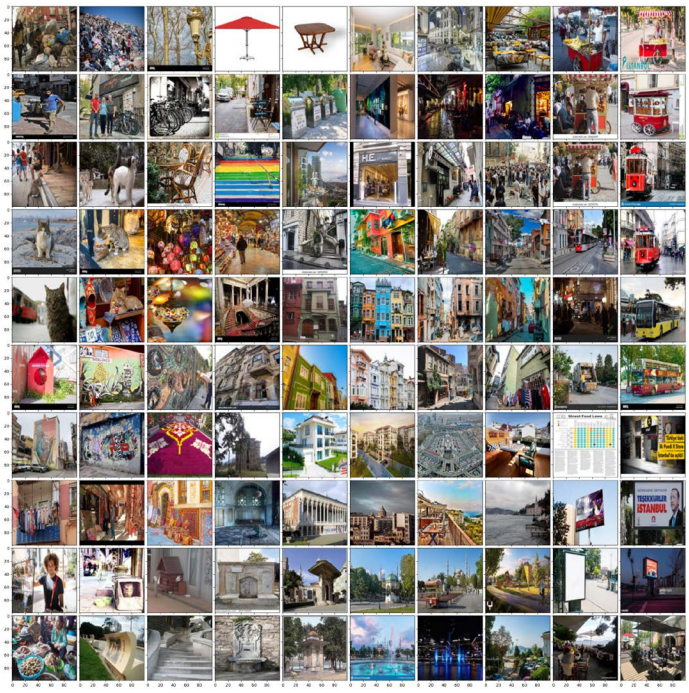
Cell 19

The dimensions of the SOM grid is set and the SOM is initialised. A vector is chosen at random from the set of training data.

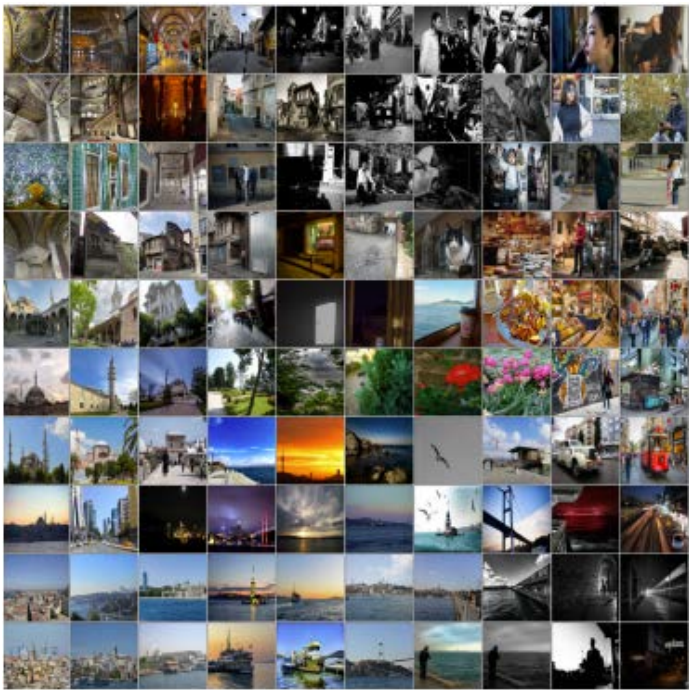
The SOM is trained with different epoch values. Through the train_SOM funtion, very node is examined to calculate which one's weights are most like the input vector. The winning node is known as the BMU. The neighbourhood of the BMU is then calculated. The amount of neighbours decreases over time. The winning weight is rewarded with becoming more like the sample vector. The neighbours also become more like the sample vector. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbour is from the BMU, the less it learns.

The images corresponding to the BMU is assigned to the grid of SOMimages by going through the each image's feature vector, resulting in a large image dictionary, in association to their specific location.

https://drive.google.com/file/d/1ez5AJNsbreBl9o3QhhXNXLQtrolsRKRY/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/DesignOfAlgorithms/SOMandPCA_ScrapedImages.docx



SOM trained images scraped from Google and Instagram (4544)



SOM trained images scraped from Google and Instagram (4275 + 2845)

SOM was carried out on images scraped from Google and Instagram in the project in order to examine how the data interacts with each other. Although the grid only provides information of 100 images, the folders containing the images most similar to each other gave insight on how objects and scenes are connected.

PRINCIPAL COMPONENT ANALYSIS (PCA)

Principal component analysis (PCA) is a technique for analyzing large datasets containing a high number of dimensions/features per observation, increasing the interpretability of data while preserving the maximum amount of information, and enabling the visualization of multidimensional data. PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

PCA reduces the dimensionality of a dataset by linearly transforming the data into a new coordinate system where (most of) the variation in the data can be described with fewer dimensions than the initial data.

To qualitatively analyse the images grouped through SOM, PCA was carried out.

```
all_images = np.array(all_images)
num_images, h, w, c = all_images.shape

flattened_images = np.reshape(all_images, (num_images,
h*w*c))

pca = PCA(n_components=2)
transformed_data = pca.fit_transform(flattened_images)
```

Cell 28

The images that were grouped after running SOM on scraped images were converted into a 2-D array and transformed into 1-D vectors. PCA was then carried out with no. of components set as 2. The fit_transform method computes the principal components and performs dimensionality reduction on the images. It returns the transformed data, where each image is now represented by a 2-dimensional vector, corresponding to the values of the first two principal components.

https://drive.google.com/file/d/1ez5AJNsbreBl9o3QhhXNXLQtrolsRKRY/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/DesignOfAlgorithms/SOMandPCA_ScrapedImages.docx

Projections of PCA carried out on images grouped in SOM

VECTORISATION OF TEXTS

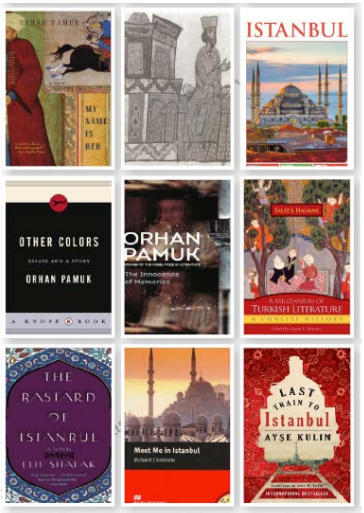
Vectorization of text refers to the process of converting text data into numerical vectors, which can be used as input to machine learning models. This is a crucial step in natural language processing (NLP) tasks, such as text classification, sentiment analysis, and language translation.

There are several techniques for vectorizing text data. One of the most popular methods is the **Bag of Words (BoW)** approach, which involves representing a text document as a set of its constituent words, ignoring the order in which they appear. This is done by creating a vocabulary of all unique words in the corpus and assigning each word a numerical value based on its frequency in each document. Each document is then represented by a vector of the same length as the vocabulary, with each element corresponding to the frequency of the corresponding word in the document.

Another popular method is the **Term Frequency-Inverse Document Frequency (TF-IDF)** approach, which also takes into account the importance of each word in the corpus. TF-IDF assigns a weight to each word in a document based on its frequency in the document and its rarity in the corpus as a whole. This method often results in more informative vector representations than the BoW approach.

Other techniques for vectorizing text data include **word embeddings**, which represent words as dense, low-dimensional vectors that capture semantic relationships between words, and neural network-based approaches such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

The choice of vectorization technique depends on the specific NLP task at hand and the nature of the text data being analyzed.



Books were scraped from various websites. The criteria of the project was to use books authored by people of Istanbul and books written about Istanbul.

The analysis centred around street objects and spaces in Istanbul. The intension was to find how other words interact with these objects and spaces, and create a narrative of the street objects with reference to its relevance on how they have been described.

```
sentences = break_sentences(text)
sentences = [clean_string(s) for s in sentences]
```

Cell 8

```
myCorpus, book_index = read_books(folder_path)
```

Cell 9

The books are processed (read_books) and broken down (break_sentences) into sentences based on the punctuation marks (.?!) used and empty spaces and lines. They are also cleaned off of unnecessary spaces. The output results in a list of lists of sentences (myCorpus) and an index (book_index) of the sentence along with the name of the book it is extracted from.

```
words_to_remove = 'and or but nor yet so been now you your
yours yourself me we us yourselves could am was too is do will
are were would should did dont ever with was had have has
make for as the this from although because before again would
even though if in order that provided that since so that than
though unless until when whenever where which whichever where-
as wherever whether while'
```

Cell 11

A list of words (words_to_remove) that include conjunctions, pronouns, and prepositions are put together. These words, known as stop words, occur frequently in sentences, adding a lot of noise to the corpus. Removing them reduces the dimensionality of the space used to represent the text and hence, provide more efficiency to the machine learning models.

https://drive.google.com/file/d/1Zfhlvj7ZRU7csfnTSXUC7Dot33YJjC80/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingText/NLP_TFIDF.docx

There are some cases where keeping certain stop words may be important for certain NLP tasks. For example, in sentiment analysis, negation words such as “not” and “never” can be important to preserve, as they can completely change the sentiment of a sentence.

```
texts = [[word.replace(".","").replace(",","").re-
place("'","").replace(":","").replace("`","") for word
in document.lower().split()]
        for document in myCorpus]

texts = [[word.replace(".","").replace(",","").
replace("'","").replace(":","").replace("`","") for
word in document.lower().split()]
        for document in myCorpus]
```

Each document in the corpus is processed by converting it to lowercase, splitting it into individual words, and removing specific characters such as punctuation. The resulting processed words are stored in the texts list, maintaining the same document structure as the original corpus.
texts are filtered further by removing stopwords (words in the stoplist) and excluding words with a length of less than or equal to 2, cleaning the corpus before further analysis.

```
processed_corpus = [[token for token in text if
frequency[token] > 1] for text in texts]
pprint.pprint(processed_corpus)
```

Cell 12

The resuting processed_corpus is a list of lists containing all documents (sentences) stripped off of any unnecessary characters and words, ready for analysis.

```
processed_corpus[23458]
```

```
Output: ['grab','african','mask','wall','dash','for-
        ward','without','thinking',about','what','do-
        ing']
```

Cell 14

```
myCorpus[23458]
```

```
Output: 'I grab the African mask on the wall and dash
        forward without thinking about what I am doing'
```

Cell 15

```
dictionary = corpora.Dictionary(processed_corpus)
print(dictionary)
```

Cell 16

A dictionary based on the processed_corpus is created using the dictionary class from the corpora module is used to create a mapping between words and their unique integer IDs in the corpus.

```
bow_corpus = [dictionary.doc2bow(text) for text in
processed_corpus]
```

Cell 18

The doc2bow method is used to convert the processed_corpus into a a list of BoW representations for each document in the corpus, consisting of lists of tuples that represents a word in the document and its frequency (count) within that document.

```
tfidf = models.TfidfModel(bow_corpus)
```

Cell 19

The TF-IDF Model class is used to compute the TF-IDF weights for each term in each document. It is a bag-of-words model that considers the presence or absence of words in a document rather than their order.

```
find_similar_sentences("street", 10, "C:\\Users\\
avey2\\OneDrive\\Desktop\\topSentences\\street.txt",
myCorpus, book_index)
```

```
Output: 10 Sentences with highest similarity to 'street' are:
        Index: 117034  Score: 0.6030227.....Index: 118074  Score:
        0.57735026
        Book Last_train_to_istanbul_a_novel.txt - Sentence 1: How
        much longer, corner to corner, street by street ......
        Book Last_train_to_istanbul_a_novel.txt - Sentence 10: He
        turned quickly into the Street of the Store Clerks, and
        from there into the Street of the Lumber Sellers, after
        which he walked for some time without looking at the street
        names
```

Cell 21

The find_similar_sentences function is built in such a way where a word is used as an input query and its similarity is checked in the entire corpus using the TF-IDF model. The resulting output is a list of 'n' (here, 10) documents most similar to the query, indexed in reference to the corpus and giving scores of its similarity.
A list of sentences corresponding to these indexes are extracted in order to understand their context.

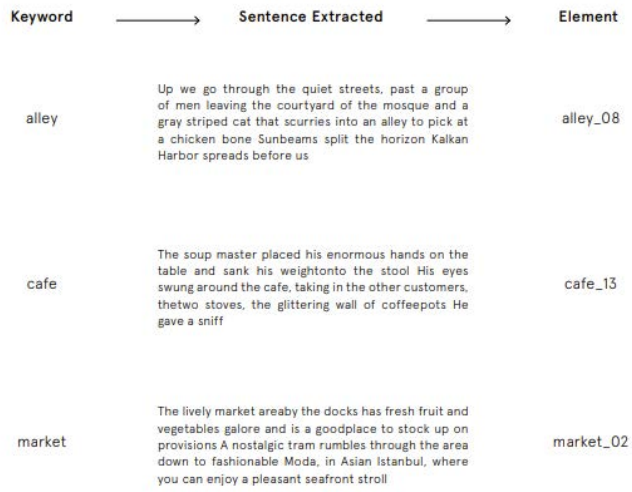
Key words, describing the street objects and spaces, were used to further extract ten sentences each that had similarity with respect to each word in the list of key words. Each sentence was then processed as a separate text file, labelled corresponding to the keyword.

```
key_words = ['Bench', 'Chair', 'Table', 'Store-
front', 'Kiosk', 'Vending cart', 'Garbage', 'Taps',
'Fountain', 'Billboard', 'Bicycle', 'Bus stop',
'Lamp', 'Tram', 'Parasol', 'Cat', 'Cathouse', 'Stair-
case', 'Mosaic', 'Graffiti', 'Carpet', 'Dilapidated',
'Doorway', 'Hawkers', 'Balcony', 'Postbox', 'Tea',
'Coffee', 'ATM', 'Café', 'Garden', 'Playground',
'Restaurant', 'Bar', 'Promenade', 'Ferry', 'Bridge',
'Mosque', 'Market', 'Library', 'Parking', 'Street',
'Footpath', 'Alley', 'Subway', 'Underpass', 'The-
atre', 'Toilet', 'Road']
```

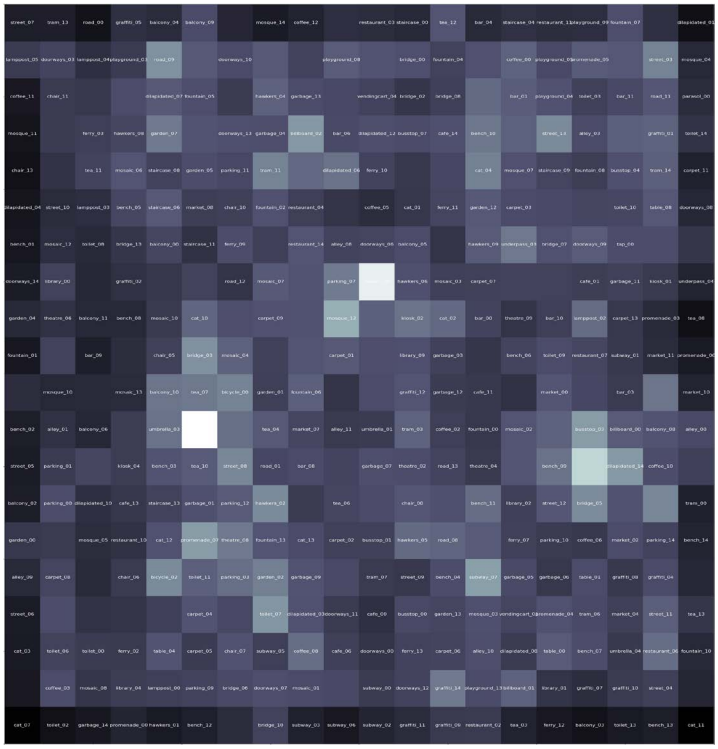
Cell 22

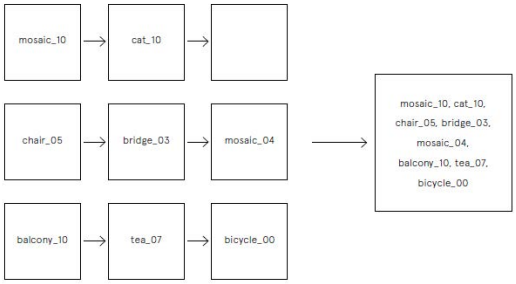
```
for kw in key_words:
    file_name = f"C:\\Users\\avey2\\OneDrive\\Desk-
top\\topSentences\\{kw}.txt"
    find_similar_sentences(kw, 10, file_name, myCorpus,
book_index)
```

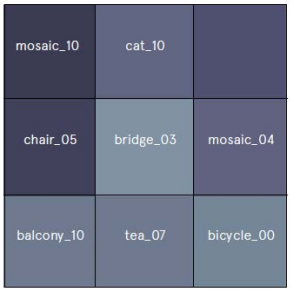
Cell 23



https://drive.google.com/file/d/1Zfhlvj7ZRU7csfnTSXUC7D0t33YJjC80/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingText/NLP_TFIDF.docx







SOM was applied on the text files of individual sentences in order to understand how the text grouped together. A cluster of nine adjoining cells are chosen to get a group of words. These sentences corresponding to the words were used as a prompt in MidJourney creating a visual representation the clustered words.

Word2Vec

In order to capture word semantics and relationships rather than focusing on word importance within a document and overall corpus (TF-IDF), the Word2Vec model is explored. Word2Vec is a shallow, two-layer neural network-based model that learns word embeddings or word vectors from large amounts of unlabeled text data. It represents each word as a dense, low-dimensional vector in a continuous vector space, capturing semantic and syntactic relationships between words.

The context in which words appear by learning from the surrounding words within a fixed-size window. It supports the notion of word similarity and can perform arithmetic operations on word vectors, such as word analogies (e.g., “king” - “man” + “woman” = “queen”).

```
model = w2v(sentences=processed_corpus)
```

Cell 18

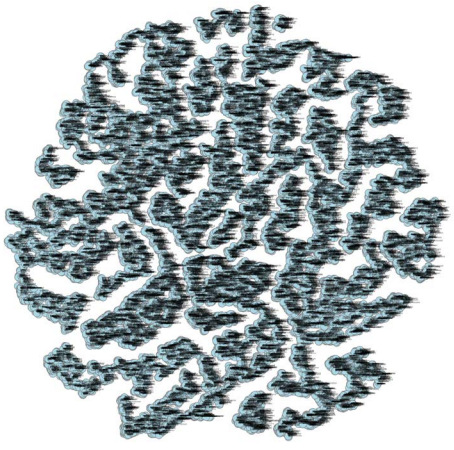
```
model.wv.most_similar(positive=['chair'], topn=10)
```

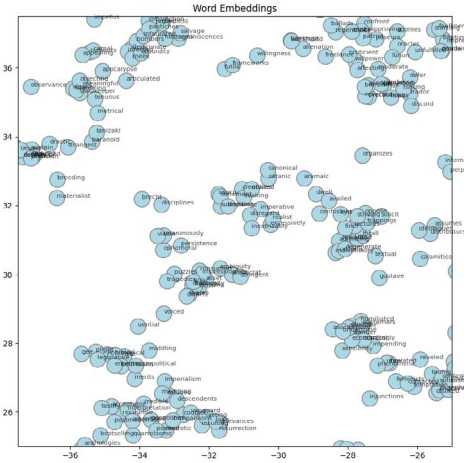
```
Output: [('sofa', 0.9273675084114075), ('bag', 0.9245791435241699), ('lap',
         0.9157040119171143), ('desk', 0.9113755822181702), ('bedroom',
         0.9086191654205322), ('grabbed', 0.9084243774414062), ('seat',
         0.9005312919616699), ('rope', 0.8999112248420715), ('knife',
         0.8949291706085205), ('basket', 0.8937400579452515)]
```

Cell 20

The Word2Vec model is trained using the processed_ corpus, using the default vector size of 100 and looking window of 5 (the algorithm looks at 5 words on both sides of the target word as the context). it also uses the Continuous-bag_of_words algorithm, where it tries to maximize the probability of the target word given the context words.

https://drive.google.com/file/d/1p8E6nxlZuEx4oiZAQeQDQAwghDcu79SW/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingText/NLP_Word2Vec.docx





PCA is carried out on the model where the dimensionality is reduced and the 2-D visualisation of the distribution of the corpus is possible.

```
w = w2v(
    new_processed_corpus,
    min_count=3,
    sg = 1,
    window=6
)
```

Cell 34

```
bench1_similar_words_scores = w.wv.most_simi-
lar('bench')
print(bench1_similar_words_scores)
bench1_similar_words = [word for word, score in
bench1_similar_words_scores]
print(bench1_similar_words)
```

Cell 35

The Skip_gram algorithm (sg = 1) is another Word2Vec algorithm where it predicts the context words given a target word. It tries to maximize the probability of the context words given the target word.

```
Output: [('treasury', 0.2589263916015625),
        ('long', 0.2546565532684326), ('restau-
        rant', 0.2514311969280243), ('came',
        0.25054702162742615), ('cush-
        ion', 0.2492130696773529), ('phone',
        0.2351987361907959), ('place',
        0.2254539132118225), ('brought',
        0.22435767948627472), ('toy',
        0.21568715572357178), ('dragged',
        0.20796334743499756)]
        ['treasury', 'long', 'restaurant', 'came',
        'cushion', 'phone', 'place', 'brought', 'toy',
        'dragged']
```

https://drive.google.com/file/d/1CJTpp-qiABKSFme5wBY8OAfBkpdKHpeY/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/blob/main/RC11_22165920_CodeBook/Julian/VectorisingText/NLP_OD_Prompts.docx

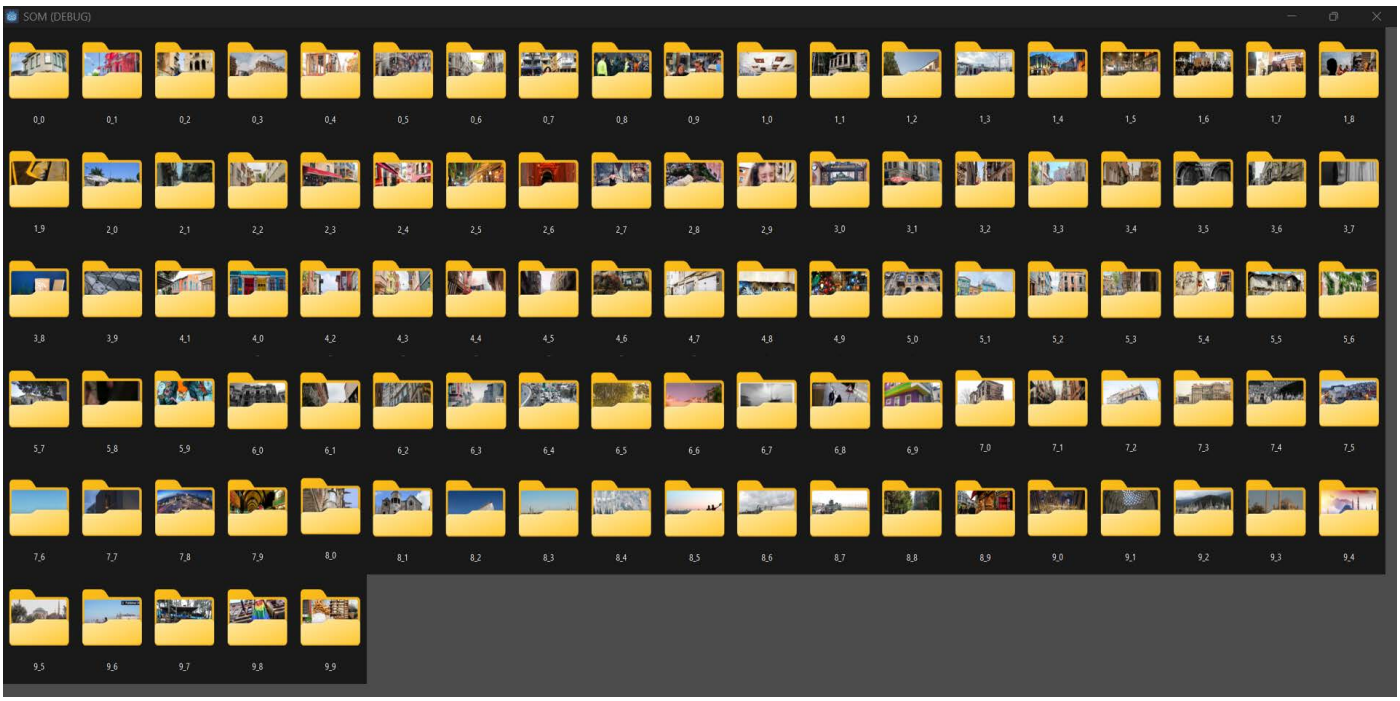
The output list of context words most similar to bench along with bench is used to create a one sentence prompt with the assistance of ChatGPT.

Prompt: "I came across a restaurant with a bench outside, where a woman sat with a cushion brought from home, playing with a toy phone while her child dragged a treasure chest along the pavement."

The intention of this exploration is to curate a prompt for Stable Diffusion in Blender that allows to render the 3D reconstruction of a Google street view. The objects detected through the custom trained Object detection model in the street view is to be used as a list of key words to obtain context words and create relevant prompts.

GODOT

The outcomces of the SOM was used as the input for visualisation. Clicking on Individual folders thumbnails on the screen to enter into the folder in order to view contents of the folder was the primary goal.



Main Scene

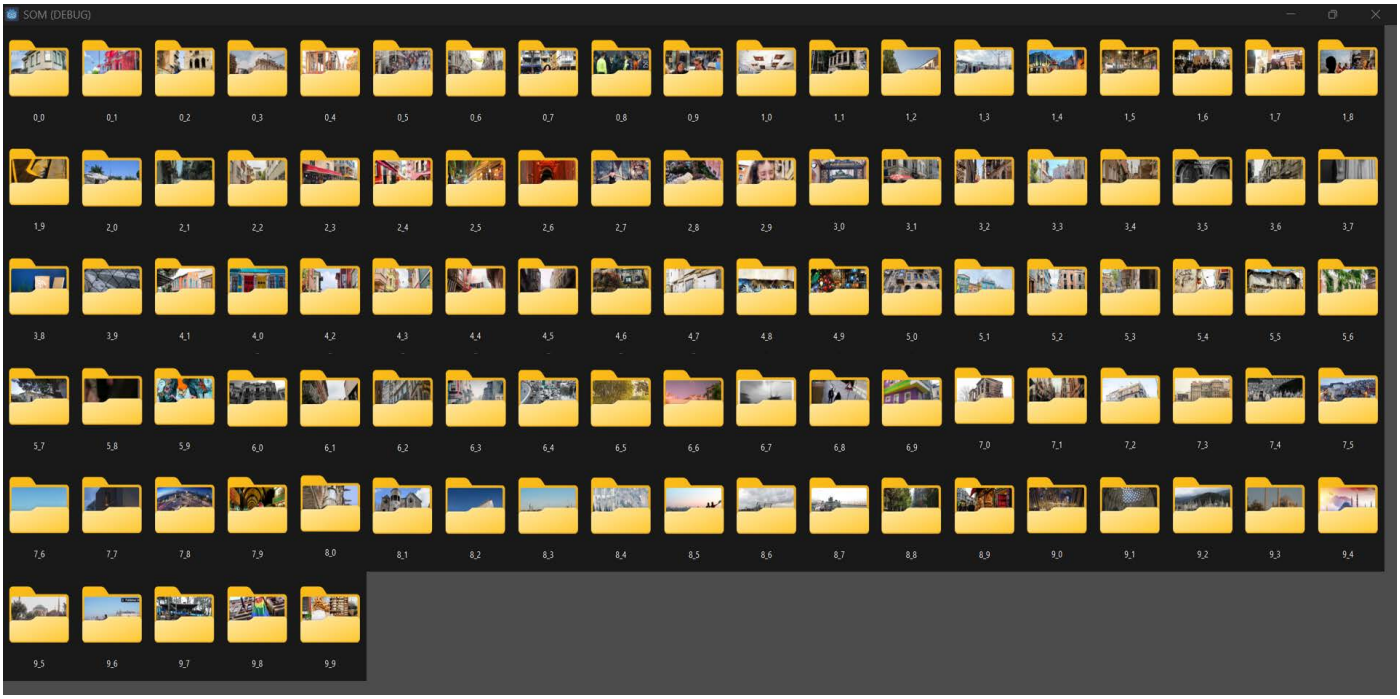


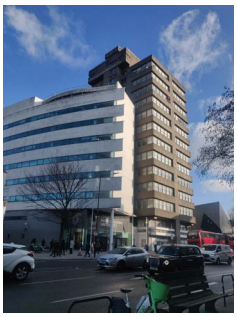
Image Scene - Folder 4_1

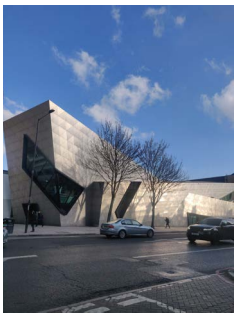
3 | 3D RECONSTRUCTION AND STABLE DIFFUSION

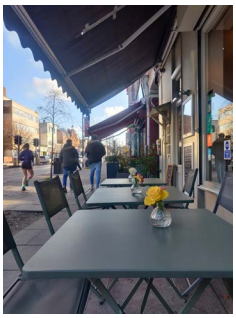
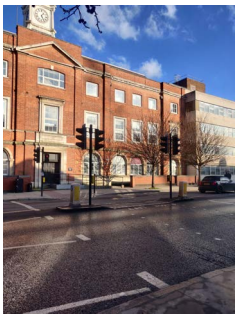
SCRAPPING AND MAPPING

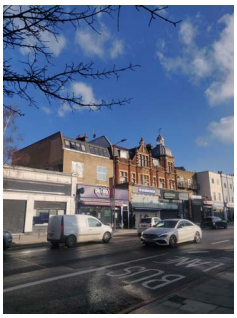
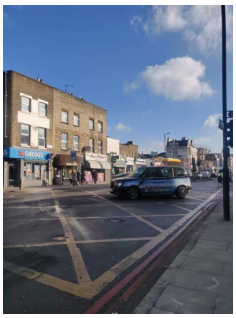
The timeline in Google Maps was used to map a route and it was imported to Blender by converting the kml file to a gpx file. The images taken along the route were processed to get the GPS data and then imported to Blender. This enabled getting a visualisation of the route with imagery. Using Blender OSM, geometry fro building and roads were also added. A camera path was set along the route.

Route taken for mapping

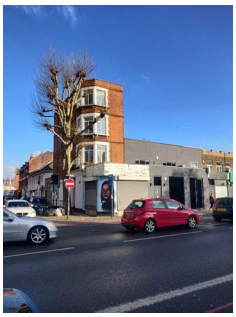





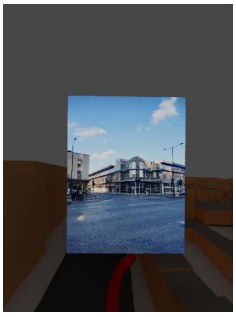
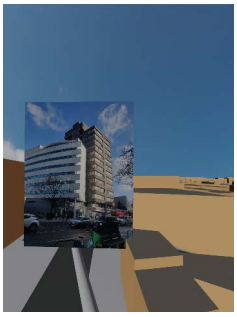
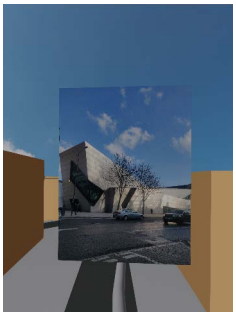
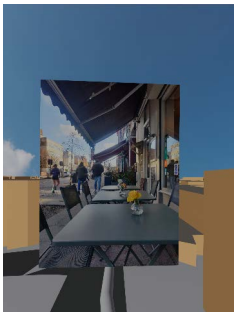
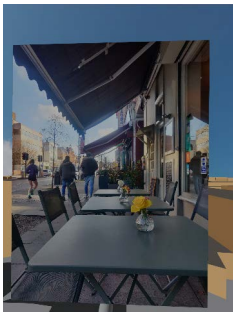




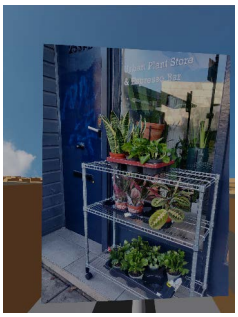
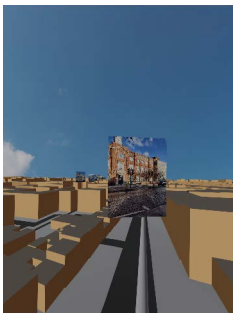
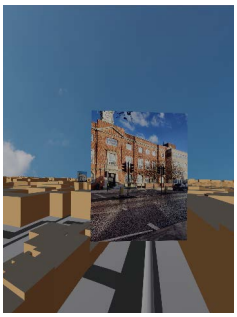
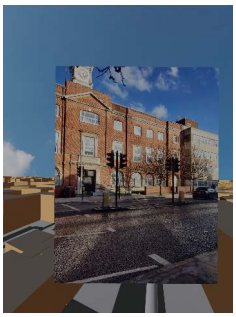
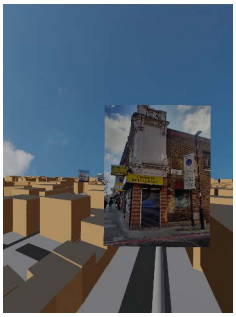
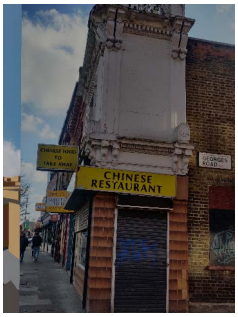


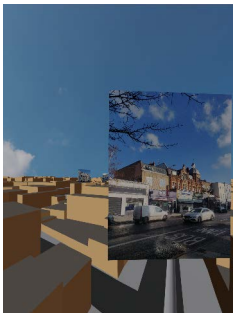
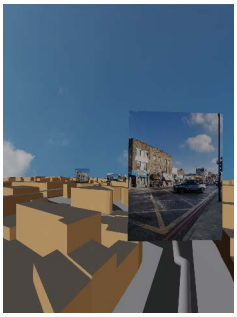
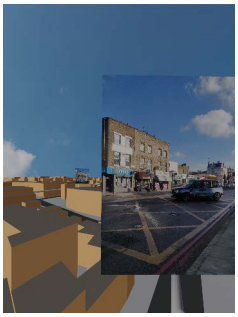
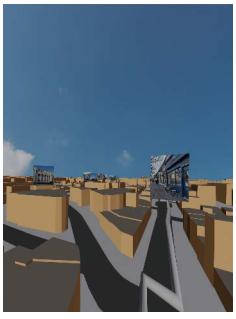


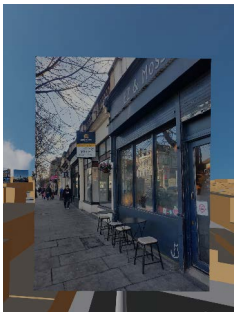
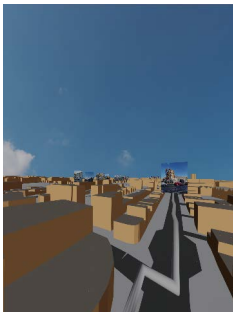
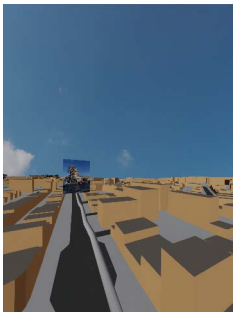

Images taken along the route (15 of 113)

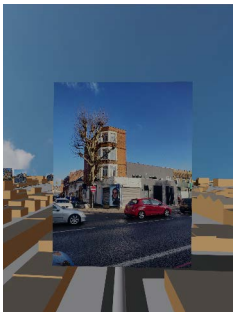
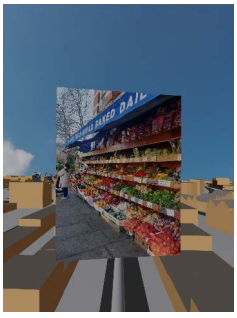
Output Frames


https://drive.google.com/file/d/1k9CvuXz5u3ZEqwZ2tlzvVYuz3F2nrw4K/view?usp=sharing
https://github.com/RC11-SkillsClass2022-23/22165920/tree/main/RC11_22165920_CodeBook/Joris/ScrapingAndMapping

3D RECONSTRUCTION USING PHOTOGRAMMETRY

Photogrammetry is the science of using photographs to extract information about the physical world and create 3D models. 3D reconstruction using photogrammetry is the process of creating a 3D model of an object or scene by using photographs taken from different angles.

The process of 3D reconstruction using photogrammetry typically involves the following steps:

- Image acquisition: Multiple images of the object or scene are captured using a camera from different angles. It is important to capture enough images to ensure that all parts of the object or scene are visible from at least two different angles.
- Image processing: The images are processed to remove any lens distortion and to identify common features in each image.
- Point cloud generation: The common features in each image are used to create a point cloud, which is a set of 3D points that represent the surface of the object or scene.
- Mesh generation: The point cloud is converted into a mesh, which is a collection of triangles that form a surface that represents the object or scene.
- Texture mapping: Texture information is added to the mesh to provide a realistic representation of the object or scene.
- Refinement: The mesh can be refined by adding additional details, filling in gaps, or removing unwanted artifacts.

Photogrammetry can be used to create 3D models of a wide range of objects and scenes, including buildings, landscapes, and even human faces. It has a number of advantages over other 3D reconstruction techniques, including its ability to create highly accurate models and its relatively low cost compared to other methods.

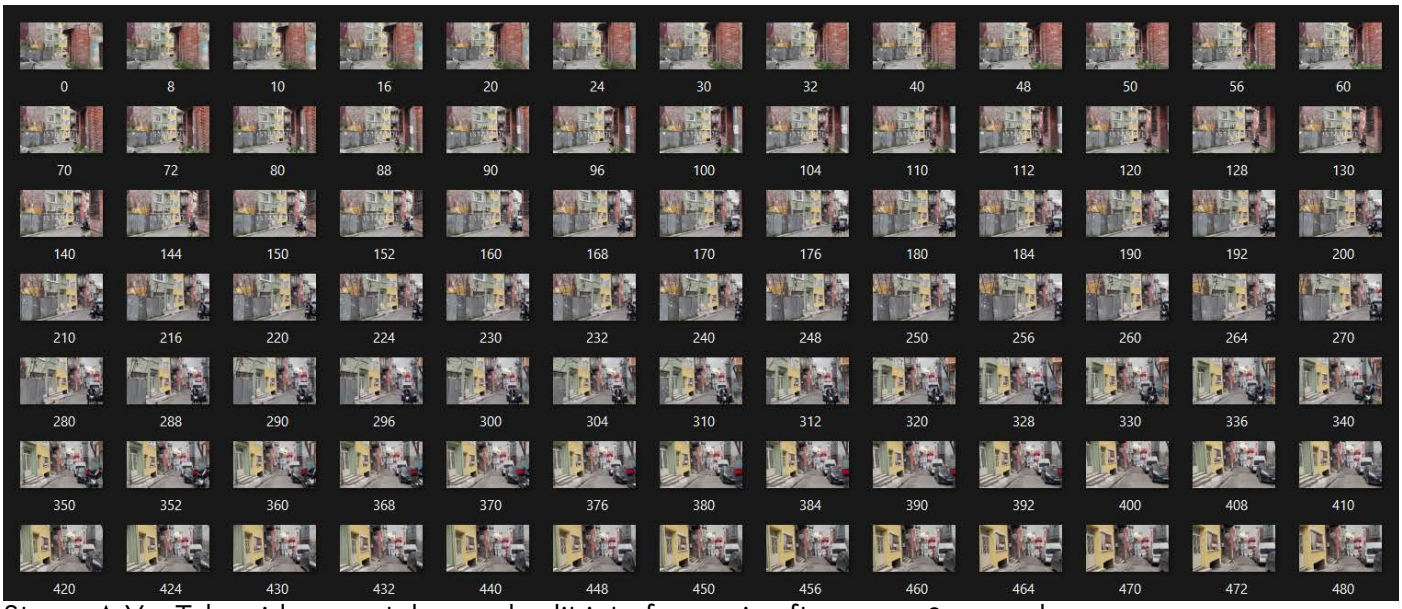
One of the main challenges of 3D reconstruction using photogrammetry is ensuring that the images are captured from the correct angles and distances. In addition, the accuracy of the resulting model can be affected by factors such as lighting conditions, camera resolution, and lens distortion. However, these challenges can be overcome through careful planning and the use of specialized software tools.

How does it work

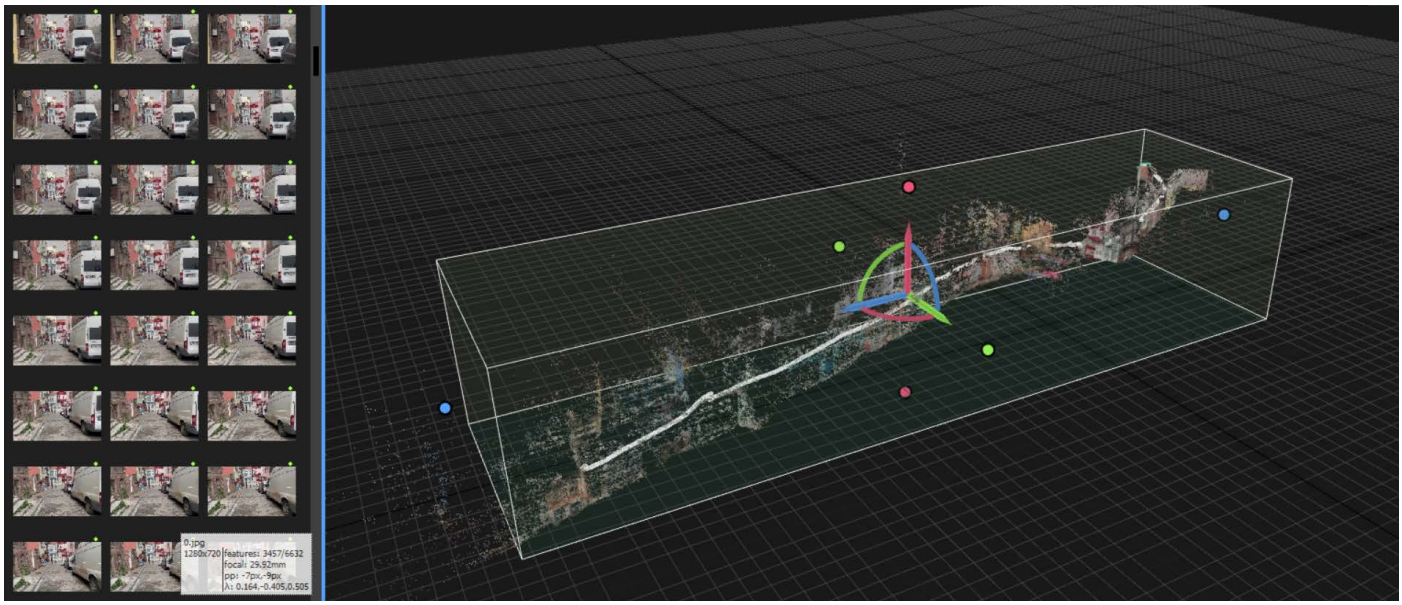
The science behind 3D reconstruction using photogrammetry lies in the use of triangulation to calculate the position of points in space based on their appearance in multiple photographs. Triangulation is a geometric technique that involves using the angles and distances between two or more points to determine their exact positions in 3D space.

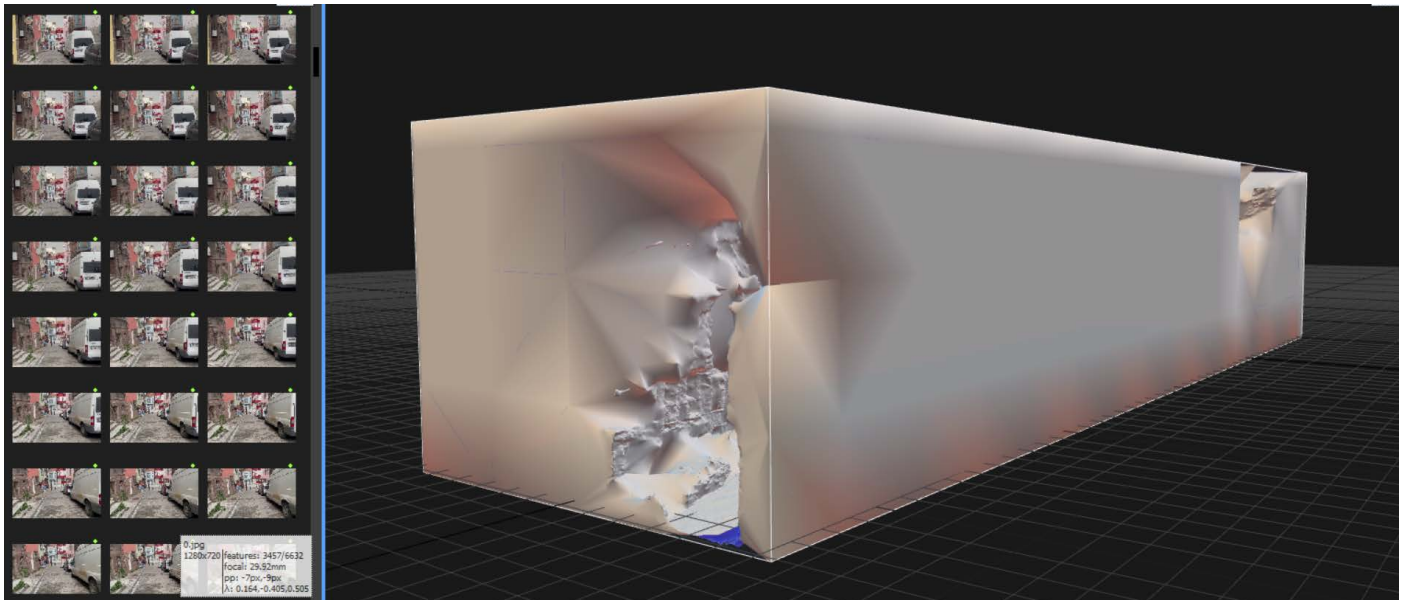
In photogrammetry, triangulation is used to calculate the position of points on the surface of an object or scene by identifying common features in multiple photographs. By matching the location of these common features in different photographs, a 3D point cloud can be generated, which represents the surface of the object or scene in three dimensions.

Once a point cloud has been generated, it can be converted into a mesh by connecting the individual points to form a surface. Texture information can then be added to the mesh to provide a realistic representation of the object or scene.

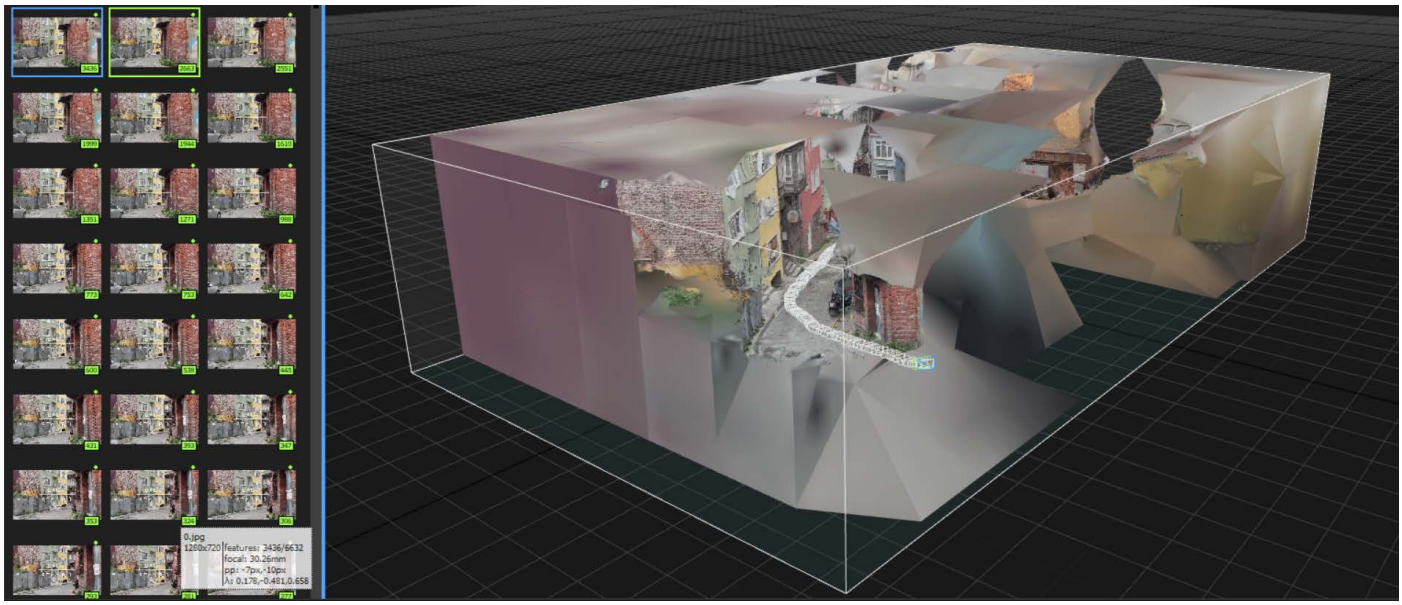


Step1 : A YouTube video was taken and split into frames in after every 8 seconds

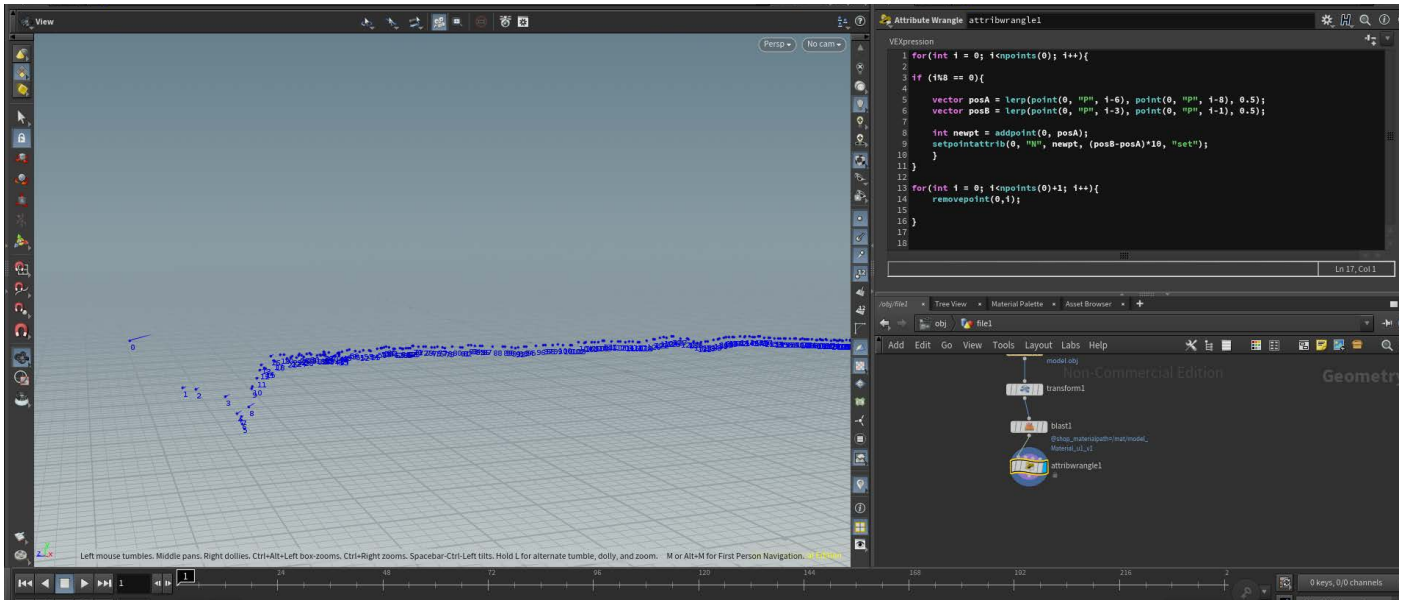

Step2 : The images are loaded into RealityCapture and the images are aligned in order to obtain a point cloud.


Step3 : The point cloud is then reconstructed into a mesh.

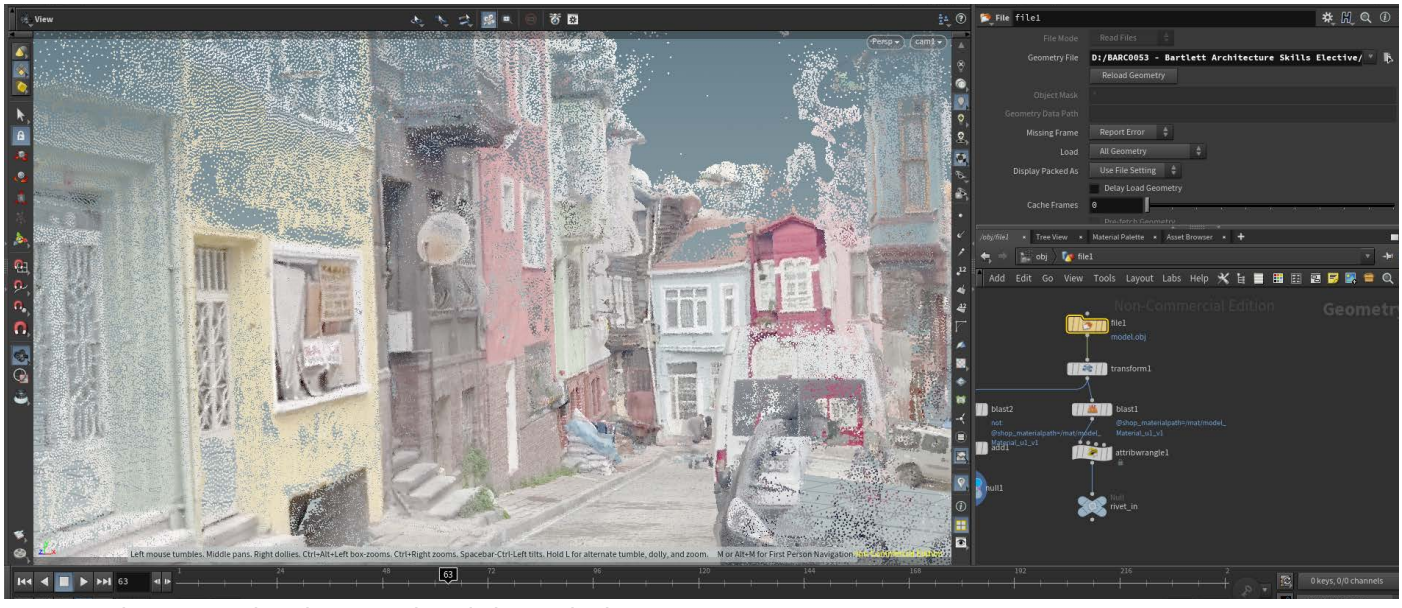
https://github.com/RC11-SkillsClass2022-23/22165920/tree/main/RC11_22165920_CodeBook/Joris/MappingAVideo



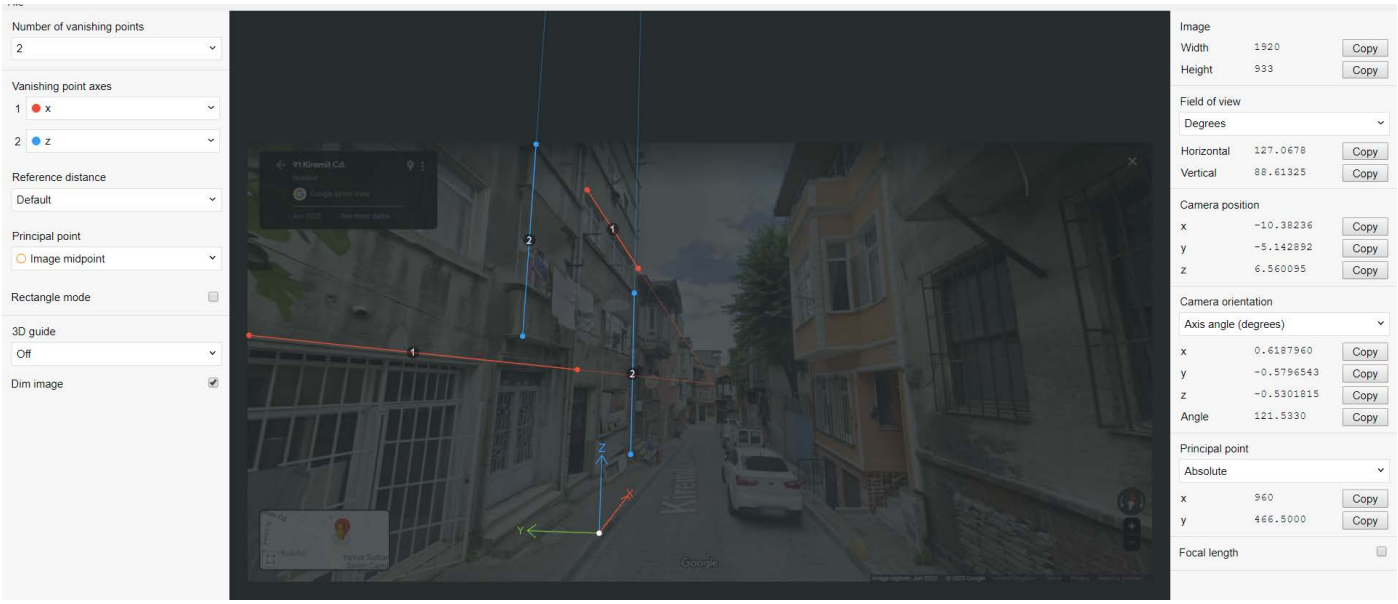
Step4 : Texturised the mesh, now the reconstructed mesh contains the camera path and textures associated to the images.



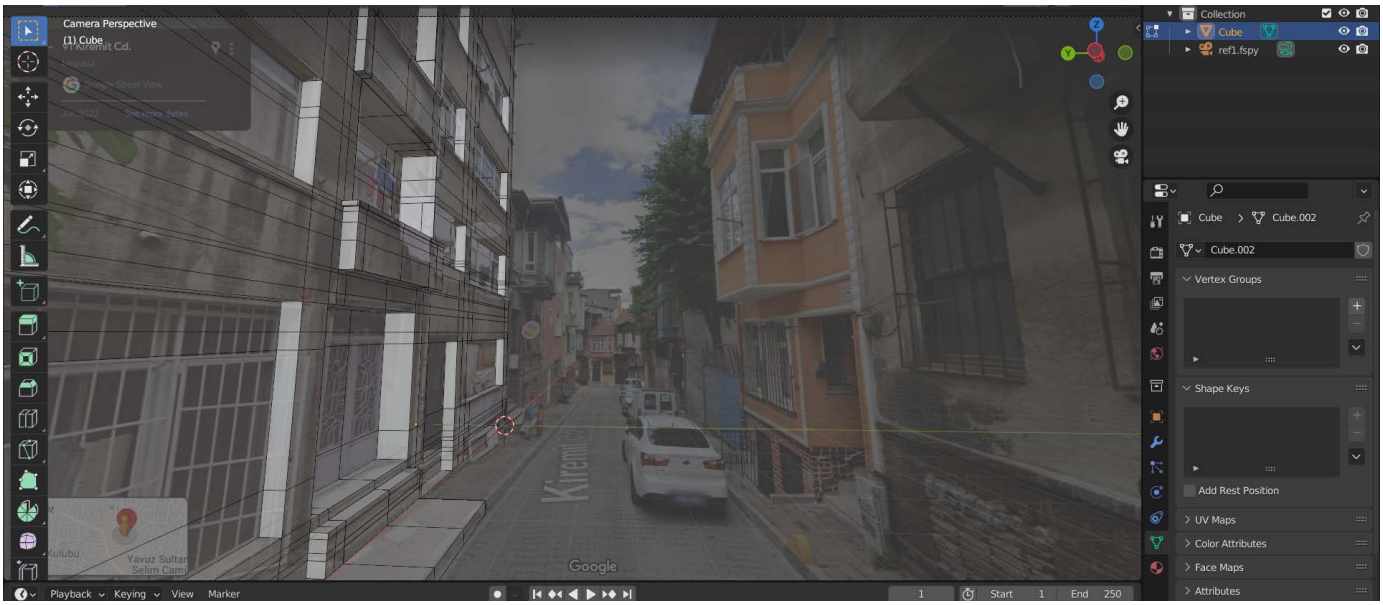
Step5 : The camera points and directions are determined to establish the camera path in Houdini



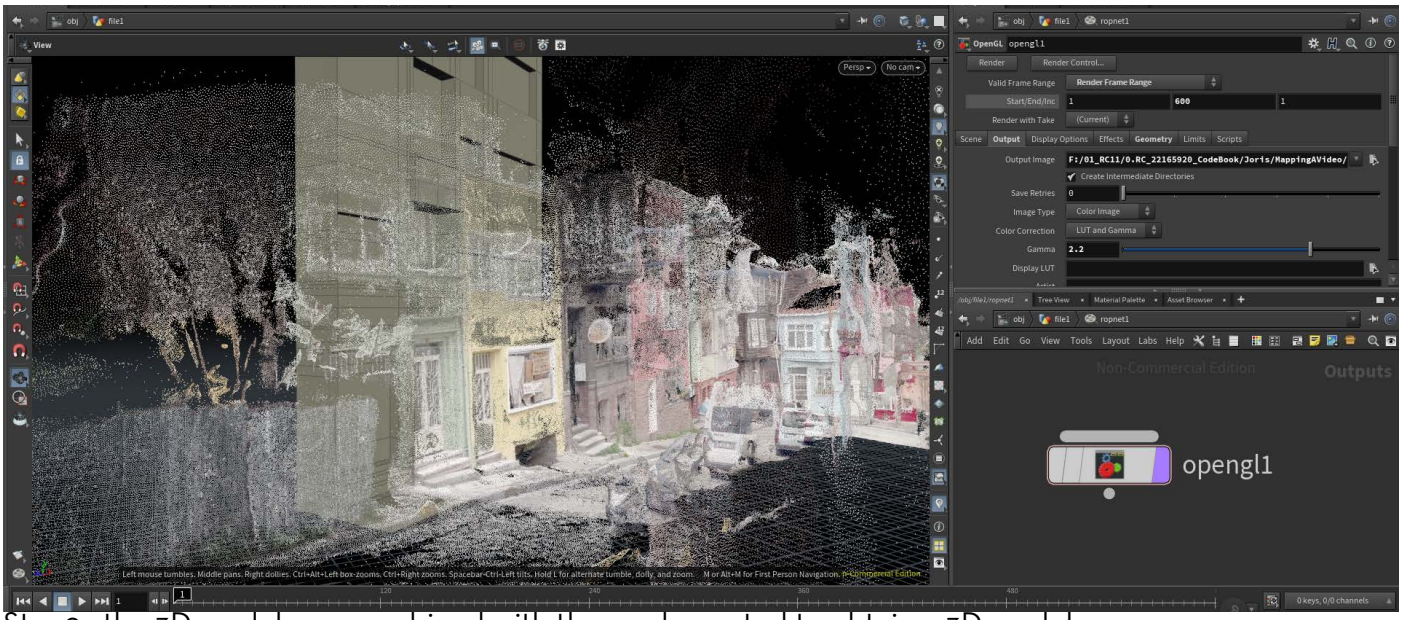
Step6 : The point cloud is visualised through the camera.



Step7 : Using fSpy, the position and orientation of the camera in the scene is adjusted with an image that is obtained from Google street view to reconstruct the buildings on Blender



Step8 : The buildings are modelled on Blender.



Step9 : the 3D models are combined with the mesh created to obtain a 3D model

https://drive.google.com/file/d/1UuxHbJiUR5YiDqtCRmKIzSQfVoXGbvXu/view?usp=sharing

STABLE DIFFUSION

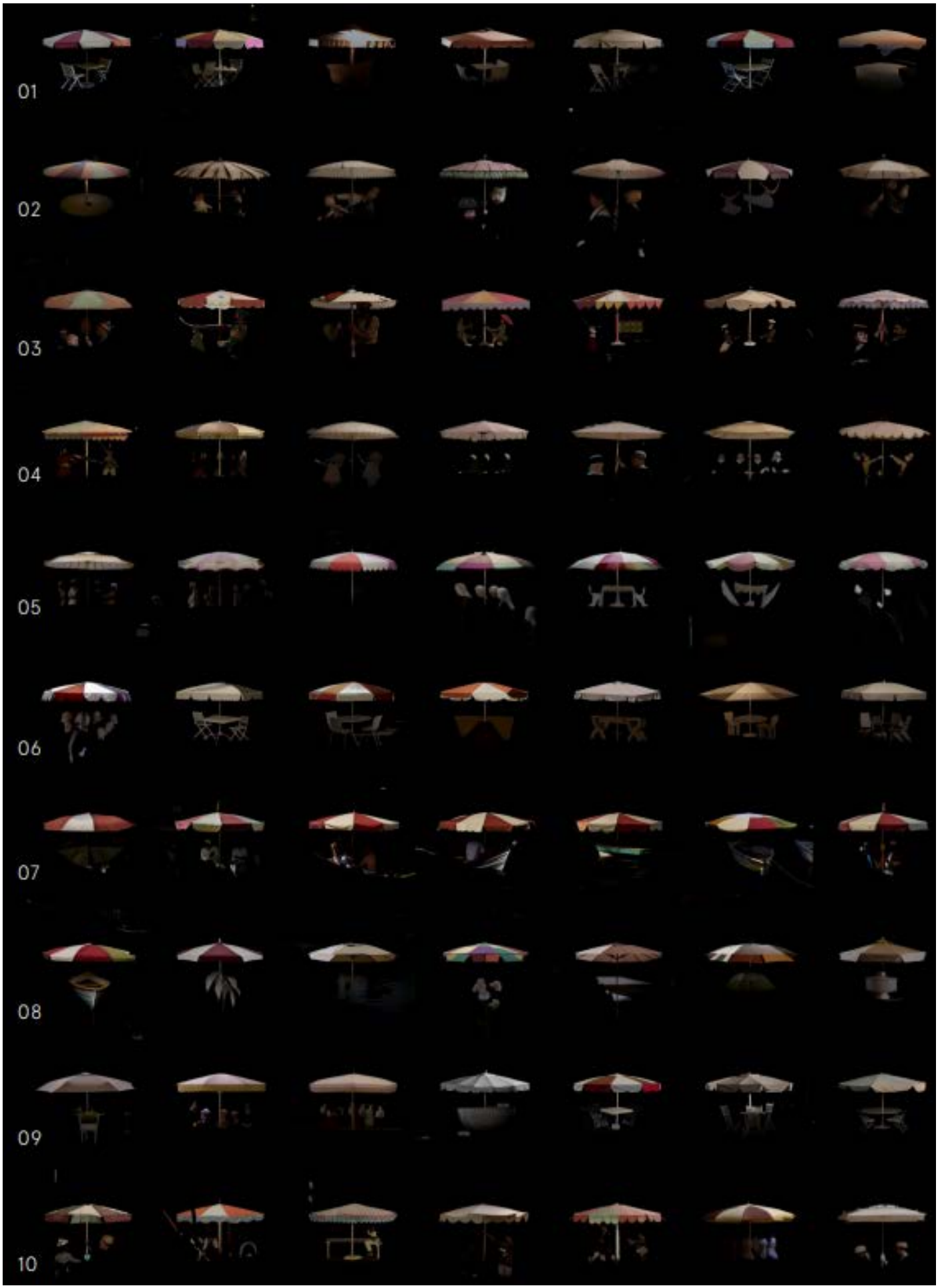
The idea was to get iterations of an object in different flavours that are guided by text that have been written in Istanbul. The name of the object is considered as the keyword and text scraped from books are processed to extract the best matching sentences that include them. A model of the object is used as an input in Blender, and rendered using Stable Diffusion. A single sentence from the sentences extracted is used as a prompt and “not Istanbul” is used as a negative prompt in the rendereing process.

Object : Parasol (Model)

Keyword : Parasol (Text)

Input model : Parasol
Prompt : sentences extracted from books
Negative prompt : Not Istanbul

Prompt 1	In an old picture each of these three rooms had a separate balcony, and today there is only one long balcony.
Prompt 2	They had a fine house on the Bosphorus, with a large balcony, almost covered by Virginian creeper, and here, going by in the steamer, I had often caught a glimpse of their heads as they sat on the balcony at work or afternoon tea.
Prompt 3	One day I stood on the balcony overlooking the Bosphorus, watching the tackmg of a ship that had caught the evening sun in its white sails Closer and closer it came, until I recognised the pale, sphinx-like face m the stern.
Prompt 4	High in the little balcony of the minaret he stands like a precentor leading the hymns of the people.
Prompt 5	At the same level is also a little garden, held up by a massive retaining wall, and a balcony with a rail of perforated marble once gave a magnificent view over the harbour.
Prompt 6	They point out to you the place whence the cock crew, and the balcony from which Pilate showed Jesus to the assembled multitude.
Prompt 7	The commencement exercises of the college were held in the large audiencerooms and the adjoining balcony overlooking the Bosphorus.
Prompt 8	A species of balcony, near this relic, commands an uninterrupted view of a face of Christ in the act of benediction, in mosaic work, on the ceiling of the half-dome, over the place where the altar formerly stood.
Prompt 9	Between this balcony and the Sultan’s box a richly latticed window, covering a space no larger than the cosey baignoires common to every French theatre, hung like a shimmering, semi-transparent curtain.
Prompt 10	Yet the arched and ivied windows high above, and the long line of jutting marble columns, once supporting an airy balcony, indicate that it was the abode of pleasure as well as of fear.

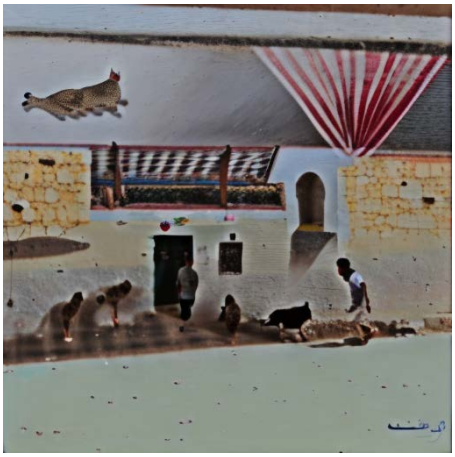
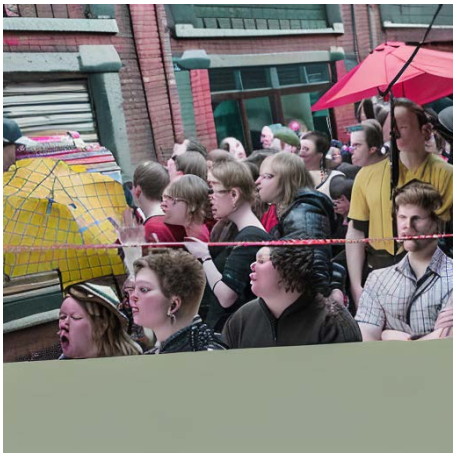


https://drive.google.com/file/d/1b4KPohJR-vb7XcEa4-2mFgJUkYe0_-zW/view?usp=share_link
https://github.com/RC11-SkillsClass2022-23/22165920/tree/main/RC11_22165920_CodeBook/Joris/StableDiffusion

A 3D model of an alley is used to render with Stable diffusion on blender with a sentence extracted from the corpus of books. Using the TF-IDF algorithm, the sentence that has most similarity with alley is used:

Prompt: "The crowd had stopped running People were standing in little groups, while those closest to the mouth of the alley had turned around and were craning their necks nervously to watch the square"

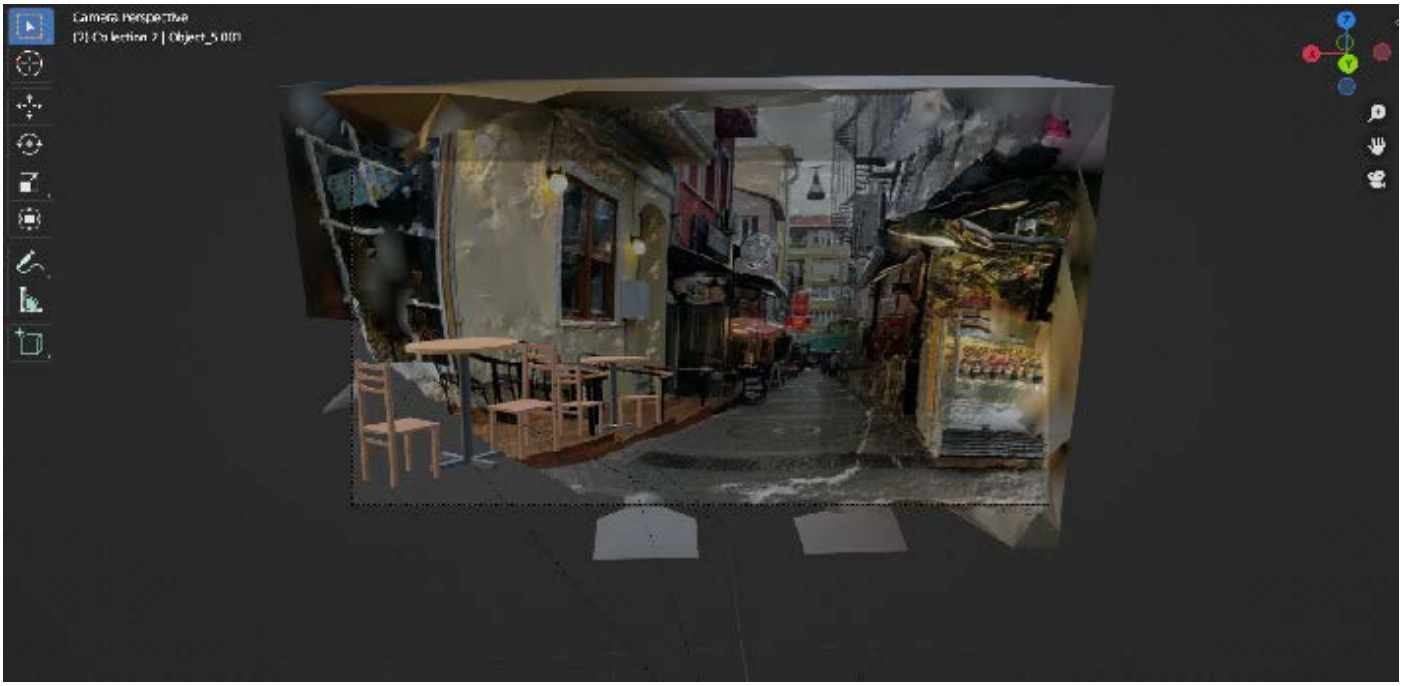
Negative Prompt: not Istanbul






A Scene from a video from Istanbul was taken and recreated through RealityCapture. Objects like chairs and tables were introduced in Blender.



Prompt: Fazıl dropped his spoon and Selva rushed to pick it up She lifted him out of his chair and took him into the kitchen while she made the coffee. The lamp atop the table is lit; next to it sits a bowl of salad and bread, all in the same basket; the table cloth is checkered What else... A plate and beans I imagine the beans, but it's not enough On the table, the same lamp is still burning Maybe a bit of yogurt Maybe a bit of life

Negative Prompt: not Istanbul

Output




https://github.com/RC11-SkillsClass2022-23/22165920/tree/main/RC11_22165920_CodeBook/Joris/StableDiffusion

Reconstruction of an image through fSpy and blender

Reference image

Rendered Image

BIBLIOGRAPY

1. Big-O Notation, Omega Notation and Big-O Notation (Asymptotic Analysis) [Internet]. [cited 2023 May 12]. Available from: https://www.programiz.com/dsa/asymptotic-notations
2. Allender E, Loui MC, Regan KW. Chapter 27 of the forthcoming CRC Handbook on Algorithms and Theory of Computation.
3. Das A. Generating Word Embeddings from Text Data using Skip-Gram Algorithm and Deep Learning in Python [Internet]. Medium. 2022 [cited 2023 May 12]. Available from: https://towardsdatascience.com/generating-word-embeddings-from-text-data-using-skip-gram-algorithm-and-deep-learning-in-python-a8873b225ab6
4. Image Classification With MobileNet | Built In [Internet]. [cited 2023 May 11]. Available from: https://builtin.com/machine-learning/mobilenet
5. Principal component analysis. In: Wikipedia [Internet]. 2023 [cited 2023 May 12]. Available from: https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=1150496893
6. Ralhan A. Self Organizing Maps [Internet]. Medium. 2018 [cited 2023 May 12]. Available from: https://medium.com/@abhinavr8/self-organizing-maps-ff5853a118d4
7. What Is Principal Component Analysis (PCA) and How It Is Used? [Internet]. Sartorius. [cited 2023 May 12]. Available from: https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186