SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT

RC11

DONGCHENDU

22174144

SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT

Contents

SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
CELL
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT

CONCEPT EXPLAINATION

Computability vs Complexity

Complexity theory is concerned with analyzing the resources required by algorithms to solve computational problems. It studies the time and **space complexity** of algorithms, and how they scale as the input size grows. This allows us to identify which problems can be solved efficiently (in **polynomial time**), and which ones are inherently difficult (requiring exponential time or more).

Computability theory, on the other hand, is concerned with understanding the fundamental limitations of computation. It studies the kinds of problems that can and cannot be solved by algorithms, and the extent to which computation can be automated. This involves analyzing the power and limitations of various models of computation, such as **Turing machines** and **lambda calculus**.

Together, complexity theory and computability theory provide a deep understanding of the nature and limitations of computation, and have many practical applications in fields such as computer science, mathematics, and engineering. Taking the code that compute the factorial of a non-negative integer n. The factorial of n is defined as the product of all positive integers less than or equal to n.

For example, the factorial of 5 (written as 5!) is 5 x 4 x 3 x 2 x 1 = 120. the code in python can be

```
def factorial(n):
    result = 1
    for i in range (1,n+1):
        result *=i
    return result
```

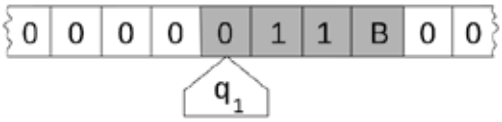
From a **COMPUTABILITY** perspective, the problem of computing the factorial of n is computable, since it can be expressed as a simple mathematical function that can be evaluated using standard arithmetic operations.

From a **COMPLEXITY** perspective, computing the factorial of n can be done using a simple loop that iterates from 1 to n, multiplying each number along the way. The time complexity of this algorithm is O(n)

Complexity and Computability are two way of thinking about algorithm, In my personal opinion computability emraces the limit of what algorithm can do, an answer to this question is also answering what algorithm is capable doing. it is a questions toward that answering the exsisting meaning for algorithm and logic. the Complexity is based on the fact that the problem itself is computable, and it is a comparison or relationship between different answers for the same issue.it is possible to lookback to the history of compurter by thinking about complexity and computability.

TURNING MARCHINE

Looing back to the history, Turning is definetely one big figures that contribute himself into the aspect of computability. A T**uring machine** is a theoretical device that can perform any computation that can be described by an algorithm. It consists of an infinite tape divided into cells, each of which can hold a symbol from a finite alphabet, a head that can read and write symbols on the tape and move left or right, and a finite set of states and rules that determine how the machine behaves based on the current state and the symbol under the head. The idea of using an finite aphabet for infinite tape is a significant example of thinking about what logic and algorithm can do without thinking about the efficiency.



Turning machine are useful for assessing computability as it can be used to model any computable function, meaning any function that can be calculated by a finite sequence of steps. However,There are some problems that cannot be solved by any algorithm. In other word, things that are not computable, One such problem is the **halting problem**, which asks whether a given Turing machine will ever stop on a given input.

Example of halting problem can be seen below:

suppose there is an algorithm H that takes a program P and an input I as inputs, and outputs either "yes" if P halts on I, or "no" if P does not halt on I. Then we can construct a program Q that takes another program R as input, and does the following:

Run H on R and R (i.e., use R as both the program and the input for H).
If H outputs "yes", then Q enters an infinite loop.
If H outputs "no", then Q halts.
Now, what happens if we run Q on itself? If Q halts on Q, then H must have outputted "no", which means Q does not halt on Q, a contradiction. If Q does not halt on Q, then H must have outputted "yes", which means Q halts on Q, another contradiction. Therefore, there is no way for H to correctly answer the question of whether Q halts on Q, and hence H cannot solve the halting problem in general.

Therefore, Turing machines are useful for studying the limits and possibilities of computation. They help us understand what kinds of problems can be solved by algorithms and what kinds cannot. They also help us compare the power and efficiency of different models of computation, such as l**ambda calculus or cellular automata**.

L**AMBDA CALCULUS**

Lambda calculus is a formal system for expressing and manipulating functions. It was invented by Alonzo Church in the 1930s as a way of studying the foundations of mathematics and logic.Lambda calculus consists of constructing lambda terms and performing reduction operations on them. A lambda term is either a variable, an abstraction, or an application. Lambda calculus can be used to model any **computable** function, meaning any function that can be calculated by a finite sequence of steps. It is equivalent in power to Turing machines, meaning that anything that can be computed by one can also be computed by the other. This result is known as the **Church-Turing thesis**.

RECURSION

Recursion is a technique of defining a function in terms of itself, or calling a function from within itself. Recursion can be used to express iterative or repetitive processes without using loops or mutable variables. it is largely affected by lamda calculas as One of the key insights of lambda calculus is that functions can be used as inputs and outputs of other functions. This means that functions can be "nested" inside other functions, which is a key feature of recursion. Below is an example of a lambda expression that takes another lambda expression as an input and applies it to a value:

```
(lambda f: f(3))(lambda x: x + 1)
```

In this example, the outer lambda expression takes an input function f and applies it to the value 3. The input function f is defined as the lambda expression lambda x: x + 1, which takes an input x and adds 1 to it.

This example illustrates the idea that functions can be passed around as values and combined with other functions. This concept of function composition is a key feature of lambda calculus and is closely related to the idea of recursion.

In fact, Church used lambda calculus to define recursive functions, which is a powerful tool in computer science. The basic idea is that a function can call itself with a smaller argument, which allows for the creation of complex algorithms that can be expressed in a simple and elegant way.

After the idea of computability developed. In the 1960s and 1970s, computer scientists began to study the efficiency of algorithms, in terms of both time and space complexity. This led to the development of the theory of computational complexity, which seeks to understand the resources (such as time and space) required to solve a problem using an algorithm.

One important result in computational complexity theory is the P vs NP problem, which asks whether all problems that can be verified in polynomial time can also be solved in polynomial time. This problem is closely related to the concept of NP-completeness, which is a measure of the complexity of a problem that is believed to be computationally intractable.

The concept of computational complexity is directly related to the Turing machine and the Church-Turing thesis, because the Turing machine provides a model for computing that can be used to analyze the time and space complexity of algorithms. In particular, the class of problems that can be solved by a Turing machine in polynomial time is known as P, and this class is widely regarded as the set of tractable problems.

SORTING ALGORITHM

Sorting algorithms are procedures to arrange a set of data in a specific order. They are used for analyzing and managing datasets. The types of sorting algorithms include insertion sort, selection sort, merge sort, quicksort, bubble sort, etc.
A simple example of sorting algorithms in python is sorting A = [4,2,6,3,5,1]. It can be presented as:

```
In: A = [4,2,6,3,5,1]
    A.sort()
    A
Out: [1, 2, 3, 4, 5, 6]
```

sorting algorithms contribute to the time **complexity** of an algorithm. The efficiency of any sorting algorithm is determined by the **time complexity** and **space complexity** of the algorithm1. Time complexity refers to the time taken by an algorithm to complete its execution with respect to the size of the input1. It can be represented in different forms such as Big-O notation (O), Omega notation (Ω), and Theta notation (θ)1. The best algorithms that make comparisons between elements usually have a complexity of O(n log n)2. This complexity means that the algorithm's run time increases slightly faster than the number of items in the vector

There are many different sorting algorithms, each with its own time complexity and trade-offs. Two common sorting algorithms are Bubble Sort and Merge Sort.

Bubble Sort has a time complexity of O(n^2), which means that its running time grows quadratically with the size of the input. Bubble Sort works by repeatedly comparing adjacent elements in the list and swapping them if they are in the wrong order. This process is repeated until the list is sorted. The main disadvantage of Bubble Sort is that it can be slow for large lists, because it compares every pair of elements in the list.

Merge Sort, on the other hand, has a time complexity of O(n log n), which means that its running time grows more slowly with the size of the input than Bubble Sort. Merge Sort works by dividing the list into smaller sublists, sorting each sublist recursively using Merge Sort, and then merging the sublists back together in the correct order. The main advantage of Merge Sort is that it is more efficient for large lists, because it only needs to compare each element once.

The time complexity of a sorting algorithm is an important factor to consider when choosing an algorithm for a specific task. If the input size is small, a slower algorithm with a lower time complexity may be sufficient. However, for large inputs, a faster algorithm with a higher time complexity may be necessary to avoid long running times.

SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
JULIAN
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT

PROJECT BRIEF 01

The goal of this project was to utilize machine learning techniques to redefine the architectural elements that comprise the city of Istanbul. Drawing upon the categories identified by Koolhaas in his book, "Elements of Architecture," I began by collecting various images of Istanbul's architectural elements using relevant keywords such as "istanbul," specific element names from Koolhaas' book, and "architecture." Subsequently, I employed unsupervised learning via Self-Organizing Maps (SOM) to explore whether this classification approach could offer new insights into how we perceive Istanbul's architectural elements.

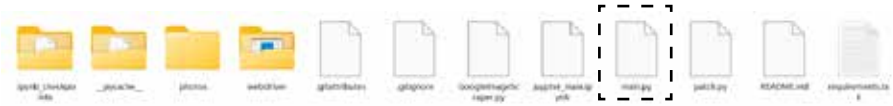
01-1 SCAPING USING GOOGLEIMAGESCRAPPER

For this experiment, I focused on collecting images of architectural elements that are easily recognizable both inside and outside buildings, including “doors”, “corridors”, “stairs”, “roofs”, “windows”, “balconies”, and “facades”.

The model I used for scraping is Google Image Scrapper https://github.com/ohyicong/Google-Image-Scraper.

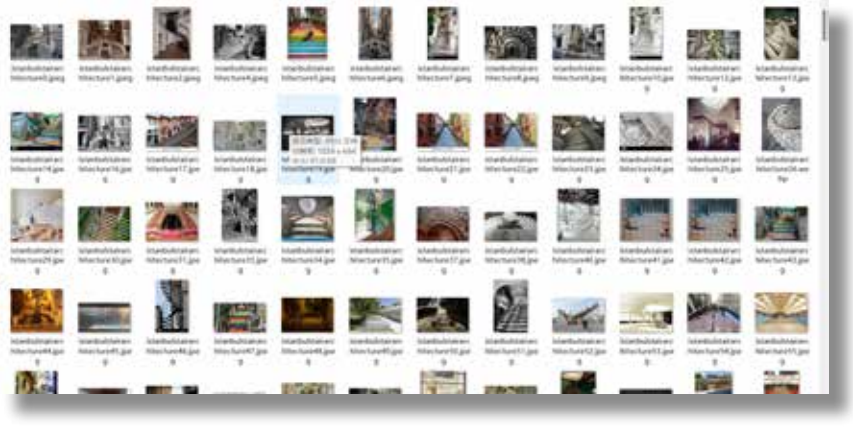
```
from GoogleImageScrapper import GoogleImageScraper
from GettyImagesScrapper import GettyImageScraper
import os
from patch import webdriver_executable
```

By adjusting the parameters in the main.py, I can change the keyword that I want to scrappe about.



```
search_keys = list(set(["istanbul door
architecture"]))

number_of_images = 1000
```

Although scape's download parameter is set to 1000, the number of images available for download is usually less than a thousand, so each folder contains 500-700 images of architectural elements from architectural elements of istanbul



01-2 VECTORIZATION OF IMAGE

Now I need to Vectorize the images and prepare them for SOM classification. Since I am going to re-establish the classification through SOM, all the images would need to be in the same folder together. There are 4727 pictures in total.

Now I need to Vectorize the images and prepare them for SOM classification. Since I am going to re-establish the classification through SOM, all the images would need to be in the same folder together. There are 4727 pictures in total.

```
len(os.listdir('momaimages'))
```

```
4247
```

The model I used for image vectorization is Mobile-net model, it is a popular convolutional neural network (CNN) architecture for deep learning, primarily used for computer vision tasks such as image classification and object detection.

```
model = tf.keras.applications.mobilenet.MobileNet()
```

Model can be inspected using Summery function

```
model.summary()
```

```
input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv1 (Conv2D)              (None, 112, 112, 32)      864
…
…
global_average_pooling2d (G  (None, 1, 1, 1024)       0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1, 1, 1024)        0

 conv_preds (Conv2D)         (None, 1, 1, 1000)        1025000

 reshape_2 (Reshape)         (None, 1000)              0

 predictions (Activation)    (None, 1000)              0
```

From the summary function, it is clear that the architecture is then shown, which consists of a series of convolutional layers, batch normalization layers, and ReLU activation layers. The model starts with an input layer, which takes a 224x224 RGB image as input.

The model experienced a pattern of depthwise convolution, batch normalization, ReLU activation, and pointwise convolution several times, with increasing numbers of filters. The final layer is a global average pooling layer, which averages the output from the previous layer across all spatial dimensions. The output of this layer is then fed to a fully connected layer with a softmax activation, which produces the final classification output.

This is the default model without removing the output layer. As for vectorization, I don't need the last fully connected layer as the feature is what I am intrested in.

The last layer of a neural network is often referred to as the output layer. It is responsible for producing the final predictions or outputs of the network. The second to last layer, on the other hand, is typically a hidden layer and is not directly responsible for producing the final outputs. Instead, it serves as an intermediate representation of the input data that is used to inform the final predictions. The Mobile-net model, is represented as 1024 dimensional vectors, by comparing those vectors, we would be able to make connections between different image features, hence helping the SOM to classify images. The last output layer of the default model can be removed through the following code.

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

After removing the last layer, image input though the model will become vectors that has 1024 dimensions. A for loop can be created to literate though 4247 images in the file in order to create 4247 of 1024 dimensional vectors.

```
features = []
for m in momaFiles:
    path = os.path.join('momaimages', m)
    f = processImage(path, model)
    features.append(f)
```

By using len(features) and len(features[0]), It is clear that the lenths of the image features list is 4247 and in each feature vector has 1024 dimensions

01-3 SOM Training

After images has been converted into feature vectors, it is possible to input the feature vectors into SOM.
The input vector is compared to the weight vectors of all the neurons in the SOM. Each neuron has a weight vector that is also 1024-dimensional and is randomly initialized at the beginning of training.

```
SOM = rand.uniform(0, 6, (m, n, 1024)).astype(float)
```

The distance between the input vector and the weight vector of each neuron is calculated. And the neuron whose weight vector is closest to the input vector is known as the Best Matching Unit (BMU). It can be identified through:

```
  distSq = (np.square(SOM - x)).sum(axis=2)
    return np.unravel_index(np.argmin(distSq, axis=None), distSq.shape)
```

Once the BMU is identified, the weights of all the neurons in the SOM are updated to move them closer to the input vector, using a learning rate that gradually decreases over time. The degree to which each neuron's weight vector is updated depends on how close it is to the BMU, with closer neurons being updated more than more distant ones.

```
def update_weights(SOM, train_ex, learn_rate, radius_sq,
                   BMU_coord, step=3):
    g, h = BMU_coord
    #if radius is close to zero then only BMU is changed
    if radius_sq < 1e-3:
        SOM[g,h,:] += learn_rate * (train_ex - SOM[g,h,:])
        return SOM
    # Change all cells in a small neighborhood of BMU
    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,:])
return SOM
```

Every epoch, 1024 dimension of 4247 images are exposed to the SOM, allowing the SOM to gradually adjust its weights to better represent the input data. once the SOM is trained, the neurons in the SOM can be interpreted as clusters of similar input vectors. Inputs that are assigned to the same neuron are considered to be part of the same cluster. In my case, the similar architecture element images would be put into the same neuron and that can be seen as a new element catagories.

After the SOM is trained, the SOM itself is still a reinterpretation of the input data, to work with original input data on top of the SOM classification, an empty array(a shadow SOM) that has the same grid dimension with the actual SOM is required.

```
SOMimages = []
for i in range(5):
    row = []
    for j in range(5):
        row.append([])
    SOMimages.append(row)
```

The shadow SOM is for the BMUS. in order to transfer the BMUS in the SOM to our shadow SOM, a Liberary need to be created for tracking purpose.By creating this, it is possible to track the features to the image names.

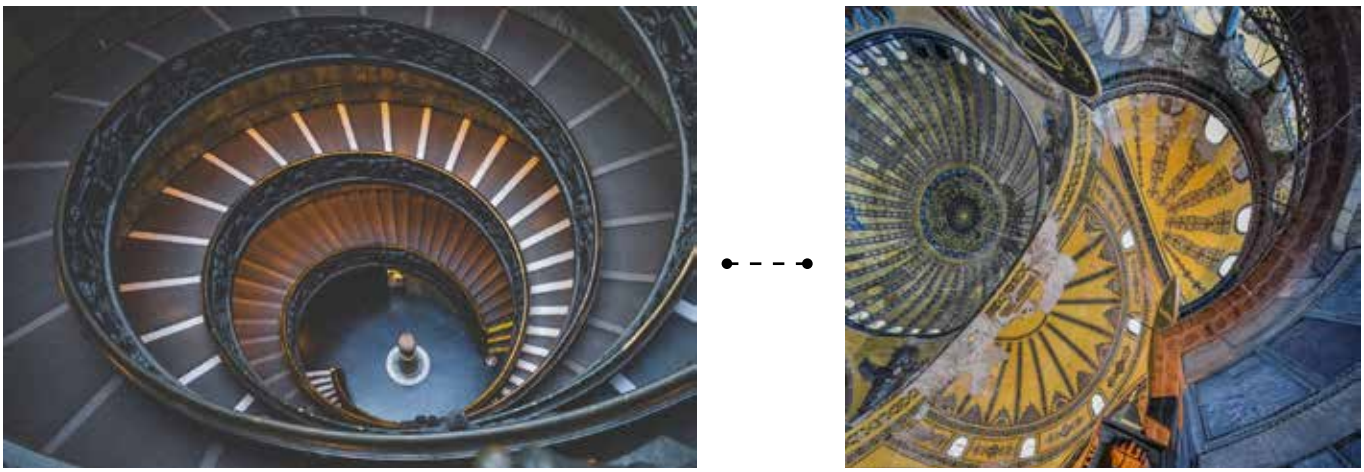
```
featureImagePairs = []
for i in range(len(features)):
    featureImage = {}
    featureImage['image'] = momaFiles[i]
    featureImage['feature'] = features[i]
featureImagePairs.append(featureImage)
```

Then by using a for loop, I can append the BMUS feature and the corresponding images name into the shadow SOM.

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

Finally, I can transfer the corresponding image from the original file to new file that name after BMUS coordinates by using the code shutil.copy

A 5x5 SOM will give out 25 categories. During the classification process, there were certain noteworthy instances. Without prior specification of categories, SOM utilized its comprehension of all input data to classify 4,247 images. The utilization of an unsupervised model presents an advantage in that it offers a form of cognition that differs from conventional notions, without imposing predetermined categories for classification purposes. Cases can be seen as following.



Through the categorization of both the stairs and the roof into a single category, we are able to transcend the individual characteristics of each element and reexamine their essence. This new perspective created by their juxtaposition generates a fresh poetic quality.

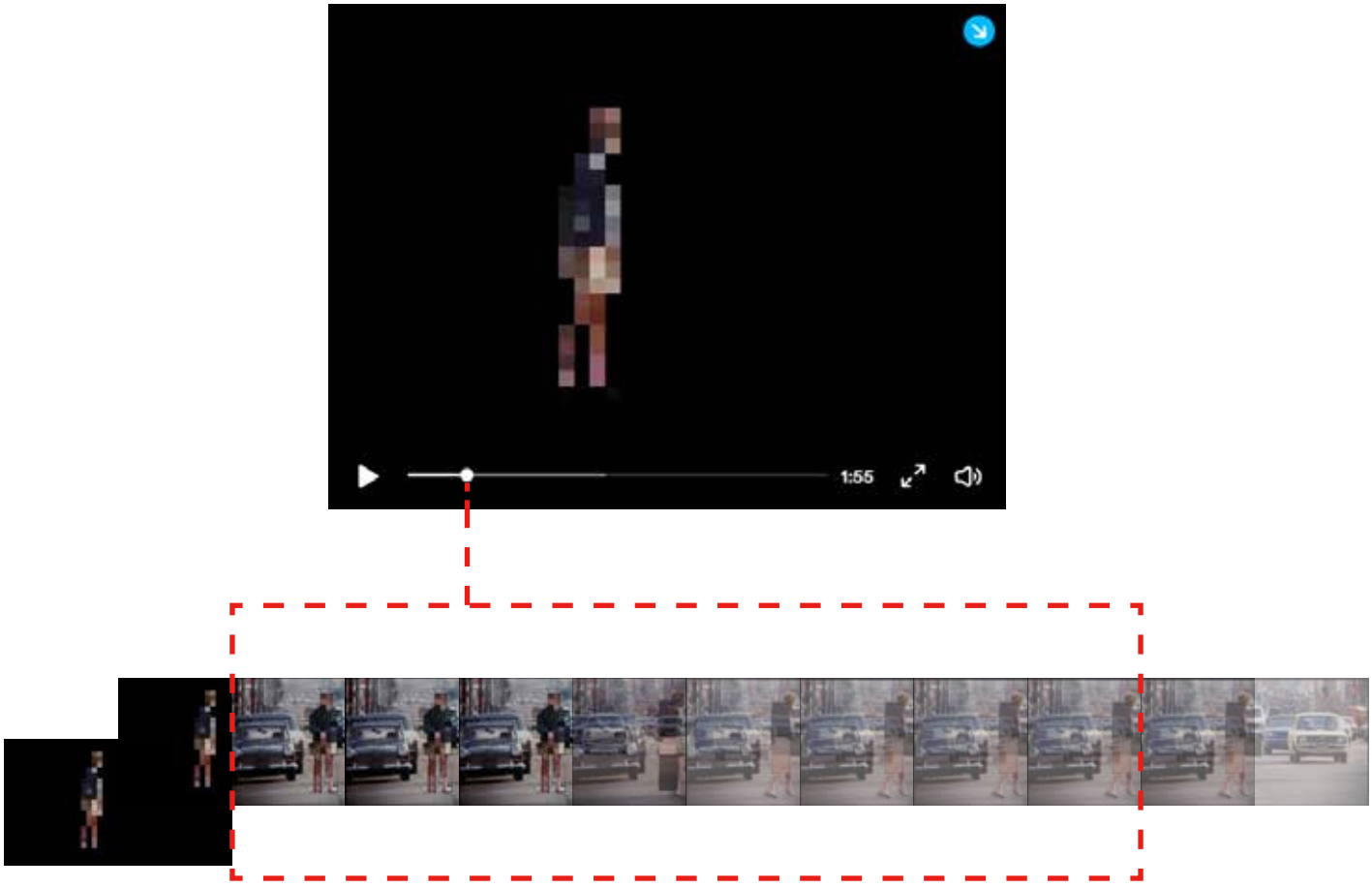
PROJECT BRIEF 02

In this project, I want to use video classification to identify activities and behaviors on istanbul street, or to provide associations of actions for activities and behaviors on the street due to the inaccuracy of the model.

On this case, a 3D-ResNet model is used, which also belongs to the family of Convolutional Neural Networks (CNNs).

```
--model /content/drive/MyDrive/RC11_2022-23/pretrainedmodel/resnet-34-kinetics.pth --model_depth 34
```

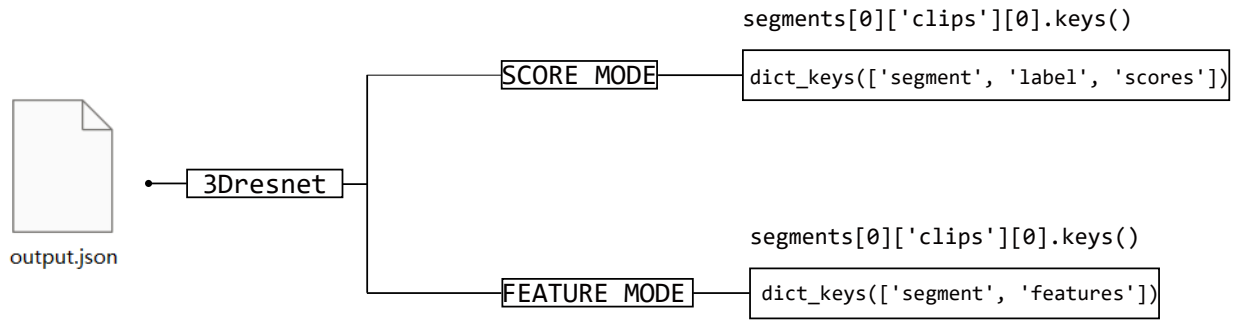
In contrast to image classification models where operates on each image, video vectorization operates on clips instead. The 3D-ResNet model takes input as every 16 frames contributing to a clip in a video.



1 Clip/16 frames

The 3Dresnet has two kinds of modes, score, and feature. For score mode, the model has the last out put layer which is trained on 400 different activities. In other words, when using score mode, the model will output a list of scores of 400 activities for every 16 frames in a video. Using following code can process the video and generate a json file.

```
! python main.py
```



Both modes will out put a json file that contains data processed by two different modes, for score mode, each clip contain segments, labels and scores. For feature, it is like using Mobile-net without last output layer, the json file contain segments and features which is a 512 dimensional feature vectors for 16 frames.

By running the following code, the json file from the score would be able to be output as a video with identified activities.

```
! python generate_result_video/generate_result_video.py
```

Input Video

Output Video

Input video link https://www.tumblr.com/mutudongdong/716882625688633344?source=share

Output video link https://www.tumblr.com/mutudongdong/716882836062404608?source=share

"play tennis"

"Juggling balls"

"Marching"

The 3Dresnet is a supervised learning model that was trained on 400 different activity categories. However, due to its inability to identify the preceding context in the video, there may be instances of inaccuracies in its predictions. Surprisingly, these inaccuracies establish unexpected connections between unrelated or seemingly random programs, with the spatial quality, character behavior, and context within the video. This amusing connection provides a fresh perspective or an additional level of imagination for interpreting the programs taking place in the video space. In the example video, I found the scene where a group of women is walking in a public space but is identified as "marching" particularly satisfying due to its dramatic quality.

PROJECT BRIEF 03

In a different project, I applied pixelization to a video in four different manners and subsequently identified three videos that were similar to each pixelized version. By placing these similar videos adjacent to the pixelized videos, the idea of information scarcity is highlighted.

The same 3Dresent model was utilized; however, to compare the direct similarity between videos, the last operation layer was removed, and the second to the last layer was retained to enable the machine to make vector comparisons.

After using feature mode to output json files, with the different feature vectors of clips, it is possible to find the best matching clips to the input clip by comparing the 512 dimensional vectors.

In order to achive this, a for loop is required to iterates through all possible starting positions in the film features list, where a sequence of the same length as input features can fit.

```
  for film in featureDictionary.keys():
            for i in range(len(filmFeatures)-len(keyFeatures)):
                distance = 0
```

For each possible starting position, the match clip can be identified through computing the Euclidean distance between the corresponding clips in keyFeatures and filmFeatures by taking the norm of the element-wise difference between them.

```
for j in range(len(keyFeatures)):
                    d = np.linalg.norm(filmFeatures[i+j]-keyFeatures[j])
                    distance += d
```

the distance is then divided by the length of the keyFeatures list to obtain the average distance between the two sequences.

```
distance = distance/len(keyFeatures)
```

At the end of each iteration of the outer for loop,current sequence would check itself that whether the filmFeatures clips is a better match than any of the previously recorded best matches. If it is, the code updates the list of best matches accordingly.

```
elif len(fragments) < nBestMatches:
                    fragments.append([film, i, len(keyFeatures), distance])
                    fragments.sort(key=lambda x: x[-1])
                elif distance < fragments[-1][-1]:
                    fragments.pop()
                    fragments.append([film, i, len(keyFeatures), distance])
                    fragments.sort(key=lambda x: x[-1])
```

This process continues until all possible sequences of film feature clips have been compared to key features. The result of the function is a list of the best matching sequences found, along with their respective distances. This is how my key feature is calculated =through my film library. And then I can use Moviepy to output the best match clips into videos.

https://www.tumblr.com/mutudongdong/703610110866489344?source=share

By using both supervised model(Mobile-net and 3DResnet) and unsupervised model(SOMs), it is clear that If I aim to uncover the global relationship of vast amounts of data or desire to explore diverse perspectives to comprehend all the data, unsupervised models are often more appropriate. On the other hand, supervised models not only benefit from their pre-trained models to classify data, but we can also remove the final operational layer to gain similarity and relationships by comparing the dimensional feature vector data in the second to the last layer. This approach is incredibly useful when I need to establish connections or associations for a singular input data.

PROJECT BRIEF 04

The objective of this project is to generate two self-organizing maps (SOMs) that embody distinct urban theories of top-down and bottom-up approaches. Each SOM is trained on 60 theoretical books from both bottom-up and top-down perspectives, creating two personas that hold different urban viewpoints. Next, articles that represent the Istanbul context are fed into both SOMs, and the resulting outputs are juxtaposed to generate tension due to the contradictions between the two schools of thought.

04-1 VECTORIZATION OF TEXT

In terms of vectorising Text, two method were used, bag of words and TF-IDF.The main difference between BoW and TF-IDF is that BoW only considers the frequency of each word in the document, while TF-IDF considers the importance of the word in the corpus as a whole. In other words, BoW gives equal weight to each word in the document, while TF-IDF gives more weight to words that are rare in the corpus and less weight to words that are common in the corpus.

BAG OF WORDS

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

Output: Dictionary(5971 unique tokens: ['abstract', 'another', 'called', 'city', 'complex']...)

```
pprint.pprint(dictionary.token2id)
```

```
Output: {'abandon': 3905,
         'abandoned': 693,
         'abandonment': 5252,
         'abercrombie': 341,
         'abilities': 2999,
         'ability': 1486,
         ...}
```

the 'words': number Represents the position number of this word in the entire Dictionary. Such as 'abandon' is the 3905th word in the Dictionary.

```
new_doc = "Cities as a progress of conflicts between top down and bottom up"
new_vec = dictionary.doc2bow(new_doc.lower().split())
print(new_vec)
```

Input: “Cities as a progress of conflicts between top down and bottom up."
Output: [(28, 1), (32, 1), (154, 1), (329, 1), (446, 1), (1621, 1), (2050, 1), (3129, 1)]

After removing some non meaning words like 'as' 'a' 'of', we convert other words into vectors. In this sample, The vector contains tuples that represent each word in the new document and its frequency in the document. For example, if the word "cities" appears once in the new document, the vector would contain a tuple like (28,1), where 28 is the index of the "cities" word in the dictionary.

In order to let SOM process our data, we shoud convert the new_vec to other form like this:
new_vec1:[(4,1), (18,1), (19,2)]
         [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0]
which means position 4 is a word that only shows once, position 19 is a words that shows twice and so on..

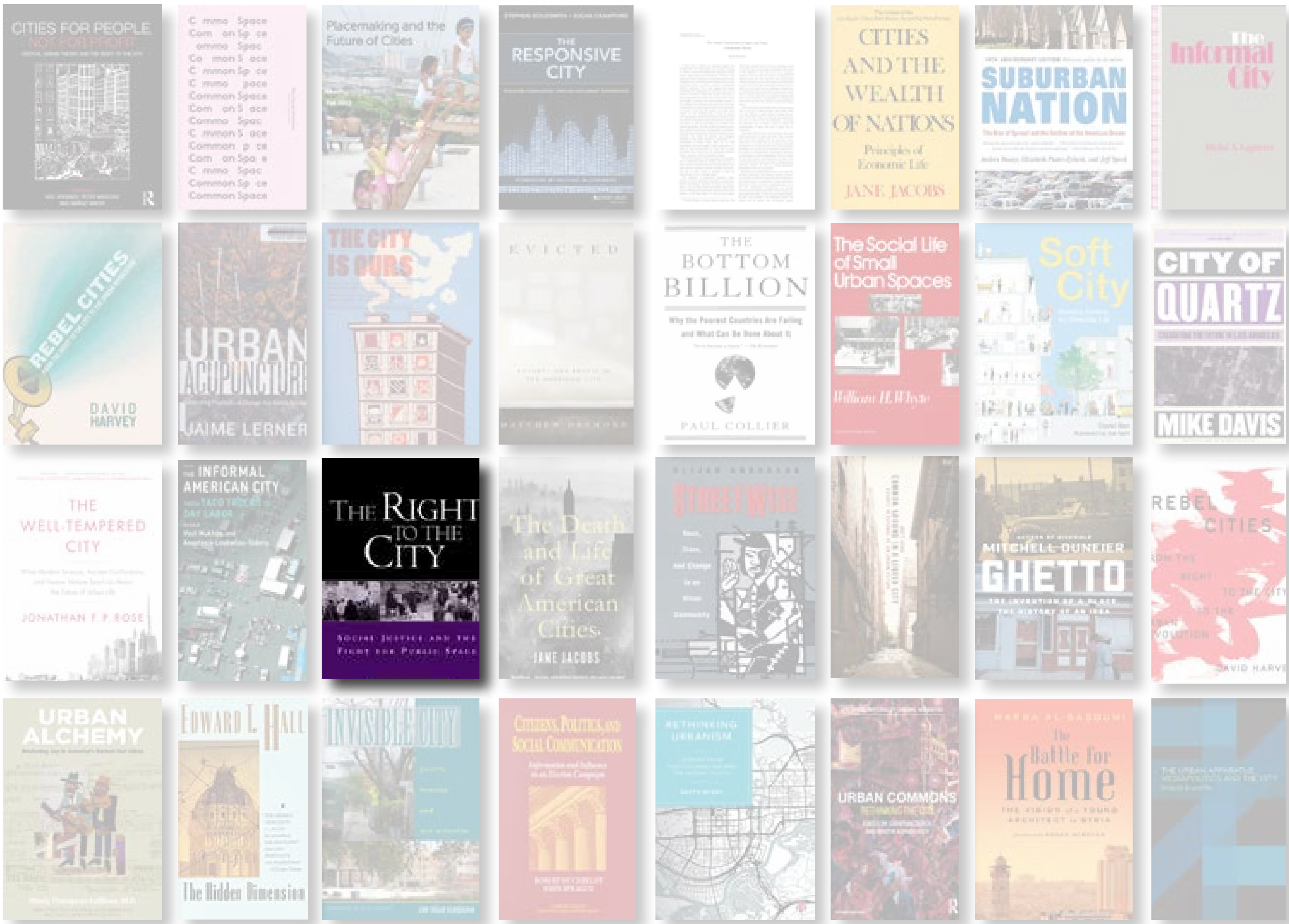
TF-IDF

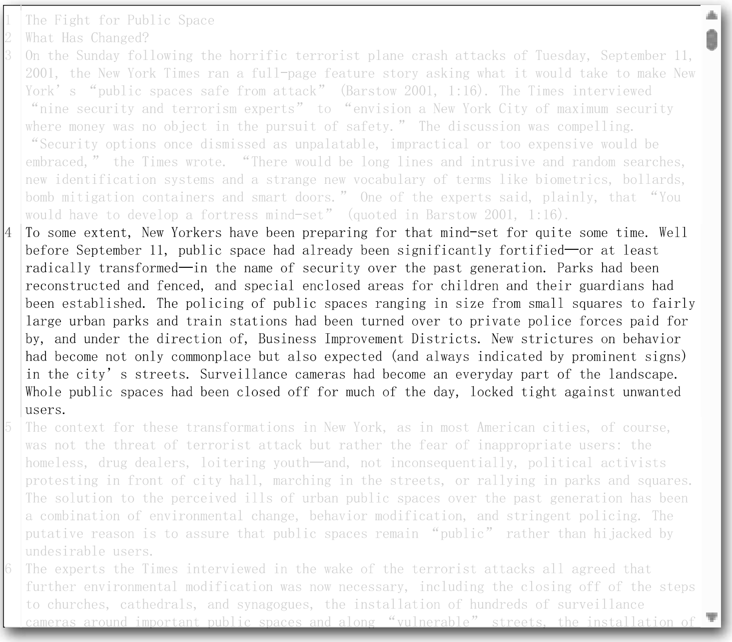
```
from gensim import models


tfidf = models.TfidfModel(bow_corpus)

feature = []
for bow_doc in bow_corpus:
    tfidf_doc = tfidf[bow_doc]
    vec = gensim.matutils.sparse2full(tfidf_doc,len(dictionary))
    vec = np.array(vec)
    feature.append(vec)

print(feature[0])
```

Output: array([0.5528216 , 0.11300873, 0.13386995, ..., 0.   , 0.   ,   0.   ], dtype=float32)

feature contains the numerical feature representation of each document in the corpus, where each feature vector has the same length as the dictionary used to create the bag-of-words corpus. This feature representation can be used for a variety of tasks, such as document classification or clustering.

Since TF-IDF the importance of the word in the corpus as a whole during vectorization, it is provide a better vectorization, hence was chosen for vectorization of our text.




1 The Fight for Public Space
2 What Has Changed?
3 On the Sunday following the horrific terrorist plane crash attacks of Tuesday, September 11, 2001, the New York Times ran a full-page feature story asking what it would take to make New York's "public spaces safe from attack" (Barstow 2001, 1:16). The Times interviewed "nine security and terrorism experts" to "envision a New York City of maximum security where money was no object in the pursuit of safety." The discussion was compelling. "Security options once dismissed as unpalatable, impractical or too expensive would be embraced," the Times wrote. "There would be long lines and intrusive and random searches, new identification systems and a strange new vocabulary of terms like biometrics, bollards, bomb mitigation containers and smart doors." One of the experts said, plainly, that "You would have to develop a fortress mind-set" (quoted in Barstow 2001, 1:16).
4 To some extent, New Yorkers have been preparing for that mind-set for quite some time. Well before September 11, public space had already been significantly fortified—or at least radically transformed—in the name of security over the past generation. Parks had been reconstructed and fenced, and special enclosed areas for children and their guardians had been established. The policing of public spaces ranging in size from small squares to fairly large urban parks and train stations had been turned over to private police forces paid for by, and under the direction of, Business Improvement Districts. New strictures on behavior had become not only commonplace but also expected (and always indicated by prominent signs) in the city's streets. Surveillance cameras had become an everyday part of the landscape. Whole public spaces had been closed off for much of the day, locked tight against unwanted users.
5 The context for these transformations in New York, as in most American cities, of course, was not the threat of terrorist attack but rather the fear of inappropriate users: the homeless, drug dealers, loitering youth—and, not inconsequentially, political activists protesting in front of city hall, marching in the streets, or rallying in parks and squares. The solution to the perceived ills of urban public spaces over the past generation has been a combination of environmental change, behavior modification, and stringent policing. The putative reason is to assure that public spaces remain "public" rather than hijacked by undesirable users.
6 The experts the Times interviewed in the wake of the terrorist attacks all agreed that further environmental modification was now necessary, including the closing off of the steps to churches, cathedrals, and synagogues, the installation of hundreds of surveillance cameras around important public spaces and along "vulnerable" streets, the installation of

$$\text{TD-IDF}(t,d,D)_d = \underbrace{\frac{f_{t,d}}{\sum_{t'\in d} f_{t',d}}}_{\text{Term Frequency}} \cdot \log \underbrace{\frac{|D|}{|\{d\in D: t\in d\}| + 1}}_{\text{Inv. Document Frequency}}$$

TF-IDF takes into account both the frequency of a word in a document (term frequency) and the frequency of the word in the entire collection of documents (inverse document frequency).

01 original texts

"...Parks had been reconstructed and fenced, and special enclosed areas for children and their guardians had been established. The policing of public spaces ranging in size from small squares to fairly large urban parks and train stations had been turned over to private police forces paid for by, and under the direction of, Business Improvement Districts..."

02 extract words

[Parks], [reconstructed], [fenced], [special], [enclosed], [areas], [children], [guardians], [established], [policing], [public], [spaces], [ranging], [size], [small], [squares], [fairly], [large], [urban], [train], [stations], [turned], [private], [police], [forces], [paid], [direction], [Business], [Improvement], [Districts]

03 covert to vectors

[(75, 1), (76, 1), (161, 1), (170, 1), (195, 1), (339, 2), (514, 1), (624, 1), (676, 1), (682, 1), (863, 1), (957, 1), (1630, 1), (1725, 1), (1802, 1), (2362, 1), (2483, 1), (2498, 1), (2717, 1), (2828, 1), (3040, 1), (3055, 1), (3564, 1), (3778, 1), (4983, 1), (5010, 1), (5043, 1)]

04 bag of words

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]

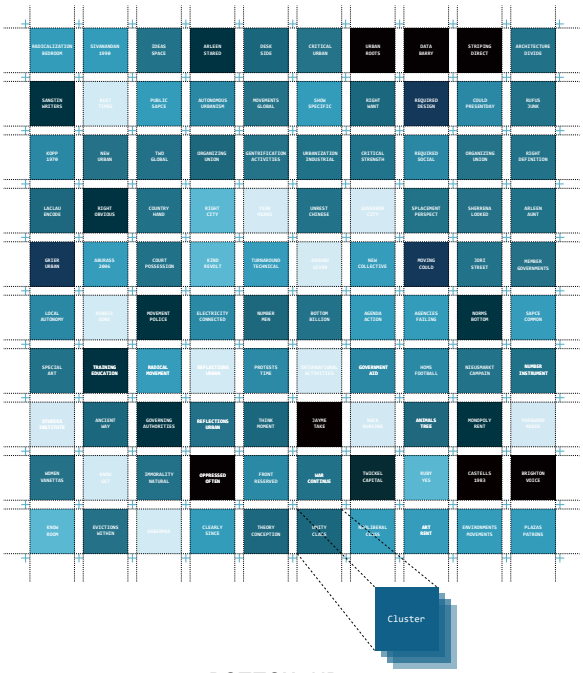
05 TF-IDF

array([0.01856529, 0.13930303, 0., ..., 0., 0., 0.], dtype=float32)

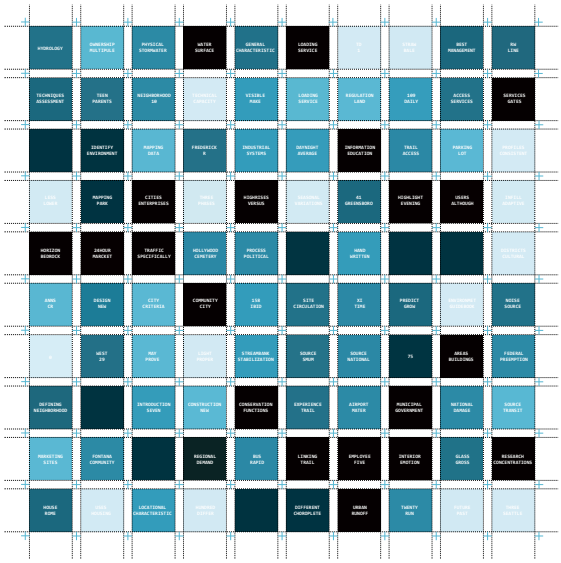
04-2 TRANIN SOM

Since the progress of training SOMs has been explained in the previous section, I will not repeat it here.

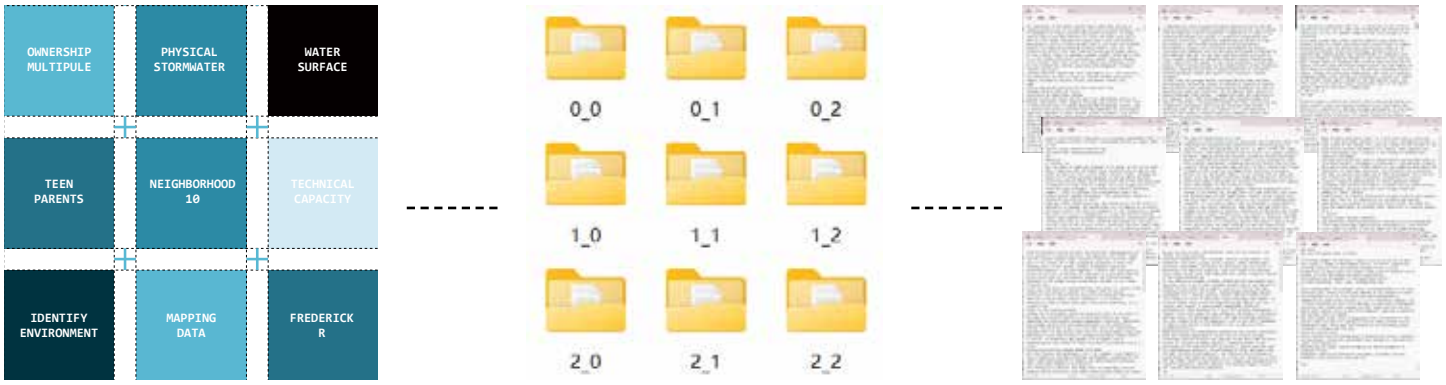
Here we input 36,836 related botoom up urban policy collections. After SOM training and visualization, we can see that these paragraphs are stored in 10*10 units after training, and each unit is these similar themes collection of paragraphs, the Key words of those paragraphs is displayed on the center of the unit.Also, here we input 40,866 related Top Down urban policy collections.



BOTTOM-UP



TOPDOWN



Units Folders Paragraphs

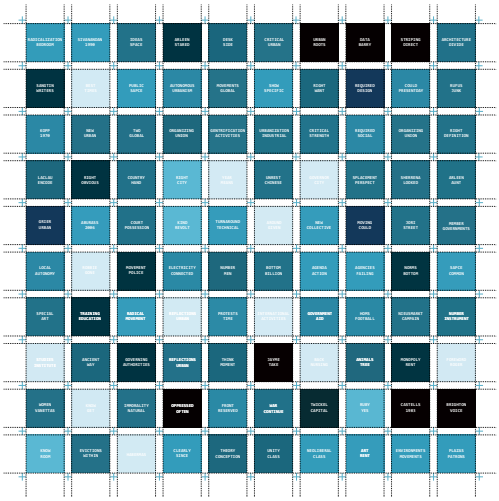
04-3 USING BMUS FROM INPUT TEXT TO CREATE TENSION

The Curated Text was input into the two SOMs to find BMU, which is then traced back to find the original theoretical text.By juxtaposing two theories derived from the same text, this creates tension from ongoing debates for Cities.I am satisfied with the result as it does give two theories along with input texts. and there is tension between two output theories.

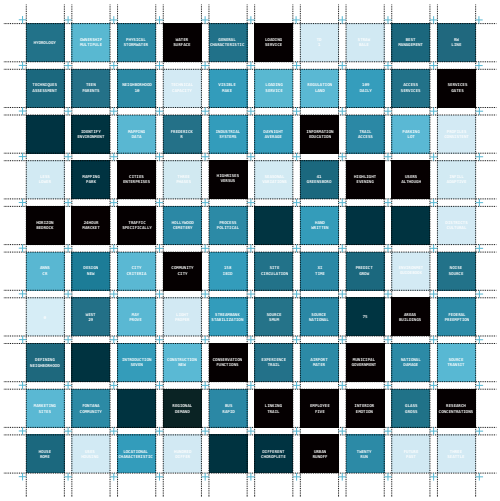
INPUT_TEXT[0]

"... Of the old Bosphorus ferries moored to deserted stations in the middle of winter, wheresleepy sailors scrub the decks, pail in hand and one eye on the black-and-white television inthe distance; of the old booksellers who lurch from one financial crisis to the next and thenwait shivering all day for a customer to appear; of the barbers who complain that men don'tshave as much after an economic crisis; of the children who play bal between the cars oncobblestoned streets; of the covered women who stand at remote bus stops clutching plasticshopping bags and speak to no one as they wait for the bus that never arrives; of the emptyboathouses of the old Bosphorus villas; of the teahouses packed to the rafters with unemployed men, of the patient pimps striding up and down the city's greatest square onsummer evenings in search of one last drunken tourist; of the broken seesaws in empty parksof one last drunken tourist; of the broken seesaws in empty parks; of ship horns boomingthrough the fog..."

BUTTOM-UP SOM



TOP-DOWN SOM



BUTTOM-UP[0]

...gray spaces have become a dominant feature of contemporary urbanism, mainly, but far from solely, in the less developed world. While the concept also covers the creation of informal spaces “from above” by powerful groups linked to the centers of power (Yiftachel, 2009), this chapter focuses on the most common expression of this phenomenon – the creation of peripheral, weakened, and marginalized spaces. Yet, communities subject to “gray spacing” are far from powerless recipients of urban policies, as they generate new mobilizations and insurgent identities, employ innovative tactics of survival, and use gray spaces as bases for self organization, negotiation, and empowerment. To be sure, power relations are heavily skewed in favor of the state, developers, or middle classes. Yet the “invisible” population of informal settlement is indeed an important actor in shaping cities and regions. portant actor in shaping cities and regions.

TOP-DOWN[0]

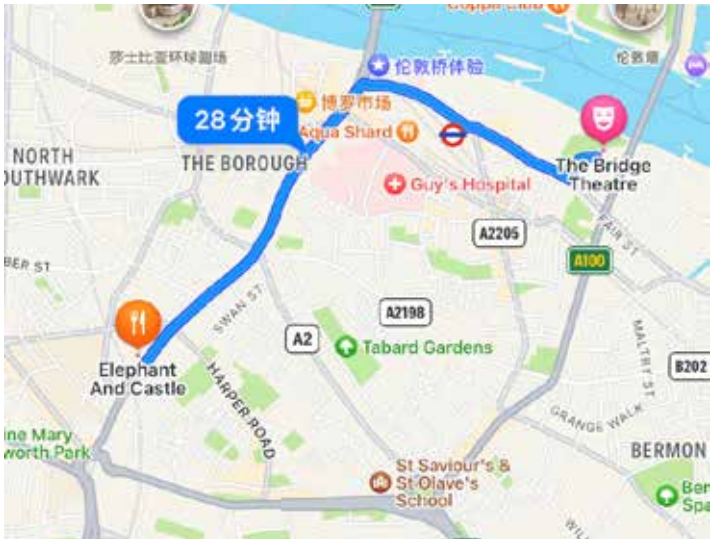
...These are linked with each other by a public transport system of higher order, for instance light rail transit (LRT). Districts in turn may form clusters around a centre of provision of yet higher order, for instance town centres linked with each other by a transport system of yet higher order, for instance the railway; and so on. In short, the micro-structure of the city is expected to be hierarchical with regard to both the development of clusters (from neighbourhood to districts, town, city, each with appropriate centres of provision) and the transport systems (from bus to LRT to railway with appropriate nodes of transport intersections at the centres of the respective spatial units). This structure and the nature of the different spatial units need to be investigated in more detail.

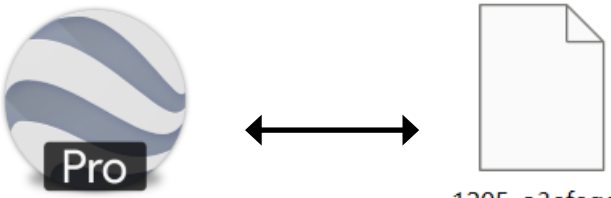
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
JORIS
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT
SKILL_REPORT

PROJECT BRIEF 01

Graffiti is a distinctive means of self-expression, and neighborhoods serve as a unique canvas for this art form. Graffiti artists frequently create multiple pieces of work as they traverse through the neighborhood, and their artwork often serves as a source of inspiration for others to create similar pieces. I undertook an exercise that involved mapping Asian graffiti throughout the London Bridge area, drawing from my personal experience of walking through various nodes of Asian art within a limited distance. Through this exercise, I aimed to recreate the cultural journey that Asian street artists encounter while creating art in the area.

01-1 CREATE PATHWAY IN BLENDER

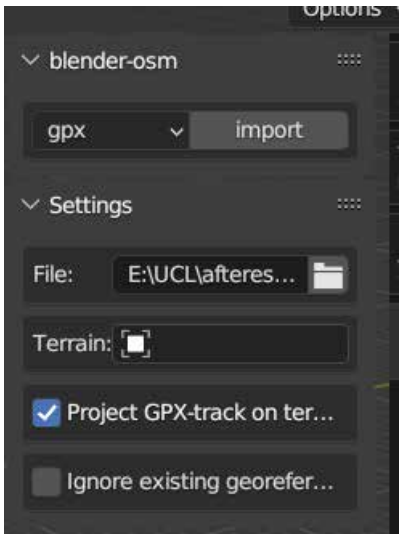


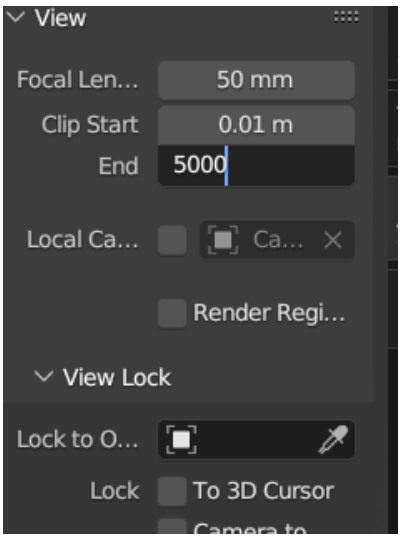


history-2023-04-19.kml

1205_o3efeqg6h_history-2023-04-19.gpx

Export the kml file using the timeline function of google map, and convert the kml file into a gpx file using the online converter function.





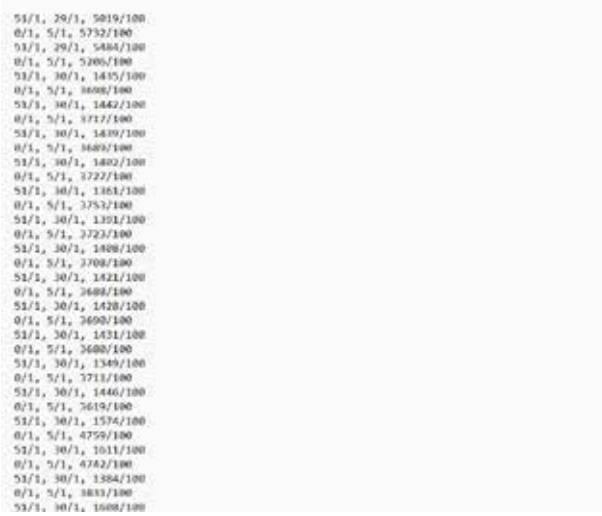
After obtaining the GPX files, they can be imported into Blender using its OSM addon feature to create a pathway. To adjust the entire pathway to fit within the viewport, I modify the Clip End slider under the "View" option.

01-2 IMPORT PATH IMAGE INTO BLENDER

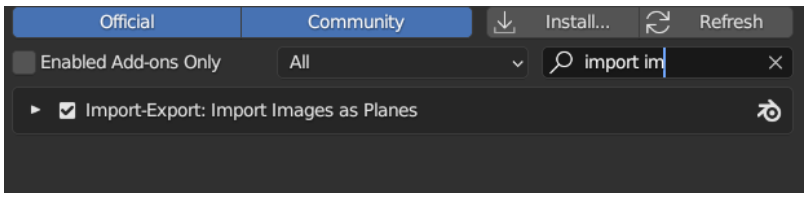
It is crucial for the imported images to have their embedded GPX information checked before they are imported into Blender. The GPX data can be easily examined by using the "get bash" function with a line of code: "magick identify -verbose+ image name".

After examination of the geographical data within images.The script list below, a text file that contain all the meta data,including geographical data for all images in the file can be created.

```
#!/bin/bash
for i in {1..27}
do
    if [ -f "IMAG$i.jpg" ];
        then
        magick IMAG$i.jpg -format "%[exif:GPSLatitude]" info: >> data.txt
        echo \ >> data.txt
        magick IMAG$i.jpg -format "%[exif:GPSLongitude]" info: >> data.txt
        echo \ >> data.txt
    fi
done
```

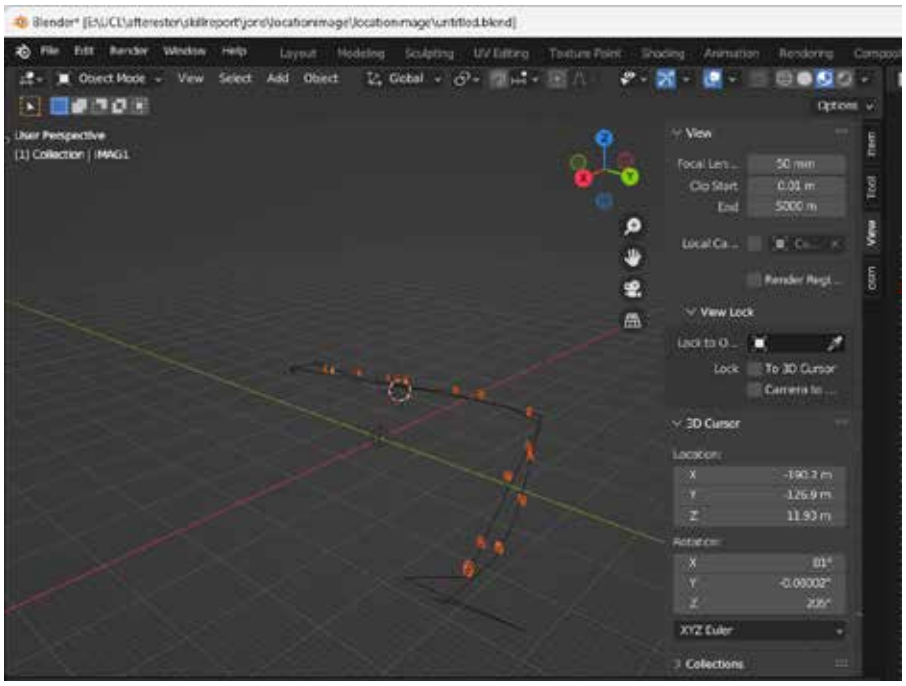


Once the geographic information of all the images has been obtained, the import image addon in Blender can be checked and another script can be used to import all the images into Blender based on their geographic coordinates.

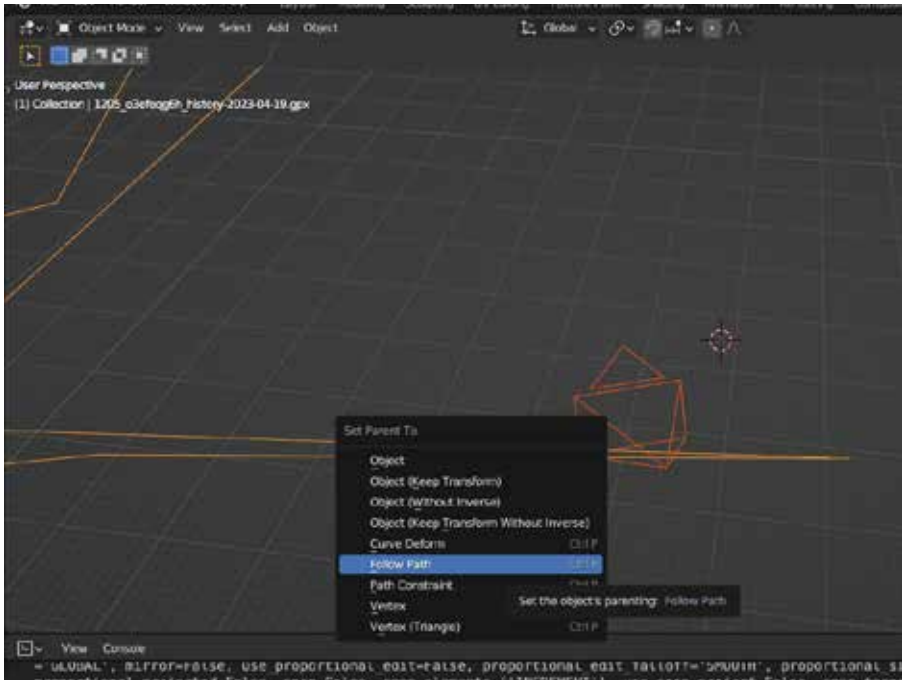


```
for i in range(1,27):
    loc = "E:\\UCL\\afterester\\skillreport\\joris\\locationimage\\locationimage\\IMAG"
    loc += str(i)
    loc += ".jpg"
    if os.path.isfile(loc) :

        # converting from the geographical coordinates to the Blender global coordinate system:
        (x, y) = projection.fromGeographic(parse_dms(f.readline()), parse_dms(f.readline()))

        #install the inmages as plane addon
        plane = bpy.ops.import_image.to_plane(shader='SHADELESS', files=[{'name':loc}])
        ob = bpy.context.object
        ob.rotation_euler[0] = math.radians(90)    rotate round x coordinate
        ob.rotation_euler[1] = math.radians(90)    rotate round y coordinate
        ob.rotation_euler[2] = math.radians(90)    rotate round z coordinate
        ob.location = ( x/100, y/100, 15 )
        ob.scale = (10,10,10)

        red = makeMaterial("Red",(1,0,0),(1,1,1),1)
        #bpy.ops.mesh.primitive_uv_sphere_add(location=(x,y,35))
        #bpy.ops.transform.translate(value=(1,0,0))
        # setMaterial(bpy.context.object, red)


f.close
```

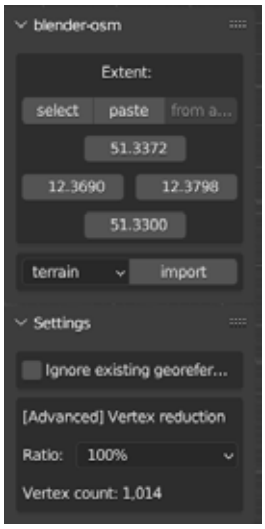
After the rotation parameters of the imported image relative to the path have been adjusted, generate.py can be run to geolocate the image.



A moving camera is necessary to create a journey. After creating a camera, it should be dragged close to the path and Blender's follow path feature can be utilized to move the camera along the entire path.

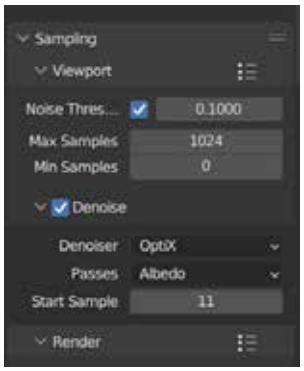


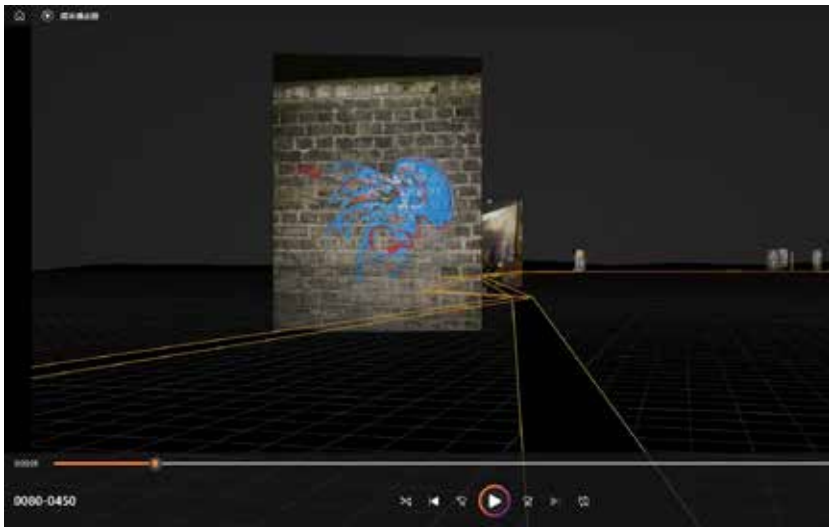
01-3 IMPORT TERRAIN INTO BLENDER



Once the path, pictures, and moving camera have been obtained, the ground can be added to the entire scene. The ground can be added by utilizing the import terrain function in the OSM addon, and the selection bottom on the top can be used to navigate into Open Street Map and copy the coordinates into the OSM addon. A region where the path is located was selected, and the coordinates was copied into the OSM addon to add the ground with the actual height difference to the entire scene.

After further adjusting the lens and overall style, the entire journey was rendered. It is worth noting that the rendering animation in viewport was chosen, which not only highlighted the path in the rendering but also maintained the display style that was selected.





Such tools are especially useful in recreating urban memories and experiences. When I was in undergraduate, I always like to use collage to express my memory of a place, and such a tool is undoubtedly a dynamic means for expressing spatial memories. If there is a way to further edit images with geographic information or add emotional processing to videos. The outcome should be very interesting.It has the potential to be a new way for urban analysis.

https://www.tumblr.com/mutudongdong/717092619709251584?source=share

PROJECT BRIEF 02

In a lot of renovation projects, fitting the context is always left to the imagination of the designer. However, capture reality plus houdini's workflow allow me to think how the context of a former relic can be overlay onto the design proposal in a renovation project. in this exercise, rather than creating a exterior 3d construction, I would like to create a 3d reconstruction of a interior for a church.

02-01 USING A FOOTAGE VIDEO FOR 3D CONSTRUCTION

To begin with, my first task is to locate a sufficiently high-quality video of the interior of the church. For this purpose, I have chosen a segment from the indoor footage captured in the video available at https://www.youtube.com/watch?v=qZmXVtH7IGM&t=1s, which was filmed on foot.

I used pytube Library for downloading youtube videos.

```
from pytube import YouTube
```

Since there are different specifications for the same video online, we can use the following code to view the downloadable specifications of the video and select the highest resolution to download

```
for video in videos:
    print(video)
```

```
<Stream: itag="17" mime_type="video/3gpp" res="144p" fps="8fps" vcodec="mp4v.20.3" acodec="mp4a.40.2" progressive="True"
type="video">
<Stream: itag="18" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.42001E" acodec="mp4a.40.2" progressive="True"
type="video">
<Stream: itag="22" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.64001F" acodec="mp4a.40.2" progressive="True"
type="video">
<Stream: itag="136" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.64001f" progressive="False" type="video">
<Stream: itag="247" mime_type="video/webm" res="720p" fps="30fps" vcodec="vp9" progressive="False" type="video">
<Stream: itag="135" mime_type="video/mp4" res="480p" fps="30fps" vcodec="avc1.4d401f" progressive="False" type="video">
<Stream: itag="244" mime_type="video/webm" res="480p" fps="30fps" vcodec="vp9" progressive="False" type="video">
<Stream: itag="134" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.4d401e" progressive="False" type="video">
```

In this case, I chose 720p for the download and used the following code as a filter to help me select the video I wanted

```
os = yt.streams.filter(res='720p',mime_type="video/mp4")
```

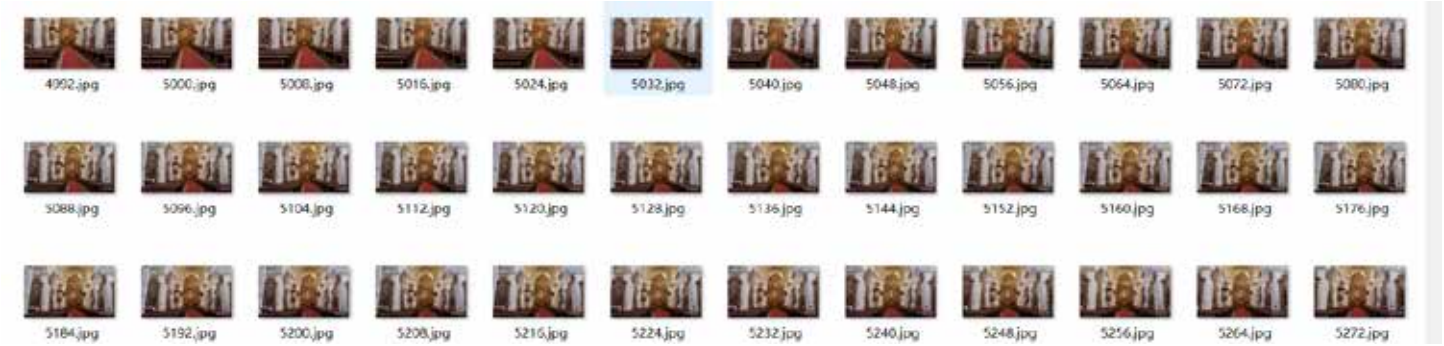
Once I have applied the filter, I can use the following code to download video:

```
video.download()
```

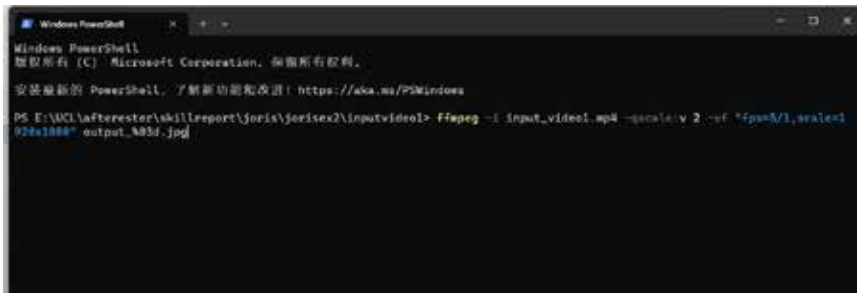
And by using cv2 library, I can also get the frame for every 10 frames along with the download video.

```
frame_no = 0
    while(cap.isOpened()):
        ret, frame = cap.read()
        if not ret:
            break
        if frame_no % 8 == 0:
            target = str(imagepath / f'{frame_no}.jpg')
            cv2.imwrite(target, frame)
        frame_no += 1
```

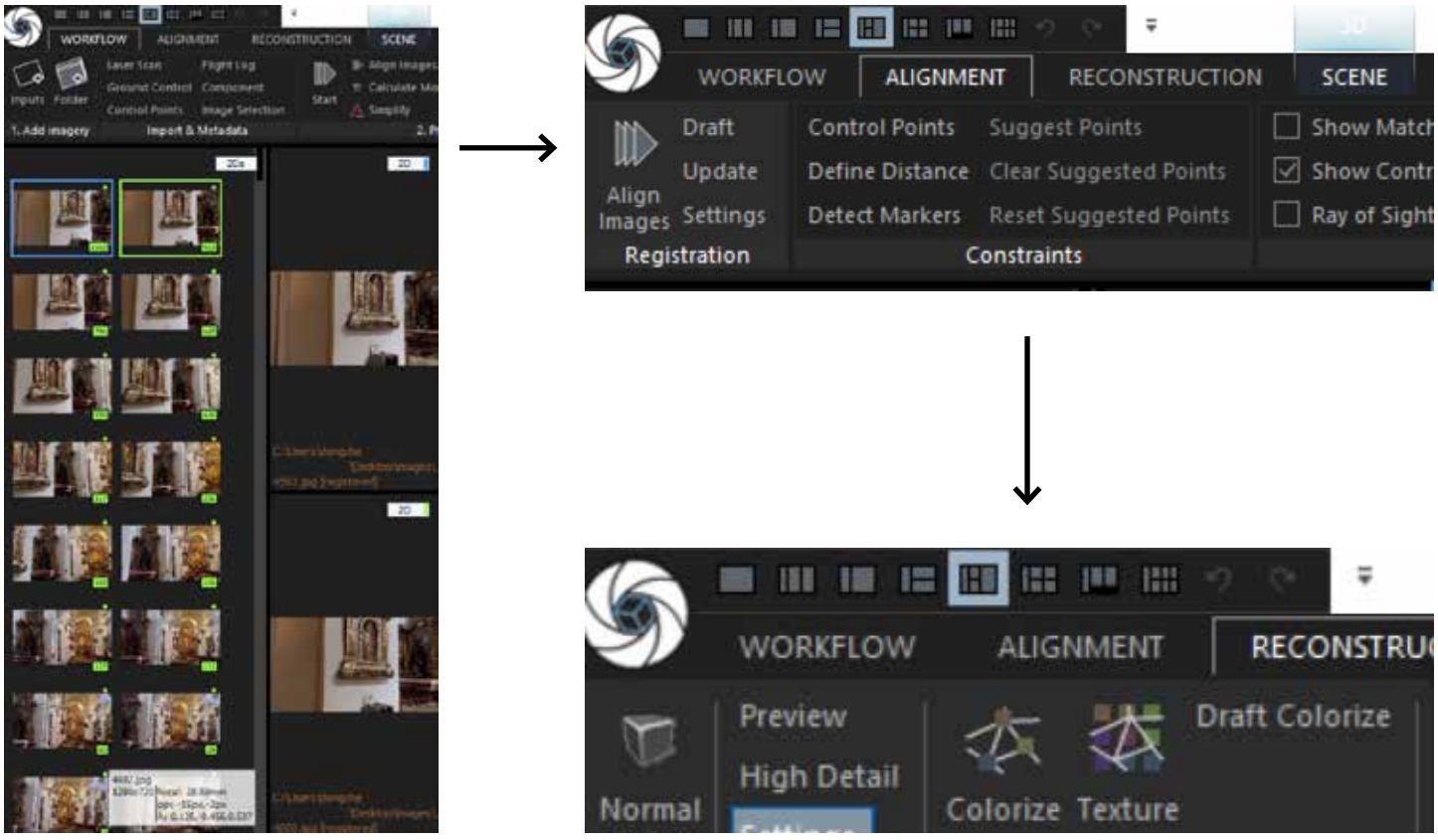
By using above code, I have scaped a lot images enough for recreating my 3d space.



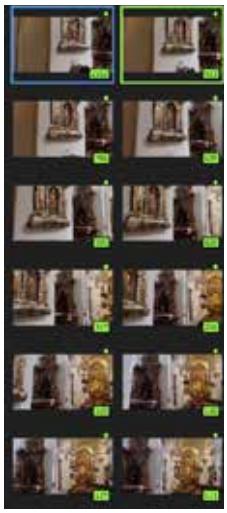
For some video, images are not able to download, I used ffmpeg when the image cannot be captured by cv2.

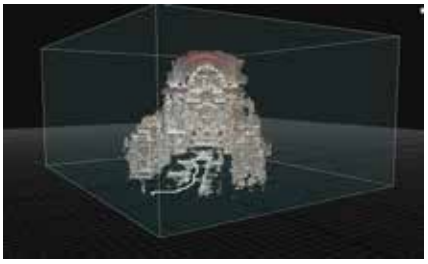


By selecting a short part of the video that satisfy my need, I then import the video into RealityCapture for further recreating the space.The RealityCapture follows a workflow of import images, align those images, reconstruct those images into space and finally give them material according to the input images.



point clouds would be generated after the images has been alligned, and volumn would be genrated after the space reconstruction. I was really satisfy with my result of reconstructing the church interior.





INPUT INTERMEDIA OUTPUT

Importing the model into Houdini enables me to produce a captivating tour of the church's interior. To achieve this, I must first manipulate the camera direction element generated in RealityCapture, such that it can be utilized as a camera path within Houdini.

```
vector up = set(0,1,0);


for(int i = 0; i<npoints(0); i++){


    if(i%8==0){
            vector posA = lerp(point(0,"P",i-6), point(0,"P",i-8),0.5);
            vector posB = lerp(point(0,"P",i-3), point(0,"P",i-1),0.5);

            int newpt = addpoint(0, posA);
            setpointattrib(0,"N",newpt,(posB-posA)*10,"set");
            setpointattrib(0,"up",newpt,up,"set");
        }
    }

for (int i=0; i<npoints(0)+1; i++){
    removepoint(0,i);
}
```

Using the code above,I can connect the center points of the camera shape objects and use the number assigned to the coner each camera corner to find the direction to create a directional path, and the camera in houdini would be able to move along the path

By adjust parameters and Rendering, I will be able to gain a walk experience in my receation of the church, and see how it can overlay with other renovation design proposal on top of it. This workflow allow me to have the ability to recreate part of the site context, and the journey produced in the end can recreate emotion when wonder around the site. More over, Learning how to use CaptureReality and Houdini give me more potential for data driven design.