

OneDrive Links

Joris homework3 frames in

Onedrive_link: <https://drive.google.com/drive/folders/1Vo1HCeUskOg529PFZuuUe81bp-rVhdP9?usp=sharing>

Julian Folderpath involved in the code

“C:\Users\xx\MAGA.pkl”

in Github

“D:\ntlive\merged_subtitles.pkl”

in Github

“D:\ntlive\1”

Onedrive_link: https://liveuclac-my.sharepoint.com/:f/g/personal/ucbvaix_ucl_ac_uk/EvUyzORcISB-Mk-1QtckBr1kBtPRD7iqov2N_gzY7X9JyZA?e=8lh1Px

“D:\sp\spanish stage play”

Onedrive_link: https://liveuclac-my.sharepoint.com/:f/g/personal/ucbvaix_ucl_ac_uk/EmkeNz9Nw9JKt-splXLXEz9wB0ZSBhLftbVRaEE03YsbFPQ?e=XgmOkt

“C:\Users\xx\Desktop\term3\1\222.png”

in Github

“E:\sp\GoogleNews-vectors-negative300.bin.gz”

Onedrive_link: https://liveuclac-my.sharepoint.com/:u/g/personal/ucbvaix_ucl_ac_uk/ERhuCQcTDpVM-sewkb_p7VXIBajY1jAsAsmRFY17EoxB98g?e=fAhBAq

“D:\sp\SpainTheatre.pkl”

Onedrive_link: https://liveuclac-my.sharepoint.com/:u/g/personal/ucbvaix_ucl_ac_uk/EaC-joY54HNNvrS-00b2e-jQBZLThL7gHITEv0cbNChehRg?e=OJUc8r

“D:\sp\matched frames”

Onedrive_link: https://liveuclac-my.sharepoint.com/:f/g/personal/ucbvaix_ucl_ac_uk/EpHsJGGd3D9Gtc-soD4MtRjsBBBbf8OvUHZ03PbrjwAJUhg?e=5d1lG5

“D:\sp\11”

Onedrive_link: https://liveuclac-my.sharepoint.com/:f/g/personal/ucbvaix_ucl_ac_uk/EjRaveCToa1Og-BmuRG7fhI4B7BcuGCU9PG86ChnoupkoiQ?e=k03R3h

Julian

Interpretation of documents

RC11

Student Number: 23086801

Path:

Input **text** - processed by Text Model - get associated **words** or similar **tweets**

Input **text** - processed by TheaterLine Model - get similar **Theater Line**

(Find the **theatre scene** corresponding to Theater)

Input **image** - processed by Theater Model - get similar **Stage Scenes**

These three searchers can become inputs and outputs for each other, either entering text to get an image or entering an image to get text

Summary Process

1 Input text - get associated words or similar tweets

Tweets Model

```
: from text_model_class import TextModel  
  
: tweets_model = TextModel(r"C:\Users\          \MAGA.pkl", min_df=3)  
  
: tweets_model.get_key_words(tweets_model.vectorize('justice'), n=10)  
['court',  
 'justice',  
 'law',  
 'supreme',  
 'judge',  
 'stand',  
 'two',  
 'bcm',  
 'mueller',  
 'voting']  
  
: tweets_model.find_similar_items('women, feminism, equality, equity, rights, advocacy, fairness, activism, inclusion', top_n=9)  
['BLACK LIVES MATTER Leader Arrested On Charges Of CHILD TRAFFICKING http://po.st/BUnu8M #TrumpsterWarriors #MAGA',  
 "In honor of Black History month Brandon is giving out crack pipes in the name of racial equity. That's what he thinks is helping black folk? It's what he thinks of black folks. Have you had enough yet? #AmericaFirst",  
 'Illegal immigrants are getting better treatment than our veterans. People who were willing to put their life on the line vs illegals who break the law We need to put America 1st & her citizens #ILLEGALimmigrants #illegals #BorderCrisis #maga #ULTRAMA GA #patriot #AmericaFirst',  
 'Life is so much brighter when we focus on what truly matters-Like put #AmericaFirst so @POTUS & We #MakeAmericaGreatAgain ht @Caligirl11111',  
 '#MAGA folks be like:',  
 '#Trump not only sucked money from his #MAGA supporters, #Trump sucked the life out of the rest of us! #IndictArrestAndConvictTrump',  
 '#MAGA Flowers loves #FREEDOM and @KWHKirkland stands for Freedom in Washington State! Come down to the Freedom Protest in #Bellevue today to see the diverse crowd here to Protest for Freedom!',  
 'I support Trump. #MAGA',  
 "If these Black Lives Matter Protesters had Jobs in a good economy they wouldn't be available to George Soros $ #Trump #MAGA"]
```

vectorization method _LSA (Latent Semantic Analysis) :

Dimensionality Reduction: LSA effectively reduces the number of features in a dataset, minimizing noise and capturing the most significant patterns, which facilitates subsequent machine learning processes.

Semantic Discovery: It reveals latent semantic structures within documents by capturing relationships among words, even if they do not co-occur in the same documents.

Improved Information Retrieval: By emphasizing semantic similarity over mere word matching, LSA enhances the quality of information retrieval, making it useful for applications like search engines and document clustering.

2 Input text - get similar Theater Line

Theater Line Model

```
from theaterline_model_class import SubtitleProcessor
merged_subtitles_path = r"D:\ntlive\merged_subtitles.pkl"
subtitle_processor = SubtitleProcessor(merged_subtitles_path)
query = '''One party believes women have equal rights. #MAGA doesn't. Vote accordingly. ...
@Joy_Villa is a #1 Billboard Artist, actress, human rights activist, a strong conservative and a proud member of the ...
@NRRA . You may have seen her at the Grammy Awards rocking a #MAGA dress. ...
Joy understands that gun rights are women's rights. #CPAC2019 @CPAC . ...
@HouseDemocrats are focused on delivering results #ForThePeople. It's time @SenateMajLdr McConnell holds votes on our bills ...
on healthcare, Rx prices, gun violence, equal rights for ALL Americans, ...
#DrainTheSwamp & much more. No federal funds for schools that do not employ an equal amount ...
of conservative instructors and administrators. #MAGA @realDonaldTrump Gun rights are civil rights. With the ...
@guncollective 's Jon Patton on #AMERICAFIRST @SalemMediaGrp'''
```

```
similar_results = subtitle_processor.find_similar(query)

for idx, (index, subtitle, score) in enumerate(similar_results):
    print(f"Rank {idx + 1}, Similarity Score: {score}")
    print(f"Index: {index}, Text: {'.join(subtitle['text'])}")
    print(f"Start Time: {subtitle['start_time']}, End Time: {subtitle['end_time']}, file_name: {subtitle['file_name']}\n")
```

```
Rank 1, Similarity Score: 0.2013974979337379
Index: 27146, Text: I wan buy gun. Gun. You go give me? - Are you stupid? - Dey kill my husband. Mutumina.
Start Time: 01:09:17,800, End Time: 01:09:27,040, file_name: Three Sisters

Rank 2, Similarity Score: 0.18698038648020998
Index: 27308, Text: Madam, just one gun. Things will be better in the morning. I promise.
Start Time: 01:42:45,200, End Time: 01:42:56,320, file_name: Three Sisters

Rank 3, Similarity Score: 0.1838359792227731
Index: 18639, Text: She is no equal for his birth you may do the part of an honest man in it. How know you he loves her? I heard him swear his affection.
Start Time: 00:29:15,670, End Time: 00:29:21,600, file_name: Much.Adio.About.Nothing.2011.01

Rank 4, Similarity Score: 0.17655006307929613
Index: 17423, Text: uh it was very exciting and and uh I think they they're so focused and they're working so hard on delivering the play
delivering the text delivering
Start Time: 00:02:33,080, End Time: 00:02:43,840, file_name: MacbethIntro

Rank 5, Similarity Score: 0.17609651237667978
```

```
similar_results
```

```
[(27146,
  {'index': 1353,
   'text': ['I wan buy gun.',
            'Gun. You go give me?',
            '- Are you stupid?',
            '- Dey kill my husband.',
            'Mutumina.'],
   'start_time': '01:09:17,800',
   'file_name': 'Three Sisters',
   'end_time': '01:09:27,040'},
  0.2013974979337379),
 (27308,
  {'index': 2001,
   'text': ['Madam,',
            'just one gun.',
            'Things will be better in the morning.'],
   'start_time': '01:42:45,200',
   'file_name': 'Three Sisters',
   'end_time': '01:42:56,320'},
  0.18698038648020998),
 (18639,
  {'index': 18639,
   'text': ['She is no equal for his birth you may do the part of an honest man in it. How know you he loves her? I heard him swear his affection.'],
   'start_time': '00:29:15,670',
   'file_name': 'Much.Adio.About.Nothing.2011.01',
   'end_time': '00:29:21,600'},
  0.1838359792227731),
 (17423,
  {'index': 17423,
   'text': ['uh it was very exciting and and uh I think they they\'re so focused and they\'re working so hard on delivering the play
delivering the text delivering'],
   'start_time': '00:02:33,080',
   'file_name': 'MacbethIntro',
   'end_time': '00:02:43,840'},
  0.17655006307929613),
 (None,
  {'text': [''],
   'start_time': '',
   'file_name': '',
   'end_time': ''},
  0.17609651237667978)]
```

vectorization method _TFIDF :

Relevance Highlighting: It emphasizes words that are frequent in a particular document but not common across all documents, helping to identify key terms that make documents unique.

Scalability: TF-IDF can be efficiently applied to large datasets of text, such as collections of subtitles.

Compatibility with Machine Learning: The numeric representation of text as TF-IDF vectors is ideal for feeding into various machine learning algorithms, particularly for tasks like similarity detection and classification.

3 (Find the theatre scene corresponding to Theater)

Theatre lines to corresponding theatre scenes

```
import os

folder_path = r"D:\ntlive\1"

if similar_results:

    matched_drama = similar_results[0][1]['file_name']
    print(matched_drama)
    mp4_name = f'{matched_drama}.mp4'

    mp4_path = os.path.join(folder_path, mp4_name)
    if os.path.exists(mp4_path):
        print(f"Found: {mp4_path}")
    else:
        print(f"No matching file found for: {mp4_name}")
```

Three Sisters
Found: D:\ntlive\1\Three Sisters.mp4

```
from moviepy.video.io.ffmpeg_tools import ffmpeg_extract_subclip
from moviepy.editor import VideoFileClip
from IPython.display import display, Video
import imageio

mp4_path = os.path.join(folder_path, mp4_name)

start_time_parts = similar_results[0][1]['start_time'].split(',')
start_seconds = int(start_time_parts[0].split(':')[1]) + 60 * int(start_time_parts[0].split(':')[1]) + 3600 * int(start_time_parts[0].split(':')[1])

end_time_parts = similar_results[0][1]['end_time'].split(',')
end_seconds = int(end_time_parts[0].split(':')[1]) + 60 * int(end_time_parts[0].split(':')[1]) + 3600 * int(end_time_parts[0].split(':')[1])

print(start_seconds)
print(end_seconds)
```

Relational interpretation:

Stage scene design is often closely related to the lines, consistent with the mood of the characters, the plot, the atmosphere, so there is a correlation between the stage scene and the lines.

4 Input image - get similar Stage Scenes

Theater Model

```
: import tensorflow as tf
import pickle
from theater_model_class import FilmProcessor

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

processor = FilmProcessor(r"D:\sp\spanish stage play", model)
processor.test('q')

WARNING:tensorflow:From C:\Users          (AppData)\Roaming\Python\Python311\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From C:\Users          (AppData)\Roaming\Python\Python311\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users          (AppData)\Roaming\Python\Python311\site-packages\keras\src\layers\normalization\batch_normalization.py:979: The name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

: 'q'

: queryFeatures = processor.processImage(r"C:\Users          (Desktop\term3\1\222.png")
queryFeatures

1/1 [=====] - 0s 353ms/step

: array([0.09042002, 1.2170194 , 0.         , ..., 0.5831064 , 0.33640075,
: matches = processor.findTopFramesByFeatures(queryFeatures, top_n=400)

from PIL import Image
for i, match in enumerate(matches):
    film_path, second, frame_path = match
    print(f'Top Match {i + 1} - Film: {film_path}, Second: {second}, Frame Image Path: {frame_path}')

    save_path = r'D:\sp\matched frames\{}'.format(os.path.basename(frame_path))
    Image.open(frame_path).save(save_path)
    print(f'Saved Matched Frame Image to: {save_path}')

Top Match 1 - Film: E:\sp\spanish stage play\TEATRO 'Llueve en Barcelona' (2009).mp4, Second: 840, Frame Image Path: Frames\TEATRO_Llueve_en_Barcelona_2009.mp4_14.jpg
Saved Matched Frame Image to: D:\sp\matched frames\TEATRO_Llueve_en_Barcelona_2009.mp4_14.jpg
Top Match 2 - Film: E:\sp\spanish stage play\Los dramáticos orígenes de las galaxias espirales (CDN, 2016).mp4, Second: 660, Frame Image Path: Frames\Los_dramáticos_orígenes_de_las_galaxias_espirales_CDN_2016.mp4_11.jpg
Saved Matched Frame Image to: D:\sp\matched frames\Los_dramáticos_orígenes_de_las_galaxias_espirales_CDN_2016.mp4_11.jpg
Top Match 3 - Film: E:\sp\spanish stage play\d.juan@simetrico.es La burladora de Sevilla y el Tenorio del siglo XXI · Jesús Campos Teatro A Teatro.mp4, Second: 3120, Frame Image Path: Frames\d.juan@simetrico.es_La_burladora_de_Sevilla_y_el_Tenorio_del_siglo_XXI_Jesús_Campos_Teatro_A_Teatro.mp4_52.jpg
Saved Matched Frame Image to: D:\sp\matched frames\d.juan@simetrico.es_La_burladora_de_Sevilla_y_el_Tenorio_del_siglo_XXI_Jesús_Campos_Teatro_A_Teatro.mp4_52.jpg
Top Match 4 - Film: E:\sp\spanish stage play\Los dramáticos orígenes de las galaxias espirales (CDN, 2016).mp4, Second: 3540, Frame Image Path: Frames\Los_dramáticos_orígenes_de_las_galaxias_espirales_CDN_2016.mp4_59.jpg
Saved Matched Frame Image to: D:\sp\matched frames\Los_dramáticos_orígenes_de_las_galaxias_espirales_CDN_2016.mp4_59.jpg
Top Match 5 - Film: E:\sp\spanish stage play\Las troyanas de Eurípides Ernesto Alteiro 2017.mp4, Second: 1860, Frame Image Path: Frames\Las_troyanas_de_Eurípides_Ernesto_Alteiro_2017.mp4_31.jpg
Saved Matched Frame Image to: D:\sp\matched frames\Las_troyanas_de_Eurípides_Ernesto_Alteiro_2017.mp4_31.jpg
Top Match 6 - Film: E:\sp\spanish stage play\TEATRO 'Llueve en Barcelona' (2009).mp4, Second: 2820, Frame Image Path: Frames\TEATRO_Llueve_en_Barcelona_2009.mp4_47.jpg
```

vectorization method _MobileNet :

High-Level Feature Extraction: Leveraging a pre-trained deep learning model allows for the extraction of complex, high-level features from images, which are more effective for tasks such as classification, recognition, and similarity assessment compared to basic pixel data.

Efficiency in Processing: The use of TensorFlow and MobileNet provides an optimized, efficient way to process and analyze large volumes of visual data, making it scalable for extensive video datasets.

Accuracy in Similarity and Retrieval: The vectorized features facilitate accurate similarity measurements and retrieval tasks through methods like Euclidean distance, enabling precise matching and ranking of video frames based on visual content.

vectorization method comparing :

TF-IDF (Term Frequency-Inverse Document Frequency):

TF-IDF emphasizes words that are important to a specific document but not common across all documents. This method is straightforward and scales well to large datasets, making it useful for tasks where identifying unique or significant words is crucial. However, it doesn't account for the context or order of words.

Word2Vec:

This approach learns to represent words in a continuous vector space based on their context within sentences, capturing both the semantic and syntactic essence of words. Word2Vec requires a large amount of text to perform well and focuses on word-level patterns, making it less effective for getting direct insights at the document level.

Detailed process (including SOM)

1 Input text - get associated words or similar tweets

1.1 Functions

Load functions

```
def preprocess(paragraph):
    stop_words = set(stopwords.words("english"))
    lemmatizer = WordNetLemmatizer()

    words = gensim.utils.simple_preprocess(paragraph, min_len=3, deacc=True)
    processed_words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    processed_doc = " ".join(processed_words)
    return processed_doc

def load_list_from_file(file_name):
    try:
        with open(file_name, 'rb') as file:
            lst = pickle.load(file)
        return lst
    except Exception as e:
        print(f"An error occurred: {e}")
        return None
```

The Load function is responsible for loading the required files;

SOM functions

```
def calculateQE(SOM, data):
    sumSqDist = 0
    for d in data:
        g, h = find_BMU(SOM, d)
        v1 = SOM[g, h]
        v2 = d
        sumSqDist += scipy.spatial.distance.cdist([v1], [v2], 'cosine')[0][0]
    QE = sumSqDist/len(data)
    return QE

def find_BMU_2(SOM, x):
    somShape = SOM.shape
    simSOM = SOM.reshape((-1, len(x)))
    cos_sims = cosine_similarity([x], simSOM).reshape(somShape[:2])
    return np.unravel_index(np.argmax(cos_sims, -2, axis=None)[-2], cos_sims.shape)
```

The SOM function is responsible for the architecture of the subsequent SOM;

Visualization functions

```
def visualize_epochs_som(som_versions, epochs, file_name_base):
    """ Visualizes SOMs at each epoch directly in Jupyter Notebook without saving to files. """
    for epoch, som in enumerate(som_versions):
        plt.figure(figsize=(8, 8))
        u_matrix_values = u_matrix(som) # Ensure that 'u_matrix' is called with the correct SOM state
        plt.imshow(u_matrix_values, cmap=custom_cmap, aspect='equal')
        plt.title(f'U-Matrix at Epoch {epoch + 1}')
        plt.colorbar(label='Distance')
        plt.xticks([])
        plt.yticks([])

        # Show the plot in Jupyter Notebook
        plt.show()

def visualize_som_with_words(SOM, pca_u_matrix, words_list, u_matrix_values, file_name, dpi=300):
    neuron_word_mapping = {}
    for i in range(SOM.shape[0]):
        for j in range(SOM.shape[1]):
            neuron_word_mapping[(i, j)] = []
```

The visualisation function is responsible for the subsequent visualisation of the SOM;

Search functions

```
def preprocess_text(text):
    """ Tokenizes, removes stopwords, and lemmatizes the input text. """
    tokens = word_tokenize(text.lower())
    tokens = [WordNetLemmatizer().lemmatize(token) for token in tokens if token.isalpha() and token not in stopwords.words('en')]
    return " ".join(tokens)

def vectorize_and_transform_text(input_text, word2vec_model, pca):
    # Tokenize and vectorize the input text using Word2Vec
    words = input_text.split() # Simple split; consider more complex preprocessing
    word_vectors = [word2vec_model[word] for word in words if word in word2vec_model]

    # If no words found in the model, handle this case
    if not word_vectors:
        return np.zeros(pca.n_components_)

    # Average the vectors (or another method, like sum)
    average_vector = np.mean(word_vectors, axis=0).reshape(1, -1)

    # Apply PCA transformation
    transformed_vector = pca.transform(average_vector)

    return transformed_vector

def activate_and_visualize_SOM(SOM, input_vector, textual_content_mapping, title="SOM Activation"):
    # Ensure the input vector is properly shaped and normalized
    norm_input_vector = input_vector / np.linalg.norm(input_vector) if np.linalg.norm(input_vector) > 0 else input_vector

    # Reshape and normalize the SOM for cosine similarity calculation
```

The search function is responsible for entering words to find BMUs and texts near BMUs from the SOM;

1.2 Load

Load and prepare

```
# Load data
MAGA = load_list_from_file(r"C:\Users\...\MAGA.pkl")
texts = [preprocess(text) for text in MAGA]
joined_text = " ".join(texts)

# Load Word2Vec model
model_path = r"E:\sp\GoogleNews-vectors-negative300.bin.gz"
model = KeyedVectors.load_word2vec_format(model_path, binary=True)

# Create TF-IDF model
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(texts)
feature_names = vectorizer.get_feature_names_out()

# Identify the top 1000 high-frequency words
high_frequency_1000_index = tfidf_matrix.toarray()[0].argsort()[-1000:]
words_list = [feature_names[i] for i in high_frequency_1000_index]

# Generate word vectors for these high-frequency words
word_vectors = [model[word] for word in words_list if word in model]
word_vectors_matrix = np.array(word_vectors)

# PCA to reduce dimensionality
pca = PCA(n_components=4)
pca_u_matrix = pca.fit_transform(word_vectors_matrix)
pca_u_matrix = normalize(pca_u_matrix, norm='l2')
```

This step loads the file and vectorises it.

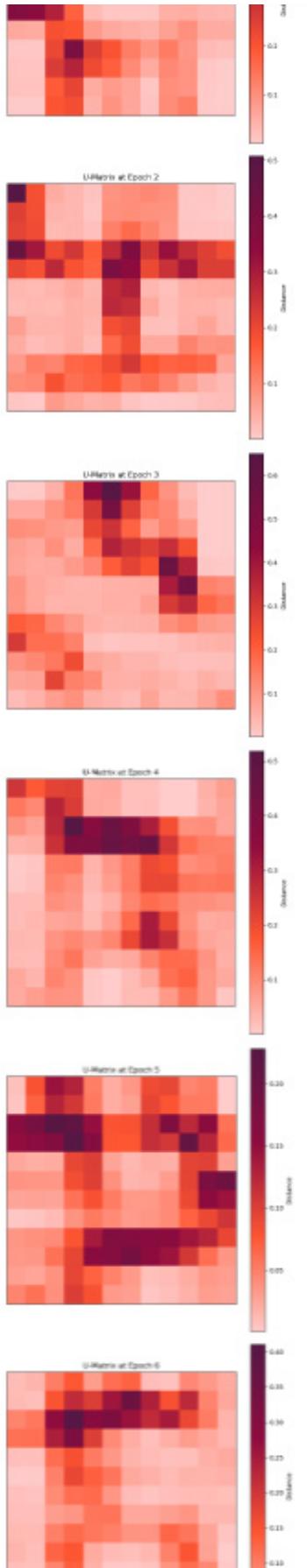
```

learn_rate = 0.5
radius = 5
epochs = 10
SOM = np.random.rand(12, 12, 4)
train_data = pca_u_matrix

best_som, best_errors, som_versions, neuron_texts = train_SOM(SOM, train_data, learn_rate, radius**2, 0.1, 0.1, epochs, texts)

visualize_epochs_som(som_versions, epochs, 'SOM_Visualization')

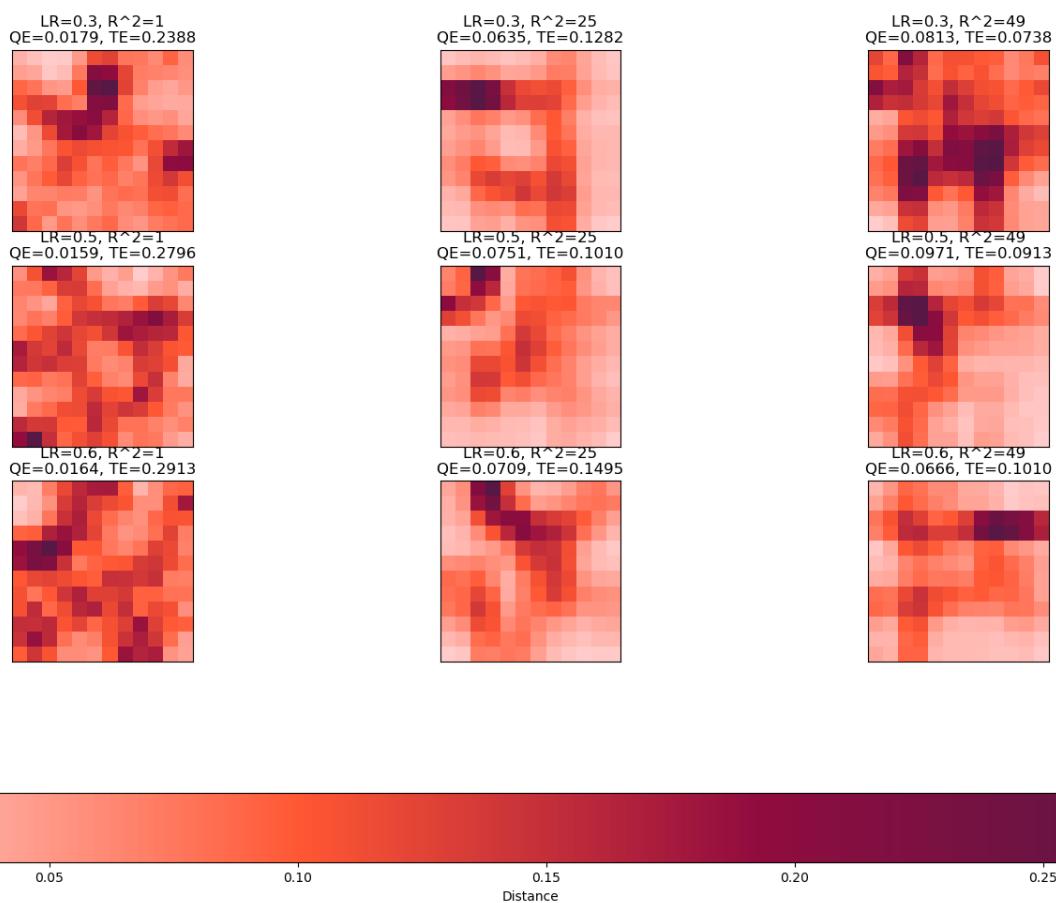
```



1.3 Visualize for different epochs

This step shows the SOM corresponding to different epochs

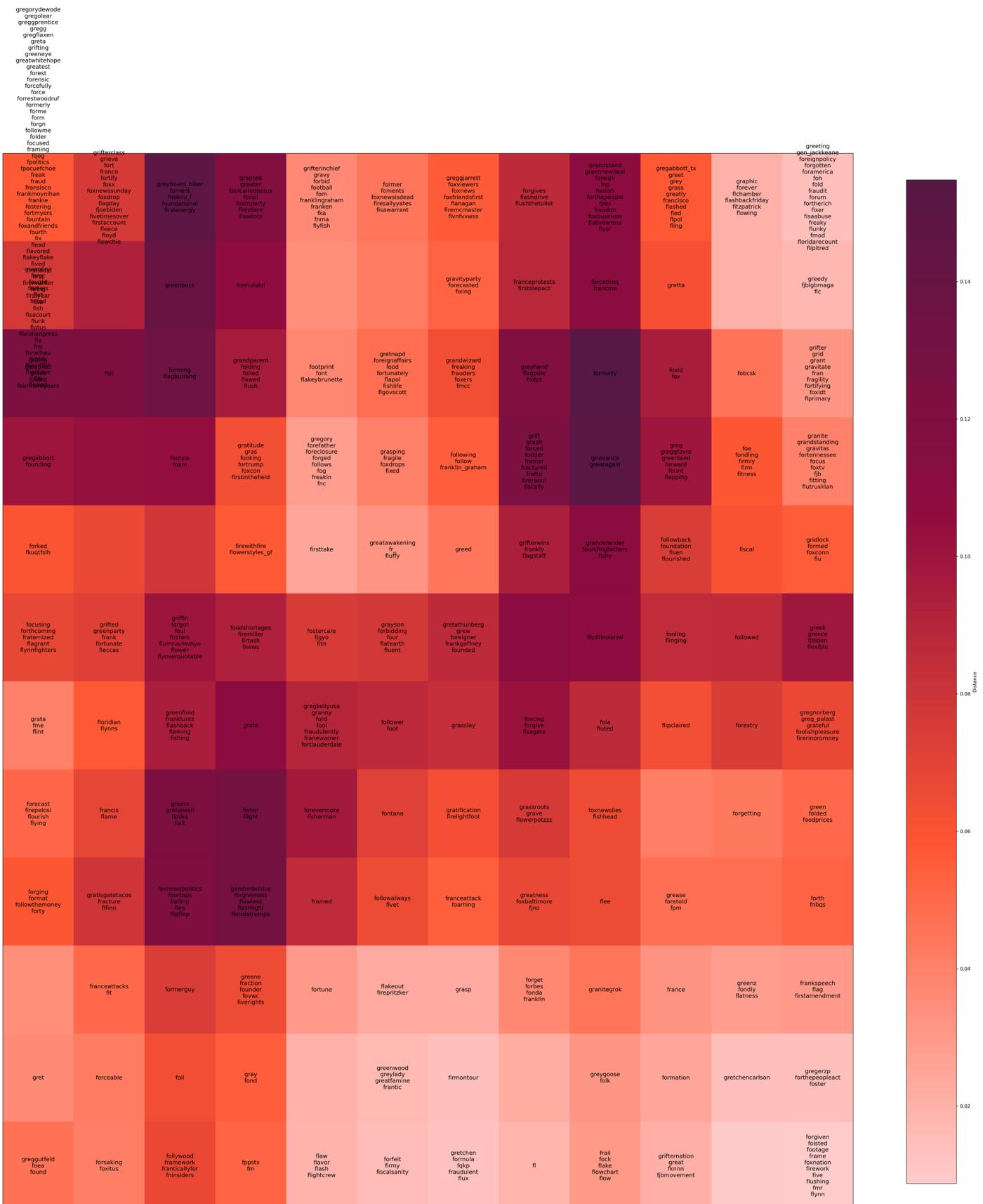
```
: best_overall_som, best_overall_error = run_and_visualize_som_training(pca_u_matrix, [0.3, 0.5, 0.6], [1, 5, 7], 12, 12, 4, 20, 1)
```



This step shows the SOMs corresponding to different learning rates and radii, since the learning rate and radius need to be smaller at the same time, I set the SOM with the smallest sum of learning rate and radius to be `best_overall_SOM`.

1.3 Visualize for different lr and r2

```
u_matrix_values = u_matrix(best_overall_som)
visualize_som_with_words(best_overall_som, pca_u_matrix, words_list, u_matrix_values, 'SOM_Visualization.png')
```



This step shows the text attached to the SOM when it is visualised.

1.3 Visualize with words

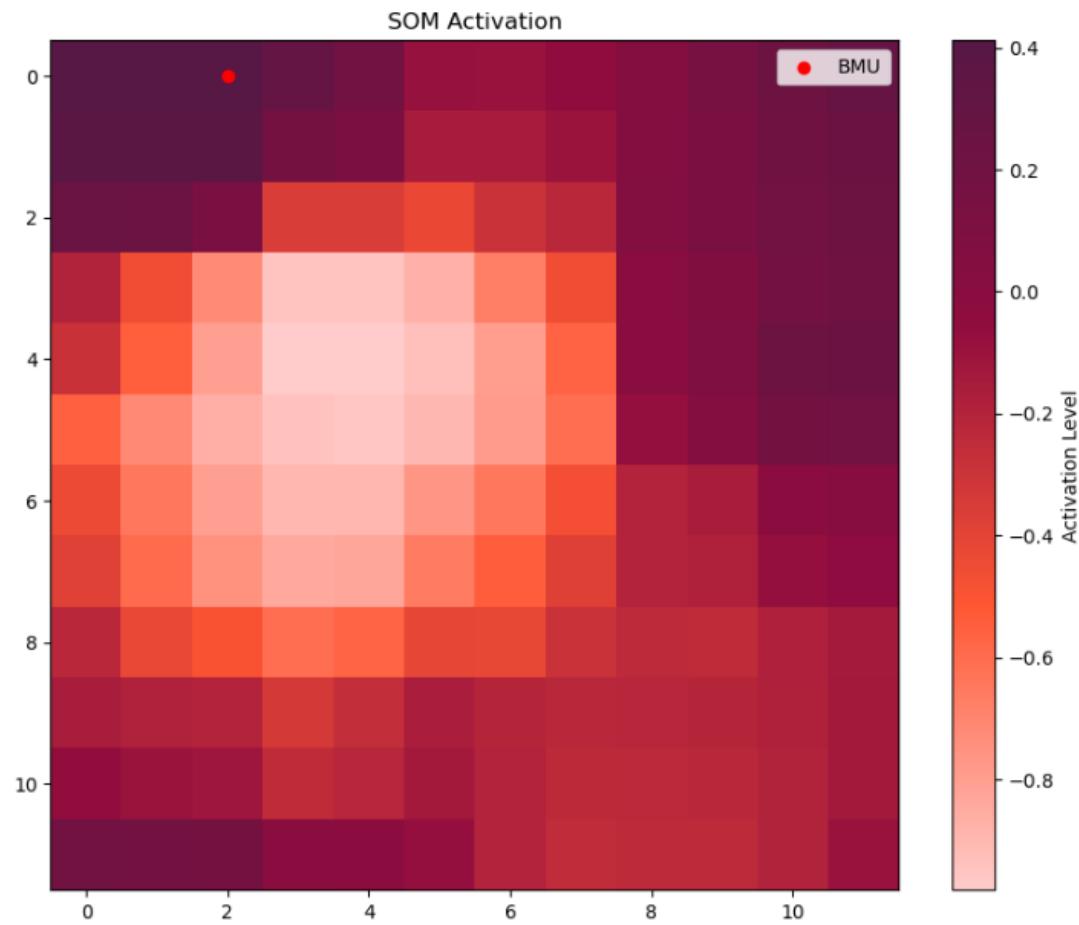
Search

```
# Train your SOM and receive the mapping of texts to each neuron
best_som, best_errors, som_versions, neuron_texts = train_SOM(SOM, train_data, learn_rate, radius**2, 0.1, 0.1, epochs, texts)

# Create textual content mapping from neuron_texts
textual_content_mapping = {}
for (i, j), texts in neuron_texts.items():
    textual_content_mapping[(i, j)] = ", ".join(texts) if texts else "No associated text"

# Prepare input text and transform it using previously defined vectorization and PCA transformation
input_text = "BLACK LIVES MATTER Leader Arrested On Charges Of CHILD TRAFFICKING http://po.st/BUnu8M #TrumpsterWarriors #MAGA"
input_vector = vectorize_and_transform_text(input_text, model, pca)

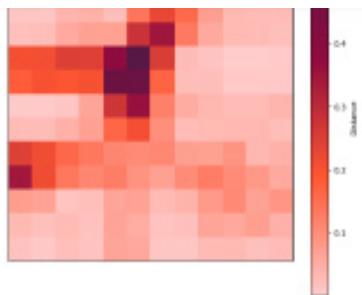
# Use the trained SOM and the input vector to find BMU and display associated text
BMU_index, BMU_content = activate_and_visualize_SOM(best_som, input_vector, textual_content_mapping)
print("BMU Index:", BMU_index)
print("BMU Content:", BMU_content)
```



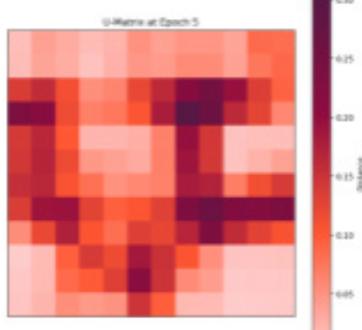
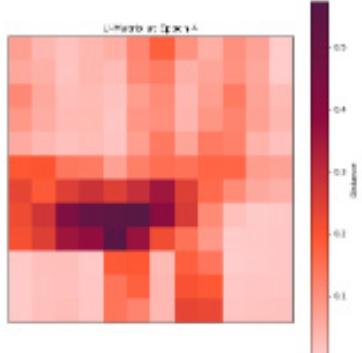
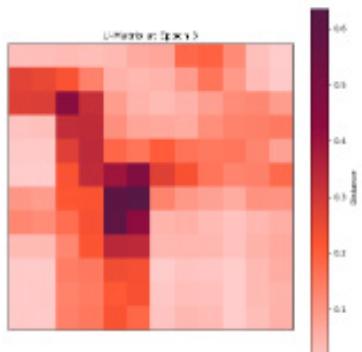
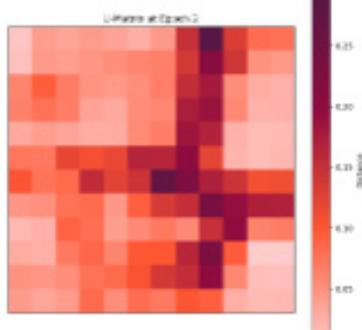
This step demonstrates a search function where text is entered and the most relevant text is obtained from BMU.

1.4 Search function

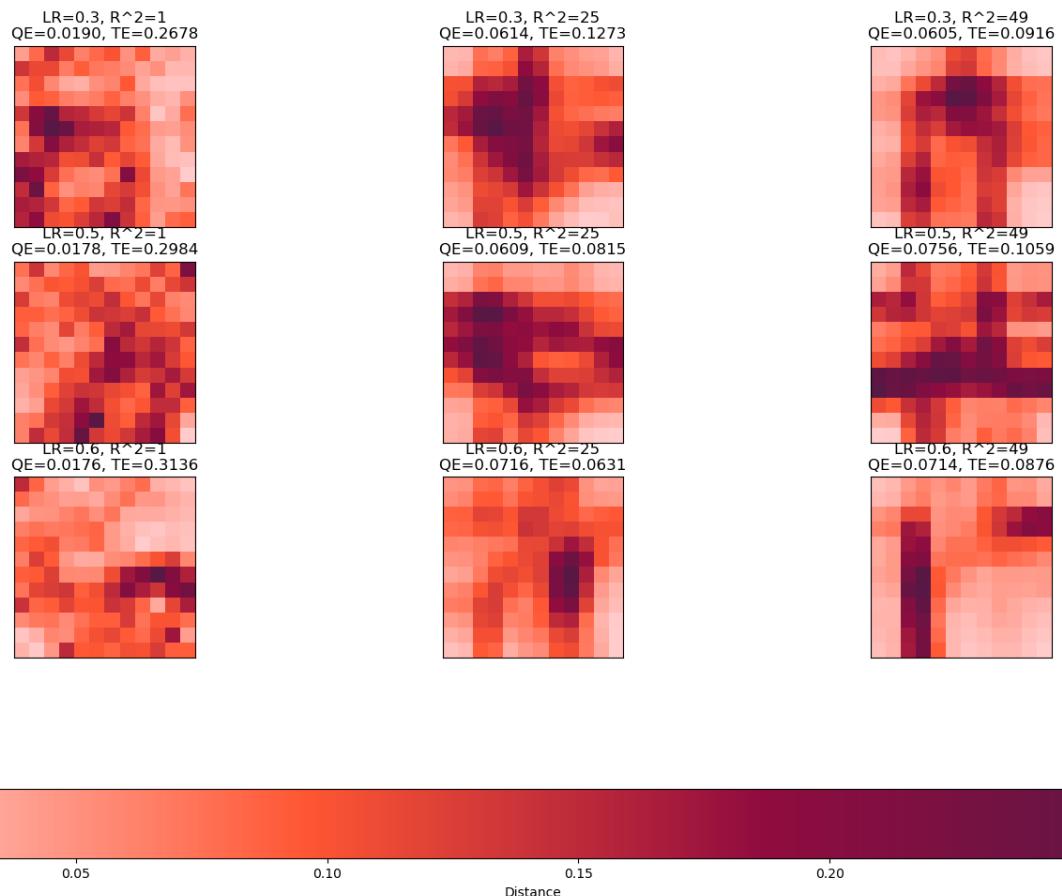
2 Input text - get similar Theater Line



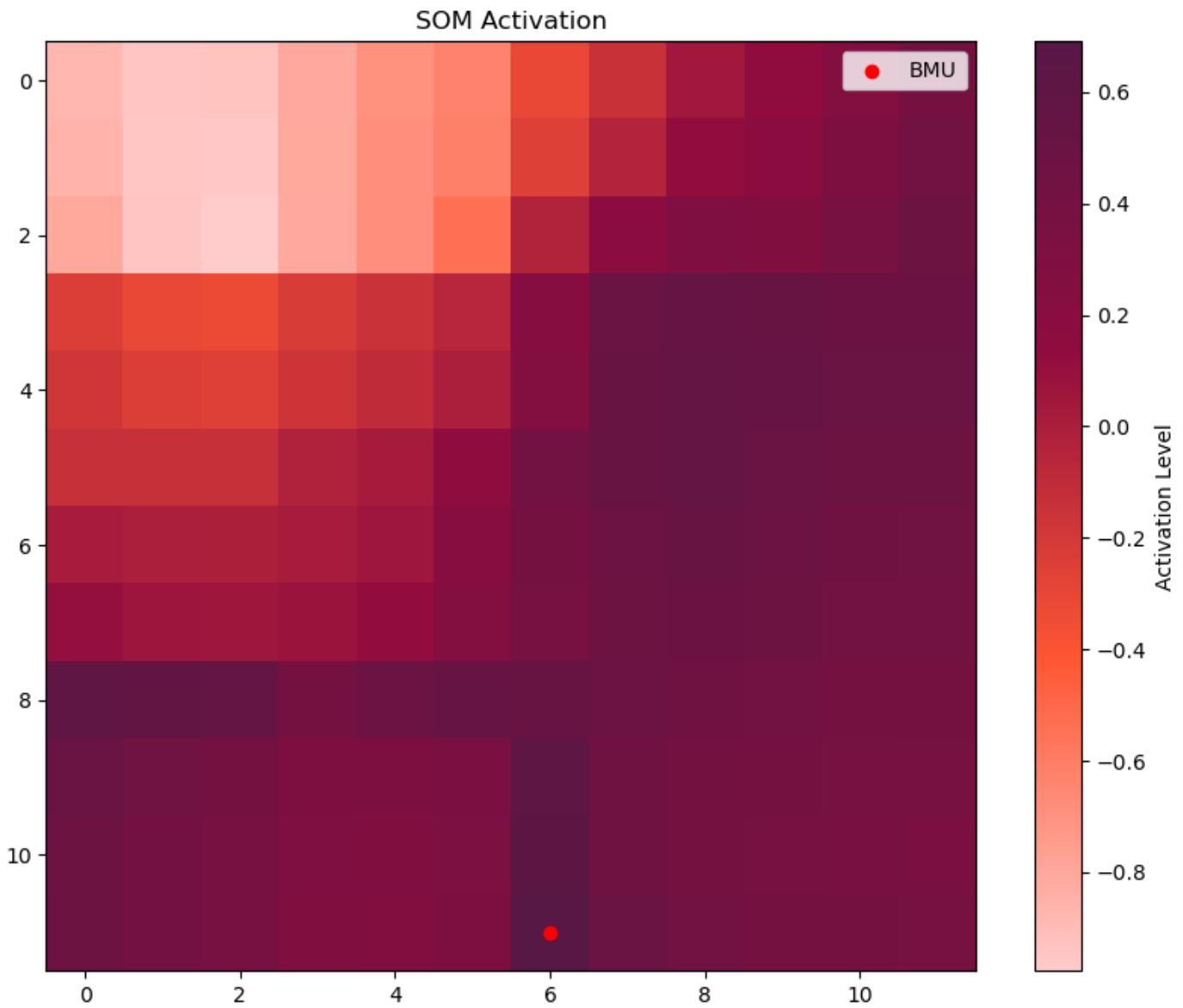
The process is exactly the same as on the previous two pages, so I won't repeat it again, but only show the results.



2.1 Visualize for different epochs



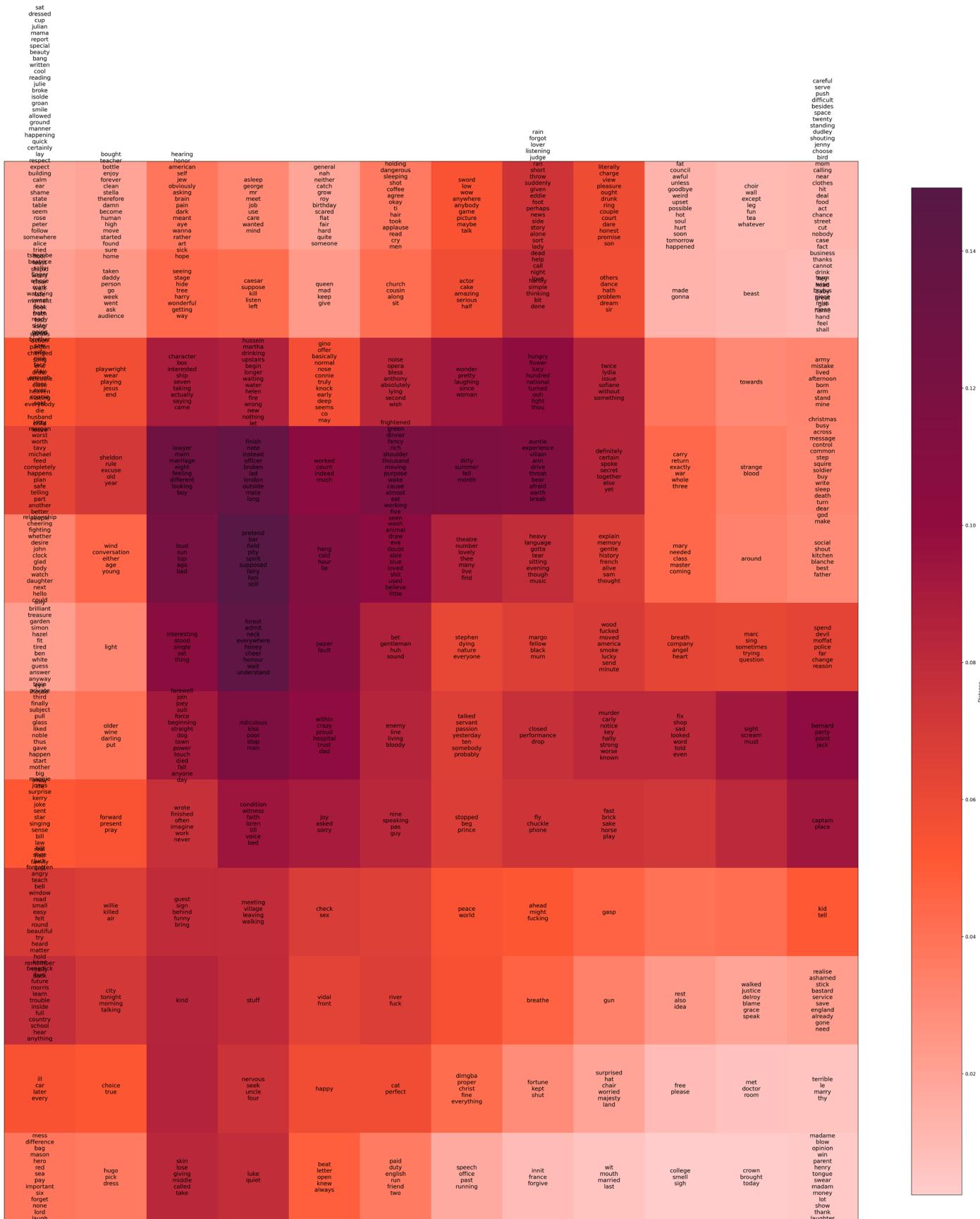
2.2 Visualize for different lr and r2



BMU Index: (11, 6)

BMU Content: eddie suppose somebody asks living mean ask, invitation auditorium imagine nobody ever done show, lot young people also older people lot people like art, horse town skinnier goat, yeah mean always making like remark like know, think theatre entertainment, man ride great machine man responsible man exists, think little bit eddie please crazy start work

2.3 Search function



2.4 Visualize with words

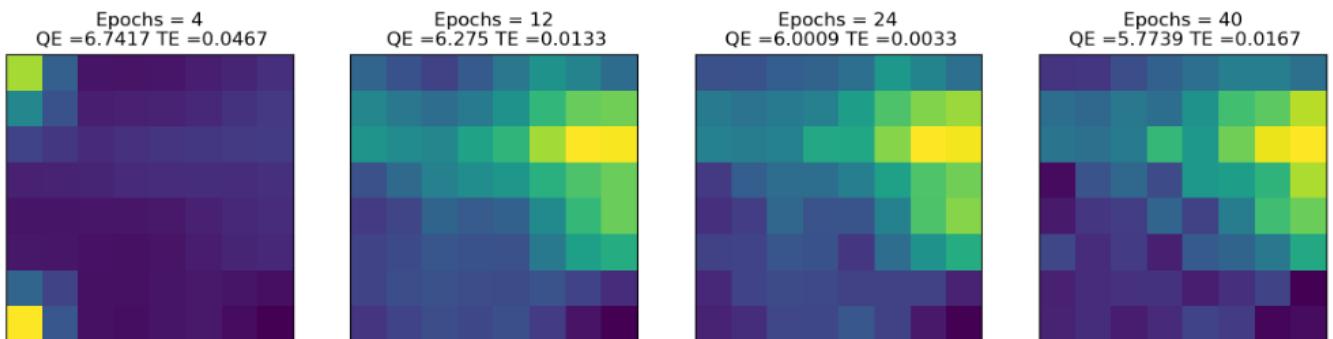
3 (Find the theatre scene corresponding to Theater)

No SOM was done for this step.

4 Input image - get similar Stage Scenes

4.1 Visualize for different epochs

```
fig, ax = plt.subplots(
    nrows=1, ncols=4, figsize=(15, 3.5),
    subplot_kw=dict(xticks=[], yticks=[]))
total_epochs = 0
SOMS = []
for epochs, i in zip([4, 8, 12, 16], range(0, 4)):
    total_epochs += epochs
    SOM = train_SOM(SOM, n_train_data, learn_rate = .1, radius_sq = 2, epochs = epochs)
    SOMS.append(SOM)
    QE = round(calculateQE(SOM, n_train_data), 4)
    TE = round(calculateTE(SOM, n_train_data), 4)
    ax[i].imshow(u_matrix(SOM))
    ax[i].title.set_text('Epochs = ' + str(total_epochs) + '\n QE = ' + str(QE) + ' TE = ' + str(TE))
```





4.2 Visualize with pictures

```

: # 读取并预处理图像
selected_image_filename = r"D:\sp\matched frames\TLA_-_Opera_Marina_Obra_Completa.mp4_38.jpg"
selected_image = Image.open(selected_image_filename)

# 将图像的特征提取出来
selected_image_features = processImage(selected_image_filename, model)

# 使用 activate 函数计算激活区域
activated_SOM = activate(selected_image_features, SOM, normalise(train_data, selected_image_features))

# 显示原始图像
plt.subplot(1, 3, 1)
plt.imshow(selected_image)
plt.title('Original Image')
plt.axis('off')

# 显示激活区域
plt.subplot(1, 3, 2)
plt.imshow(np.expand_dims(activated_SOM, axis=2), cmap='viridis')
plt.title('Activation Map')
plt.axis('off')

plt.show()

```

1/1 [=====] - 0s 27ms/step



4.3 Search function

Search and Save function

```
: output_directory = r"D:\sp\l11"

normalized_selected_features = normalise(train_data, selected_image_features)
bmu_index = find_BMU(SOM, normalized_selected_features)
neighbors = get_neighbors(bmu_index, som_height, som_width, radius=1) # You can adjust radius as needed

neighbor_images = []
for (i, j) in neighbors:
    if SOMimages[i][j]:
        neighbor_images.extend(SOMimages[i][j])

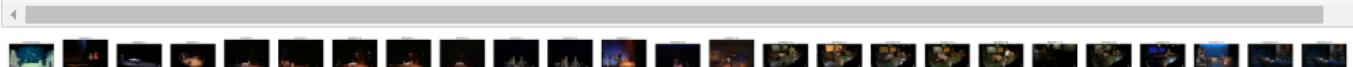
fig, axes = plt.subplots(1, len(neighbor_images) + 1, figsize=(5 * (len(neighbor_images) + 1), 10)) # Adjusted for larger display
axes[0].imshow(selected_image)
axes[0].set_title('Original Image')
axes[0].axis('off')

original_image_save_path = os.path.join(output_directory, 'Original_Image.jpg')
selected_image.save(original_image_save_path)

for idx, fi in enumerate(neighbor_images):
    img_path = os.path.join(r"D:\sp\matched frames", fi['image'])
    img = Image.open(img_path)
    axes[idx + 1].imshow(img)
    axes[idx + 1].set_title(f"Neighbor {idx + 1}")
    axes[idx + 1].axis('off')

    # Save each neighbor image
    neighbor_image_save_path = os.path.join(output_directory, f"Neighbor_{idx + 1}.jpg")
    img.save(neighbor_image_save_path)

plt.show()
```

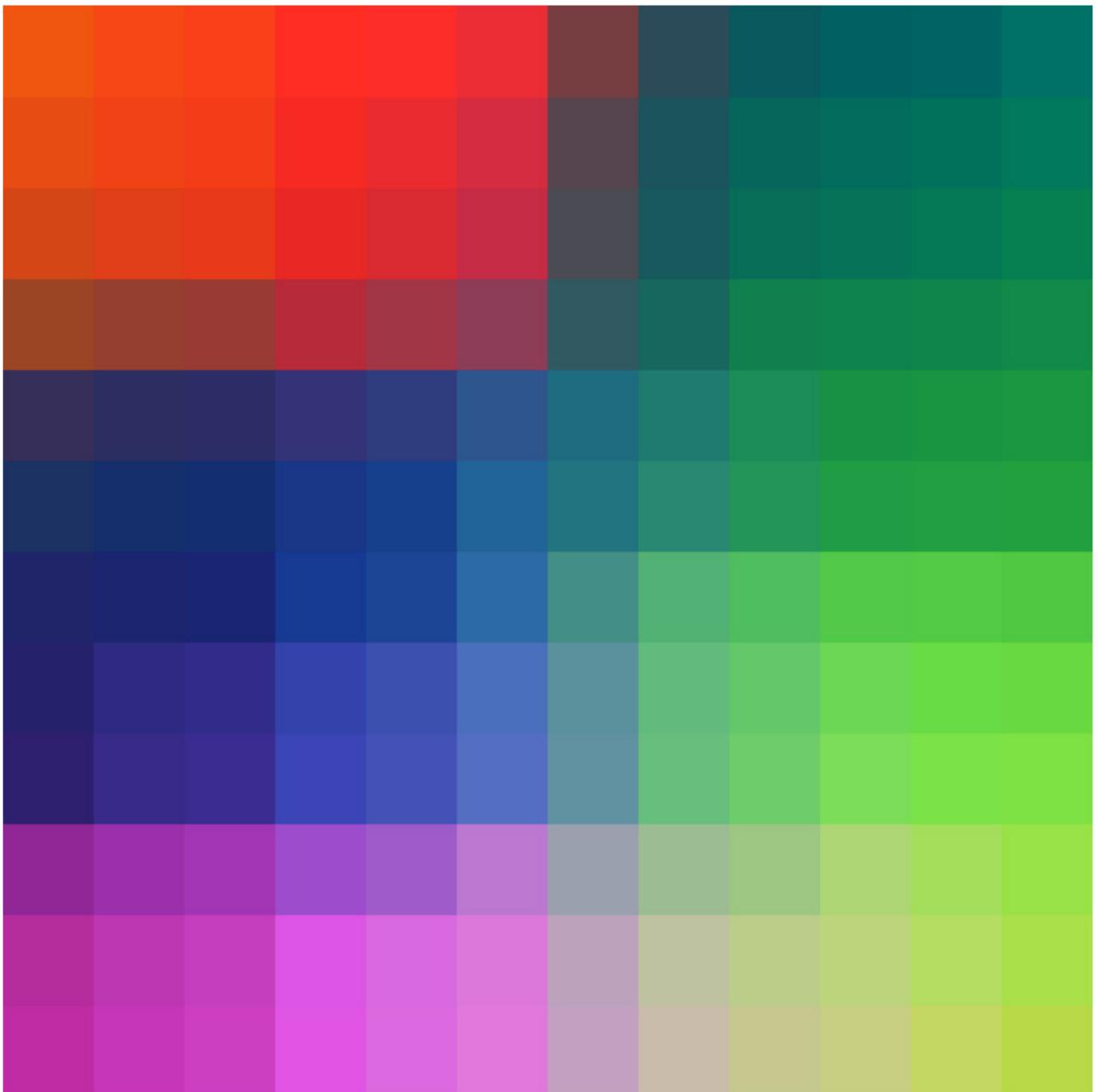


Enter an image, search for its corresponding BMU, and return twenty images on the BMU.

4.3 Search function

PCA

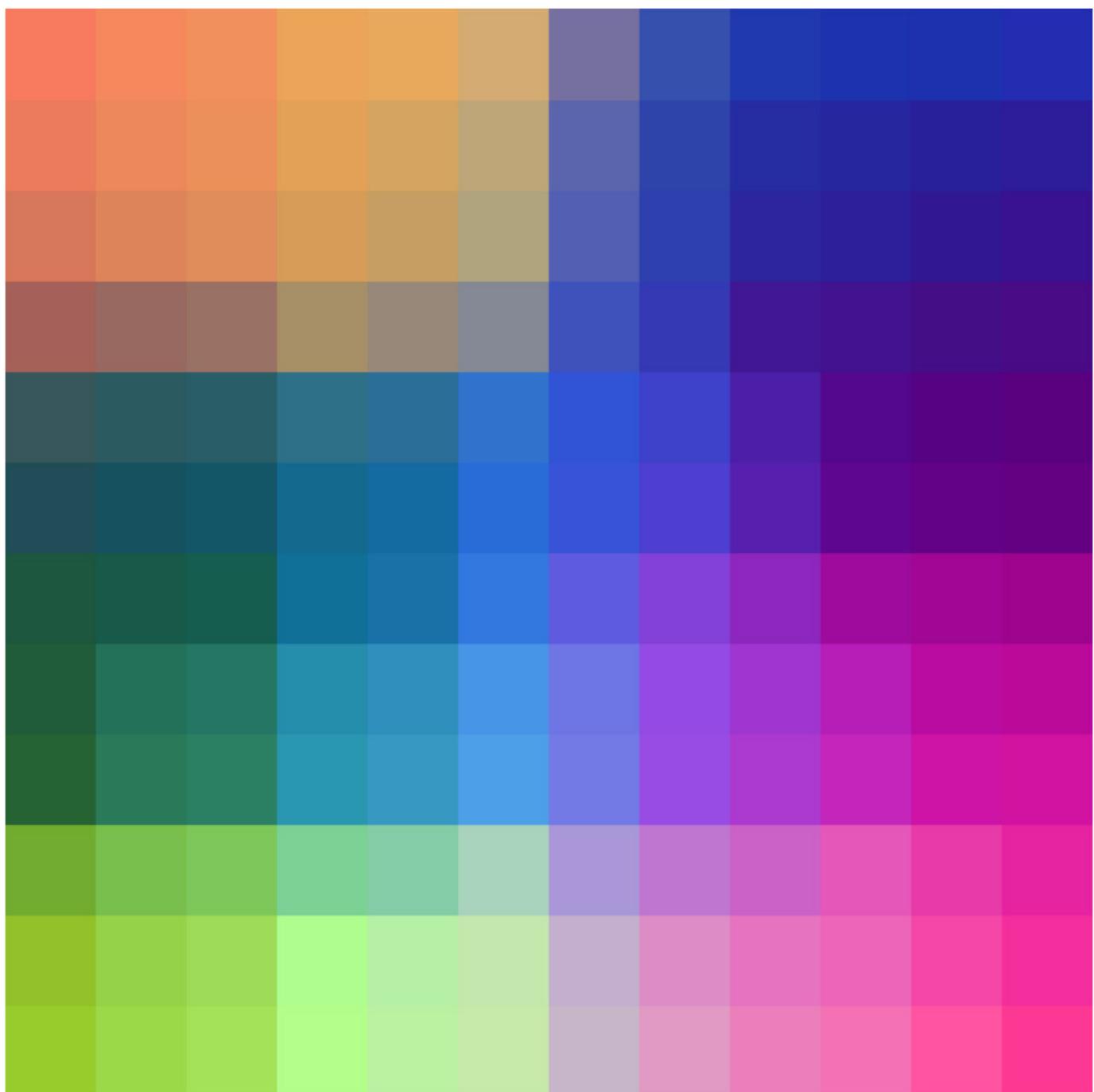
1 PCA on all cells



Fit PCA on All Cells of the SOM: In this method, you apply PCA directly to the feature vectors of all cells in the SOM grid. This approach treats each cell as an individual data point, calculating the principal components based on the variation observed directly in the SOM's representation of the data.

Fit PCA on the Entire Training Set, Then Apply to SOM Cells: Here, PCA is first fit on the entire dataset used to train the SOM. This captures the principal components that reflect the broad patterns and variance in the overall training data. The transformation defined by this PCA is then applied to each SOM cell to obtain their RGB values.

1 PCA on entire training

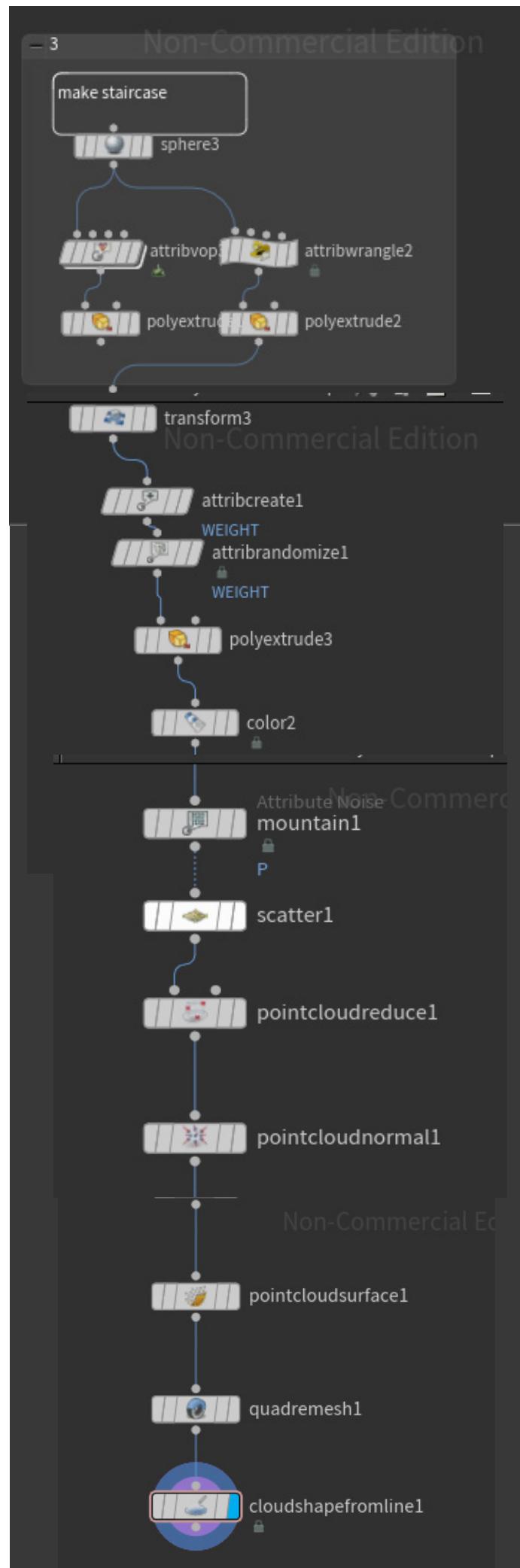


Joris

exercisel

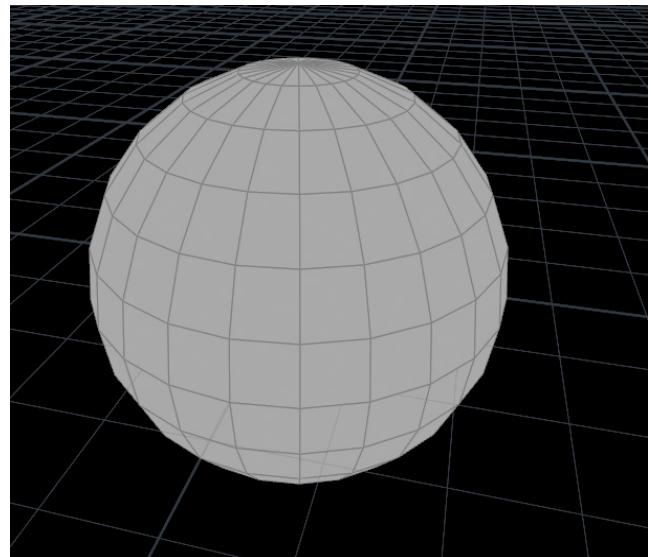
Master Node Diagram

(Detailed on back page)



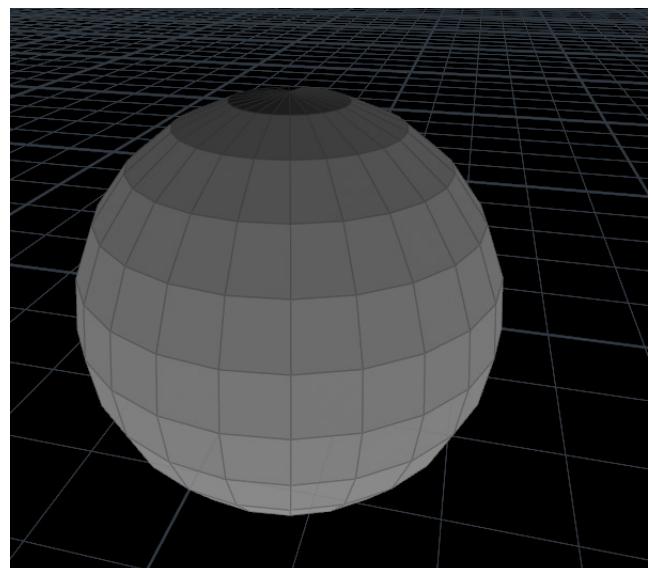
1 Sphere SOP

In Houdini, I started by creating a Sphere SOP. This node generates a sphere using the primitive type ‘polygon’.



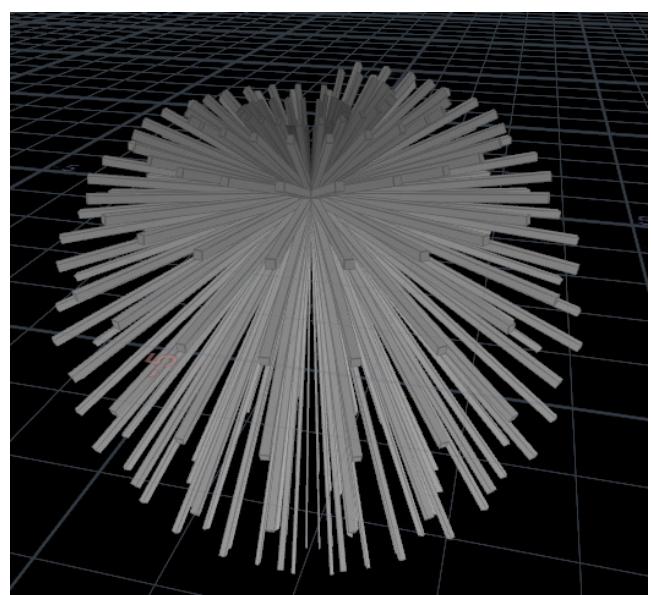
2Attribute Wrangle SOP

Following the sphere, an Attribute Wrangle SOP was added. Here, I used VEX to calculate a float attribute named increment, defined as the ratio of the current primitive number (@primnum) to the total number of primitives minus one (@numprim - 1).



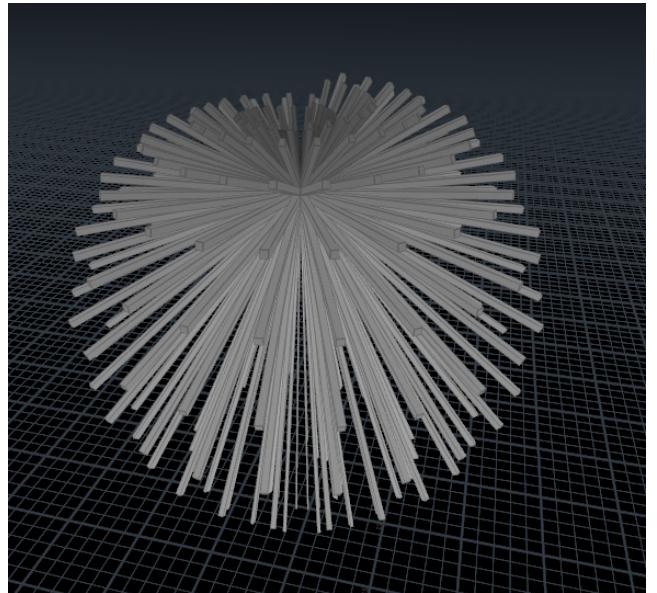
3 PolyExtrude SOP

Next, a PolyExtrude SOP was connected to extrude the sphere. The extrusion distance was set to 6.9 units.



4 Transform SOP

A Transform SOP was then used to translate the entire geometry upward by 30 units. This alteration repositions the extruded sphere.

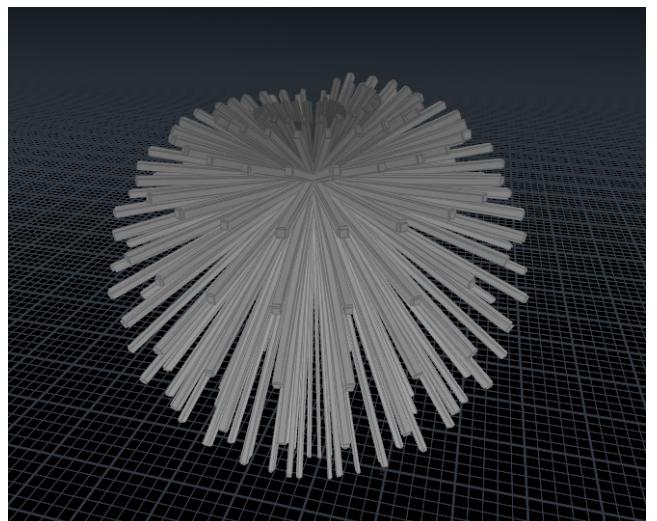


5 Attribute Create SOP

Following the transformation, an Attribute Create SOP was employed to introduce a new attribute named WEIGHT. This attribute is intended for later use in controlling other procedural effects.

6 Attribute Randomize SOP

An Attribute Randomize SOP was connected next, tasked with assigning random values between 0 and 1 to the WEIGHT attribute on each primitive.

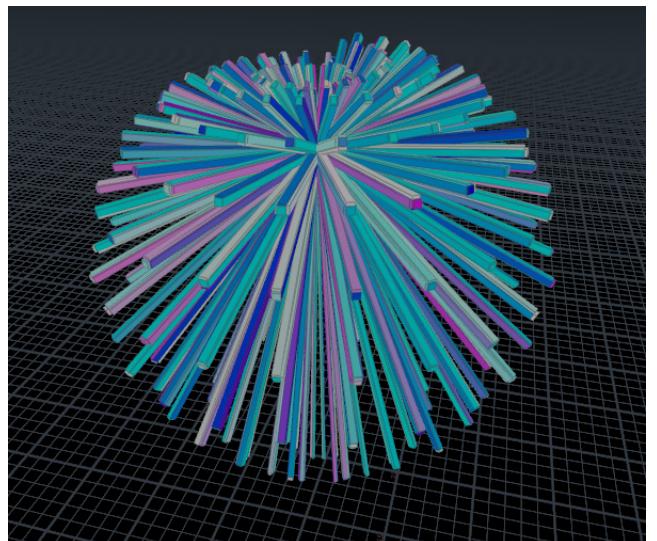


7 Second PolyExtrude SOP

Another PolyExtrude SOP was added, setting the extrusion distance to a subtle 0.054 units. This extrusion was applied selectively to primitives.

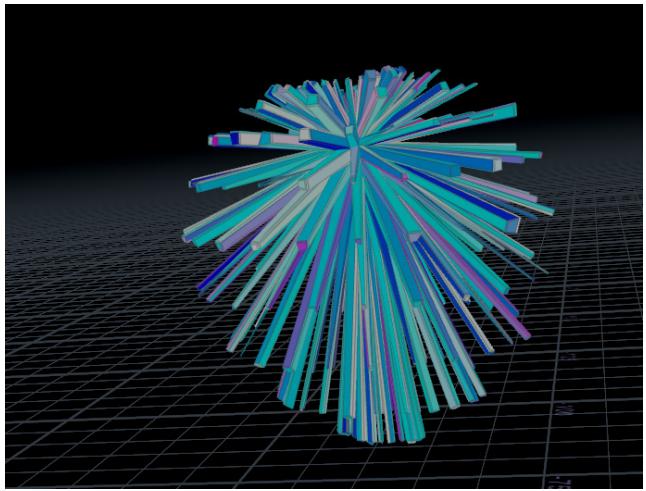
8 Color SOP

Finally, a Color SOP was used to colorize the geometry. The color assignment was driven by the WEIGHT attribute, affecting each primitive individually.



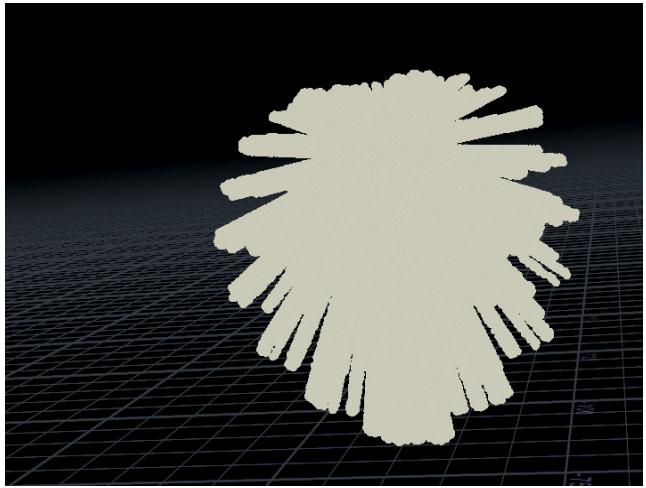
9 Mountain SOP

Following the previous steps, a Mountain SOP was utilized to add procedural displacement to the geometry's surface.



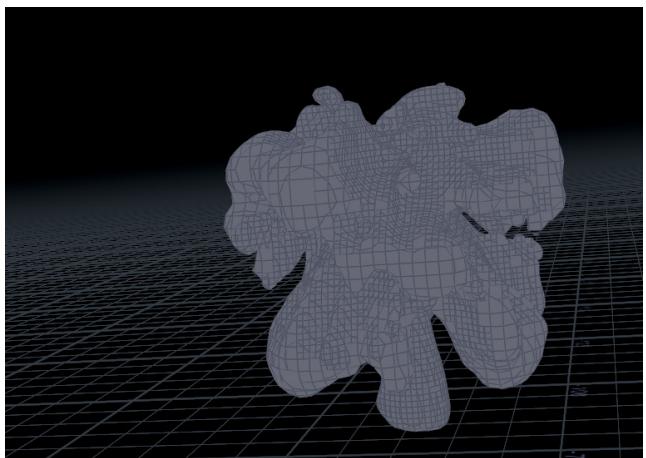
10 Scatter, PointCloudReduce, and PointCloudNormal SOPs

The next steps involved using the Scatter SOP to distribute points across the surface of the geometry, creating a dense point cloud. This was followed by a PointCloudReduce SOP, which optimizes the point cloud by reducing its density. The PointCloudNormal SOP was then applied to calculate and assign normals to the points.



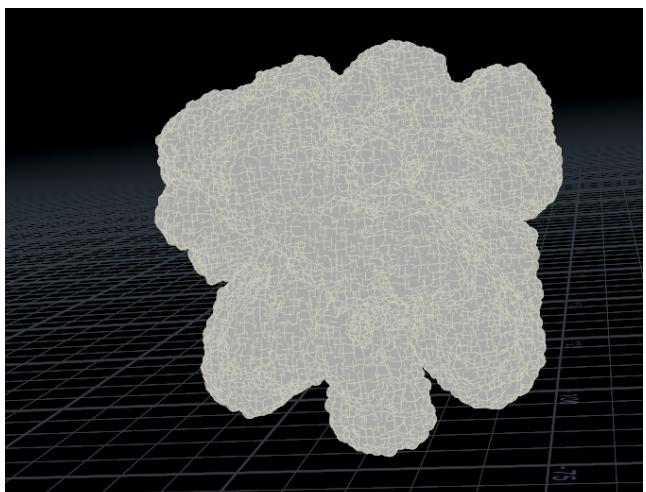
11 PointCloudSurface and QuadRemesh SOPs

The PointCloudSurface SOP was used next to generate a smooth surface from the optimized point cloud.

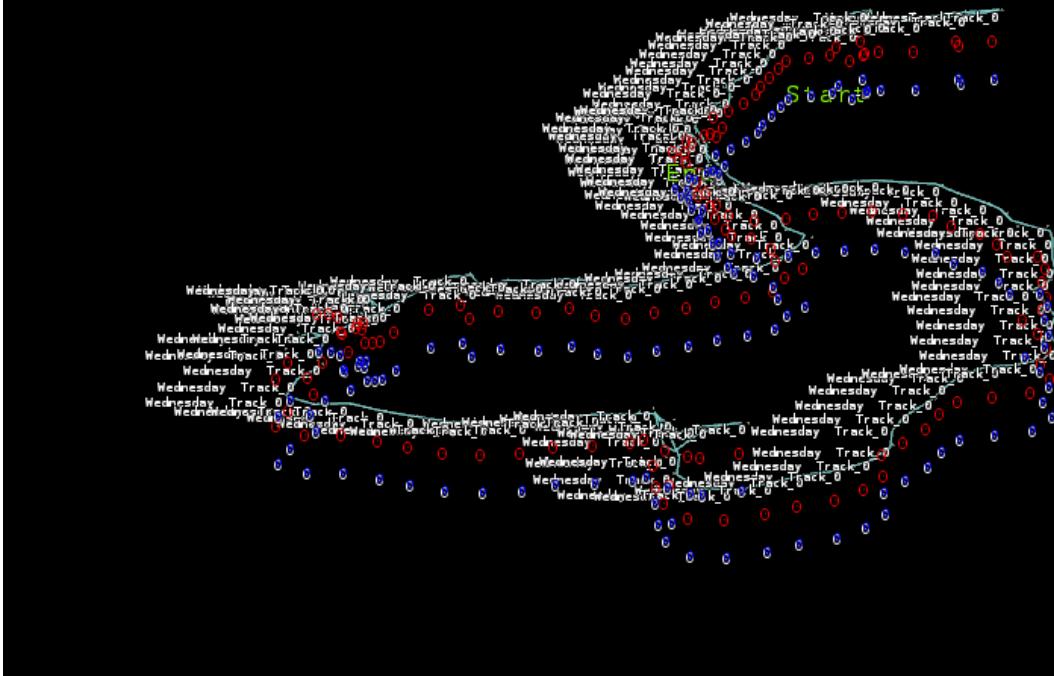


12 CloudShapeFromLine SOP

Lastly, the CloudShapeFromLine SOP was added to convert line geometry into volumetric clouds.

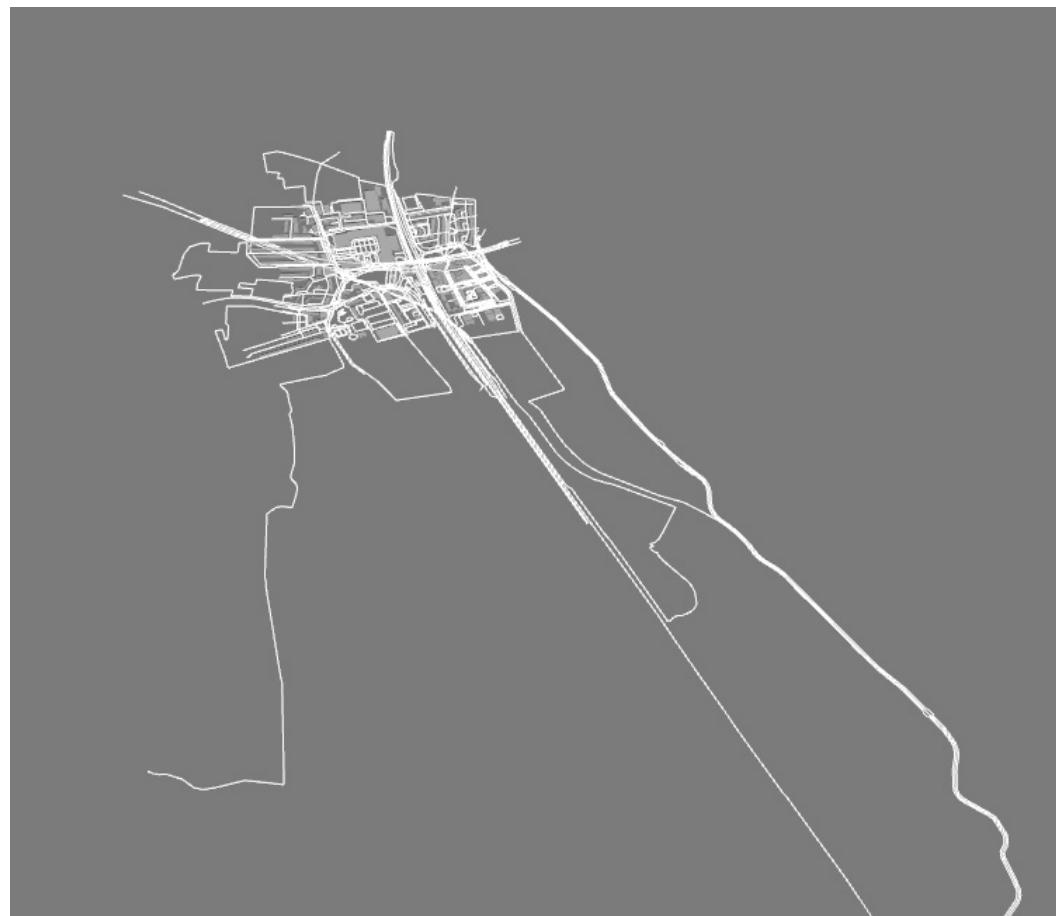


exercise2

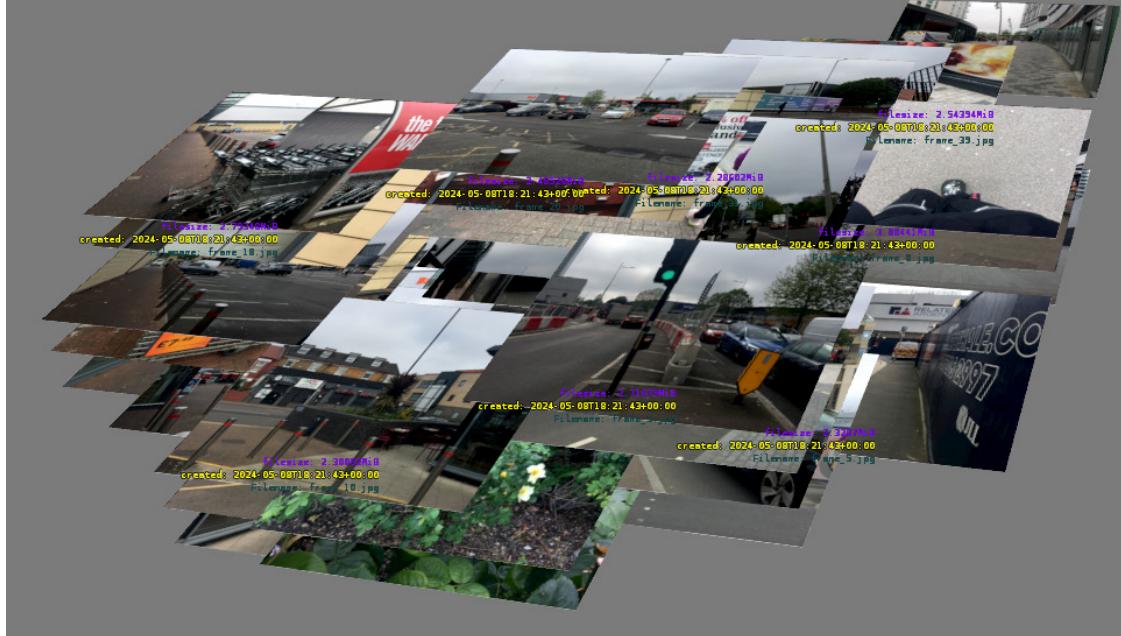


First load the GPX data. It was supposed to be converted from the KML format exported by Google Maps.

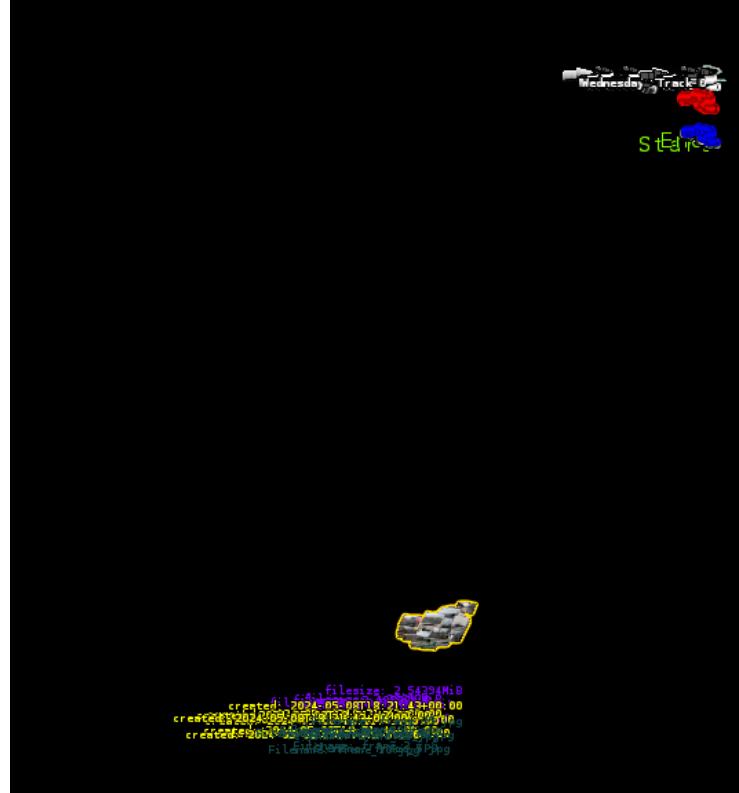
But my Google account showed that the Google Timeline function was not available, so I downloaded a GPS Tracker APP myself and got the path data exported by the APP.



Then load the osm data loaded from open map street.



Finally, the photo data is loaded. The photo information contains GPS related data, according to which the photos are placed and mapped according to their texture attributes.



But because of the format of the GPS data, there was no way to stack the three that were imported in.I tried my best toI tried my best to fix it.I tried my best to fix it.I did my best to fix that.I did my best to fix it, butI tried my best to fix it, but itI did my best to fix it, but it's justI tried my best to fix it, but it really doesn't workI tried my best to fix it, but it just doesn't work!I tried my best to fix it, but it just doesn't work.

exercise3

1

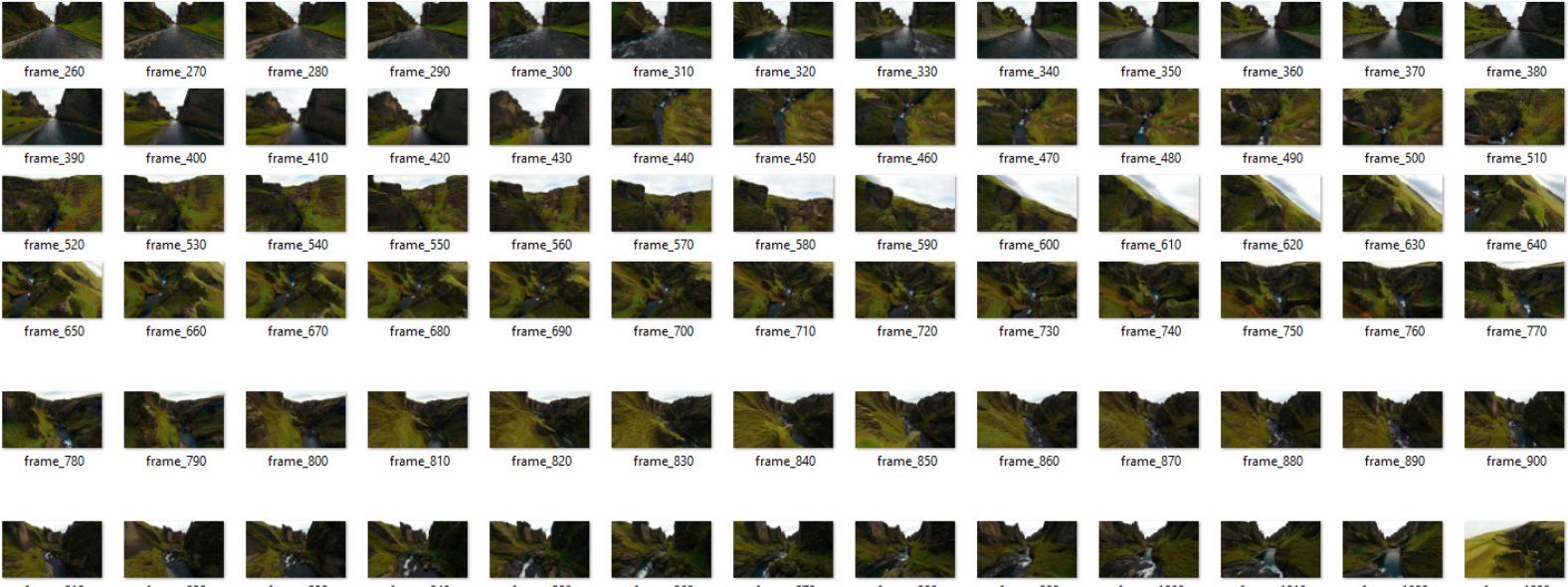
First, I downloaded a video clip and used Houdini's "geometry" node to decompose it into a sequence of frames. Then, I utilized a third-party tool called CapturingReality to perform 3D reconstruction on the video, exporting an OBJ model, texture maps, and camera data.

2

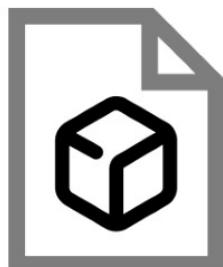
Next, I created a new scene file in Houdini and imported the geometry and camera data exported from CapturingReality using the "file" node. Since the model and camera data were merged, I used the "object merge" node to separate them into two different node trees.

I proceeded to reconstruct and optimize the camera path. I first used the "add" node to convert the camera position data into a set of points, then applied the "Sort" node to adjust the order of the points. To smoothen the camera motion path, I utilized the "Refine" node to interpolate and insert additional rotation and position keyframes between the existing keyframes.

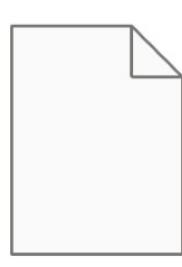
At this stage, I could play back the scene on the timeline to review the video reconstruction achieved by the combination of AI and Houdini.



mesh_Model_1



mesh_Model_1



mesh_Model_1.obj.rcinfo

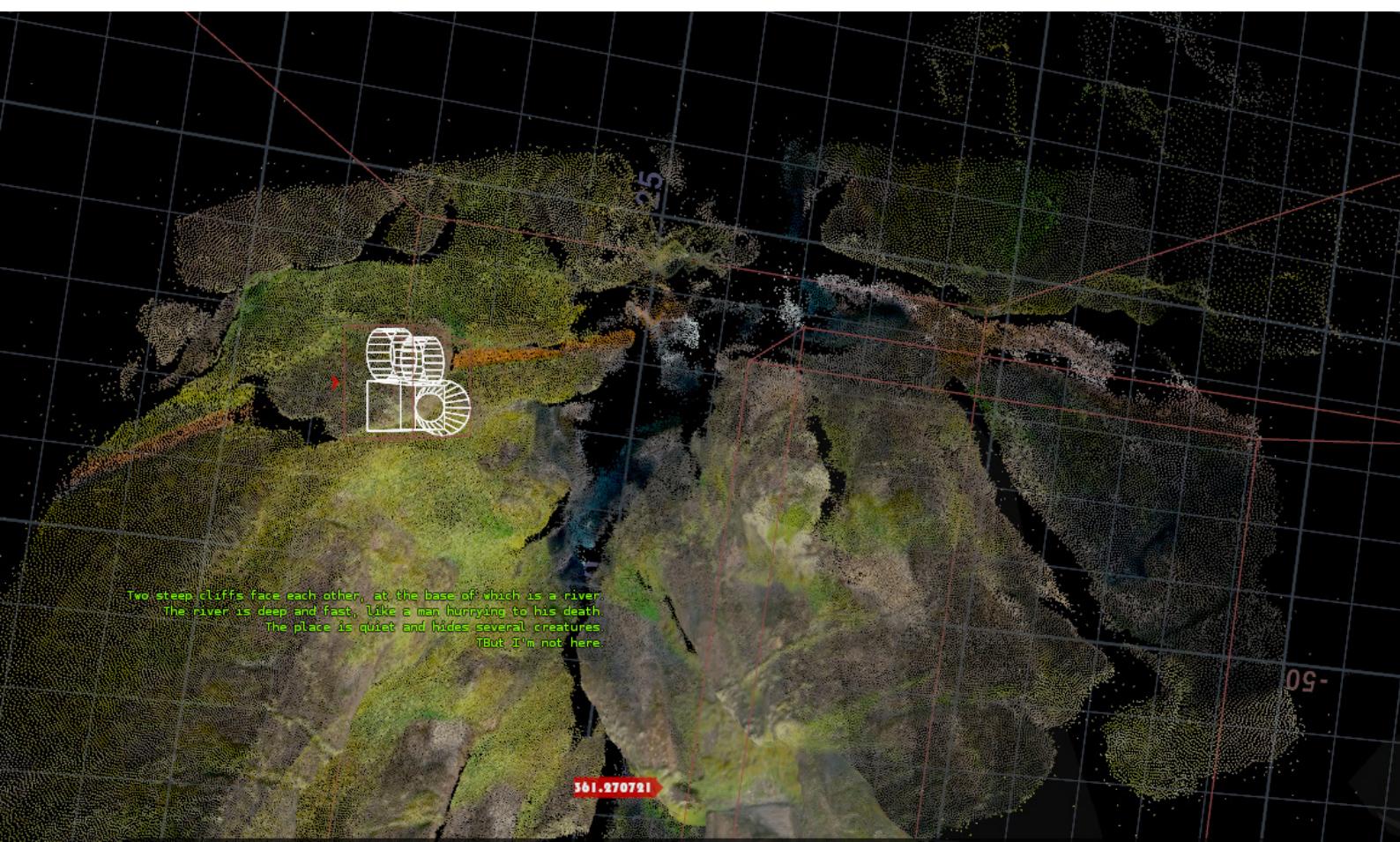
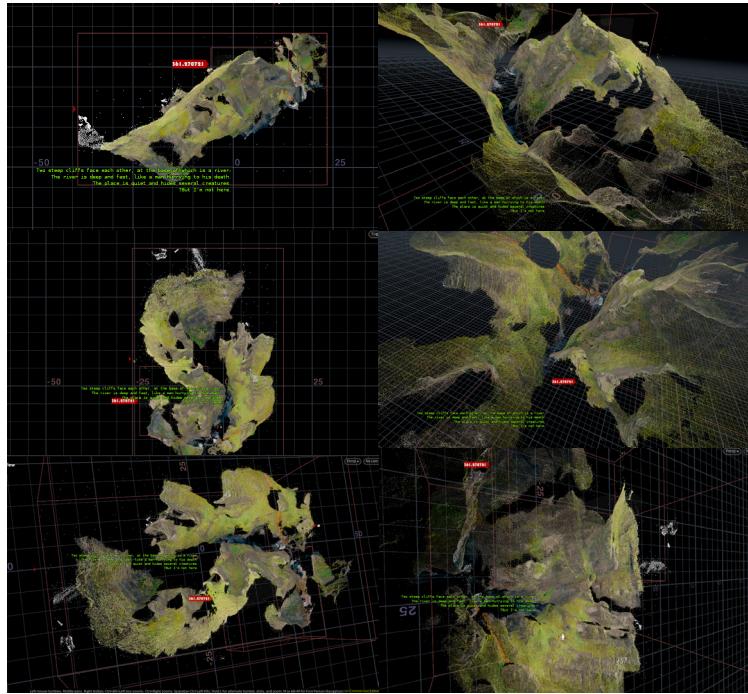


mesh_Model_1_u1_v1_diffuse



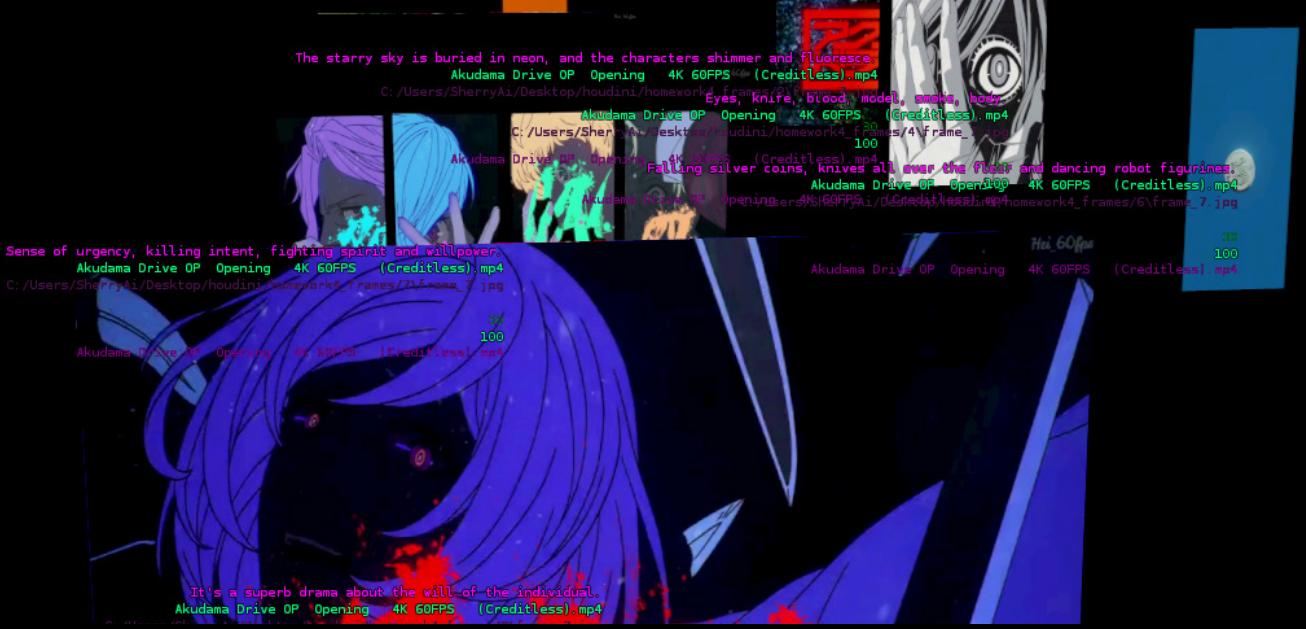
mesh_Model_1_u2_v1_diffuse

Subsequently, I began adding additional attribute data to the model object in the scene. Using the “Attribute” node, I created two new point attributes for each point: volume and descriptive text. I read the relevant data from the previously parsed JSON file and assigned it to the newly created attributes using the VEX language. Through these steps, I successfully reconstructed the video scene in Houdini and added extra information to the objects within the scene. Building upon this, I can proceed with further visual effects compositing and rendering work.



exercise4





CeeI

1

I define a function sma_multi that takes two NumPy arrays A and B as input and computes their matrix multiplication using nested loops. The result is stored in a new NumPy array C and returned by the function.

```
import numpy as np

def sma_multi(A, B):
    n = A.shape[0]
    C = np.zeros((n, n), dtype=int)
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i, j] += A[i, k] * B[k, j]
    return C

A = np.array([[2, 4], [0, 5]])
B = np.array([[3, 2], [1, 6]])
C = sma_multi(A, B)
print("Result:\n", C)
print("Time complexity is O(n^3) since there are 3 nested loops each iterating n times.")
```

Result:

```
[[10 28]
 [ 5 30]]
```

Time complexity is $O(n^3)$ since there are 3 nested loops each iterating n times.

```
def matrix_mult_list(A, B):
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])

    if cols_A != rows_B:
        print("not able to mult")
        return

    C = [[0 for col in range(cols_B)] for row in range(rows_A)]

    for i in range(rows_A):
        for j in range(cols_B):
            for k in range(cols_A):
                C[i][j] += A[i][k] * B[k][j]

    return C

C_numpy = sma_multi(np.array(A), np.array(B))
C_list = matrix_mult_list(A, B)

print("NumPy result:\n", C_numpy)
print("List result:\n", np.array(C_list))
```

NumPy result:

```
[[10 28]
 [ 5 30]]
```

List result:

```
[[10 28]
 [ 5 30]]
```

I compared the result from your nested list implementation to the NumPy built-in matrix multiplication function.

```
: A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

C1 = A @ B
C2 = np.matmul(A, B)
C3 = A.dot(B)

print("A @ B:\n", C1)
print("np.matmul(A, B):\n", C2)
print("A. dot(B):\n", C3)
```

```
A @ B:
[[19 22]
 [43 50]]
np.matmul(A, B):
[[19 22]
 [43 50]]
A. dot(B):
[[19 22]
 [43 50]]
```

I created a function to multiply multiple matrices together.

```
: def multi_matrix_mult(matrices):
    result = matrices[0]
    for matrix in matrices[1:]:
        result = sma_multi(result, matrix)
    return result

matrices = [np.random.randint(10, size=(3, 3)) for _ in range(4)]
result = multi_matrix_mult(matrices)
print("Multi matrix multiplication result:\n", result)

M1 = np.random.randint(10, size=(3, 3))
M2 = np.random.randint(10, size=(3, 3))
M3 = np.random.randint(10, size=(3, 3))
M4 = np.random.randint(10, size=(3, 3))

print("Matrix 1:\n", M1)
print("Matrix 2:\n", M2)
print("Matrix 3:\n", M3)
print("Matrix 4:\n", M4)

result = multi_matrix_mult([M1, M2, M3, M4])
print("Result of M1 * M2 * M3 * M4:\n", result)
```

Multi matrix multiplication result:

```
[[20823 4413 8727]
[29184 6154 12406]
[11480 2380 5152]]
```

Matrix 1:

```
[[6 4 3]
[2 3 1]
[9 9 1]]
```

Matrix 2:

```
[[5 3 4]
[5 1 5]
[9 4 0]]
```

Matrix 3:

```
[[7 7 3]
[3 8 1]
[6 0 0]]
```

Matrix 4:

```
[[3 2 6]
[6 1 0]
[1 8 2]]
```

Result of M1 * M2 * M3 * M4:

```
[[ 7846 4741 5960]
[ 3412 2092 2720]
[10312 6307 8468]]
```

2

I break down the matrices into smaller submatrices (quarters), recursively multiplying those submatrices, and then combining the results to form the final product matrix.

```

def sma_mult_recursive(A, B, indent=0):
    n = A.shape[0]
    indent_str = '    ' * indent

    C = np.zeros((n, n), dtype=int)

    if n == 1:
        C[0, 0] = A[0, 0] * B[0, 0]
    else:
        A11, A12, A21, A22 = split_m(A)
        B11, B12, B21, B22 = split_m(B)

        C11 = sma_mult_recursive(A11, B11, indent+1)
        C12 = sma_mult_recursive(A11, B12, indent+1)
        C21 = sma_mult_recursive(A21, B11, indent+1)
        C22 = sma_mult_recursive(A21, B12, indent+1)

        tmp1 = sma_mult_recursive(A12, B21, indent+1)
        tmp2 = sma_mult_recursive(A12, B22, indent+1)
        tmp3 = sma_mult_recursive(A22, B21, indent+1)
        tmp4 = sma_mult_recursive(A22, B22, indent+1)

        C11 += tmp1
        C12 += tmp2
        C21 += tmp3
        C22 += tmp4

    C = combine_m(C11, C12, C21, C22)

    return C

def split_m(M):
    n = M.shape[0]//2
    M11 = M[:n, :n]
    M12 = M[:n, n:]
    M21 = M[n:, :n]
    M22 = M[n:, n:]
    return M11, M12, M21, M22

def combine_m(C11, C12, C21, C22):
    n = C11.shape[0]
    C = np.zeros((2*n, 2*n), dtype=int)
    C[:n, :n] = C11
    C[:n, n:] = C12
    C[n:, :n] = C21
    C[n:, n:] = C22
    return C

C = sma_mult_recursive(A, B)
print("\nResult:\n", C)
print("")

Recursiveness: Base case is when n=1, a 1x1 matrix multiply; Recursion step divides matrices into quarter

```

Divide and Conquer: Split matrices A and B into 4 quarters each; Recursively multiply quarters; Add multipi

Result:

```
[[19 22]
 [43 50]]
```

Recursiveness: Base case is when n=1, a 1x1 matrix multiply; Recursion step divides matrices into quarters and multiplies quarters recursively; Recursion reduces to base case when matrices get divided all the way down to 1x1

Divide and Conquer: Split matrices A and B into 4 quarters each; Recursively multiply quarters; Add multiplied quarter results to get final matrix C

Complexity Analysis: Base case is $O(1)$ when n=1; Divide step is $O(1)$ as it just splits the matrices; Conquer step has 8 recursive calls of size $n/2$, so $O(8*(n/2)^2) = O(n^2)$; Combine step has 4 matrix additions of $n/2 \times n/2$ matrices, so $O(4*(n/2)^2) = O(n^2)$. Putting it together using the master theorem, with $a=8$, $b=2$, $f(n) = O(n^2)$:

$$\begin{aligned} T(n) &= 8T(n/2) + O(n^2) \\ &= O(n^3) \text{ using case 2 of the master theorem} \end{aligned}$$

So the overall time complexity is $O(n^3)$, same as the iterative solution.

```
print("")
```

The recursion tree has a branching factor of 8, since each recursive call splits the problem into 8 subproblems of size $n/2$ (the 8 multiplications of quarter matrices). At each level of the recursion tree, the work done is $O(n^2)$ from the splitting and combining of matrices. Since the recursion bottoms out at matrices of size 1x1, the depth of the recursion tree is $O(\log n)$, giving an overall time complexity of $O(n^3)$.

```
""")
```

The recursion tree has a branching factor of 8, since each recursive call splits the problem into 8 subproblems of size $n/2$ (the 8 multiplications of quarter matrices). At each level of the recursion tree, the work done is $O(n^2)$ from the splitting and combining of matrices.

Since the recursion bottoms out at matrices of size 1x1, the depth of the recursion tree is $O(\log n)$, giving an overall time complexity of $O(n^3)$.

3

I implemented the Strassen algorithm for matrix multiplication and analyzed the time complexity of the Strassen algorithm.

```
def strassen(A, B):
    n = A.shape[0]
    C = np.zeros((n, n), dtype=int)
    if n == 1:
        C[0, 0] = A[0, 0] * B[0, 0]
    else:
        A11, A12, A21, A22 = split_m(A)
        B11, B12, B21, B22 = split_m(B)
        S1 = B12 - B22
        S2 = A11 + A12
        S3 = A21 + A22
        S4 = B21 - B11
        S5 = A11 + A22
        S6 = B11 + B22
        S7 = A12 - A22
        S8 = B21 + B22
        S9 = A11 - A21
        S10 = B11 + B12
        P1 = strassen(A11, S1)
        P2 = strassen(S2, B22)
        P3 = strassen(S3, B11)
        P4 = strassen(A22, S4)
        P5 = strassen(S5, S6)
        P6 = strassen(S7, S8)
        P7 = strassen(S9, S10)
        C11 = P5 + P4 - P2 + P6
        C12 = P1 + P2
        C21 = P3 + P4
        C22 = P5 + P1 - P3 - P7
        C = combine_m(C11, C12, C21, C22)
    return C

A = np.array([[6, 3, 10, 4],
              [10, 3, 6, 10],
              [6, 9, 2, 5],
              [4, 2, 9, 4]])

B = np.array([[8, 1, 6, 4],
              [7, 5, 9, 2],
              [6, 3, 7, 8],
              [4, 9, 3, 1]])

C = strassen(A, B)
print("\nResult:\n", C)

print("""
Reflections:
Addition/subtraction are O(n^2) operations as just iterate through matrix once; Multiplication is O(n^3) w
Complexity Analysis:
Base case is O(1) when n=1 ; Divide step has 7 multiplications of n/2 x n/2 = O(7*(n/2)^3) = O(n^3/8); Con
Putting together using the master theorem, with a=7, b=2, f(n) = O(n^3/8) + O(5n^2/4) = O(n^3/4):
T(n) = 7T(n/2) + O(n^3/4)""")
```

Result:

```
[[145 87 145 114]
 [177 133 159 104]
 [143 102 146 63]
 [116 77 117 96]]
```

Reflections:

Addition/subtraction are $O(n^2)$ operations as just iterate through matrix once; Multiplication is $O(n^3)$ with nested loops

Complexity Analysis:

Base case is $O(1)$ when $n=1$; Divide step has 7 multiplications of $n/2 \times n/2 = O(7*(n/2)^3) = O(n^{3/8})$; Conquer step has 10 additions of $n/2 \times n/2 = O(10*(n/2)^2) = O(5n^{2/4})$; Combine step has 8 additions of $n/2 \times n/2 = O(8*(n/2)^2) = O(n^2)$

Putting together using the master theorem, with $a=7$, $b=2$, $f(n) = O(n^{3/8}) + O(5n^{2/4}) = O(n^{3/4})$:

$$T(n) = 7T(n/2) + O(n^{3/4})$$