3. Autoencoder DEC

January 27, 2024

1 Deep Embedded Clustering using Autoencoders

[ ]:
```
import numpy as np
import h5py
import os, re, glob
import math
from scipy import signal
from scipy.signal import butter, lfilter
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D,␣
↪Conv1D, MaxPooling1D, UpSampling1D, Flatten, Dropout, Reshape
from keras.layers import Bidirectional, BatchNormalization, ZeroPadding1D,␣
↪Conv2DTranspose
from keras.models import Model
from keras import backend as K
from tensorflow.keras.optimizers import SGD, Adam, schedules
from keras import regularizers
from tensorflow.keras.layers import Layer, InputSpec
from keras.callbacks import ModelCheckpoint, LearningRateScheduler,␣
↪ReduceLROnPlateau, EarlyStopping
from keras.initializers import VarianceScaling
from keras.callbacks import CSVLogger
from scipy.optimize import linear_sum_assignment as linear_assignment
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from matplotlib.pyplot import savefig
import seaborn as sns
from keras.utils import plot_model
from IPython.display import Image, display
import tensorflow as tf

sns.set_style('darkgrid')
sns.set_palette('muted')

import tensorflow as tf
gpus = tf.config.list_physical_devices('GPU')
if gpus:
```

```python
try:
  # Currently, memory growth needs to be the same across GPUs
  for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
  logical_gpus = tf.config.list_logical_devices('GPU')
  print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
except RuntimeError as e:
  # Memory growth must be set before GPUs have been initialized
  print(e)
```

```
2024-01-27 17:01:28.329156: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-01-27 17:01:28.329198: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-01-27 17:01:28.330300: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-01-27 17:01:28.336860: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-01-27 17:01:29.270819: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

```
2 Physical GPUs, 2 Logical GPUs
```

```
2024-01-27 17:01:30.472445: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.472724: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.495094: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
```

https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.495947: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.496944: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.498369: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.675927: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.676191: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.676419: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.676636: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2024-01-27 17:01:30.676866: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at

```
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.677084: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.686928: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.687220: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.687453: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.687684: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.687910: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-27 17:01:30.688086: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1929] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 6368 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 2080 SUPER, pci bus id: 0000:0a:00.0, compute
capability: 7.5
2024-01-27 17:01:30.688588: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
```

```
pci#L344-L355
2024-01-27 17:01:30.688755: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1929] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 6799 MB memory:  -> device: 1,
name: NVIDIA GeForce RTX 2080 SUPER, pci bus id: 0000:0b:00.0, compute
capability: 7.5
```

1.1 Autoencoder Architecture

[ ]:
```
from numpy.random import seed
seed(46)

import tensorflow as tf
tf.random.set_seed(46)

from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Reshape,␣
↪UpSampling2D
from keras.models import Model, Sequential
import keras.backend as K

initializer = tf.keras.initializers.GlorotUniform(seed=46)

# Change input shape
inp = Input(shape=(128, 32, 1))

# Adjust encoder layers
e = Conv2D(8, (7, 5),strides=[2,2], activation='elu',␣
↪kernel_initializer=initializer, padding='same')(inp)
e = Conv2D(16, (5, 3),strides=[2,2], activation='elu',␣
↪kernel_initializer=initializer, padding='same')(e)
e = Conv2D(32, (5, 3),strides=[2,2], activation='elu',␣
↪kernel_initializer=initializer, padding='same')(e)
e = Conv2D(64, (5, 3),strides=[2,2], activation='elu' ,␣
↪kernel_initializer=initializer, padding='same')(e)

# Get shape before flattening
shape_before_flattening = K.int_shape(e)

# Encode to a dense layer
encoded1 = Flatten()(e)
encoded2 = Dense(24, activation="elu")(encoded1)
# Remove redundant dense layer
fc = Dense(np.prod(shape_before_flattening[1:]), activation="elu")(encoded2)

# Decoder layers
d = Reshape(shape_before_flattening[1:])(fc)
```

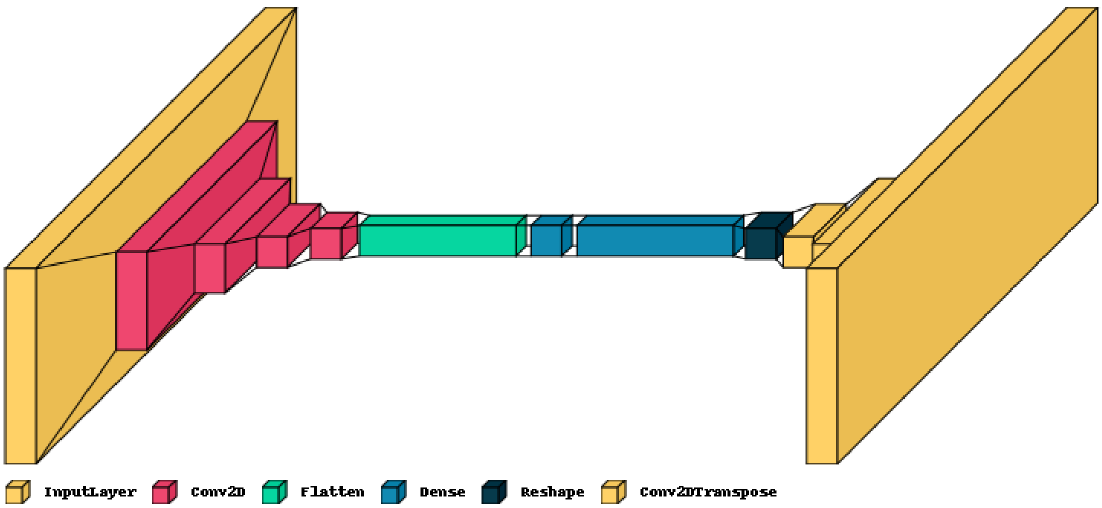
```python
d = Conv2DTranspose(32, (5, 3), strides=[2,2], activation='elu',␣
 ↪kernel_initializer=initializer, padding='same')(d)
d = Conv2DTranspose(16, (5, 3), strides=[2,2], activation='elu',␣
 ↪kernel_initializer=initializer, padding='same')(d)
d = Conv2DTranspose(8, (5, 3),strides=[2,2] , activation='elu',␣
 ↪kernel_initializer=initializer, padding='same')(d)
decoded = Conv2DTranspose(1, (7, 5),strides=[2,2] , activation='linear',␣
 ↪kernel_initializer=initializer, padding='same')(d)

autoencoder = Model(inputs=inp, outputs=decoded, name="autoencoder")
encoder = Model(inputs=inp, outputs=encoded2, name="encoder")

autoencoder.summary()
```

```
Model: "autoencoder"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 32, 1)]      0

 conv2d (Conv2D)             (None, 64, 16, 8)         288

 conv2d_1 (Conv2D)           (None, 32, 8, 16)         1936

 conv2d_2 (Conv2D)           (None, 16, 4, 32)         7712

 conv2d_3 (Conv2D)           (None, 8, 2, 64)          30784

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 24)                24600

 dense_1 (Dense)             (None, 1024)              25600

 reshape (Reshape)           (None, 8, 2, 64)          0

 conv2d_transpose (Conv2DTr  (None, 16, 4, 32)         30752
 anspose)

 conv2d_transpose_1 (Conv2D  (None, 32, 8, 16)         7696
 Transpose)

 conv2d_transpose_2 (Conv2D  (None, 64, 16, 8)         1928
 Transpose)

 conv2d_transpose_3 (Conv2D  (None, 128, 32, 1)        281
 Transpose)
```

```
=================================================================
Total params: 131577 (513.97 KB)
Trainable params: 131577 (513.97 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
import visualkeras
from PIL import ImageFont

font = ImageFont.load_default()
visualkeras.layered_view(autoencoder,legend=True, font=font)
```

[ ]:



1.2 Data Loading

[ ]:
```
from sklearn.model_selection import train_test_split

# Access your file using the path
data = np.load('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪Preprocessed/Train_Test_Scipy.npy')
# Split the data into training and testing sets
train_data, test_data = train_test_split(data, test_size=0.2, random_state=46,␣
↪shuffle=True)
print ('size-data='+str (data.shape), 'size-train='+str (train_data.
↪shape),'size-test='+str (test_data.shape) )
```

```
size-data=(2329, 128, 32, 1) size-train=(1863, 128, 32, 1) size-test=(466, 128,
32, 1)
```

1.3 Training

[ ]:
```
lr_schedule = schedules.ExponentialDecay(
    initial_learning_rate= 0.001,
    decay_steps=1000,
    decay_rate=0.8)

### Adapting the learning rate of the optimizer using an exponential decay␣
↪schedule

optimizer = Adam(learning_rate=lr_schedule)

es = EarlyStopping( monitor='val_loss', mode='min', verbose=1, patience=30),␣
↪CSVLogger('Pretrain_log.csv')

autoencoder.compile(optimizer=optimizer, loss='mse')
autoencoder.fit(train_data, train_data, batch_size=32, epochs=1000␣
↪,validation_data=(test_data, test_data), callbacks=[es])
```

```
Epoch 1/1000

2024-01-27 17:19:51.304071: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.

59/59 [==============================] - 5s 25ms/step - loss: 48.5460 -
val_loss: 32.9689
Epoch 2/1000
59/59 [==============================] - 1s 10ms/step - loss: 35.2235 -
val_loss: 26.5075
Epoch 3/1000
59/59 [==============================] - 1s 9ms/step - loss: 30.5754 - val_loss:
25.6012
Epoch 4/1000
59/59 [==============================] - 1s 9ms/step - loss: 28.8158 - val_loss:
24.8975
Epoch 5/1000
59/59 [==============================] - 1s 9ms/step - loss: 27.2092 - val_loss:
24.0504
Epoch 6/1000
59/59 [==============================] - 1s 9ms/step - loss: 26.7016 - val_loss:
23.7613
Epoch 7/1000
59/59 [==============================] - 0s 8ms/step - loss: 25.8492 - val_loss:
23.4665
Epoch 8/1000
59/59 [==============================] - 1s 9ms/step - loss: 25.1013 - val_loss:
23.3054
Epoch 9/1000
```

```
59/59 [==============================] - 1s 8ms/step - loss: 24.7146 - val_loss:
23.0975
Epoch 10/1000
59/59 [==============================] - 1s 9ms/step - loss: 24.6291 - val_loss:
22.9684
Epoch 11/1000
59/59 [==============================] - 1s 9ms/step - loss: 24.1388 - val_loss:
23.2566
Epoch 12/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.9749 - val_loss:
22.8803
Epoch 13/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.6849 - val_loss:
23.6194
Epoch 14/1000
59/59 [==============================] - 1s 9ms/step - loss: 24.4928 - val_loss:
22.9626
Epoch 15/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.4650 - val_loss:
22.8031
Epoch 16/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.2709 - val_loss:
22.4994
Epoch 17/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.9409 - val_loss:
22.6554
Epoch 18/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.2073 - val_loss:
22.8168
Epoch 19/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.1757 - val_loss:
22.5054
Epoch 20/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.0666 - val_loss:
22.5274
Epoch 21/1000
59/59 [==============================] - 1s 9ms/step - loss: 23.2894 - val_loss:
22.4981
Epoch 22/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.7977 - val_loss:
22.3926
Epoch 23/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.6405 - val_loss:
22.5897
Epoch 24/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.6271 - val_loss:
22.3539
Epoch 25/1000
```

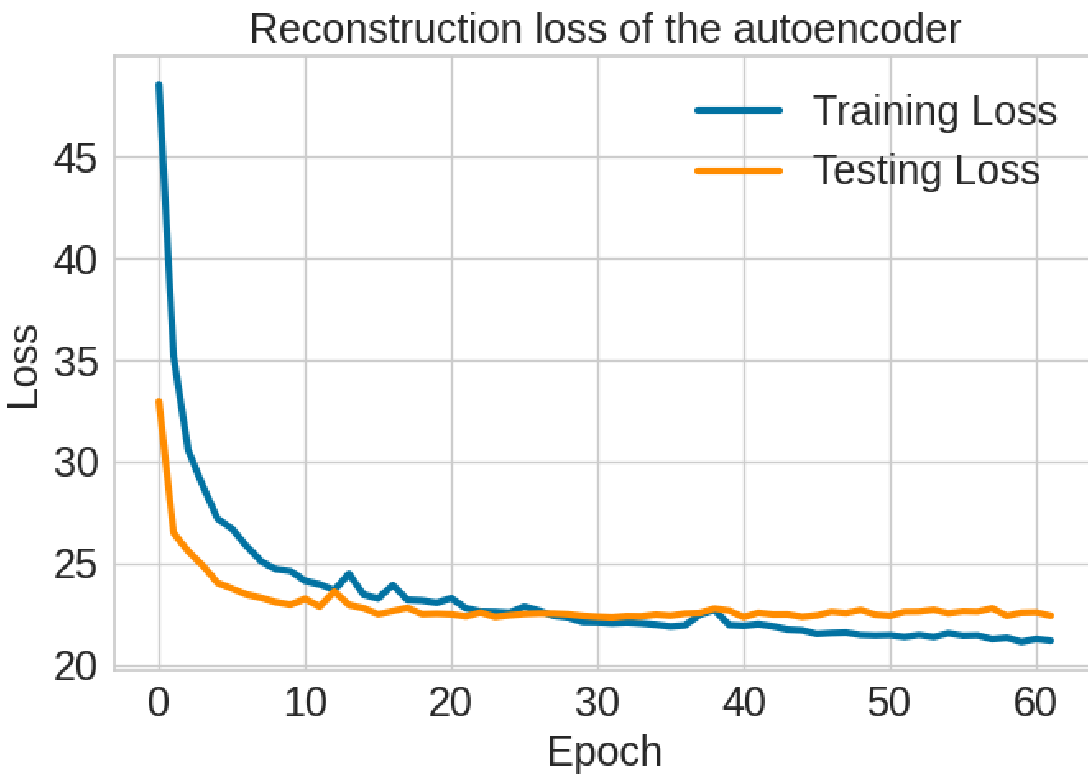
```
59/59 [==============================] - 1s 9ms/step - loss: 22.5850 - val_loss:
22.4483
Epoch 26/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.8854 - val_loss:
22.5124
Epoch 27/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.6811 - val_loss:
22.5440
Epoch 28/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.4173 - val_loss:
22.5390
Epoch 29/1000
59/59 [==============================] - 1s 10ms/step - loss: 22.3322 -
val_loss: 22.5007
Epoch 30/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.1101 - val_loss:
22.4147
Epoch 31/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.1020 - val_loss:
22.3641
Epoch 32/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.0598 - val_loss:
22.3292
Epoch 33/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.0960 - val_loss:
22.3997
Epoch 34/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.0535 - val_loss:
22.3901
Epoch 35/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.9877 - val_loss:
22.4867
Epoch 36/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.8994 - val_loss:
22.4287
Epoch 37/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.9546 - val_loss:
22.5372
Epoch 38/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.4764 - val_loss:
22.5695
Epoch 39/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.6973 - val_loss:
22.7785
Epoch 40/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.9706 - val_loss:
22.6770
Epoch 41/1000
```

```
59/59 [==============================] - 1s 9ms/step - loss: 21.9325 - val_loss:
22.3548
Epoch 42/1000
59/59 [==============================] - 1s 9ms/step - loss: 22.0170 - val_loss:
22.5773
Epoch 43/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.9061 - val_loss:
22.4891
Epoch 44/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.7553 - val_loss:
22.4925
Epoch 45/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.7165 - val_loss:
22.3626
Epoch 46/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.5307 - val_loss:
22.4403
Epoch 47/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.5769 - val_loss:
22.6238
Epoch 48/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.6012 - val_loss:
22.5568
Epoch 49/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.4814 - val_loss:
22.7086
Epoch 50/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.4572 - val_loss:
22.4817
Epoch 51/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.4752 - val_loss:
22.4208
Epoch 52/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.3886 - val_loss:
22.6213
Epoch 53/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.4870 - val_loss:
22.6295
Epoch 54/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.3846 - val_loss:
22.7185
Epoch 55/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.5650 - val_loss:
22.5487
Epoch 56/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.4471 - val_loss:
22.6424
Epoch 57/1000
```

```
59/59 [==============================] - 1s 9ms/step - loss: 21.4645 - val_loss:
22.6255
Epoch 58/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.2753 - val_loss:
22.7963
Epoch 59/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.3472 - val_loss:
22.4211
Epoch 60/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.1375 - val_loss:
22.5798
Epoch 61/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.2838 - val_loss:
22.5963
Epoch 62/1000
59/59 [==============================] - 1s 9ms/step - loss: 21.1925 - val_loss:
22.4208
Epoch 62: early stopping
```

[ ]:
```
<keras.src.callbacks.History at 0x7f68102549d0>
```

[ ]:
```python
import pandas as pd
df = pd.read_csv('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
 ↪notebook/Pretrain_log.csv')
fig= plt.figure(figsize=(7, 5))
plt.plot(df['epoch'],df['loss'], color='b',label='Training Loss', linewidth=3.0)
plt.plot(df['epoch'],df['val_loss'], color='darkorange',label='Testing Loss',␣
 ↪linewidth=3.0)

plt.ylabel('Loss', fontsize= 18)
plt.xlabel('Epoch', fontsize= 18)
plt.title('Reconstruction loss of the autoencoder', fontsize= 18)
plt.yticks (fontsize= 18)
plt.xticks (fontsize= 18)

plt.legend(loc= 1, frameon= False, fontsize= 18)
plt.tight_layout()
plt.show ()
```



[ ]:
```
### Save the model
from tensorflow.keras.models import save_model
autoencoder.save ('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/Models/
↪autoencoder-model2')
```

1.3.1 Kmeans clustering based on extracted features from the autoencoder

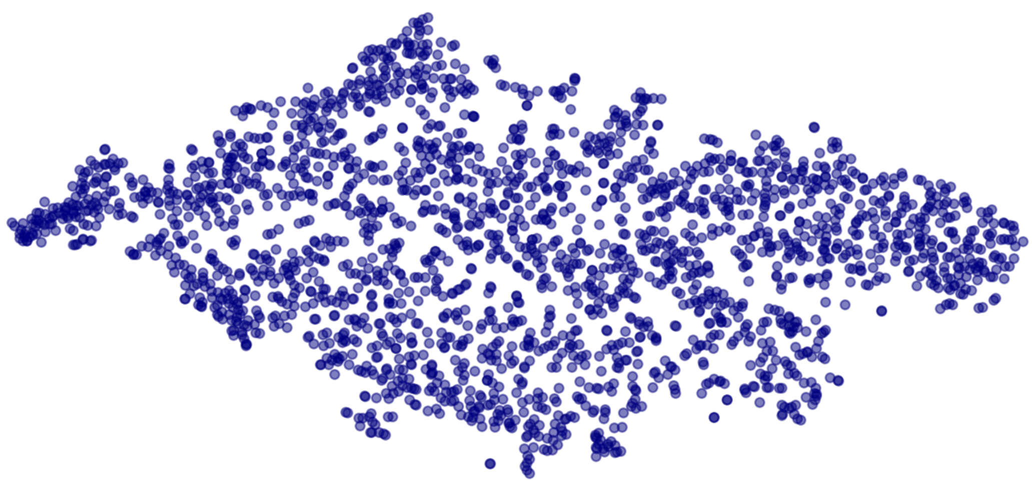
[ ]:
```
from tensorflow.keras.models import load_model

autoencoder = load_model('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
↪Models/autoencoder-model2')
autoencoder_input = autoencoder.input
encoder_output = autoencoder.get_layer('dense_1').output
encoder = Model(inputs=autoencoder_input, outputs=encoder_output)
```

[ ]:
```
kmeans = KMeans(n_clusters=1, random_state=46, n_init=20).fit(encoder.
↪predict(data))
y = kmeans.predict(encoder.predict(data))

def plotter(S, y):
    '''
    function to visualize the outputs of t-SNE
```

```
    '''

    lw = 2
    # create a scatter plot.
    f = plt.figure(figsize=(22, 10))
    ax = f.add_subplot(111)
    plt.scatter(S[y == 0, 0], S[y == 0, 1],color='navy', alpha=.5, lw=lw, s=100)
    ax.axis('off')
    ax.axis('tight')
    plt.show()

    return f, ax

enc = encoder.predict(data)
from sklearn.manifold import TSNE
redu = TSNE(random_state=123).fit_transform(enc)
plotter(redu, y)
```

```
2024-01-27 17:01:40.909631: I
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:454] Loaded cuDNN
version 8904
```

```
73/73 [==============================] - 1s 4ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
```



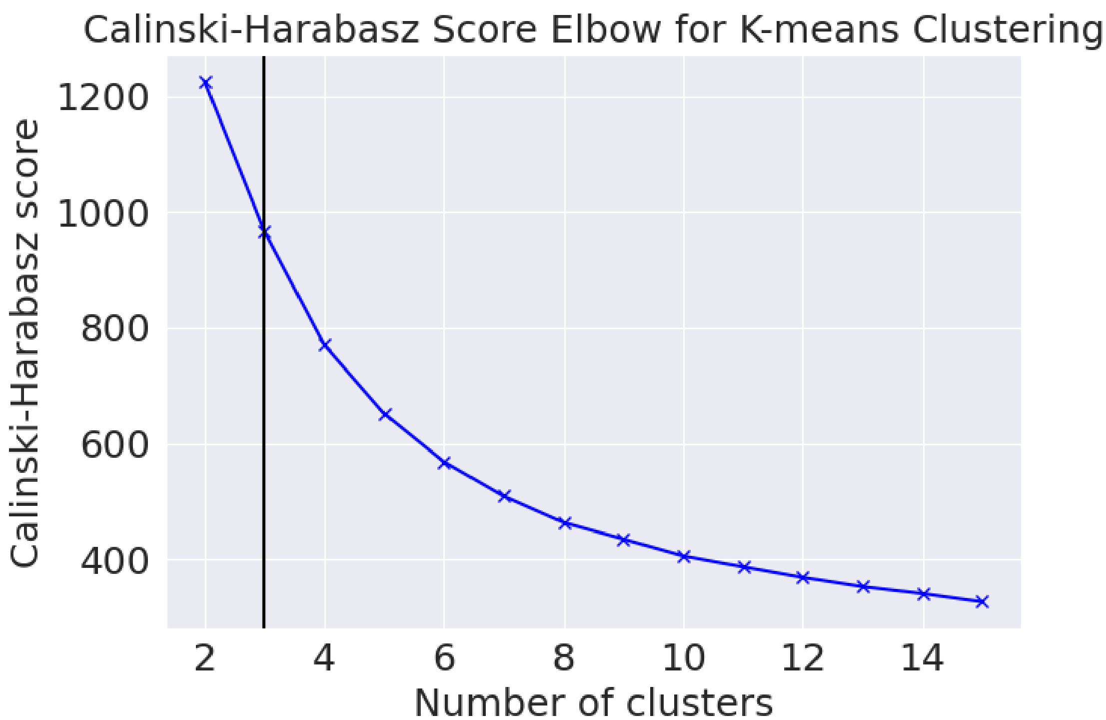
[ ]:
```
(<Figure size 2200x1000 with 1 Axes>, <Axes: >)
```

Determining optimal number of clusters

```
[ ]: from sklearn.metrics import calinski_harabasz_score
cal = []
K = range(2,16)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=20).fit(encoder.predict(data))
    labelskm = kmeans.predict(encoder.predict(data))
    cal.append(calinski_harabasz_score(encoder.predict(data), labelskm))
fig= plt.figure(figsize=(7, 5))
plt.plot(K, cal, 'bx-')
plt.xlabel('Number of clusters',fontsize= 18)
plt.ylabel('Calinski-Harabasz score',fontsize= 18)
plt.title('Calinski-Harabasz Score Elbow for K-means Clustering',fontsize= 18)
plt.yticks (fontsize= 18)
plt.xticks (fontsize= 18)
plt.axvline(x = 3, color = 'black')
plt.tight_layout()
plt.show()
```

```
73/73 [==============================] - 0s 2ms/step

73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
```

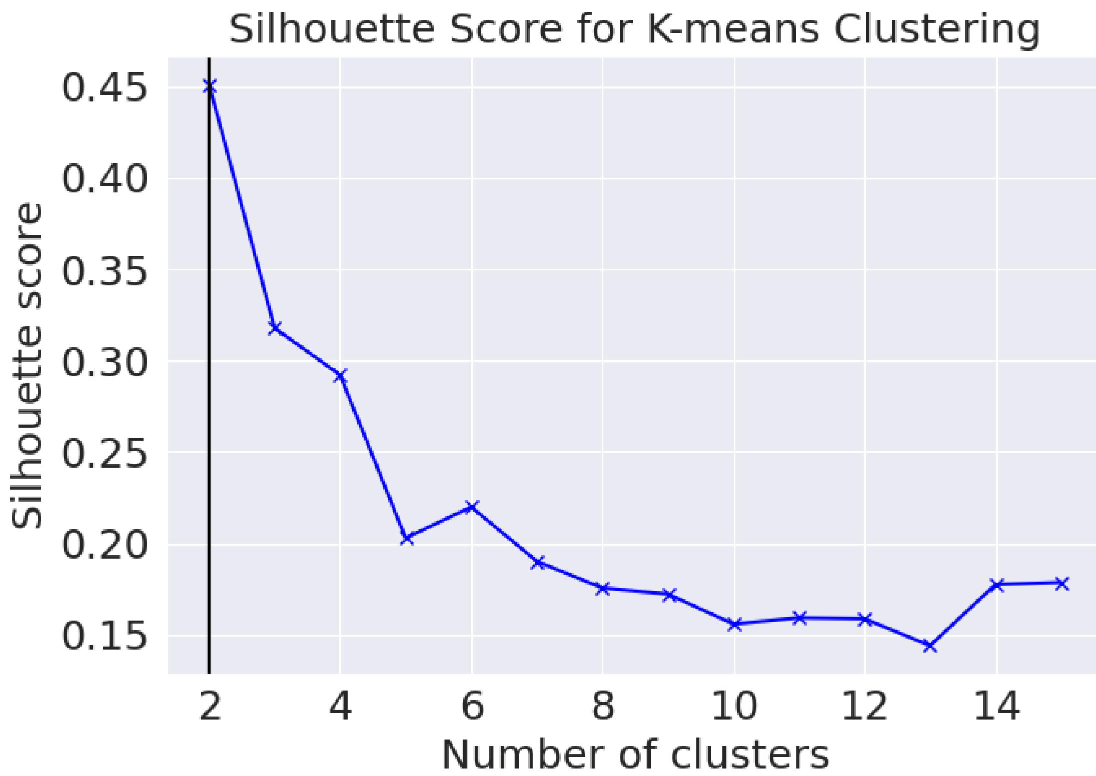
```
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
```



[ ]:
```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

silhouette_scores = []
K = range(2, 16)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=20).fit(encoder.
 ↪predict(data))
```

```python
    labels = kmeans.labels_
    silhouette_scores.append(silhouette_score(encoder.predict(data), labels))

# Plotting the silhouette scores
fig = plt.figure(figsize=(7, 5))
plt.plot(K, silhouette_scores, 'bx-')
plt.xlabel('Number of clusters', fontsize=18)
plt.ylabel('Silhouette score', fontsize=18)
plt.title('Silhouette Score for K-means Clustering', fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.axvline(x=silhouette_scores.index(max(silhouette_scores)) + 2, color='black')  # +2 because range starts at 2
plt.tight_layout()
plt.show()
```

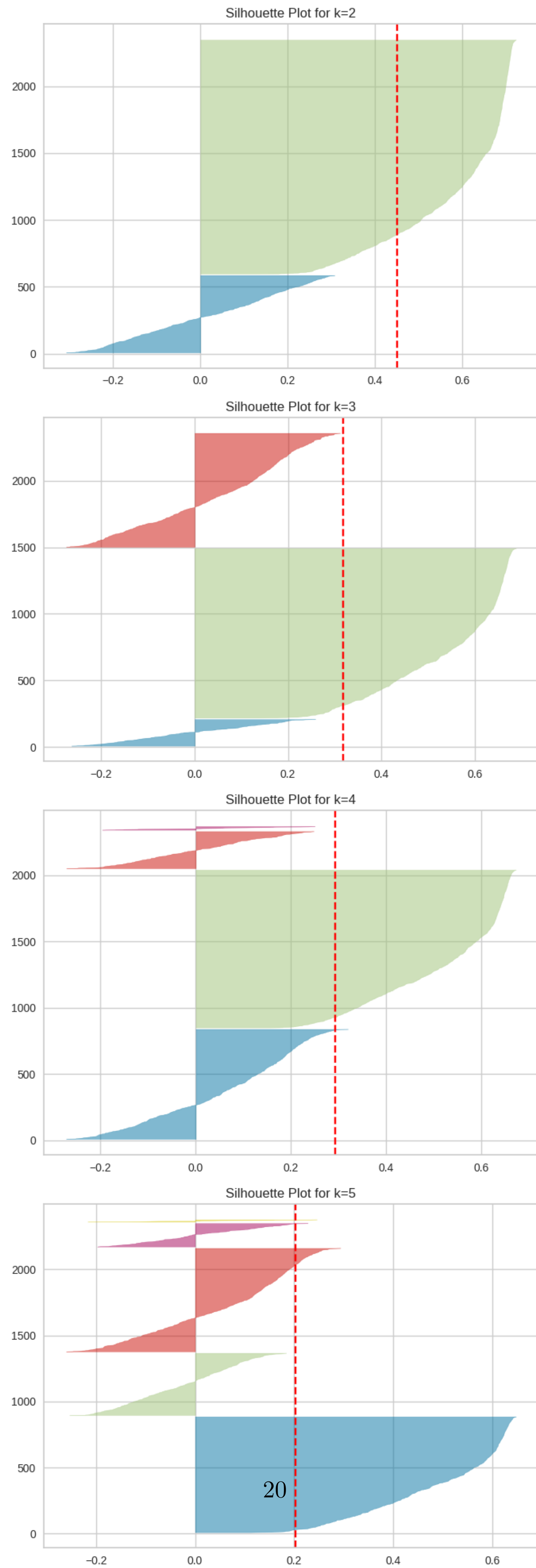
```
73/73 [==============================] - 0s 2ms/step

73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
```

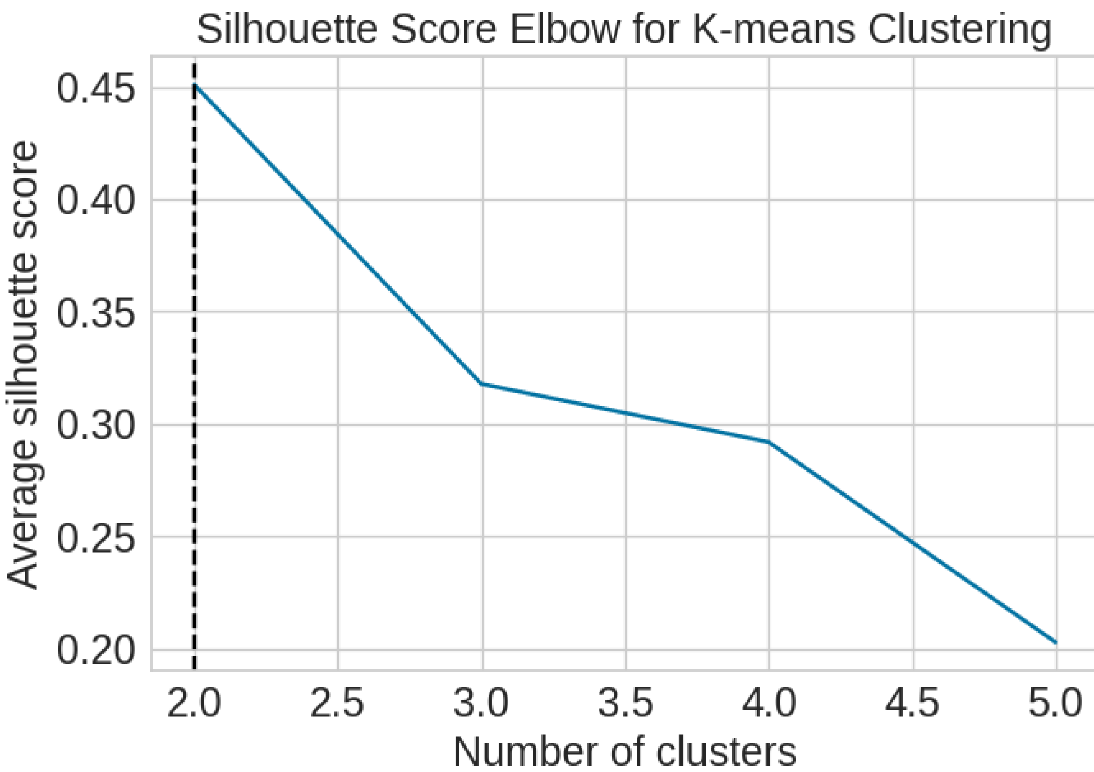


[ ]:
```python
from tensorflow.keras.models import load_model
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from yellowbrick.cluster import SilhouetteVisualizer
import matplotlib.pyplot as plt

# Calculate the encoded features only once
encoded_data = encoder.predict(data)

# Generate encoded data
encoded_data = encoder.predict(data)

# Range of clusters to evaluate
K = range(2, 6)
silhouette_avg_scores = []  # To store the average silhouette scores

print("Silhouette scores for different numbers of clusters:")
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=20).fit(encoded_data)
    labels = kmeans.labels_
    score = silhouette_score(encoded_data, labels)
    silhouette_scores.append(score)
```

```
    print(f"Clusters: {k}, Silhouette Score: {score:.4f}")

# Create subplots for silhouette visualizers
fig, axes = plt.subplots(len(K), 1, figsize=(7, 5*len(K)))

# Loop over the range of cluster numbers to fit KMeans and calculate silhouette␣
↪scores
for idx, k in enumerate(K):
    # Fit KMeans and predict the cluster labels
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=20)
    cluster_labels = kmeans.fit_predict(encoded_data)

    # Calculate the average silhouette score and append to the list
    silhouette_avg = silhouette_score(encoded_data, cluster_labels)
    silhouette_avg_scores.append(silhouette_avg)

    # Initialize the SilhouetteVisualizer with the KMeans model
    visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick',␣
↪ax=axes[idx])
    visualizer.fit(encoded_data)
    axes[idx].set_title(f'Silhouette Plot for k={k}', fontsize=12)

# Adjust the layout
plt.tight_layout()

# Plot the average silhouette scores
fig, ax = plt.subplots(figsize=(7, 5))
ax.plot(K, silhouette_avg_scores, 'bx-')
ax.set_xlabel('Number of clusters', fontsize=18)
ax.set_ylabel('Average silhouette score', fontsize=18)
ax.set_title('Silhouette Score Elbow for K-means Clustering', fontsize=18)
ax.axvline(x=K[silhouette_avg_scores.index(max(silhouette_avg_scores))],␣
↪color='black', linestyle='--')
ax.tick_params(axis='x', labelsize=18)  # Corrected line for setting x-tick␣
↪label sizes
ax.tick_params(axis='y', labelsize=18)  # Corrected line for setting y-tick␣
↪label sizes
plt.tight_layout()
plt.show()
```

```
73/73 [==============================] - 0s 2ms/step

73/73 [==============================] - 0s 2ms/step
Silhouette scores for different numbers of clusters:
Clusters: 2, Silhouette Score: 0.4509
Clusters: 3, Silhouette Score: 0.3178
Clusters: 4, Silhouette Score: 0.2920
Clusters: 5, Silhouette Score: 0.2029
```


Silhouette Plot for k=2
Silhouette Plot for k=3
Silhouette Plot for k=4
Silhouette Plot for k=5
0
500
1000
1500
2000
−0.2
0.0
0.2
0.4
0.6

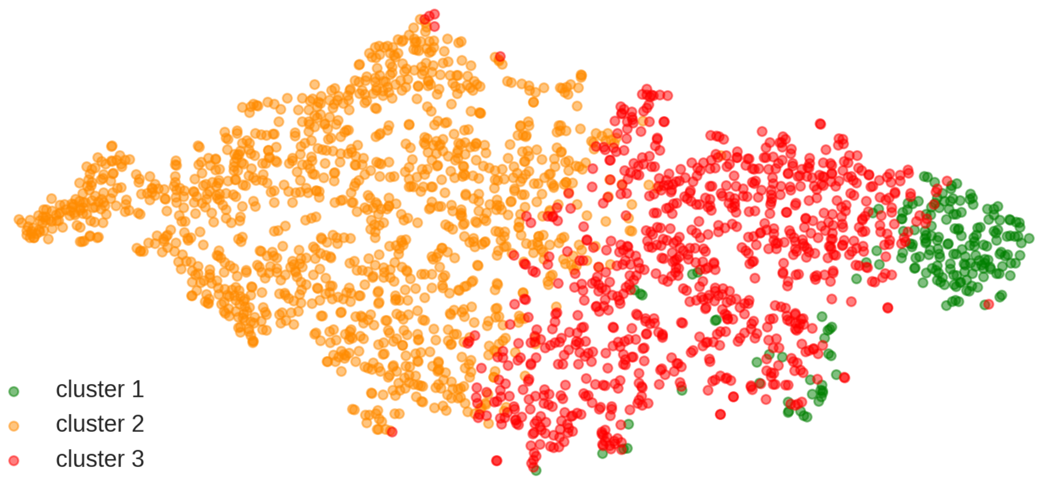


1.3.2 T-sne visualizations of seismic event clusters in feature domain after pretraining using K-Means

[ ]:
```
kmeans = KMeans(n_clusters=3, random_state=46, n_init=20).fit(encoder.
 ↪predict(data))
y = kmeans.predict(encoder.predict(data))

def plotter(S, y, target_names):
    '''
    function to visualize the outputs of t-SNE
    '''
    # choose a color palette with seaborn.
    colors = ['green','darkorange','red']

    lw = 2
    # create a scatter plot.
    f = plt.figure(figsize=(22, 10))
    ax = f.add_subplot(111)
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):
```

```
        plt.scatter(S[y == i, 0], S[y == i, 1], color=color, alpha=.5, lw=lw,␣
 ↪s=100, label=target_name)
    plt.legend(loc='lower left', shadow=False, scatterpoints=1, prop={'size':␣
 ↪26})
    ax.axis('off')
    ax.axis('tight')
    plt.show()

    return f, ax

enc = encoder.predict(data)
from sklearn.manifold import TSNE
redu = TSNE(random_state=123).fit_transform(enc)
target_names = ['cluster 1', 'cluster 2', 'cluster 3']

# Print the cluster counts
(unique, counts) = np.unique(y, return_counts=True)
cluster_counts = dict(zip(unique, counts))
print("Cluster counts:", cluster_counts)

plotter(redu, y, target_names)
```

```
27/73 [==========>…] - ETA: 0s

73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
73/73 [==============================] - 0s 2ms/step
Cluster counts: {0: 200, 1: 1273, 2: 856}
```




```
[ ]: (<Figure size 2200x1000 with 1 Axes>, <Axes: >)
```

```
[ ]: ### Save the labels
np.savetxt('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
  ↪Km-n3-ft_Pretrain.txt', y, fmt='%i', delimiter=',')
```

```
[ ]: # Path to the file
file_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
  ↪Km-n3-ft_Pretrain.txt'

# Initialize a dictionary to count cluster occurrences
cluster_counts = {}

# Read the file line by line
with open(file_path, 'r') as file:
    for line in file:
        cluster = line.strip()
        if cluster in cluster_counts:
            cluster_counts[cluster] += 1
        else:
            cluster_counts[cluster] = 1

# Print the counts for each cluster
for cluster, count in cluster_counts.items():
    print(f"Cluster {cluster}: {count} occurrences")
```

```
Cluster 1: 1273 occurrences
Cluster 2: 856 occurrences
Cluster 0: 200 occurrences
```

1.3.3 Loading the Pre-trained Model

```
[ ]: from tensorflow.keras.models import load_model
autoencoder = load_model("/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
  ↪Models/autoencoder-model2")
```

```
[ ]: import keras.backend as K
get_all_layer_outputs = K.function([autoencoder.layers[0].input],
                                  [l.output for l in autoencoder.layers[1:]])

layer_output = get_all_layer_outputs([data]) # return the same thing
```

```
[ ]: n_clusters=3
```

1.3.4 Integrating clustering layer into autoencoder bottelneck

```
[ ]: from numpy.random import seed
sd=46
seed(sd)
import tensorflow
```

```python
import tensorflow as tf
tensorflow.random.set_seed(sd)

initializer = tf.keras.initializers.GlorotUniform(seed=sd)

#### clustering layers
class ClusteringLayer(Layer):
    def __init__(self, n_clusters, weights=None, alpha=1, **kwargs):
        if 'input_shape' not in kwargs and 'input_dim' in kwargs:
            kwargs['input_shape'] = (kwargs.pop('input_dim'),)
        super(ClusteringLayer, self).__init__(**kwargs)
        self.n_clusters = n_clusters
        self.alpha = alpha
        self.initial_weights = weights
        self.input_spec = InputSpec(ndim=2)

    def build(self, input_shape):
        assert len(input_shape) == 2
        input_dim = input_shape[1]
        self.input_spec = InputSpec(dtype=K.floatx(), shape=(None, input_dim))
        self.clusters = self.add_weight(shape=(self.n_clusters, input_dim),␣
 ↪initializer= initializer , name='clusters')
        if self.initial_weights is not None:
            self.set_weights(self.initial_weights)
            del self.initial_weights

    def call(self, inputs, **kwargs):
        q = 1.0 / (1.0 + (K.sum(K.square(K.expand_dims(inputs, axis=1) - self.
 ↪clusters), axis=2) / self.alpha))
        q **= (self.alpha + 1.0) / 2.0
        q = K.transpose(K.transpose(q) / K.sum(q, axis=1))
        return q

    def compute_output_shape(self, input_shape):
        assert input_shape and len(input_shape) == 2
        return input_shape[0], self.n_clusters

    def get_config(self):
        config = {'n_clusters': self.n_clusters}
        base_config = super(ClusteringLayer, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))


print('...Finetuning...')
clustering_layer = ClusteringLayer(n_clusters, name='clustering')(autoencoder.
 ↪layers[6].output)
model = Model(inputs=autoencoder.layers[0].output, outputs=clustering_layer)
```

```
model.compile(loss='kld', loss_weights=0.1,  optimizer=SGD(learning_rate=0.01, ␣
↪momentum=0.9))
```

…Finetuning…

```
[ ]: ### Initializing the weights using Kmean and assigning them to the model

kmeans = KMeans(n_clusters=n_clusters,  random_state=46, n_init=20)
y_pred = kmeans.fit_predict(layer_output[5])
y_pred_last = np.copy(y_pred)
model.get_layer(name='clustering').set_weights([kmeans.cluster_centers_])
```

1.4 Finetuning Pre-Trained Model Parameters

```
[ ]: ## parameters for the finetuning
x = data
batch_size=32
tol = 0.0001 # tolerance threshold to stop training
loss = 0
index = 0
maxiter = 100000
update_interval = 500
index_array = np.arange(x.shape[0])

################################################################################
### simultaneous optimization and clustering
def target_distribution(q):
    weight = q ** 2 / q.sum(0)
    return (weight.T / weight.sum(1)).T

for ite in range(int(maxiter)):
    if ite % update_interval == 0:
        q = model.predict(data, verbose=0)
        p = target_distribution(q)  # update the auxiliary target distribution p
        y_pred = q.argmax(1) # evaluate the clustering performance

        loss = np.round(loss, 5)
        print('Iter %d: ' % (ite), ' ; loss=', loss)

        # check stop criterion
        delta_label = np.sum(y_pred != y_pred_last).astype(np.float32) / y_pred.
↪shape[0]
        y_pred_last = np.copy(y_pred)

        if ite > 0 and delta_label < tol:
            print('delta_label ', delta_label, '< tol ', tol)
            break
```

```
        IN = layer_output[5]

    idx = index_array[index * batch_size: min((index+1) * batch_size, data.
↪shape[0])]
    loss = model.train_on_batch(x=x[idx], y=p[idx])
    index = index + 1 if (index + 1) * batch_size <= x.shape[0] else 0
```

```
Iter 0:   ; loss= 0
```

```
2024-01-27 17:05:26.013225: I external/local_xla/xla/service/service.cc:168] XLA
service 0x7f67e0b49c00 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
2024-01-27 17:05:26.013256: I external/local_xla/xla/service/service.cc:176]
StreamExecutor device (0): NVIDIA GeForce RTX 2080 SUPER, Compute Capability 7.5
2024-01-27 17:05:26.013262: I external/local_xla/xla/service/service.cc:176]
StreamExecutor device (1): NVIDIA GeForce RTX 2080 SUPER, Compute Capability 7.5
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1706349926.114602 2952019 device_compiler.h:186] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.
```

```
Iter 500:   ; loss= 0.00473
Iter 1000:   ; loss= 0.00479
Iter 1500:   ; loss= 0.00549
Iter 2000:   ; loss= 0.00921
Iter 2500:   ; loss= 0.00472
Iter 3000:   ; loss= 0.00586
Iter 3500:   ; loss= 0.00427
Iter 4000:   ; loss= 0.01082
Iter 4500:   ; loss= 0.00371
Iter 5000:   ; loss= 0.00701
Iter 5500:   ; loss= 0.00833
Iter 6000:   ; loss= 0.00449
Iter 6500:   ; loss= 0.00858
Iter 7000:   ; loss= 0.00721
Iter 7500:   ; loss= 0.012
Iter 8000:   ; loss= 0.00478
Iter 8500:   ; loss= 0.00804
Iter 9000:   ; loss= 0.01666
Iter 9500:   ; loss= 0.00602
Iter 10000:   ; loss= 0.00418
Iter 10500:   ; loss= 0.01335
Iter 11000:   ; loss= 0.00963
Iter 11500:   ; loss= 0.00804
Iter 12000:   ; loss= 0.00799
Iter 12500:   ; loss= 0.00456
Iter 13000:   ; loss= 0.00605
Iter 13500:   ; loss= 0.00519
```

```
Iter 14000:   ; loss= 0.00591
Iter 14500:   ; loss= 0.00433
Iter 15000:   ; loss= 0.01088
Iter 15500:   ; loss= 0.00685
Iter 16000:   ; loss= 0.00619
Iter 16500:   ; loss= 0.0062
Iter 17000:   ; loss= 0.01172
Iter 17500:   ; loss= 0.00599
Iter 18000:   ; loss= 0.00594
Iter 18500:   ; loss= 0.00733
Iter 19000:   ; loss= 0.00569
Iter 19500:   ; loss= 0.0063
Iter 20000:   ; loss= 0.00482
Iter 20500:   ; loss= 0.0109
Iter 21000:   ; loss= 0.00592
Iter 21500:   ; loss= 0.01062
Iter 22000:   ; loss= 0.00432
Iter 22500:   ; loss= 0.00611
Iter 23000:   ; loss= 0.00635
Iter 23500:   ; loss= 0.00374
Iter 24000:   ; loss= 0.0108
Iter 24500:   ; loss= 0.00374
Iter 25000:   ; loss= 0.01116
Iter 25500:   ; loss= 0.00897
Iter 26000:   ; loss= 0.00599
Iter 26500:   ; loss= 0.00561
Iter 27000:   ; loss= 0.0085
Iter 27500:   ; loss= 0.00544
Iter 28000:   ; loss= 0.00346
Iter 28500:   ; loss= 0.00691
Iter 29000:   ; loss= 0.00688
Iter 29500:   ; loss= 0.00667
Iter 30000:   ; loss= 0.00311
Iter 30500:   ; loss= 0.0077
Iter 31000:   ; loss= 0.00347
Iter 31500:   ; loss= 0.00847
Iter 32000:   ; loss= 0.00646
Iter 32500:   ; loss= 0.00332
Iter 33000:   ; loss= 0.00797
Iter 33500:   ; loss= 0.00899
Iter 34000:   ; loss= 0.01172
Iter 34500:   ; loss= 0.00376
Iter 35000:   ; loss= 0.00677
Iter 35500:   ; loss= 0.0042
Iter 36000:   ; loss= 0.00504
Iter 36500:   ; loss= 0.00385
Iter 37000:   ; loss= 0.00572
Iter 37500:   ; loss= 0.00581
```

```
Iter 38000:   ; loss= 0.00495
Iter 38500:   ; loss= 0.00803
Iter 39000:   ; loss= 0.00371
Iter 39500:   ; loss= 0.00861
Iter 40000:   ; loss= 0.00351
Iter 40500:   ; loss= 0.01012
Iter 41000:   ; loss= 0.0063
Iter 41500:   ; loss= 0.00885
Iter 42000:   ; loss= 0.009
Iter 42500:   ; loss= 0.00397
Iter 43000:   ; loss= 0.00909
Iter 43500:   ; loss= 0.00851
Iter 44000:   ; loss= 0.01053
Iter 44500:   ; loss= 0.00335
Iter 45000:   ; loss= 0.00719
Iter 45500:   ; loss= 0.01257
Iter 46000:   ; loss= 0.00379
Iter 46500:   ; loss= 0.00362
Iter 47000:   ; loss= 0.01061
Iter 47500:   ; loss= 0.00729
Iter 48000:   ; loss= 0.00619
Iter 48500:   ; loss= 0.0079
Iter 49000:   ; loss= 0.00547
Iter 49500:   ; loss= 0.00688
Iter 50000:   ; loss= 0.00424
Iter 50500:   ; loss= 0.00789
Iter 51000:   ; loss= 0.00387
Iter 51500:   ; loss= 0.00599
Iter 52000:   ; loss= 0.00661
Iter 52500:   ; loss= 0.00563
Iter 53000:   ; loss= 0.00675
Iter 53500:   ; loss= 0.00837
Iter 54000:   ; loss= 0.00416
Iter 54500:   ; loss= 0.00425
Iter 55000:   ; loss= 0.00571
Iter 55500:   ; loss= 0.006
Iter 56000:   ; loss= 0.00518
Iter 56500:   ; loss= 0.00354
Iter 57000:   ; loss= 0.0109
Iter 57500:   ; loss= 0.00608
Iter 58000:   ; loss= 0.00772
Iter 58500:   ; loss= 0.0053
Iter 59000:   ; loss= 0.0044
Iter 59500:   ; loss= 0.00532
Iter 60000:   ; loss= 0.00449
Iter 60500:   ; loss= 0.00813
Iter 61000:   ; loss= 0.004
Iter 61500:   ; loss= 0.00716
```

```
Iter 62000:   ; loss= 0.007
Iter 62500:   ; loss= 0.00297
Iter 63000:   ; loss= 0.00744
Iter 63500:   ; loss= 0.00763
Iter 64000:   ; loss= 0.0041
Iter 64500:   ; loss= 0.00454
Iter 65000:   ; loss= 0.00647
Iter 65500:   ; loss= 0.00406
Iter 66000:   ; loss= 0.00523
Iter 66500:   ; loss= 0.00201
Iter 67000:   ; loss= 0.0068
Iter 67500:   ; loss= 0.00301
Iter 68000:   ; loss= 0.00666
Iter 68500:   ; loss= 0.00723
Iter 69000:   ; loss= 0.00123
Iter 69500:   ; loss= 0.00691
Iter 70000:   ; loss= 0.00848
Iter 70500:   ; loss= 0.00716
Iter 71000:   ; loss= 0.00263
Iter 71500:   ; loss= 0.00658
Iter 72000:   ; loss= 0.00429
Iter 72500:   ; loss= 0.00326
Iter 73000:   ; loss= 0.00284
Iter 73500:   ; loss= 0.00745
Iter 74000:   ; loss= 0.00577
Iter 74500:   ; loss= 0.00489
Iter 75000:   ; loss= 0.00617
delta_label  0.0 < tol  0.0001
```

[ ]:
```
import keras.backend as K
get_all_layer_outputs = K.function([autoencoder.layers[0].input],
                                  [l.output for l in autoencoder.layers[1:]])

layer_output = get_all_layer_outputs([x]) # return the same thing
```

[ ]:
```
autoencoder.save('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/Models/
 ↪autoencoder-model2_finetuned')
```

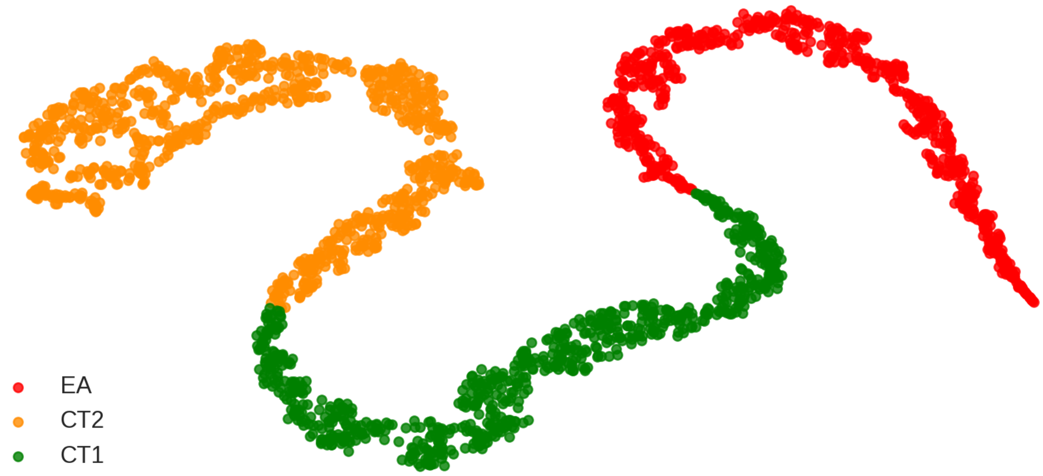
```
INFO:tensorflow:Assets written to:
/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/Models/autoencoder-
model2_finetuned/assets
```


```
INFO:tensorflow:Assets written to:
/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/Models/autoencoder-
model2_finetuned/assets
```

1.4.1 T-sne visualizations of seismic event clusters in feature domain after finetuning

[ ]:
```
y=y_pred

def plotter(S, y, target_names):
    '''
    function to visualize the outputs of t-SNE
    '''
    # choose a color palette with seaborn.
    colors = ['red','darkorange','green']

    lw = 2
    # create a scatter plot.
    f = plt.figure(figsize=(22, 10))
    ax = f.add_subplot(111)
    for color, i, target_name in zip(colors, range(len(target_names)),␣
 ↪target_names):
        plt.scatter(S[y == i, 0], S[y == i, 1], color=color, alpha=.8, lw=lw,␣
 ↪s=100, label=target_name)
    plt.legend(loc='lower left', shadow=False, scatterpoints=1, prop={'size':␣
 ↪26})
    ax.axis('off')
    ax.axis('tight')
    plt.show()

    return f, ax

enc = layer_output[5]
from sklearn.manifold import TSNE
redu = TSNE(random_state=123).fit_transform(enc)
target_names = ['EA', 'CT2', 'CT1']

# Print the cluster counts
(unique, counts) = np.unique(y, return_counts=True)
cluster_counts = dict(zip(unique, counts))
print("Cluster counts:", cluster_counts)

plotter(redu, y, target_names)
```

```
Cluster counts: {0: 642, 1: 916, 2: 771}
```



[ ]: (<Figure size 2200x1000 with 1 Axes>, <Axes: >)

1.4.2 Rewriting Cluser Result for Temporal Analysis

[ ]:
```
### Save the labels
np.savetxt('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/Km-n3-ft.
↪txt', y, fmt='%i', delimiter=',')
```

[ ]:
```
# Path to the file
file_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪Km-n3-ft.txt'

# Initialize a dictionary to count cluster occurrences
cluster_counts = {}

# Read the file line by line
with open(file_path, 'r') as file:
    for line in file:
        cluster = line.strip()
        if cluster in cluster_counts:
            cluster_counts[cluster] += 1
        else:
            cluster_counts[cluster] = 1

# Print the counts for each cluster
for cluster, count in cluster_counts.items():
    print(f"Cluster {cluster}: {count} occurrences")
```

```
Cluster 2: 771 occurrences
Cluster 0: 642 occurrences
```

```
Cluster 1: 916 occurrences
```

[ ]:
```
# Change the order of the cluster numbers (just for a nice representation)

with open('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/Km-n3-ft.
 ↪txt', 'r') as file :
  filedata = file.read()

# Replace the target string
filedata = filedata.replace('2', 'x')
filedata = filedata.replace('0', '2')
filedata = filedata.replace('1', '0')
filedata = filedata.replace('x', '1')

# Re-write the output
with open('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
 ↪Km-n3-ft_Rewritten.txt', 'w') as file:
  file.write(filedata)
```

[ ]:
```
# Path to the file
file_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
 ↪Km-n3-ft_Rewritten.txt'

# Initialize a dictionary to count cluster occurrences
cluster_counts = {}

# Read the file line by line
with open(file_path, 'r') as file:
    for line in file:
        cluster = line.strip()
        if cluster in cluster_counts:
            cluster_counts[cluster] += 1
        else:
            cluster_counts[cluster] = 1

# Print the counts for each cluster
for cluster, count in cluster_counts.items():
    print(f"Cluster {cluster}: {count} occurrences")
```

```
Cluster 1: 771 occurrences
Cluster 2: 642 occurrences
Cluster 0: 916 occurrences
```

1.5 Adding new unforseen data

1.5.1 Validation Data Loading

[ ]:
```
new_data = np.load('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪Preprocessed/valid_Scipy.npy')
print ('size-data='+str (new_data.shape))
```

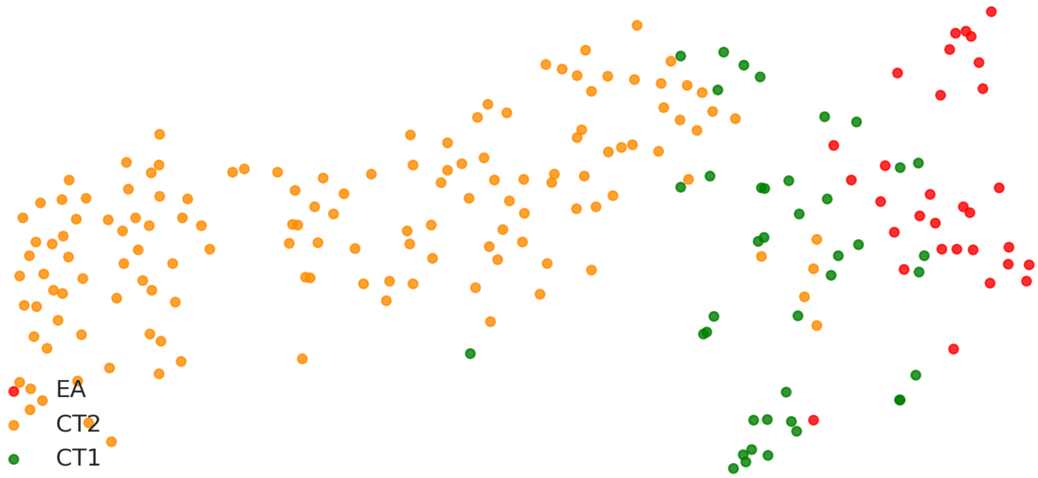
```
size-data=(211, 128, 32, 1)
```

[ ]:
```
from tensorflow.keras.models import load_model

# Load the finetuned model
autoencoder = load_model('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
↪Models/autoencoder-model_finetuned')
```

[ ]:
```
from joblib import load
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Predict encoded features for new_data
encoded_features = encoder.predict(new_data)

# Apply t-SNE
tsne_output = TSNE(random_state=123).fit_transform(encoded_features)

# Predict cluster assignments for new_data using the full model
cluster_assignments = model.predict(new_data)
y = np.argmax(cluster_assignments, axis=1)  # convert soft assignments to hard

# Define the plotter function
def plotter(S, y, target_names):
    '''
    function to visualize the outputs of t-SNE
    '''
    # choose a color palette with seaborn.
    colors = ['red', 'darkorange', 'green']

    lw = 2
    # create a scatter plot.
    f = plt.figure(figsize=(22, 10))
    ax = f.add_subplot(111)
    for color, i, target_name in zip(colors, range(len(target_names)),␣
↪target_names):
        plt.scatter(S[y == i, 0], S[y == i, 1], color=color, alpha=.8, lw=lw,␣
↪s=100, label=target_name)
    plt.legend(loc='lower left', shadow=False, scatterpoints=1, prop={'size':␣
↪26})
```

```
    ax.axis('off')
    ax.axis('tight')
    plt.show()
    f.savefig ('Tsne-km-n3-ft.png', dpi=100, bbox_inches="tight")

    return f, ax

# Target names for clusters
target_names = ['EA', 'CT2', 'CT1']  # Adjust as per your actual cluster names

# Visualize
plotter(tsne_output, y, target_names)
```

```
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 3ms/step
```



[ ]: (<Figure size 2200x1000 with 1 Axes>, <Axes: >)

[ ]:
```
### Save the labels
np.savetxt('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪Valid-n3-ft.txt', y, fmt='%i', delimiter=',')
```

[ ]:
```
# Path to the file
file_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪Valid-n3-ft.txt'

# Initialize a dictionary to count cluster occurrences
cluster_counts = {}
```

```
# Read the file line by line
with open(file_path, 'r') as file:
    for line in file:
        cluster = line.strip()
        if cluster in cluster_counts:
            cluster_counts[cluster] += 1
        else:
            cluster_counts[cluster] = 1

# Print the counts for each cluster
for cluster, count in cluster_counts.items():
    print(f"Cluster {cluster}: {count} occurrences")
```

```
Cluster 1: 211 occurrences
```

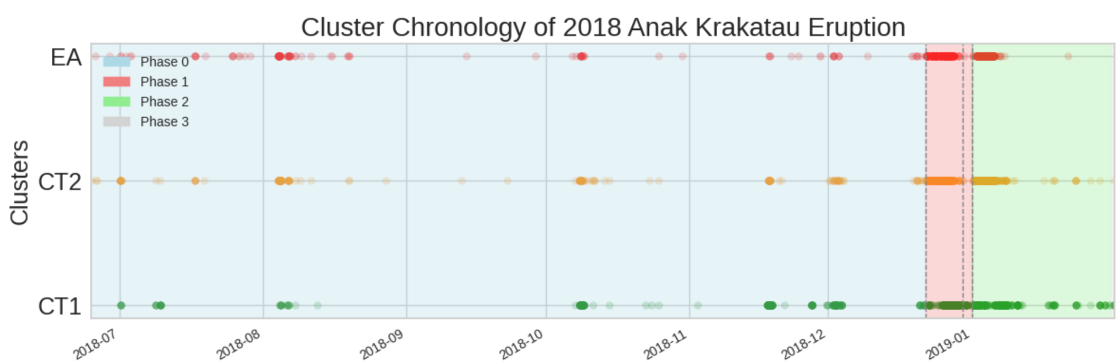
[ ]:
```
# Change the order of the cluster numbers (just for a nice representation)

with open('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
 ↪Valid-n3-ft.txt', 'r') as file :
  filedata = file.read()

# Replace the target string
filedata = filedata.replace('2', 'x')
filedata = filedata.replace('0', '2')
filedata = filedata.replace('1', '0')
filedata = filedata.replace('x', '1')

# Re-write the output
with open('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
 ↪Valid-n3-ft_Rewritten.txt', 'w') as file:
  file.write(filedata)
```

2 Temporal Visualization of Cluster Distribution

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# Load the numpy array containing the datetime index
datetime_index_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
 ↪data/Preprocessed/train_test_datetime.npy'
y_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
 ↪Km-n3-ft_Rewritten.txt'
filtered_vona_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
 ↪data/filtered_VONA.csv'  # Update this path to your local file
```

```
datetime_index = np.load(datetime_index_path, allow_pickle=True)
datetime_index = pd.to_datetime(datetime_index).tz_localize(None)  # Remove␣
 ↪timezone information to make it tz-naive

# Specific dates for axvline and phases
specific_dates = [
    '2018-12-22 00:00:00',
    '2018-12-30 00:00:00',
    '2019-01-01 00:00:00',
    '2019-05-22 00:00:00'
]

phase_intervals = [
    ('2018-06-18 00:00:00', '2018-12-22 00:00:00'),  # Phase 0
    ('2018-12-22 00:00:00', '2019-01-01 00:00:00'),  # Phase 1
    ('2019-01-01 00:00:00', '2019-01-31 00:00:00'),  # Phase 2
    ('2019-01-31 00:00:00', '2019-05-22 00:00:00')   # Phase 3 (I added this)
]

# Define the colors for each phase
phase_colors = ['lightblue', 'lightcoral', 'lightgreen', 'lightgrey']  #␣
 ↪lightcoral is a light red color

# Create the plot
fig, ax = plt.subplots(figsize=(12, 4))

# Set up the colormap for three clusters
colors = ['green','darkorange','red']
cmap = LinearSegmentedColormap.from_list('my_list', colors, N=3)

# Create scatter plot
scatter = ax.scatter(datetime_index, y, c=y, cmap=cmap, s=30, alpha=0.2)

# Set y-ticks and labels for three clusters
ax.set_yticks([0, 1, 2])
ax.set_yticklabels(['CT1', 'CT2', 'EA'], fontsize=18)

plt.ylabel('Clusters', fontsize=18)
ax.xaxis_date()  # Set x-axis to use date format
fig.autofmt_xdate()  # Auto-format the date labels

# Set x-axis limits
plt.xlim(datetime_index[0], datetime_index[-1])

# Add vertical lines at specific dates
for date in specific_dates:
```

```
    ax.axvline(pd.to_datetime(date), color='grey', linestyle='--', linewidth=1)

# Add background color and labels for each phase
for i, (interval, color) in enumerate(zip(phase_intervals, phase_colors)):
    start_date, end_date = pd.to_datetime(interval[0]), pd.
 ↪to_datetime(interval[1])
    ax.axvspan(start_date, end_date, color=color, alpha=0.3, label=f'Phase {i}')

# Create a legend for the phases
handles, labels = ax.get_legend_handles_labels()
# Create custom legend for phases
from matplotlib.patches import Patch
custom_handles = [Patch(facecolor=pc, edgecolor=pc) for pc in phase_colors]
plt.legend(custom_handles, [f'Phase {i}' for i in range(len(phase_intervals))],␣
 ↪loc='upper left')

# Set the title
plt.title('Cluster Chronology of 2018 Anak Krakatau Eruption', fontsize=20)

plt.tight_layout()
plt.show()
```

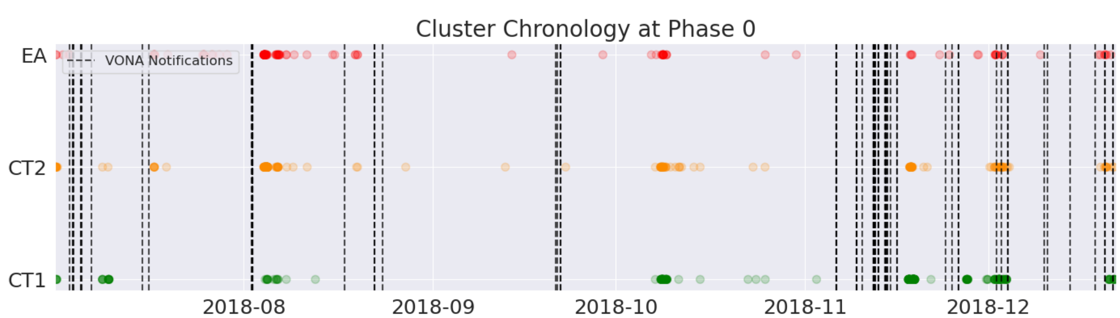


2.1 Phase 0

[ ]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import pandas as pd
import numpy as np

# Load the necessary data
datetime_index_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
 ↪data/Preprocessed/train_test_datetime.npy'
```

```
y_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪Km-n3-ft_Rewritten.txt'
filtered_vona_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
↪data/filtered_VONA.csv'  # Update this path to your local file

datetime_index = np.load(datetime_index_path, allow_pickle=True)
datetime_index = pd.to_datetime(datetime_index).tz_localize(None)  # Remove␣
↪timezone information to make it tz-naive
y = np.loadtxt(y_path)
filtered_vona_df = pd.read_csv(filtered_vona_path)

# Explicitly convert 'Issued' column to datetime
filtered_vona_df['Issued'] = pd.to_datetime(filtered_vona_df['Issued'])

# Determine the date range from the data itself
date_range = (datetime_index.min().strftime('%Y-%m-%d'), datetime_index.max().
↪strftime('%Y-%m-%d'))

# Set up the colormap
colors = ['green', 'darkorange', 'red']
cmap = LinearSegmentedColormap.from_list('my_list', colors)

# Define your date range
date_range = ('2018-07-01', '2018-12-22')

# Create a figure
fig, ax = plt.subplots(figsize=(14, 4))

# Convert Pandas Timestamp to NumPy datetime64[ns]
dates_filtered_np = datetime_index.to_numpy()

# Filter the data for the current date range
mask = (dates_filtered_np >= pd.to_datetime(date_range[0])) &␣
↪(dates_filtered_np <= pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]

# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,␣
↪s=50, alpha=0.2)

    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2', 'EA'], fontsize=18)
```

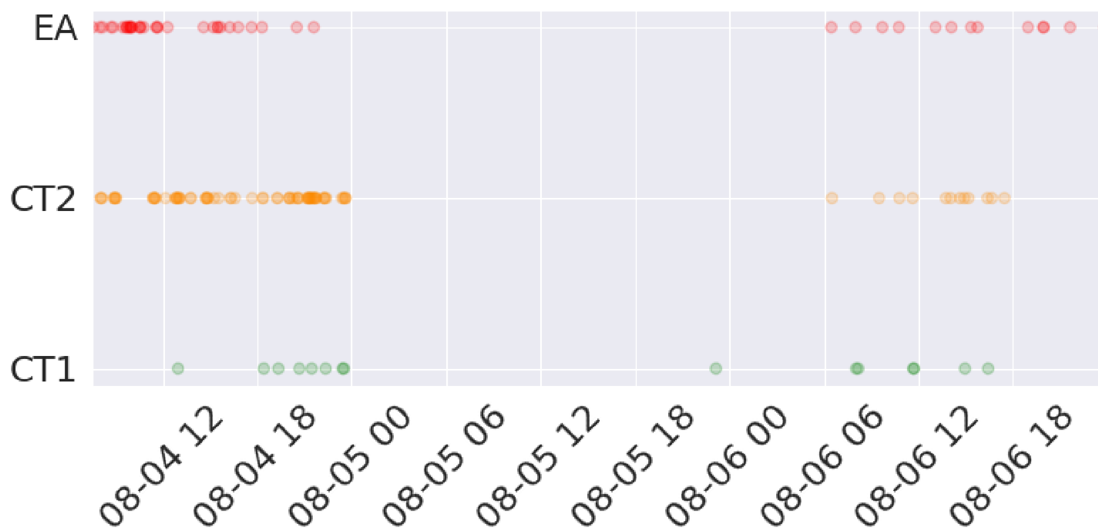
```
    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to the range of filtered dates

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7, label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0] else "")

    # Adding a legend for VONA Notifications
    ax.legend(loc='upper left', fontsize='large')

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) # Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center', verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 0", fontsize=20)

plt.tight_layout()
plt.show()
```



[ ]:

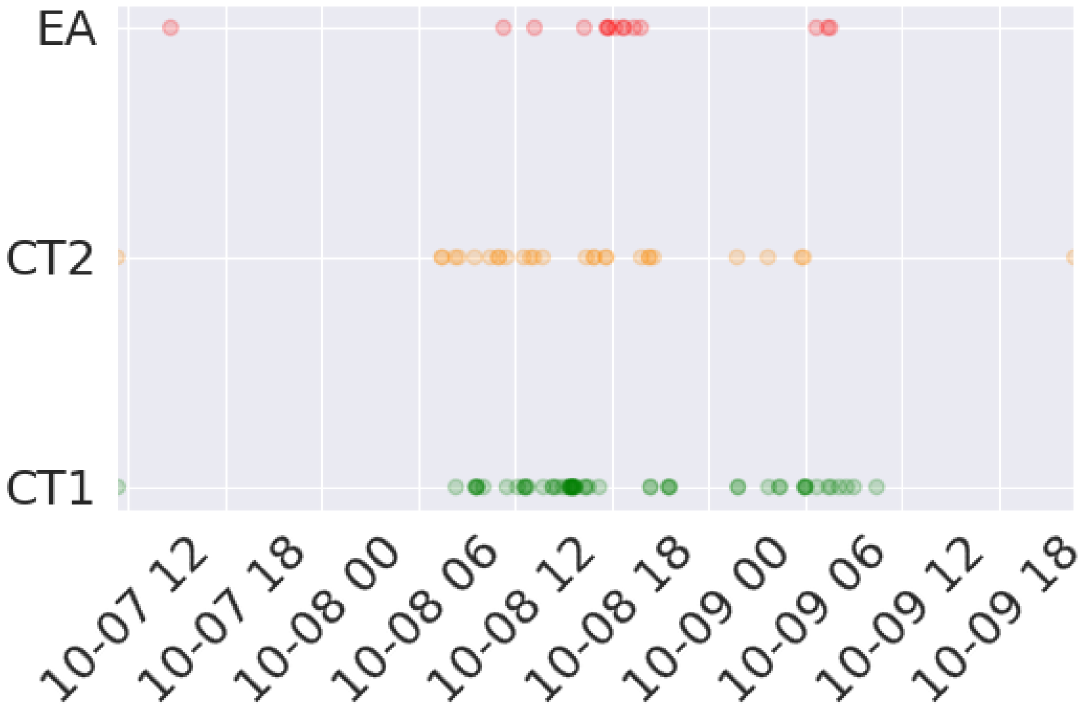
```python
import matplotlib.dates as mdates


# Define your date range
date_range = ('2018-08-03', '2018-08-07')

# Create a figure
fig, ax = plt.subplots(figsize=(8, 4))

# Filter the data for the current date range
mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=␣
 ↪pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]

# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,␣
 ↪s=30, alpha=0.2)

    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2', 'EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to␣
 ↪the range of filtered dates

    # Decrease the x-ticks frequency
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%d %H'))
    plt.xticks(rotation=45)

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.
 ↪Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7,␣
 ↪label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0]␣
 ↪else "")


    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
```

```
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) ␣
↪# Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center',␣
↪verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data␣
↪is available

# Add a title to the plot

plt.tight_layout()
plt.show()
```



[ ]:
```
import matplotlib.dates as mdates


# Define your date range
date_range = ('2018-10-07 00:00:00', '2018-10-10 00:00:00')

# Create a figure
fig, ax = plt.subplots(figsize=(6, 4))

# Filter the data for the current date range
mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=␣
↪pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]

# Plot the data
```

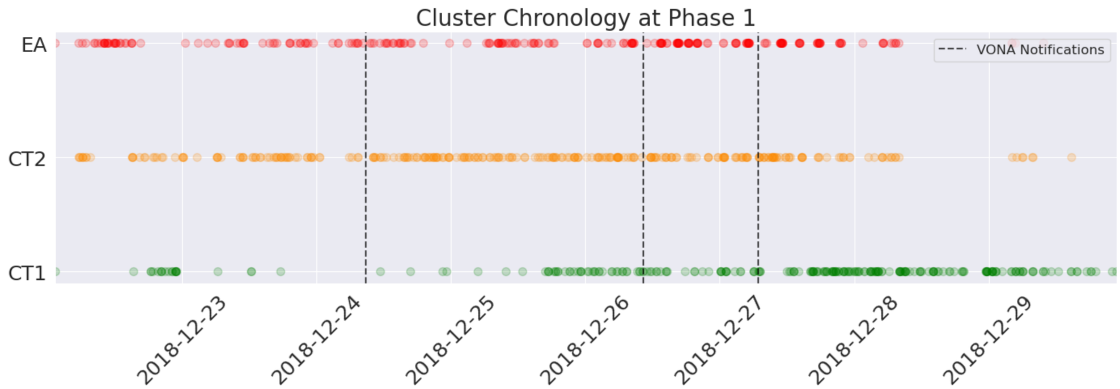
```python
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap, s=30, alpha=0.2)

    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2', 'EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to the range of filtered dates

    # Decrease the x-ticks frequency
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%d %H'))
    plt.xticks(rotation=45)

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7, label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0] else "")


    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) # Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center', verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data is available

# Add a title to the plot

plt.tight_layout()
plt.show()
```



2.2 Phase 1

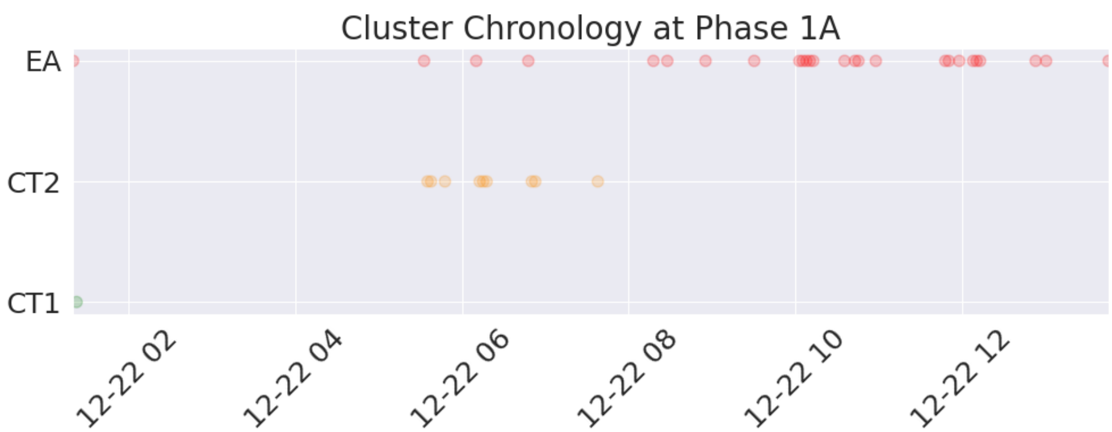
[ ]:
```
date_range = ('2018-12-22', '2018-12-30')

# Set up the colormap
colors = ['green','darkorange','red']
cmap = LinearSegmentedColormap.from_list('my_list', colors)

# Create a figure
fig, ax = plt.subplots(figsize=(14, 5))

# Filter the data for the current date range
mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=␣
↪pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]

# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,␣
↪s=50, alpha=0.2)
```

```
    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2', 'EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to the range of filtered dates
    plt.xticks(rotation=45)  # Give inclination to the x-axis labels

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7, label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0] else "")

    # Adding a legend for VONA Notifications
    ax.legend(loc='upper right', fontsize='large')

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) # Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center', verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 1", fontsize=20)

plt.tight_layout()
plt.show()
```



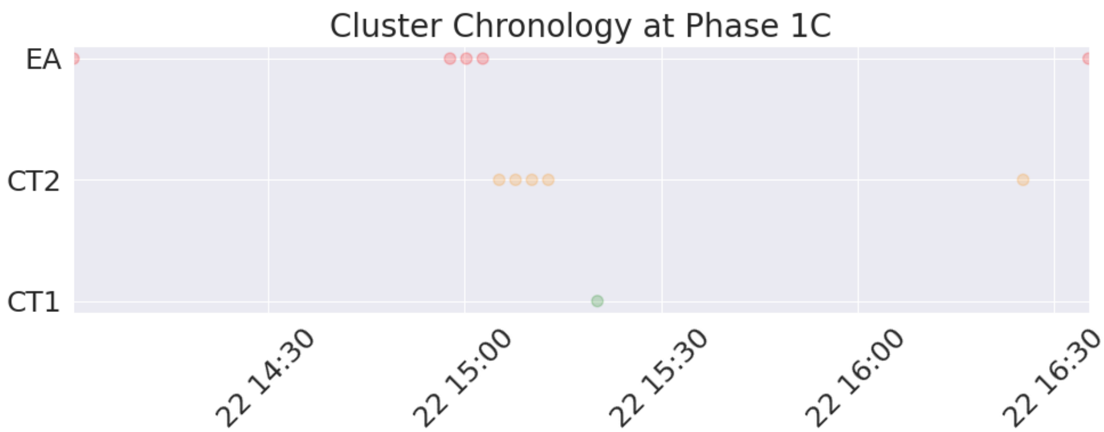

```
[ ]: date_range = ('2018-12-22 00:00:00', '2018-12-22 13:50:00')

     # Create a figure
     fig, ax = plt.subplots(figsize=(10, 4))

     # Filter the data for the current date range
     mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=
      ↪pd.to_datetime(date_range[1]))
     dates_filtered = datetime_index[mask]
     y_filtered = y[mask]

     # Plot the data
     if not dates_filtered.empty:
         # Create scatter plot for the filtered data
         scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,
      ↪s=50, alpha=0.2)

         # Set y-ticks and labels
         ax.set_yticks([0, 1, 2])
         ax.set_yticklabels(['CT1', 'CT2','EA'], fontsize=18)

         # Formatting the x-axis
         ax.xaxis_date()  # Set x-axis to use date format
         ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to
      ↪the range of filtered dates
         plt.xticks(rotation=45)  # Give inclination to the x-axis labels

         # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
         shifted_issued_dates = filtered_vona_df['Issued'] + pd.
      ↪Timedelta(seconds=3311)
         for issued_date in shifted_issued_dates:
```

```
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7,␣
↪label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0]␣
↪else "")

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) ␣
↪# Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center',␣
↪verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data␣
↪is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 1A", fontsize=20)

plt.tight_layout()
plt.show()
```



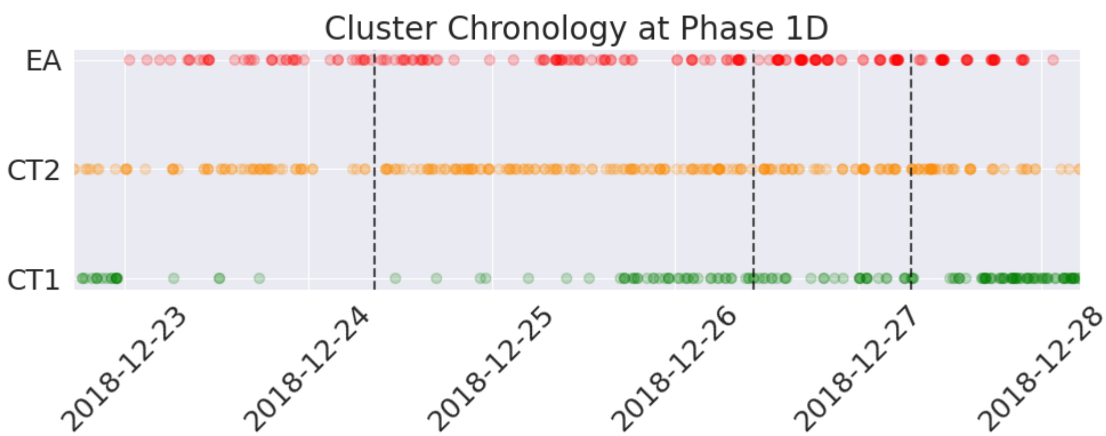

```
[ ]: date_range = ('2018-12-22 14:00:00', '2018-12-22 16:55:00')

# Create a figure
fig, ax = plt.subplots(figsize=(10, 4))

# Filter the data for the current date range
mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=␣
↪pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]
```

```python
# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,␣
↪s=50, alpha=0.2)

    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2','EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to␣
↪the range of filtered dates
    plt.xticks(rotation=45)  # Give inclination to the x-axis labels

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.
↪Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7,␣
↪label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0]␣
↪else "")

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) ␣
↪# Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center',␣
↪verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data␣
↪is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 1C", fontsize=20)

plt.tight_layout()
plt.show()
```



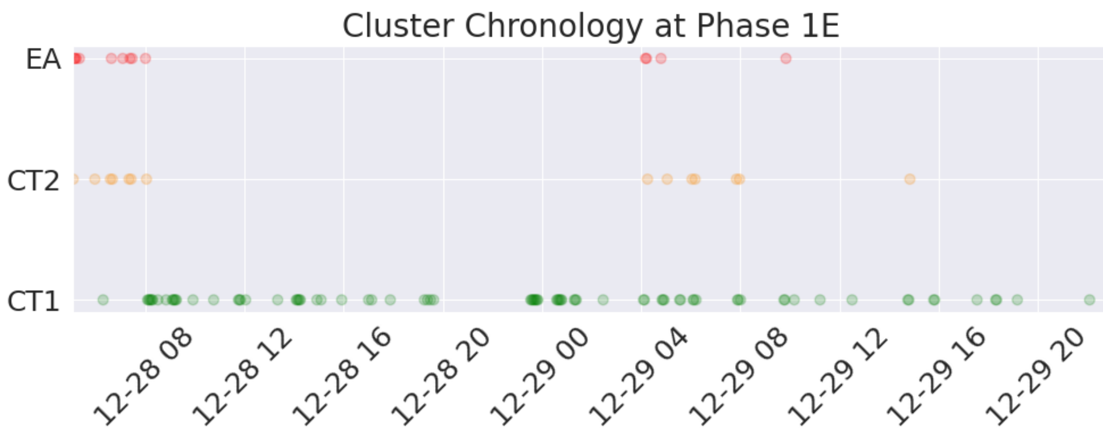
[ ]:
```
date_range = ('2018-12-22 16:55:00', '2018-12-28 05:05:00')

# Create a figure
fig, ax = plt.subplots(figsize=(10, 4))

# Filter the data for the current date range
mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=␣
↪pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]

# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,␣
↪s=40, alpha=0.2)

    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2','EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to␣
↪the range of filtered dates
    plt.xticks(rotation=45)  # Give inclination to the x-axis labels

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.
↪Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
```

```
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7,␣
↪label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0]␣
↪else "")

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) ␣
↪# Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center',␣
↪verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data␣
↪is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 1D", fontsize=20)

plt.tight_layout()
plt.show()
```



[ ]:
```
# Define your date range
date_range = ('2018-12-28 05:05:00', '2018-12-30 00:00:00')

# Create a figure
fig, ax = plt.subplots(figsize=(10, 4))

# Filter the data for the current date range
mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=␣
↪pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]
```

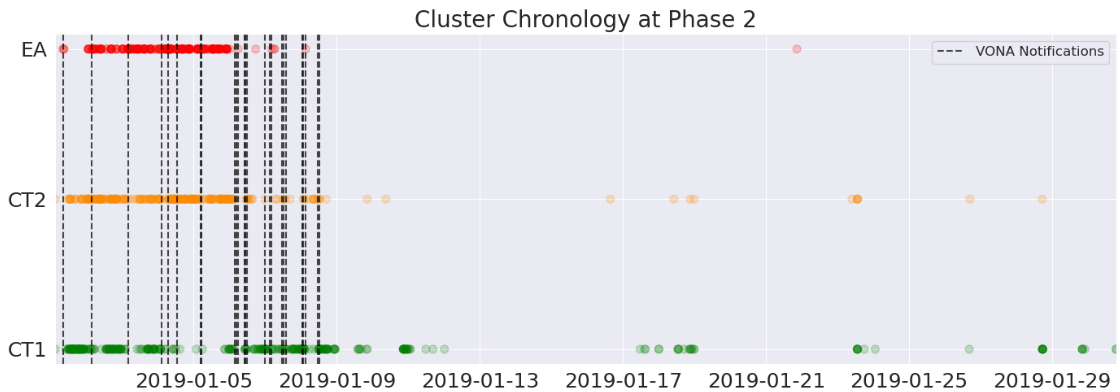
```
# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,␣
 ↪s=40, alpha=0.2)

    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2','EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to␣
 ↪the range of filtered dates
    plt.xticks(rotation=45)  # Give inclination to the x-axis labels

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.
 ↪Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7,␣
 ↪label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0]␣
 ↪else "")

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) ␣
 ↪# Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center',␣
 ↪verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data␣
 ↪is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 1E", fontsize=20)

plt.tight_layout()
plt.show()
```



2.3 Phase 2

[ ]:
```
date_range = ('2019-01-01 00:00:00', '2019-01-31 00:00:00')

# Create a figure
fig, ax = plt.subplots(figsize=(14, 5))

# Filter the data for the current date range
mask = (datetime_index >= pd.to_datetime(date_range[0])) & (datetime_index <=
 ↪pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]

# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap,
 ↪s=50, alpha=0.2)

    # Set y-ticks and labels
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2', 'EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to
 ↪the range of filtered dates

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.
 ↪Timedelta(seconds=3311)
```

```
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7,␣
↪label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0]␣
↪else "")

    # Adding a legend for VONA Notifications
    ax.legend(loc='upper right', fontsize='large')

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) ␣
↪# Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center',␣
↪verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data␣
↪is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 2", fontsize=20)

plt.tight_layout()
plt.show()
```

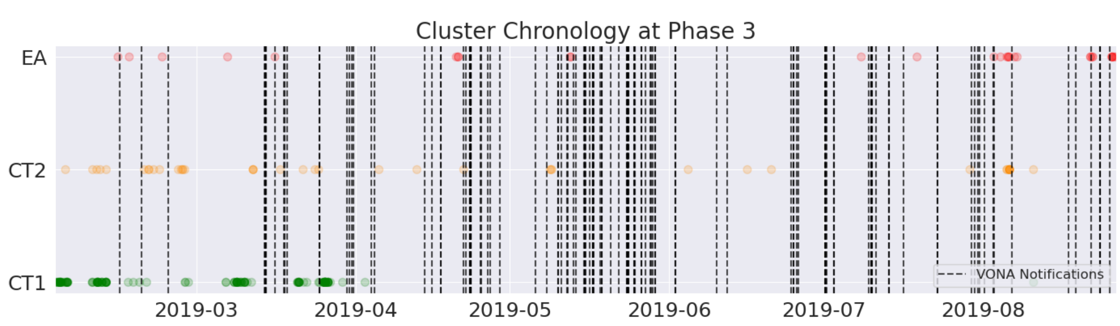


2.4 Phase 3 (Validation Data)

[ ]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import pandas as pd
import numpy as np

# Load the necessary data
```

```python
datetime_index_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/Preprocessed/valid_datetime.npy'
y_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/Valid-n3-ft_Rewritten.txt'
filtered_vona_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/filtered_VONA.csv'  # Update this path to your local file

datetime_index = np.load(datetime_index_path, allow_pickle=True)
datetime_index = pd.to_datetime(datetime_index).tz_localize(None)  # Remove timezone information to make it tz-naive
y = np.loadtxt(y_path)
filtered_vona_df = pd.read_csv(filtered_vona_path)

# Explicitly convert 'Issued' column to datetime
filtered_vona_df['Issued'] = pd.to_datetime(filtered_vona_df['Issued'])

# Determine the date range from the data itself
date_range = (datetime_index.min().strftime('%Y-%m-%d'), datetime_index.max().strftime('%Y-%m-%d'))

# Set up the colormap
colors = ['green', 'darkorange', 'red']
cmap = LinearSegmentedColormap.from_list('my_list', colors)

# Define your date range
date_range = ('2019-02-01', '2019-09-01')

# Create a figure
fig, ax = plt.subplots(figsize=(14, 4))

# Convert Pandas Timestamp to NumPy datetime64[ns]
dates_filtered_np = datetime_index.to_numpy()

# Filter the data for the current date range
mask = (dates_filtered_np >= pd.to_datetime(date_range[0])) & (dates_filtered_np <= pd.to_datetime(date_range[1]))
dates_filtered = datetime_index[mask]
y_filtered = y[mask]

# Plot the data
if not dates_filtered.empty:
    # Create scatter plot for the filtered data
    scatter = ax.scatter(dates_filtered, y_filtered, c=y_filtered, cmap=cmap, s=50, alpha=0.2)

    # Set y-ticks and labels
```

```
    ax.set_yticks([0, 1, 2])
    ax.set_yticklabels(['CT1', 'CT2', 'EA'], fontsize=18)

    # Formatting the x-axis
    ax.xaxis_date()  # Set x-axis to use date format
    ax.set_xlim(dates_filtered.min(), dates_filtered.max())  # Set x limits to␣
↪the range of filtered dates

    # Shift 'Issued' dates backwards by 3311 seconds and plot vertical lines
    shifted_issued_dates = filtered_vona_df['Issued'] + pd.
↪Timedelta(seconds=3311)
    for issued_date in shifted_issued_dates:
        ax.axvline(x=issued_date, color='black', linestyle='--', alpha=0.7,␣
↪label='VONA Notifications' if issued_date == shifted_issued_dates.iloc[0]␣
↪else "")

    # Adding a legend for VONA Notifications
    ax.legend(loc='lower right', fontsize='large')

    ax.tick_params(axis='x', labelsize=18)  # Increase x-axis label size

else:
    ax.set_xlim(pd.to_datetime(date_range[0]), pd.to_datetime(date_range[1])) ␣
↪# Use the date range for the limits
    ax.text(0.5, 0.5, 'No data for this range', horizontalalignment='center',␣
↪verticalalignment='center',
            transform=ax.transAxes, fontsize=18)  # Show message when no data␣
↪is available

# Add a title to the plot
ax.set_title("Cluster Chronology at Phase 3", fontsize=20)

plt.tight_layout()
plt.show()
```



3 Input-Decoded Visualization of Clusters

[ ]:
```python
from keras.models import load_model


autoencoder = load_model("/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
↪Models/autoencoder-model2")
import keras.backend as K
get_all_layer_outputs = K.function([autoencoder.layers[0].input],
                                   [l.output for l in autoencoder.layers[1:]])
layer_output = get_all_layer_outputs([data]) # return the same thing
decoded_imgs = autoencoder.predict(data)
```

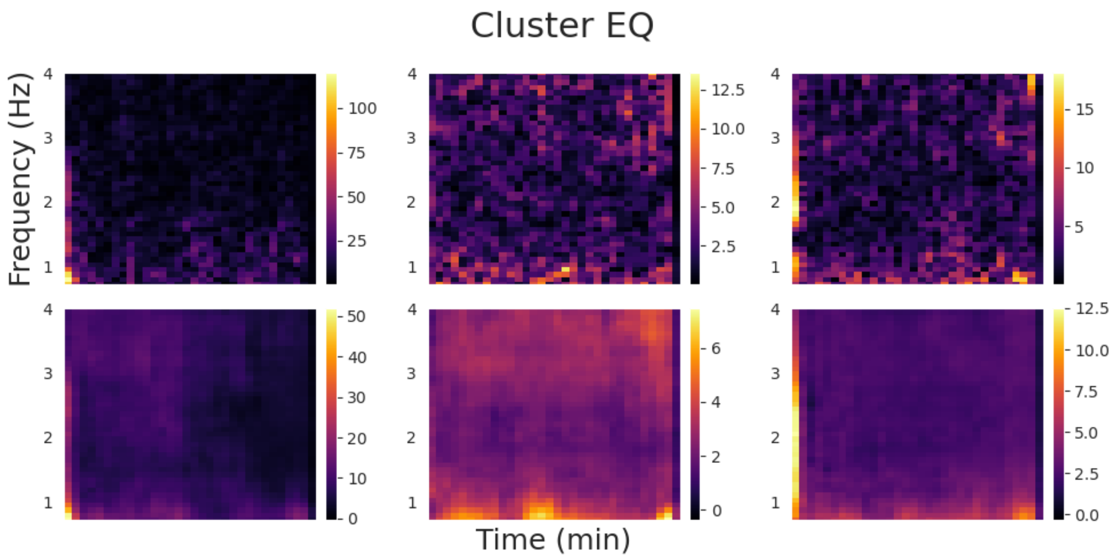
```
2024-01-09 02:25:11.135842: I
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:454] Loaded cuDNN
version 8904
```

```
73/73 [==============================] - 1s 6ms/step
```

[ ]:
```python
import librosa
import librosa.display
import matplotlib.pyplot as plt
from matplotlib.pyplot import savefig
import matplotlib.gridspec as gridspec

sr = 20  # Replace with the actual sample rate

# Assuming n_fft is known (required to calculate frequency bins accurately)
n_fft = 254  # Replace with the actual n_fft used for STFT computation

# Calculate frequency bins
frequencies = librosa.fft_frequencies(sr=sr, n_fft=n_fft)

# Find the index of the frequency bin closest to 0.7 Hz and 4 Hz
fmin_idx = np.abs(frequencies - 0.7).argmin()
fmax_idx = np.abs(frequencies - 4).argmin()

# Create a figure and define the subplot grid
fig = plt.figure(figsize=(10, 5))
spec = gridspec.GridSpec(2, 3)

# First Subplot
ax1 = fig.add_subplot(spec[0, 0])
librosa.display.specshow(data[50, :, :, 0], cmap='inferno', y_axis='linear',␣
↪sr=sr)
plt.colorbar(pad=0.03)
plt.ylim(frequencies[fmin_idx], frequencies[fmax_idx])
plt.ylabel('Frequency (Hz)', fontsize=18)
```

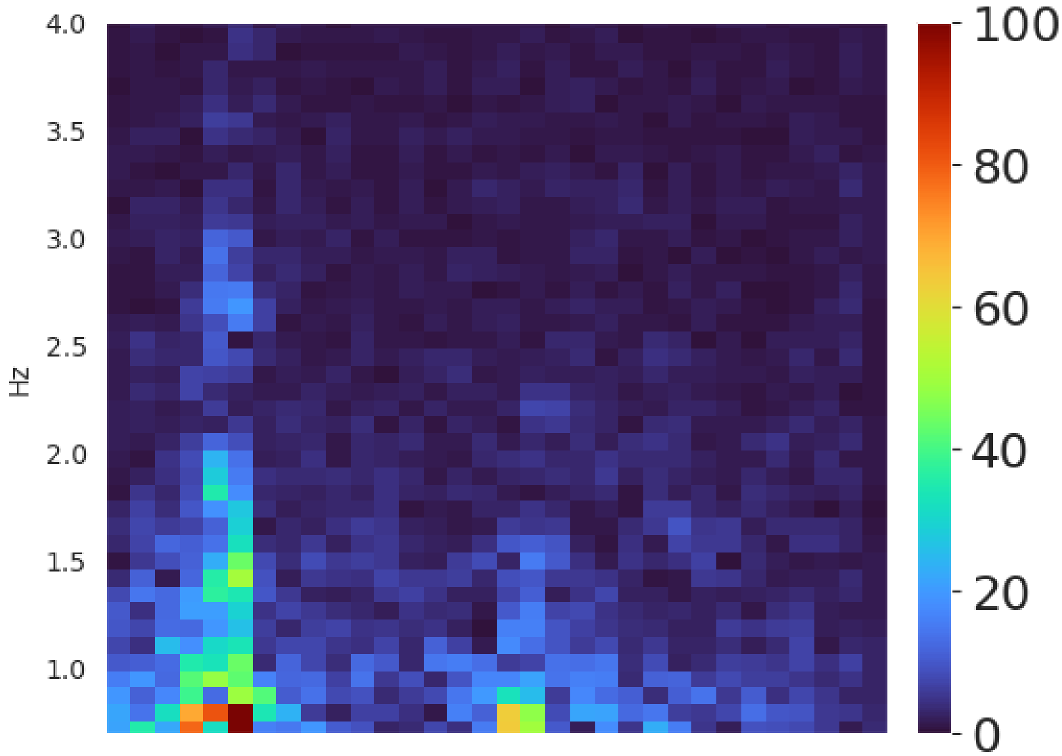
```
# Second Subplot
ax2 = fig.add_subplot(spec[0, 1])
librosa.display.specshow(data[44, :, :, 0], cmap='inferno',y_axis='linear',␣
↪sr=sr)
plt.colorbar(pad=0.03)
plt.ylim(frequencies[fmin_idx], frequencies[fmax_idx])
# Remove the y-label
ax2.set(ylabel=None)

# Third Subplot
ax3 = fig.add_subplot(spec[0, 2])
librosa.display.specshow(data[45, :, :, 0], cmap='inferno', y_axis='linear',␣
↪sr=sr)
plt.colorbar(pad=0.03)
plt.ylim(frequencies[fmin_idx], frequencies[fmax_idx])
# Remove the y-label
ax3.set(ylabel=None)

# Fourth Subplot (Decoded Images)
ax4 = fig.add_subplot(spec[1, 0])
librosa.display.specshow(decoded_imgs[56, :, :, 0], cmap='inferno',␣
↪y_axis='linear', sr=sr)
plt.colorbar(pad=0.03)
plt.ylim(frequencies[fmin_idx], frequencies[fmax_idx])
# Remove the y-label
ax4.set(ylabel=None)

# Fifth Subplot (Decoded Images)
ax5 = fig.add_subplot(spec[1, 1])
librosa.display.specshow(decoded_imgs[44, :, :, 0], cmap='inferno',␣
↪y_axis='linear', sr=sr)
plt.colorbar(pad=0.03)
plt.ylim(frequencies[fmin_idx], frequencies[fmax_idx])
# Remove the y-label and set x-label
ax5.set(ylabel=None)
plt.xlabel("Time (min)", fontsize=18)

# Sixth Subplot (Decoded Images)
ax6 = fig.add_subplot(spec[1, 2])
librosa.display.specshow(decoded_imgs[45, :, :, 0], cmap='inferno',␣
↪y_axis='linear', sr=sr)
plt.colorbar(pad=0.03)
plt.ylim(frequencies[fmin_idx], frequencies[fmax_idx])
# Remove the y-label
ax6.set(ylabel=None)
```

```
# Set the figure's title
fig.suptitle('Cluster EQ', fontsize=22)

# Adjust layout
plt.tight_layout()
```

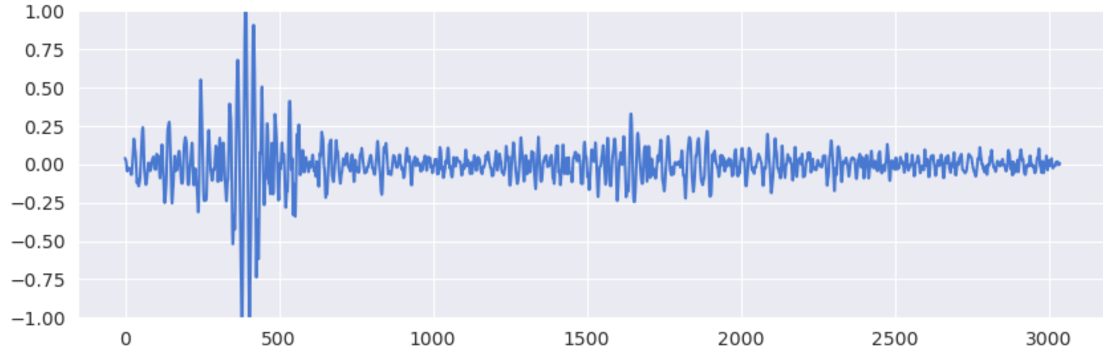


[ ]:

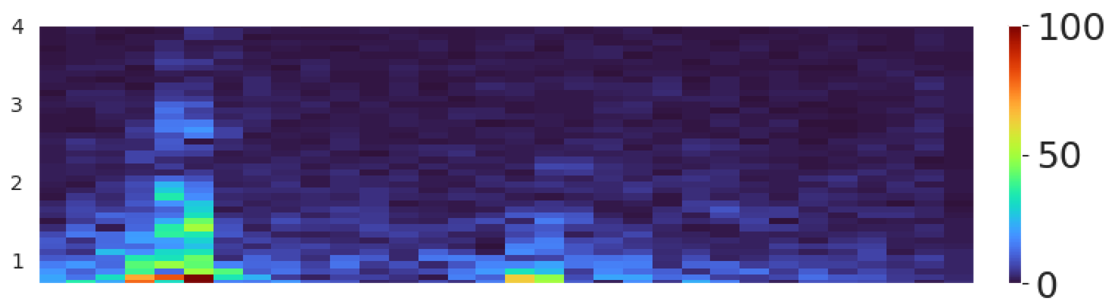
```
ax1 = fig.add_subplot(spec[0, 0])

librosa.display.specshow(data[1304, :, :, 0], cmap='turbo', y_axis='linear',␣
↪sr=20, vmin=0, vmax=100)
cbar= plt.colorbar(pad= 0.03)
cbar.ax.tick_params(labelsize=18, rotation=0)
plt.ylim(0.7, 4)

ax1.set(ylabel=None)
plt.show()
```




```
[ ]: import matplotlib.pyplot as plt
     import librosa

     n_fft = 254
     hop_length = 98

     # Perform ISTFT
     signal = librosa.istft(data[1304, :, :, 0], hop_length=hop_length,␣
      ↪win_length=n_fft)

     # Set the figure size
     fig, ax = plt.subplots(figsize=(10, 3))  # You can adjust the width and height␣
      ↪as needed

     ax.plot(signal)
     ax.set_ylim(-1, 1)  # Set the y-axis limits to -1 and 1
     plt.show()
```

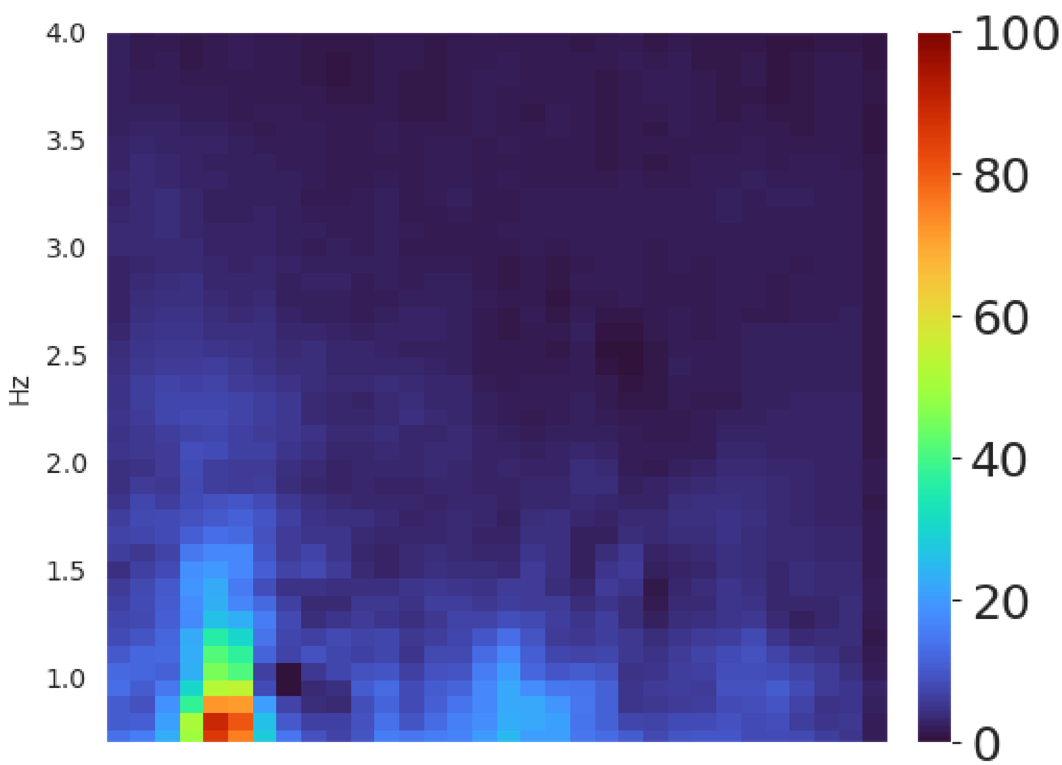


[ ]:
```
import matplotlib.pyplot as plt
import librosa.display

# Define your spec and ax1

# Set the figure size
fig = plt.figure(figsize=(10, 5))  # You can adjust the width and height as
 ↪needed

ax1 = fig.add_subplot(spec[0, 0])

librosa.display.specshow(data[1304, :, :, 0], cmap='turbo', y_axis='linear',
 ↪sr=20, vmin=0, vmax=100)
cbar = plt.colorbar(pad=0.03)
cbar.ax.tick_params(labelsize=18, rotation=0)
plt.ylim(0.7, 4)

ax1.set(ylabel=None)
plt.show()
```



[ ]:
```
# Second subplot for the waveform (time-domain signal)
ax2 = fig.add_subplot(spec[1, 0])
# Inverse STFT to obtain the time-domain signal
# Replace 'data[120, :, :, 0]' with the appropriate slice of your actual data␣
↪variable.
signal = librosa.istft(data[1304, :, :, 0], hop_length=hop_length,␣
↪win_length=n_fft)
ax2.plot(signal, color='blue')
ax2.set_xlabel('Time (samples)')
ax2.set_ylabel('Amplitude')
ax2.set_title('Waveform')
plt.show()
```

[ ]:
```
ax1 = fig.add_subplot(spec[0, 0])

librosa.display.specshow(decoded_imgs[1304, :, :, 0], cmap='turbo',␣
↪y_axis='linear', sr=20, vmin=0, vmax=100)
cbar= plt.colorbar(pad= 0.03)
cbar.ax.tick_params(labelsize=18, rotation=0)
plt.ylim(0.7, 4)
ax1.set(ylabel=None)
plt.show()
```



4 Dataset Creation for Predictive Maintanance

[ ]:
```
from tensorflow.keras.models import load_model
import numpy as np
import pandas as pd
import keras.backend as K
from sklearn.preprocessing import MinMaxScaler

# Load data
data = np.load('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪Preprocessed/Train_Test_Scipy.npy')
datetime_index = np.load('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
↪data/Preprocessed/train_test_datetime.npy', allow_pickle=True)
clustering_results = np.loadtxt('/run/media/viblab/Markov2/Haykal/
↪AnakKrakatauEWSFinal/data/Km-n3-ft_Rewritten.txt', dtype=str)

# Load the autoencoder model
autoencoder = load_model('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/
↪Models/autoencoder-model2')

# Function to get all layer outputs
get_all_layer_outputs = K.function([autoencoder.layers[0].input], [l.output for␣
↪l in autoencoder.layers[1:]])

# Get the encoded layer output (assuming it's the output of 'dense_1')
layer_output = get_all_layer_outputs([data])
encoded_features = layer_output[5]  # Adjust index 5 to match the 'dense_1'␣
↪layer in your model

# Apply MinMax scaling to the encoded features
scaler = MinMaxScaler()
scaled_encoded_features = scaler.fit_transform(encoded_features)

# Convert datetime array to pandas datetime and round up to the nearest minute
datetime_index = pd.to_datetime(datetime_index).ceil('min')

# Create DataFrame for scaled encoded features with 'enc_' prefix
scaled_encoded_features_df = pd.DataFrame(scaled_encoded_features)
scaled_encoded_features_df.columns = ['enc_' + str(col) for col in␣
↪scaled_encoded_features_df.columns]

# Add cluster labels to the DataFrame
clustering_labels_df = pd.DataFrame({'cluster': clustering_results})

# Combine scaled encoded features and cluster labels
combined_df = pd.concat([scaled_encoded_features_df, clustering_labels_df],␣
↪axis=1)
```

```python
# Set the rounded datetime as the DataFrame index and name it 'timestamp'
combined_df.index = datetime_index
combined_df.index.name = 'timestamp'

combined_df.head()
```

[ ]:
```
                           enc_0     enc_1     enc_2     enc_3     enc_4  \
timestamp
2018-06-24 22:50:00+00:00    0.0  0.153880  0.131726  0.020461  0.042775
2018-06-25 20:31:00+00:00    0.0  0.142621  0.073355  0.028357  0.051817
2018-06-26 02:51:00+00:00    0.0  0.128460  0.097686  0.023503  0.060460
2018-06-26 02:59:00+00:00    0.0  0.093154  0.065859  0.017310  0.042119
2018-06-28 22:33:00+00:00    0.0  0.223360  0.212247  0.114154  0.199226

                           enc_5     enc_6  enc_7     enc_8  enc_9  …  \
timestamp                                                            …
2018-06-24 22:50:00+00:00    0.0  0.000225    0.0  0.103674    0.0  …
2018-06-25 20:31:00+00:00    0.0  0.028261    0.0  0.065544    0.0  …
2018-06-26 02:51:00+00:00    0.0  0.022133    0.0  0.017686    0.0  …
2018-06-26 02:59:00+00:00    0.0  0.012361    0.0  0.033170    0.0  …
2018-06-28 22:33:00+00:00    0.0  0.037208    0.0  0.164736    0.0  …

                           enc_15    enc_16    enc_17        enc_18  enc_19  \
timestamp
2018-06-24 22:50:00+00:00     0.0  0.031588  0.097323  2.227753e-02     0.0
2018-06-25 20:31:00+00:00     0.0  0.093951  0.063795  3.866509e-03     0.0
2018-06-26 02:51:00+00:00     0.0  0.084417  0.066729  2.910383e-11     0.0
2018-06-26 02:59:00+00:00     0.0  0.032732  0.048599  1.637670e-02     0.0
2018-06-28 22:33:00+00:00     0.0  0.109253  0.182266  1.183589e-04     0.0

                           enc_20    enc_21    enc_22    enc_23  cluster
timestamp
2018-06-24 22:50:00+00:00     0.0  0.014235  0.209731  0.013914        1
2018-06-25 20:31:00+00:00     0.0  0.001169  0.178872  0.047715        2
2018-06-26 02:51:00+00:00     0.0  0.000000  0.173401  0.054453        1
2018-06-26 02:59:00+00:00     0.0  0.007294  0.102231  0.016622        1
2018-06-28 22:33:00+00:00     0.0  0.021254  0.277943  0.000000        2

[5 rows x 25 columns]
```

[ ]:
```python
combined_df.to_csv('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
↪DEC_Results/Encoded_Forecasting_Dataset_Normalized.csv', index=True)
```

[ ]:
```python
import pandas as pd

# Load the dataset
```

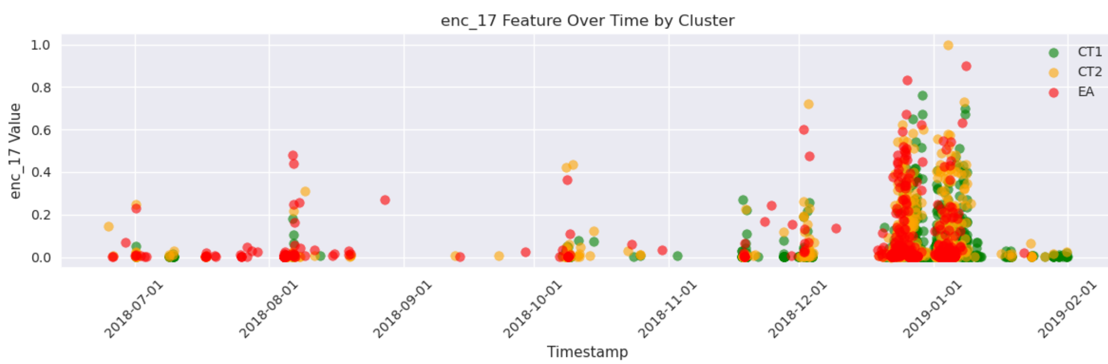
```
data_cluster = pd.read_csv('/run/media/viblab/Markov2/Haykal/
↪AnakKrakatauEWSFinal/data/DEC_Results/
↪Encoded_Forecasting_Dataset_NotNormalized.csv')

# Filter columns that start with "enc_"
enc_features = [col for col in data_cluster.columns if col.startswith('enc_')]
data_enc = data_cluster[['cluster'] + enc_features]

# Group by 'cluster' and calculate min, mean, median, and max for the filtered␣
↪features
enc_cluster_summary = data_enc.groupby('cluster').agg(['min', 'mean', 'median',␣
↪'max'])

# Reformatting the DataFrame for the desired structure
# Swap levels and sort columns to make clusters as primary column headers
enc_cluster_summary_adjusted = enc_cluster_summary.stack(level=0).
↪unstack(level=0).swaplevel(axis=1).sort_index(axis=1)

# Display the first few rows of the adjusted summary
enc_cluster_summary_adjusted.head()
```

[ ]:
```
cluster            0                                           1              \
                 max         mean       median  min          max        mean
enc_0      -0.998544    -0.999998    -1.000000 -1.0      -1.0000   -1.000000
enc_1     871.231450   147.304620   130.282175 -1.0    1080.1642  287.277070
enc_10   1047.830600   189.216231   171.658815 -1.0    1713.9205  369.045156
enc_11     -1.000000    -1.000000    -1.000000 -1.0      -1.0000   -1.000000
enc_12     -1.000000    -1.000000    -1.000000 -1.0      -1.0000   -1.000000

cluster                            2
             median  min         max        mean     median  min
enc_0      -1.00000 -1.0     -1.0000   -1.000000   -1.00000 -1.0
enc_1     268.14914 -1.0   2091.1790  523.670707  476.24708 -1.0
enc_10    345.77850 -1.0   3922.6042  831.440185  730.12902 -1.0
enc_11     -1.00000 -1.0     -1.0000   -1.000000   -1.00000 -1.0
enc_12     -1.00000 -1.0     -1.0000   -1.000000   -1.00000 -1.0
```

[ ]:
```
enc_cluster_summary_adjusted.to_csv('/run/media/viblab/Markov2/Haykal/
↪AnakKrakatauEWSFinal/data/DEC_Results/Summary.csv', index=True)
```

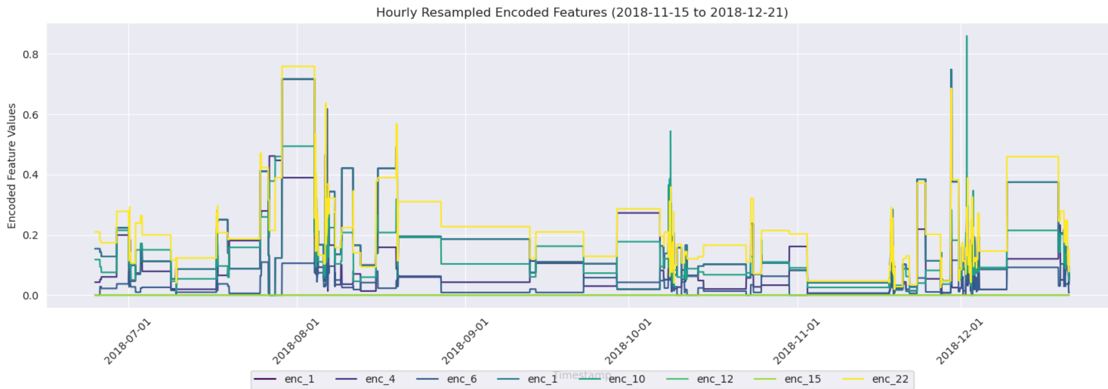
[ ]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Plot enc_17 over time with colors based on cluster labels
plt.figure(figsize=(12, 4))

# Plot each cluster with its respective color
```

```
colors = {0: 'green', 1: 'orange', 2: 'red'}
labels = {0: 'CT1', 1: 'CT2', 2: 'EA'}

for cluster in colors:
    # Filter data for each cluster
    cluster_data = combined_df[combined_df['cluster'] == str(cluster)]
    # Use the index directly for plotting
    plt.scatter(cluster_data.index, cluster_data['enc_4'],␣
 ↪color=colors[cluster], label=labels[cluster], alpha=0.6)

plt.xlabel('Timestamp')
plt.ylabel('enc_17 Value')  # Make sure this matches the correct column name in␣
 ↪your DataFrame
plt.title('enc_17 Feature Over Time by Cluster')
plt.xticks(rotation=45)  # Rotate the x-axis labels for better readability
plt.legend()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
plt.tight_layout()
plt.show()
```



[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.dates as mdates

# Load the dataset
file_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
 ↪DEC_Results/Encoded_Forecasting_Dataset_Normalized.csv'
data = pd.read_csv(file_path)

# Convert the 'timestamp' column to datetime and set it as the index
data['timestamp'] = pd.to_datetime(data['timestamp'])
```

```python
data.set_index('timestamp', inplace=True)

# Filter data between 2018-06-24 and 2018-12-21
mask = (data.index >= '2018-06-24') & (data.index <= '2018-12-21')
filtered_data = data.loc[mask]

# Resample the filtered data to an hourly frequency, taking the mean of the␣
↪features
filtered_data_hourly = filtered_data.resample('1H').max()

# Forward fill any missing values after resampling
filtered_data_hourly_ffill = filtered_data_hourly.ffill()

# Specify the features you want to plot. If empty, all features will be plotted.
# Example: features_to_plot = ['feature1', 'feature2']
features_to_plot = ['enc_1','enc_4','enc_6',␣
↪'enc_1','enc_10','enc_12','enc_15','enc_22']  # Put feature names as strings␣
↪in this list

# If no specific features are provided, plot all features
if not features_to_plot:
    features_to_plot = filtered_data.columns

# Plot specified features on a single time series plot with different colors
plt.figure(figsize=(15, 6))  # Adjust the figure size as needed

# Create a color cycle iterator
colors = plt.cm.viridis(np.linspace(0, 1, len(features_to_plot)))

# Plot each specified feature
for i, column in enumerate(features_to_plot):
    plt.plot(filtered_data_hourly_ffill.index,␣
↪filtered_data_hourly_ffill[column], label=column, color=colors[i])

# Format the x-axis to show dates nicely
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.xticks(rotation=45)

# Include other plot elements
plt.title('Hourly Resampled Encoded Features (2018-11-15 to 2018-12-21)')
plt.xlabel('Timestamp')
plt.ylabel('Encoded Feature Values')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.2), ncol=10)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Load the dataset
file_path = '/run/media/viblab/Markov2/Haykal/AnakKrakatauEWSFinal/data/
 ↪DEC_Results/Encoded_Forecasting_Dataset.csv'
data = pd.read_csv(file_path)

# Ensure the 'cluster' column is read as a category if it's not already.
data['cluster'] = data['cluster'].astype('category')

# Convert the 'timestamp' column to datetime and set it as the index
data['timestamp'] = pd.to_datetime(data['timestamp'], utc=True)
data.set_index('timestamp', inplace=True)


# Filter data between 2018-06-24 and 2018-12-21
start_date = '2018-06-24'
end_date = '2018-12-21'
mask = (data.index >= start_date) & (data.index <= end_date)
filtered_data = data.loc[mask]

# Group by 'cluster' and 'timestamp' and count occurrences within each hour
cluster_counts = filtered_data.groupby(['cluster', pd.Grouper(freq='7D')]).
 ↪size().unstack(level=0, fill_value=0)

# Define the color mapping and cluster names
colors = ['green', 'darkorange', 'red']
cluster_names = ['CT1', 'CT2', 'EA']

# Plot the hourly counts of each cluster
plt.figure(figsize=(20, 4))
```

```python
cluster_counts.plot(kind='bar', stacked=True, color=colors, ax=plt.gca())

# Rename legend labels
handles, labels = plt.gca().get_legend_handles_labels()
plt.legend(handles, cluster_names, title='Cluster')

# Set major ticks to monthly frequency
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))

# Rotate and align the tick labels so they look better
plt.setp(plt.gca().get_xticklabels(), rotation=45, ha="right")

# Set plot title and labels
plt.title('Hourly Counts of Each Cluster (2018-06-24 to 2018-12-21)')
plt.xlabel('Timestamp')
plt.ylabel('Count')

# Show plot
plt.show()
```

```
/tmp/ipykernel_2113548/4202501608.py:24: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  cluster_counts = filtered_data.groupby(['cluster',
pd.Grouper(freq='7D')]).size().unstack(level=0, fill_value=0)
```