DA_4

November 1, 2025

Name:	Tufan Kundu
Registration no:	24MDT0184
Course Name:	Deep Learning Lab
Course Code:	PMDS603P
Digital Assignment:	4

0.1 Question 1: Collect the dataset regarding ECG (Electrocardiogram) signals of different subjects given in the moodle platform and prepare an autoencoder that can perform anomaly detection. The dataset you see has two classes, normal (0) and abnormal (1) ECG signals. The labels are also provided in the same csv file. The task is to construct an autoencoder that can detect an abnormal ECG signal. Train an autoencoder on the normal ECG signals. Then use a suitable method using reconstruction error to flag abnormal ecg's. And you can check with your test set data the outputs.

0.1.1 Importing necessary libraries

[80]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, roc_auc_score,␣
↪confusion_matrix
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
```

0.1.2 Loading the dataset

[81]:
```
df = pd.read_csv("as.csv")
df
```

[81]:
```
      Unnamed: 0         0         1         2         3         4         5  \
0              0 -0.112522 -2.827204 -3.773897 -4.349751 -4.376041 -3.474986
1              1 -1.100878 -3.996840 -4.285843 -4.506579 -4.022377 -3.234368
2              2 -0.567088 -2.593450 -3.874230 -4.584095 -4.187449 -3.151462
3              3  0.490473 -1.914407 -3.616364 -4.318823 -4.268016 -3.881110
4              4  0.800232 -0.874252 -2.384761 -3.973292 -4.338224 -3.802422
…            …         …         …         …         …         …         …
4993        4993  0.608558 -0.335651 -0.990948 -1.784153 -2.626145 -2.957065
4994        4994 -2.060402 -2.860116 -3.405074 -3.748719 -3.513561 -3.006545
4995        4995 -1.122969 -2.252925 -2.867628 -3.358605 -3.167849 -2.638360
4996        4996 -0.547705 -1.889545 -2.839779 -3.457912 -3.929149 -3.966026
4997        4997 -1.351779 -2.209006 -2.520225 -3.061475 -3.065141 -3.030739

             6         7         8  …       131       132       133  \
0    -2.181408 -1.818286 -1.250522  …  0.792168  0.933541  0.796958
1    -1.566126 -0.992258 -0.754680  …  0.538356  0.656881  0.787490
2    -1.742940 -1.490659 -1.183580  …  0.886073  0.531452  0.311377
3    -2.993280 -1.671131 -1.333884  …  0.350816  0.499111  0.600345
4    -2.534510 -1.783423 -1.594450  …  1.148884  0.958434  1.059025
…          …         …         …  …         …         …         …
4993 -2.931897 -2.664816 -2.090137  …  1.757705  2.291923  2.704595
4994 -2.234850 -1.593270 -1.075279  …  1.388947  2.079675  2.433375
4995 -1.664162 -0.935655 -0.866953  … -0.472419 -1.310147 -2.029521
4996 -3.492560 -2.695270 -1.849691  …  1.258419  1.907530  2.280888
4997 -2.622720 -2.044092 -1.295874  … -1.512234 -2.076075 -2.586042

           134       135       136       137       138       139  140
0     0.578621  0.257740  0.228077  0.123431  0.925286  0.193137  1.0
1     0.724046  0.555784  0.476333  0.773820  1.119621 -1.436250  1.0
2    -0.021919 -0.713683 -0.532197  0.321097  0.904227 -0.421797  1.0
3     0.842069  0.952074  0.990133  1.086798  1.403011 -0.383564  1.0
4     1.371682  1.277392  0.960304  0.971020  1.614392  1.421456  1.0
…          …         …         …         …         …         …    …
4993  2.451519  2.017396  1.704358  1.688542  1.629593  1.342651  0.0
4994  2.159484  1.819747  1.534767  1.696818  1.483832  1.047612  0.0
4995 -3.221294 -4.176790 -4.009720 -2.874136 -2.008369 -1.808334  0.0
4996  1.895242  1.437702  1.193433  1.261335  1.150449  0.804932  0.0
4997 -3.322799 -3.627311 -3.437038 -2.260023 -1.577823 -0.684531  0.0

[4998 rows x 142 columns]
```

[82]:
```
## dropping the first column containing the serial no
df = df.drop(df.columns[0], axis=1)
df
```

[82]:
```
             0         1         2         3         4         5         6  \
0    -0.112522 -2.827204 -3.773897 -4.349751 -4.376041 -3.474986 -2.181408
```

```
1    -1.100878 -3.996840 -4.285843 -4.506579 -4.022377 -3.234368 -1.566126
2    -0.567088 -2.593450 -3.874230 -4.584095 -4.187449 -3.151462 -1.742940
3     0.490473 -1.914407 -3.616364 -4.318823 -4.268016 -3.881110 -2.993280
4     0.800232 -0.874252 -2.384761 -3.973292 -4.338224 -3.802422 -2.534510
…          …        …        …        …        …        …        …
4993  0.608558 -0.335651 -0.990948 -1.784153 -2.626145 -2.957065 -2.931897
4994 -2.060402 -2.860116 -3.405074 -3.748719 -3.513561 -3.006545 -2.234850
4995 -1.122969 -2.252925 -2.867628 -3.358605 -3.167849 -2.638360 -1.664162
4996 -0.547705 -1.889545 -2.839779 -3.457912 -3.929149 -3.966026 -3.492560
4997 -1.351779 -2.209006 -2.520225 -3.061475 -3.065141 -3.030739 -2.622720

             7         8         9  …       131       132       133  \
0    -1.818286 -1.250522 -0.477492  …  0.792168  0.933541  0.796958
1    -0.992258 -0.754680  0.042321  …  0.538356  0.656881  0.787490
2    -1.490659 -1.183580 -0.394229  …  0.886073  0.531452  0.311377
3    -1.671131 -1.333884 -0.965629  …  0.350816  0.499111  0.600345
4    -1.783423 -1.594450 -0.753199  …  1.148884  0.958434  1.059025
…          …        …        …  …       …        …        …
4993 -2.664816 -2.090137 -1.461841  …  1.757705  2.291923  2.704595
4994 -1.593270 -1.075279 -0.976047  …  1.388947  2.079675  2.433375
4995 -0.935655 -0.866953 -0.645363  … -0.472419 -1.310147 -2.029521
4996 -2.695270 -1.849691 -1.374321  …  1.258419  1.907530  2.280888
4997 -2.044092 -1.295874 -0.733839  … -1.512234 -2.076075 -2.586042

           134       135       136       137       138       139  140
0     0.578621  0.257740  0.228077  0.123431  0.925286  0.193137  1.0
1     0.724046  0.555784  0.476333  0.773820  1.119621 -1.436250  1.0
2    -0.021919 -0.713683 -0.532197  0.321097  0.904227 -0.421797  1.0
3     0.842069  0.952074  0.990133  1.086798  1.403011 -0.383564  1.0
4     1.371682  1.277392  0.960304  0.971020  1.614392  1.421456  1.0
…          …        …        …        …        …        …  …
4993  2.451519  2.017396  1.704358  1.688542  1.629593  1.342651  0.0
4994  2.159484  1.819747  1.534767  1.696818  1.483832  1.047612  0.0
4995 -3.221294 -4.176790 -4.009720 -2.874136 -2.008369 -1.808334  0.0
4996  1.895242  1.437702  1.193433  1.261335  1.150449  0.804932  0.0
4997 -3.322799 -3.627311 -3.437038 -2.260023 -1.577823 -0.684531  0.0

[4998 rows x 141 columns]
```

0.1.3 Separating the feature and target column

[83]:
```
x = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

[84]:
```
x
```

[84]: array([[-0.11252183, -2.8272038 , -3.7738969 , …, 0.12343082,
          0.92528624, 0.19313742],
        [-1.1008778 , -3.9968398 , -4.2858426 , …, 0.77381971,
          1.1196209 , -1.4362499],
        [-0.56708802, -2.5934502 , -3.8742297 , …, 0.32109663,
          0.90422673, -0.42179659],
        …,
        [-1.1229693 , -2.2529248 , -2.8676281 , …, -2.874136 ,
         -2.0083694 , -1.8083338],
        [-0.54770461, -1.8895451 , -2.8397786 , …, 1.261335 ,
          1.1504486 , 0.80493225],
        [-1.3517791 , -2.2090058 , -2.5202247 , …, -2.2600228 ,
         -1.577823 , -0.68453092]])

[85]:
```
y
```

[85]: array([1., 1., 1., …, 0., 0., 0.])

[86]:
```
print(f"Normal samples: {(y==0).sum()}, \nAbnormal samples: {(y==1).sum()}")
```

```
Normal samples: 2079,
Abnormal samples: 2919
```

0.1.4 Splitting normal and abnormal ECG samples

[87]:
```
x_normal = x[y == 0]
x_abnormal = x[y == 1]
```

0.1.5 Splitting normal data into train and test

[88]:
```
x_train,x_test_normal = train_test_split(x_normal, test_size=0.
↪2,random_state=42)
```

[89]:
```
x_test = np.vstack([x_test_normal, x_abnormal])
y_test = np.hstack([
    np.zeros(len(x_test_normal)),
    np.ones(len(x_abnormal))
])
```

[90]:
```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

[91]:
```
print(f"Train shape: {x_train_scaled.shape}, Test shape: {x_test_scaled.shape}")
```

```
Train shape: (1663, 140), Test shape: (3335, 140)
```

0.1.6 Building the model

[92]:
```
input_dim = x.shape[1]

autoencoder = Sequential([
    Dense(256, activation = 'relu', input_shape = (input_dim,)),
    Dense(128, activation = 'relu'),
    Dense(64, activation = 'relu'),
    Dense(32, activation = 'relu'),
    Dense(64, activation = 'relu'),
    Dense(128, activation = 'relu'),
    Dense(256, activation = 'relu'),
    Dense(input_dim, activation='linear')
])

autoencoder.summary()
```

```
c:\Users\TUFAN\.conda\envs\tf_env\lib\site-
packages\keras\src\layers\core\dense.py:93: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_22 (Dense)	(None, 256)	36,096
dense_23 (Dense)	(None, 128)	32,896
dense_24 (Dense)	(None, 64)	8,256
dense_25 (Dense)	(None, 32)	2,080
dense_26 (Dense)	(None, 64)	2,112
dense_27 (Dense)	(None, 128)	8,320
dense_28 (Dense)	(None, 256)	33,024
dense_29 (Dense)	(None, 140)	35,980

Total params: 158,764 (620.17 KB)

Trainable params: 158,764 (620.17 KB)

Non-trainable params: 0 (0.00 B)

[93]:
```
autoencoder.compile(optimizer='adam', loss = 'mse', metrics=['mae'])
```

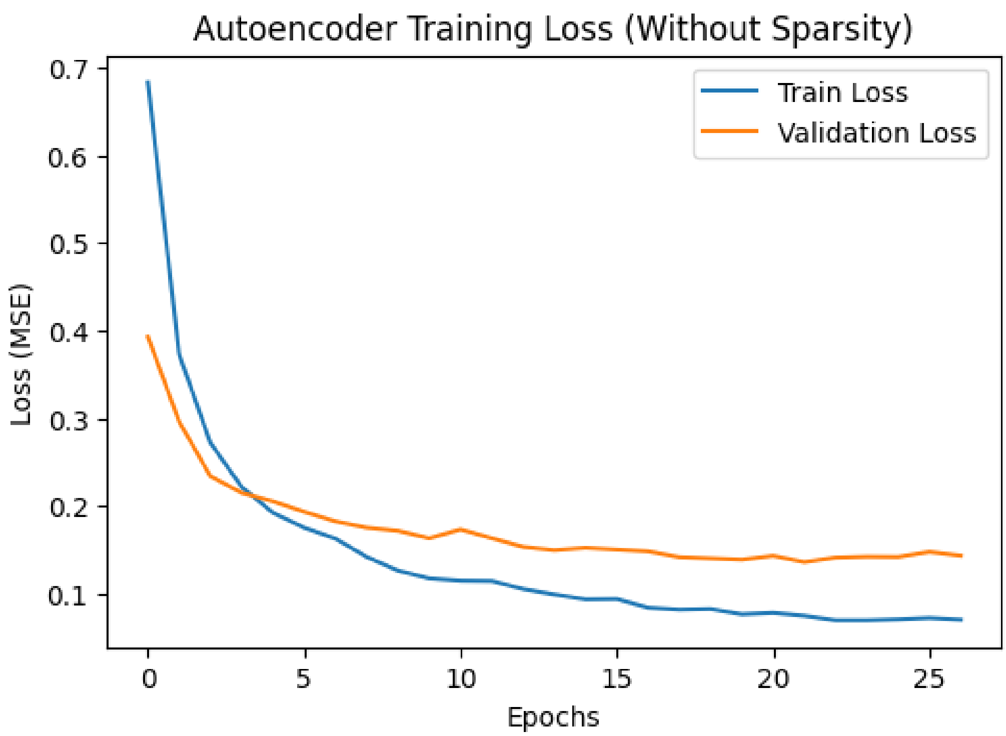
[94]:
```
estop = EarlyStopping(monitor='val_loss', patience = 5, verbose = 1,␣
 ↪restore_best_weights = True )
history = autoencoder.fit(
    x_train_scaled, x_train_scaled,
    epochs=50,
    batch_size=32,
    validation_split=0.1,
    callbacks = [estop],
    verbose=1
)
```

```
Epoch 1/50
47/47                2s 8ms/step - loss:
0.8639 - mae: 0.6321 - val_loss: 0.3930 - val_mae: 0.4205
Epoch 2/50
47/47                0s 4ms/step - loss:
0.4076 - mae: 0.4054 - val_loss: 0.2963 - val_mae: 0.3455
Epoch 3/50
47/47                0s 4ms/step - loss:
0.3001 - mae: 0.3370 - val_loss: 0.2339 - val_mae: 0.2954
Epoch 4/50
47/47                0s 5ms/step - loss:
0.2178 - mae: 0.2869 - val_loss: 0.2152 - val_mae: 0.2796
Epoch 5/50
47/47                0s 4ms/step - loss:
0.1815 - mae: 0.2631 - val_loss: 0.2054 - val_mae: 0.2663
Epoch 6/50
47/47                0s 3ms/step - loss:
0.1630 - mae: 0.2553 - val_loss: 0.1935 - val_mae: 0.2568
Epoch 7/50
47/47                0s 4ms/step - loss:
0.1763 - mae: 0.2542 - val_loss: 0.1825 - val_mae: 0.2538
Epoch 8/50
47/47                0s 4ms/step - loss:
0.1555 - mae: 0.2420 - val_loss: 0.1754 - val_mae: 0.2411
Epoch 9/50
47/47                0s 4ms/step - loss:
0.1339 - mae: 0.2251 - val_loss: 0.1718 - val_mae: 0.2370
Epoch 10/50
47/47                0s 4ms/step - loss:
0.1085 - mae: 0.2112 - val_loss: 0.1634 - val_mae: 0.2314
```

```
Epoch 11/50
47/47            0s 4ms/step - loss:
0.1132 - mae: 0.2127 - val_loss: 0.1733 - val_mae: 0.2378
Epoch 12/50
47/47            0s 4ms/step - loss:
0.1235 - mae: 0.2226 - val_loss: 0.1634 - val_mae: 0.2322
Epoch 13/50
47/47            0s 5ms/step - loss:
0.1017 - mae: 0.2078 - val_loss: 0.1535 - val_mae: 0.2197
Epoch 14/50
47/47            0s 6ms/step - loss:
0.1033 - mae: 0.2069 - val_loss: 0.1497 - val_mae: 0.2175
Epoch 15/50
47/47            0s 3ms/step - loss:
0.0908 - mae: 0.1975 - val_loss: 0.1524 - val_mae: 0.2219
Epoch 16/50
47/47            0s 4ms/step - loss:
0.0932 - mae: 0.2037 - val_loss: 0.1504 - val_mae: 0.2183
Epoch 17/50
47/47            0s 4ms/step - loss:
0.0817 - mae: 0.1921 - val_loss: 0.1484 - val_mae: 0.2172
Epoch 18/50
47/47            0s 4ms/step - loss:
0.0759 - mae: 0.1862 - val_loss: 0.1418 - val_mae: 0.2103
Epoch 19/50
47/47            0s 4ms/step - loss:
0.0825 - mae: 0.1926 - val_loss: 0.1404 - val_mae: 0.2089
Epoch 20/50
47/47            0s 4ms/step - loss:
0.0759 - mae: 0.1827 - val_loss: 0.1389 - val_mae: 0.2111
Epoch 21/50
47/47            0s 4ms/step - loss:
0.0771 - mae: 0.1878 - val_loss: 0.1433 - val_mae: 0.2133
Epoch 22/50
47/47            0s 3ms/step - loss:
0.0739 - mae: 0.1843 - val_loss: 0.1362 - val_mae: 0.2033
Epoch 23/50
47/47            0s 4ms/step - loss:
0.0649 - mae: 0.1744 - val_loss: 0.1413 - val_mae: 0.2097
Epoch 24/50
47/47            0s 3ms/step - loss:
0.0711 - mae: 0.1797 - val_loss: 0.1423 - val_mae: 0.2126
Epoch 25/50
47/47            0s 4ms/step - loss:
0.0695 - mae: 0.1794 - val_loss: 0.1420 - val_mae: 0.2087
Epoch 26/50
47/47            0s 4ms/step - loss:
0.0721 - mae: 0.1825 - val_loss: 0.1478 - val_mae: 0.2207
```

```
Epoch 27/50
47/47              0s 4ms/step - loss:
0.0702 - mae: 0.1814 - val_loss: 0.1435 - val_mae: 0.2097
Epoch 27: early stopping
Restoring model weights from the end of the best epoch: 22.
```

[95]:
```
# Plot training history
plt.figure(figsize=(6,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Autoencoder Training Loss (Without Sparsity)')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.show()
```



0.1.7 Anomaly detection using reconstruction error

[96]:
```
def reconstruction_errors(model, data):
    reconstructed = model.predict(data, verbose=0)
    errors = np.mean((data - reconstructed) ** 2, axis=1)
    return errors
```

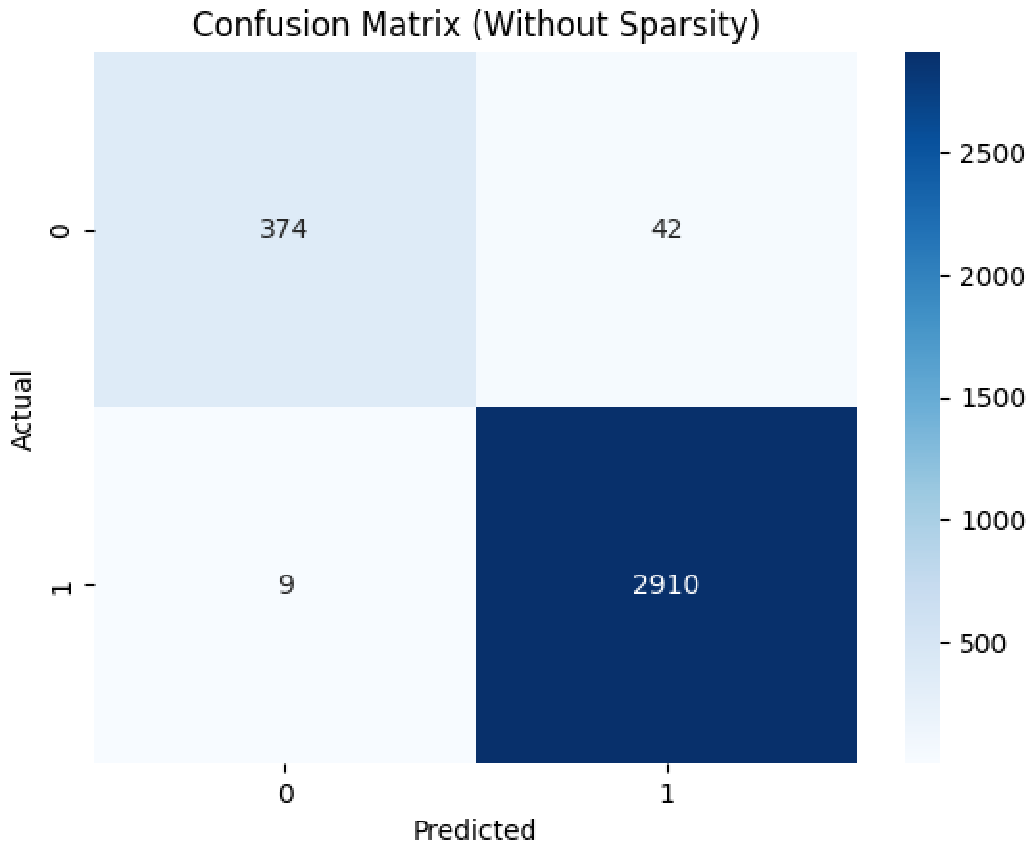
[97]:
```python
## error on test test
errors = reconstruction_errors(autoencoder, x_test_scaled)

# Computing threshold using normal data
normal_errors = reconstruction_errors(autoencoder, x_test_scaled[y_test == 0])
threshold = np.percentile(normal_errors, 90)
print(f"Detection threshold: {threshold:.6f}")

## Flagging the anomalies
y_pred = (errors > threshold).astype(int)
```

```
Detection threshold: 0.104929
```

[98]:
```python
print("\nClassification Report (Without Sparsity):")
print(classification_report(y_test, y_pred))

print(f"ROC-AUC Score: {roc_auc_score(y_test, errors):.4f}")
```

```

Classification Report (Without Sparsity):
              precision    recall  f1-score   support

         0.0       0.98      0.90      0.94       416
         1.0       0.99      1.00      0.99      2919

    accuracy                           0.98      3335
   macro avg       0.98      0.95      0.96      3335
weighted avg       0.98      0.98      0.98      3335

ROC-AUC Score: 0.9480
```

[99]:
```python
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix (Without Sparsity)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



0.2 Question 2: Include sparsity regularization and compare your results.

0.2.1 Building the sparse autoencoder model

[100]:
```
sparse_autoencoder = Sequential([
    Dense(256, activation = 'relu', input_shape = (input_dim,),
          activity_regularizer = regularizers.l1(1e-4)),
    Dense(128, activation = 'relu',activity_regularizer = regularizers.
 ↪l1(1e-4)),
    Dense(64, activation = 'relu',activity_regularizer = regularizers.l1(1e-4)),
    Dense(32, activation = 'relu',activity_regularizer = regularizers.l1(1e-4)),
    Dense(64, activation = 'relu'),
    Dense(128, activation = 'relu'),
    Dense(256, activation = 'relu'),
    Dense(input_dim, activation='linear')
])

sparse_autoencoder.summary()
sparse_autoencoder.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
c:\Users\TUFAN\.conda\envs\tf_env\lib\site-
packages\keras\src\layers\core\dense.py:93: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 256)	36,096
dense_31 (Dense)	(None, 128)	32,896
dense_32 (Dense)	(None, 64)	8,256
dense_33 (Dense)	(None, 32)	2,080
dense_34 (Dense)	(None, 64)	2,112
dense_35 (Dense)	(None, 128)	8,320
dense_36 (Dense)	(None, 256)	33,024
dense_37 (Dense)	(None, 140)	35,980

Total params: 158,764 (620.17 KB)

Trainable params: 158,764 (620.17 KB)

Non-trainable params: 0 (0.00 B)

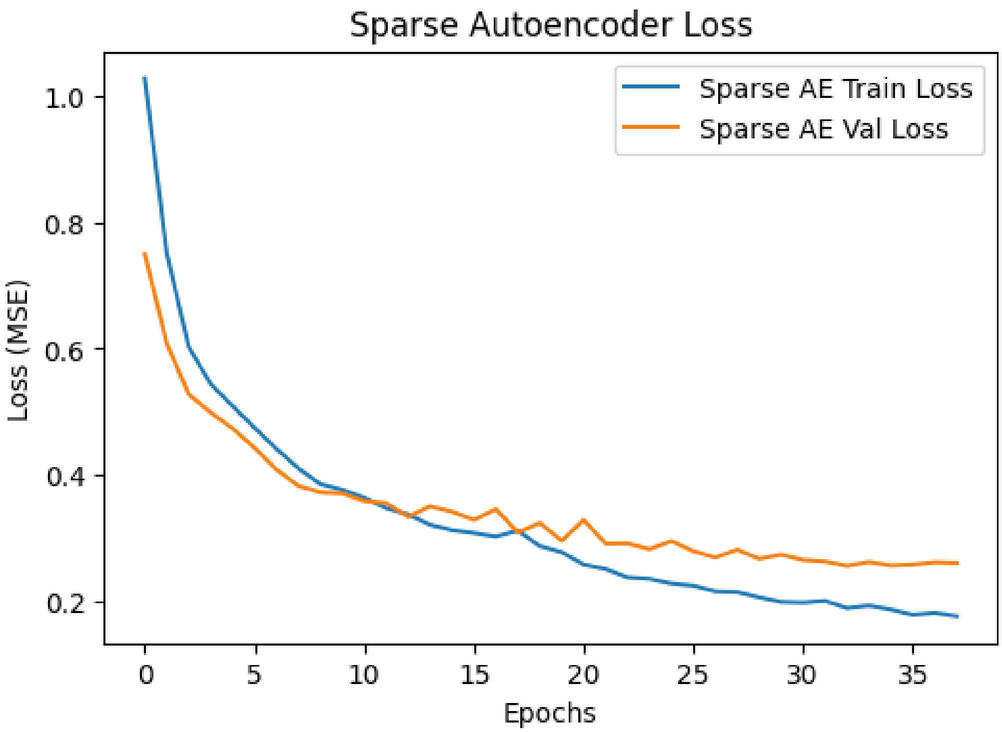
[101]:
```
history_sparse = sparse_autoencoder.fit(
    x_train_scaled, x_train_scaled,
    epochs=50,
    batch_size=32,
    validation_split=0.1,
    callbacks = [estop],
    verbose=1
)

# Plot training history comparison
plt.figure(figsize=(6,4))
```

```
plt.plot(history_sparse.history['loss'], label='Sparse AE Train Loss')
plt.plot(history_sparse.history['val_loss'], label='Sparse AE Val Loss')
plt.title('Sparse Autoencoder Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.show()
```

```
Epoch 1/50
47/47            2s 7ms/step - loss:
1.2393 - mae: 0.6909 - val_loss: 0.7497 - val_mae: 0.5672
Epoch 2/50
47/47            0s 4ms/step - loss:
0.7218 - mae: 0.5470 - val_loss: 0.6087 - val_mae: 0.4791
Epoch 3/50
47/47            0s 3ms/step - loss:
0.5971 - mae: 0.4668 - val_loss: 0.5281 - val_mae: 0.4544
Epoch 4/50
47/47            0s 4ms/step - loss:
0.5515 - mae: 0.4494 - val_loss: 0.4991 - val_mae: 0.4394
Epoch 5/50
47/47            0s 3ms/step - loss:
0.5420 - mae: 0.4375 - val_loss: 0.4740 - val_mae: 0.4176
Epoch 6/50
47/47            0s 5ms/step - loss:
0.4928 - mae: 0.4206 - val_loss: 0.4433 - val_mae: 0.4008
Epoch 7/50
47/47            0s 3ms/step - loss:
0.4494 - mae: 0.3887 - val_loss: 0.4089 - val_mae: 0.3729
Epoch 8/50
47/47            0s 4ms/step - loss:
0.4296 - mae: 0.3732 - val_loss: 0.3830 - val_mae: 0.3585
Epoch 9/50
47/47            0s 3ms/step - loss:
0.4025 - mae: 0.3652 - val_loss: 0.3729 - val_mae: 0.3507
Epoch 10/50
47/47            0s 6ms/step - loss:
0.3508 - mae: 0.3437 - val_loss: 0.3712 - val_mae: 0.3451
Epoch 11/50
47/47            0s 4ms/step - loss:
0.3764 - mae: 0.3429 - val_loss: 0.3593 - val_mae: 0.3397
Epoch 12/50
47/47            0s 3ms/step - loss:
0.3672 - mae: 0.3407 - val_loss: 0.3559 - val_mae: 0.3355
Epoch 13/50
47/47            0s 4ms/step - loss:
0.3309 - mae: 0.3244 - val_loss: 0.3342 - val_mae: 0.3296
Epoch 14/50
```

```
47/47 0s 4ms/step - loss:
0.3097 - mae: 0.3209 - val_loss: 0.3507 - val_mae: 0.3377
Epoch 15/50
47/47 0s 3ms/step - loss:
0.3028 - mae: 0.3227 - val_loss: 0.3422 - val_mae: 0.3306
Epoch 16/50
47/47 0s 4ms/step - loss:
0.2948 - mae: 0.3175 - val_loss: 0.3292 - val_mae: 0.3247
Epoch 17/50
47/47 0s 3ms/step - loss:
0.3019 - mae: 0.3177 - val_loss: 0.3462 - val_mae: 0.3338
Epoch 18/50
47/47 0s 3ms/step - loss:
0.3101 - mae: 0.3217 - val_loss: 0.3093 - val_mae: 0.3134
Epoch 19/50
47/47 0s 5ms/step - loss:
0.2767 - mae: 0.3079 - val_loss: 0.3242 - val_mae: 0.3162
Epoch 20/50
47/47 0s 4ms/step - loss:
0.2804 - mae: 0.2997 - val_loss: 0.2964 - val_mae: 0.3101
Epoch 21/50
47/47 0s 4ms/step - loss:
0.2342 - mae: 0.2929 - val_loss: 0.3292 - val_mae: 0.3179
Epoch 22/50
47/47 0s 4ms/step - loss:
0.2467 - mae: 0.2897 - val_loss: 0.2917 - val_mae: 0.3025
Epoch 23/50
47/47 0s 3ms/step - loss:
0.2239 - mae: 0.2811 - val_loss: 0.2919 - val_mae: 0.3066
Epoch 24/50
47/47 0s 4ms/step - loss:
0.2503 - mae: 0.2900 - val_loss: 0.2831 - val_mae: 0.2956
Epoch 25/50
47/47 0s 3ms/step - loss:
0.2252 - mae: 0.2788 - val_loss: 0.2957 - val_mae: 0.3025
Epoch 26/50
47/47 0s 4ms/step - loss:
0.2242 - mae: 0.2768 - val_loss: 0.2798 - val_mae: 0.2959
Epoch 27/50
47/47 0s 4ms/step - loss:
0.2072 - mae: 0.2720 - val_loss: 0.2699 - val_mae: 0.2895
Epoch 28/50
47/47 0s 3ms/step - loss:
0.1985 - mae: 0.2631 - val_loss: 0.2821 - val_mae: 0.2943
Epoch 29/50
47/47 0s 4ms/step - loss:
0.2140 - mae: 0.2714 - val_loss: 0.2678 - val_mae: 0.2838
Epoch 30/50
```

```
47/47              0s 3ms/step - loss:
0.1952 - mae: 0.2611 - val_loss: 0.2744 - val_mae: 0.2861
Epoch 31/50
47/47              0s 4ms/step - loss:
0.2044 - mae: 0.2669 - val_loss: 0.2658 - val_mae: 0.2816
Epoch 32/50
47/47              0s 3ms/step - loss:
0.1888 - mae: 0.2545 - val_loss: 0.2634 - val_mae: 0.2826
Epoch 33/50
47/47              0s 4ms/step - loss:
0.1935 - mae: 0.2542 - val_loss: 0.2565 - val_mae: 0.2803
Epoch 34/50
47/47              0s 3ms/step - loss:
0.1930 - mae: 0.2575 - val_loss: 0.2626 - val_mae: 0.2842
Epoch 35/50
47/47              0s 3ms/step - loss:
0.1817 - mae: 0.2495 - val_loss: 0.2570 - val_mae: 0.2848
Epoch 36/50
47/47              0s 4ms/step - loss:
0.1787 - mae: 0.2504 - val_loss: 0.2584 - val_mae: 0.2765
Epoch 37/50
47/47              0s 4ms/step - loss:
0.1797 - mae: 0.2509 - val_loss: 0.2621 - val_mae: 0.2856
Epoch 38/50
47/47              0s 5ms/step - loss:
0.1809 - mae: 0.2511 - val_loss: 0.2611 - val_mae: 0.2800
Epoch 38: early stopping
Restoring model weights from the end of the best epoch: 33.
```

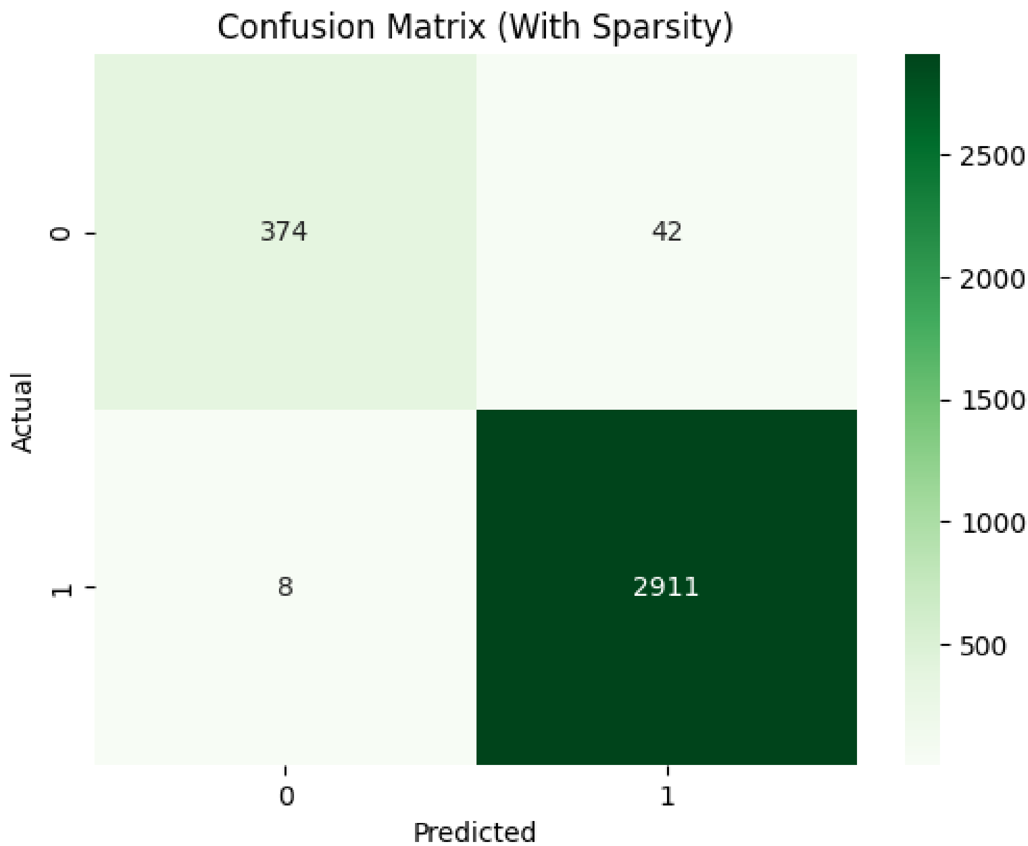


0.2.2 Evaluating Sparse Autoencoder

[103]:
```
sparse_errors = reconstruction_errors(sparse_autoencoder, x_test_scaled)

# new threshold
normal_sparse_errors = reconstruction_errors(sparse_autoencoder,␣
 ↪x_test_scaled[y_test == 0])
threshold_sparse = np.percentile(normal_sparse_errors, 90)
print(f"Sparse AE Threshold: {threshold_sparse:.6f}")

# Predictions
y_pred_sparse = (sparse_errors > threshold_sparse).astype(int)

print("\nClassification Report (With Sparsity):")
print(classification_report(y_test, y_pred_sparse))
print(f"ROC-AUC Score (Sparse AE): {roc_auc_score(y_test, sparse_errors):.4f}")

# Confusion Matrix
cm_sparse = confusion_matrix(y_test, y_pred_sparse)
sns.heatmap(cm_sparse, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix (With Sparsity)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.show()
```

```
Sparse AE Threshold: 0.259583

Classification Report (With Sparsity):
              precision    recall  f1-score   support

         0.0       0.98      0.90      0.94       416
         1.0       0.99      1.00      0.99      2919

    accuracy                           0.99      3335
   macro avg       0.98      0.95      0.96      3335
weighted avg       0.98      0.99      0.98      3335

ROC-AUC Score (Sparse AE): 0.9622
```



0.3 Comparison:

Model Type	Accuracy	ROC–AUC
Standard Autoencoder	**98%**	**0.948**
Sparse Autoencoder	**99%**	**0.962**

- Incorporating sparsity regularization significantly enhanced the model's ability to distinguish between normal and abnormal ECG signals by promoting compact and discriminative feature learning.