In [1]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
@author: M Arshad Zahangir Chowdhury

Identify experiments, plot roc, pr curve, grad cam maps etc.

"""

%matplotlib inline

import sys
import os
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy import signal
from ipywidgets import interactive
import seaborn as sns  #heat map
import glob # batch processing of images

if '../../' not in sys.path:
    sys.path.append('../../')

import math
from scipy import signal
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score

import itertools

from vocnet.misc.utils import classifier_internals
from vocnet.misc.utils import clf_post_processor


from vocnet.spectral_datasets.IR_datasets import IR_data
from vocnet.spectral_datasets.IR_datasets import spectra_to_img
from vocnet.spectral_datasets.THz_datasets import THz_data

from vocnet.misc.aperture import publication_fig
from vocnet.misc.voc_net_utils import multiclass_roc_auc_score
from vocnet.misc.voc_net_utils import plot_raw_scores
from vocnet.misc.voc_net_utils import simple_spectrum_fig
from vocnet.misc.voc_net_utils import simple_plot_raw_scores

from vocnet.misc.voc_net_utils import plot_sequential_group_prediction
```

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models


# GPU_mem_limit=1.0
# gpus = tf.config.experimental.list_physical_devices('GPU')
# if gpus:
#     try:
#         tf.config.experimental.set_virtual_device_configuration(gpus[

#     except RuntimeError as e:
#         print(e)


# !pip install git+https://github.com/tensorflow/docs

import tensorflow_docs as tfdocs
import tensorflow_docs.modeling
import tensorflow_docs.plots
import tensorflow_docs.modeling
from tensorflow.keras import regularizers

from vocnet.models.voc_net_models import get_callbacks
from vocnet.models.voc_net_models import get_optimizer
from vocnet.models.voc_net_models import compile_and_fit


from vocnet.models.voc_net_models import C1f1k3_AP1_D12
from vocnet.models.voc_net_models import C1f1k3_MP1_D12

from vocnet.models.voc_net_models import C2f1k3_AP1_D12
from vocnet.models.voc_net_models import C2f1k3_AP1_D48_D12
from vocnet.models.voc_net_models import C2f1k3_AP2_D48_D12

from vocnet.models.voc_net_models import C2f3k3_AP1_D48_D12
from vocnet.models.voc_net_models import C2f3k3_AP1_D6_D12

from vocnet.models.voc_net_models import C1f1k3_AP1_RD50_D12
from vocnet.models.voc_net_models import C1f1k3_AP1_D48_RL1_D12
from vocnet.models.voc_net_models import C2f3k3_AP1_D48_RD50_D12
from vocnet.models.voc_net_models import C2f3k3_AP1_D48_RL1_D12
from vocnet.models.voc_net_models import C2f3k3_AP1_D48_RL1_RD50_D12



from tensorflow import keras
import keras_tuner as kt

import random


#Set random seed
os.environ['PYTHONHASHSEED'] = str(42)
os.environ['TF_DETERMINISTIC_OPS'] = '1'
tf.random.set_seed(42)
tf.random.get_global_generator().reset_from_seed(42)
```

```
np.random.seed(42)
random.seed(42)



# os.environ['CUDA_VISIBLE_DEVICES'] = '-1'

if tf.test.gpu_device_name():
    print('GPU found')
else:
    print("No GPU found")
```

```
2022-07-12 11:49:36.725092: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcudart.s
o.11.0

GPU found

2022-07-12 11:49:37.664748: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcuda.so.1
2022-07-12 11:49:37.720967: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1733] Found device 0 with properties:
pciBusID: 0000:65:00.0 name: Quadro RTX 4000 computeCapability: 7.5
coreClock: 1.545GHz coreCount: 36 deviceMemorySize: 7.79GiB deviceMemo
ryBandwidth: 387.49GiB/s
2022-07-12 11:49:37.721042: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcudart.s
o.11.0
2022-07-12 11:49:37.728621: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcublas.s
o.11
2022-07-12 11:49:37.728735: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcublasLt.
so.11
2022-07-12 11:49:37.730550: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcufft.so.
10
2022-07-12 11:49:37.731001: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcurand.s
o.10
2022-07-12 11:49:37.732034: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcusolver.
so.11
2022-07-12 11:49:37.733396: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcusparse.
so.11
2022-07-12 11:49:37.733572: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcudnn.so.
8
2022-07-12 11:49:37.734733: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1871] Adding visible gpu devices: 0
2022-07-12 11:49:37.735346: I tensorflow/core/platform/cpu_feature_gua
rd.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the approp
```

```
riate compiler flags.
2022-07-12 11:49:37.737332: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1733] Found device 0 with properties:
pciBusID: 0000:65:00.0 name: Quadro RTX 4000 computeCapability: 7.5
coreClock: 1.545GHz coreCount: 36 deviceMemorySize: 7.79GiB deviceMemo
ryBandwidth: 387.49GiB/s
2022-07-12 11:49:37.739039: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1871] Adding visible gpu devices: 0
2022-07-12 11:49:37.739153: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcudart.s
o.11.0
2022-07-12 11:49:38.158292: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1258] Device interconnect StreamExecutor with strength 1 edge
matrix:
2022-07-12 11:49:38.158319: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1264]      0
2022-07-12 11:49:38.158325: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1277] 0:   N
2022-07-12 11:49:38.159552: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1418] Created TensorFlow device (/job:localhost/replica:0/tas
k:0/device:GPU:0 with 5934 MB memory) -> physical GPU (device: 0, nam
e: Quadro RTX 4000, pci bus id: 0000:65:00.0, compute capability: 7.5)
2022-07-12 11:49:38.225946: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1733] Found device 0 with properties:
pciBusID: 0000:65:00.0 name: Quadro RTX 4000 computeCapability: 7.5
coreClock: 1.545GHz coreCount: 36 deviceMemorySize: 7.79GiB deviceMemo
ryBandwidth: 387.49GiB/s
2022-07-12 11:49:38.226346: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1871] Adding visible gpu devices: 0
2022-07-12 11:49:38.226364: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1258] Device interconnect StreamExecutor with strength 1 edge
matrix:
2022-07-12 11:49:38.226368: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1264]      0
2022-07-12 11:49:38.226371: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1277] 0:   N
2022-07-12 11:49:38.226722: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1418] Created TensorFlow device (/device:GPU:0 with 5934 MB m
emory) -> physical GPU (device: 0, name: Quadro RTX 4000, pci bus id:
0000:65:00.0, compute capability: 7.5)
```

In [2]:
```
s = THz_data(resolution=0.016, verbosity = False)
s.load_THz_data()
# s.dataset_info()
X = s.spectra
y = s.targets

X=np.expand_dims(X, 1)
```

In [3]:
```
#split intro train and test set

#seeds used 123,237, 786
from sklearn.model_selection import train_test_split

TRAIN_SIZE=0.70
TEST_SIZE=1-TRAIN_SIZE
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=TR
                                                    test_size=TEST_SIZE,
                                                    random_state=786,
                                                    stratify=y
                                                    )

print("All:", np.bincount(y) / float(len(y))*100  )
print("Training:", np.bincount(y_train) / float(len(y_train))*100  )
print("Testing:", np.bincount(y_test) / float(len(y_test))*100  )
```

```
All: [8.33333333 8.33333333 8.33333333 8.33333333 8.33333333 8.3333333
3
 8.33333333 8.33333333 8.33333333 8.33333333 8.33333333 8.33333333]
Training: [8.35148874 8.35148874 8.35148874 8.35148874 8.35148874 8.35
148874
 8.35148874 8.2788671  8.35148874 8.35148874 8.2788671  8.2788671 ]
Testing: [8.29103215 8.29103215 8.29103215 8.29103215 8.29103215 8.291
03215
 8.29103215 8.46023689 8.29103215 8.29103215 8.46023689 8.46023689]
```

build the best model here

In [4]:

```
def voc_net():

    model = models.Sequential()

    # C1 Convolutional Layer
    model.add(layers.Conv1D(filters = 3 , kernel_size=3, activation='re

    # S2 Subsampling Layer
    model.add(layers.AveragePooling1D(pool_size = 2, strides = 2, paddi

    # C3 Convolutional Layer
    model.add(layers.Conv1D(filters = 3 , kernel_size=3, activation='re

    # Flatten the CNN output to feed it with fully connected layers
    model.add(layers.Flatten())

    model.add(layers.Dense(48, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(12))  # number of dense layer would be equal



    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_log
                 metrics=[
              tf.keras.losses.SparseCategoricalCrossentropy(
                  from_logits=True, name='SparseCatCrossentropy'),
              'accuracy'])

    model.summary()
```

In [ ]:
```
    return model
```

In [5]:
```
s.load_experiments()
Xexp = s.exp_spectra
yexp = s.exp_targets
SpectraFrame = pd.DataFrame(s.exp_spectra)
SpectraFrame['labels'] = [s.labels[i] for i in s.exp_targets]
SpectraFrame['targets'] =  s.exp_targets
spectraframe = SpectraFrame
Xexp = np.expand_dims(Xexp,-1)
print(yexp)

print(s.labels)
print([s.labels[i] for i in s.exp_targets])
```

```
Number of Experimental Compounds: 6
Number of Spectrum: 6
Total Number of Spectra: 36
Sample Size of training data: 229
Rows discarded: 22
[10 10 10 10 10 10  2  2  2  2  2  2  1  1  1  1  1  1  0  0  0  0  0
0
  8  8  8  8  8  8 11 11 11 11 11 11]
['CH3Cl', 'CH3OH', 'HCOOH', 'H2CO', 'H2S', 'SO2', 'OCS', 'HCN', 'CH3CN
', 'HNO3', 'C2H5OH', 'CH3CHO']
['C2H5OH', 'C2H5OH', 'C2H5OH', 'C2H5OH', 'C2H5OH', 'C2H5OH', 'HCOOH',
'HCOOH', 'HCOOH', 'HCOOH', 'HCOOH', 'HCOOH', 'CH3OH', 'CH3OH', 'CH3OH
', 'CH3OH', 'CH3OH', 'CH3OH', 'CH3Cl', 'CH3Cl', 'CH3Cl', 'CH3Cl', 'CH3
Cl', 'CH3Cl', 'CH3CN', 'CH3CN', 'CH3CN', 'CH3CN', 'CH3CN', 'CH3CN', 'C
H3CHO', 'CH3CHO', 'CH3CHO', 'CH3CHO', 'CH3CHO', 'CH3CHO']
```

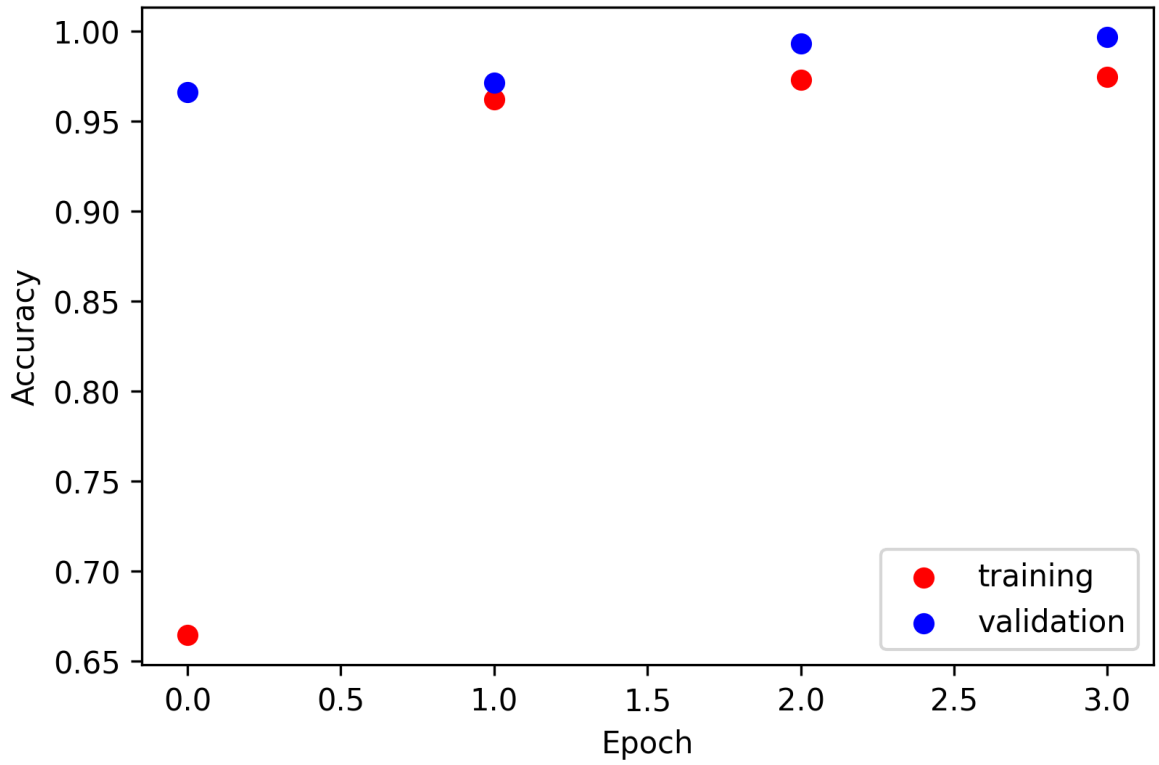
In [6]:
```
model = voc_net()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
C1 (Conv1D)                  (None, 227, 3)            12
_________________________________________________________________
S2 (AveragePooling1D)        (None, 113, 3)            0
_________________________________________________________________
C3 (Conv1D)                  (None, 111, 3)            30
_________________________________________________________________
flatten (Flatten)            (None, 333)               0
_________________________________________________________________
dense (Dense)                (None, 48)                16032
_________________________________________________________________
dropout (Dropout)            (None, 48)                0
_________________________________________________________________
dense_1 (Dense)              (None, 12)                588
=================================================================
Total params: 16,662
Trainable params: 16,662
Non-trainable params: 0
_________________________________________________________________
```

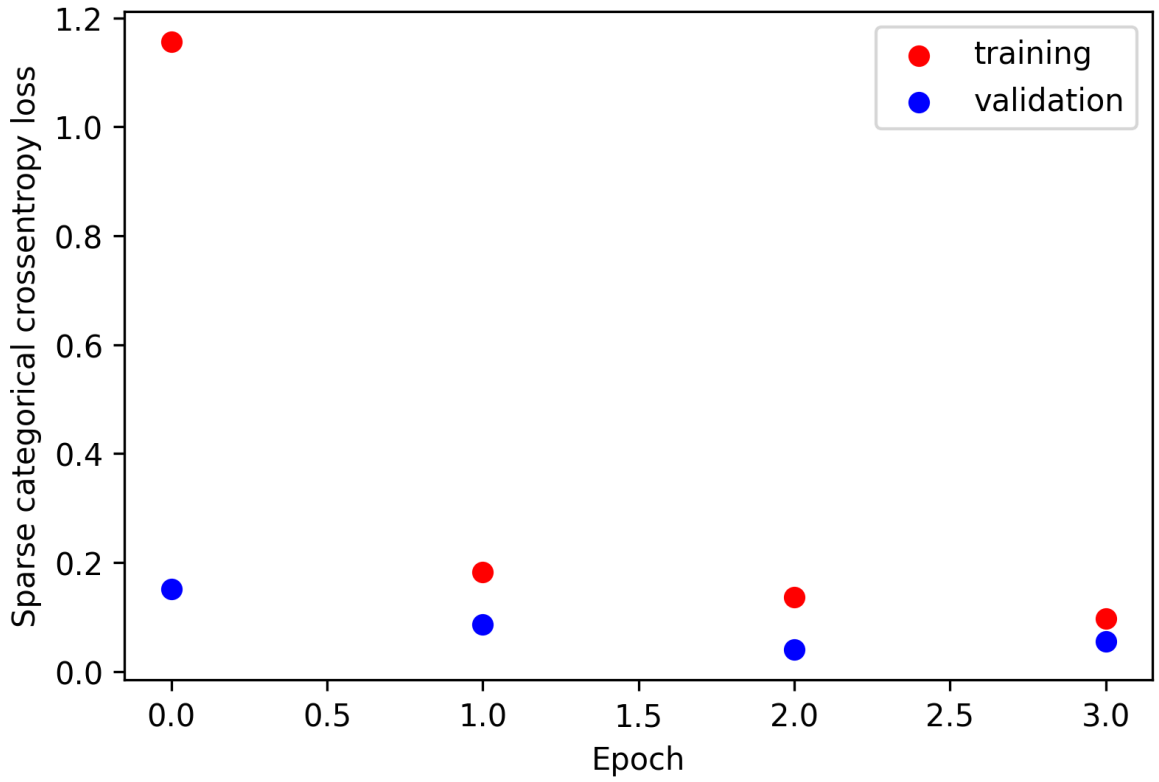
In [7]:
```
# run on CPU for reproducibility, best epoch is 4.
with tf.device('/CPU:0'):
    stop_early = tf.keras.callbacks.EarlyStopping(monitor='SparseCatCro
    history = model.fit(x_train, y_train, epochs=4, validation_data=(x_
```

```
Epoch 1/4

2022-07-12 11:49:42.998548: I tensorflow/compiler/mlir/mlir_graph_opti
mization_pass.cc:176] None of the MLIR Optimization Passes are enabled
(registered 2)
2022-07-12 11:49:43.016676: I tensorflow/core/platform/profile_utils/c
pu_utils.cc:114] CPU Frequency: 3600000000 Hz

44/44 [==============================] - 1s 12ms/step - loss: 1.1818 -
SparseCatCrossentropy: 1.1559 - accuracy: 0.6645 - val_loss: 0.1511 -
val_SparseCatCrossentropy: 0.1514 - val_accuracy: 0.9662
Epoch 2/4
44/44 [==============================] - 0s 10ms/step - loss: 0.1868 -
SparseCatCrossentropy: 0.1832 - accuracy: 0.9622 - val_loss: 0.0871 -
val_SparseCatCrossentropy: 0.0870 - val_accuracy: 0.9712
Epoch 3/4
44/44 [==============================] - 0s 9ms/step - loss: 0.1351 -
SparseCatCrossentropy: 0.1361 - accuracy: 0.9731 - val_loss: 0.0404 -
val_SparseCatCrossentropy: 0.0404 - val_accuracy: 0.9932
Epoch 4/4
44/44 [==============================] - 0s 9ms/step - loss: 0.0993 -
SparseCatCrossentropy: 0.0971 - accuracy: 0.9746 - val_loss: 0.0544 -
val_SparseCatCrossentropy: 0.0550 - val_accuracy: 0.9966
```

In [8]:
```
plt.figure(dpi=300)
plt.scatter(history.epoch,history.history['accuracy'], color = 'red', l
plt.scatter(history.epoch,history.history['val_accuracy'], color = 'blu
plt.legend(loc=4)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.savefig(r'RESULTS/results_figures/accuracies.png', bbox_inches='tig
```



In [9]:
```
plt.figure(dpi=300)
plt.scatter(history.epoch,history.history['SparseCatCrossentropy'], col
plt.scatter(history.epoch,history.history['val_SparseCatCrossentropy'],
plt.legend(loc=1)
plt.xlabel('Epoch')
plt.ylabel('Sparse categorical crossentropy loss')
plt.savefig(r'RESULTS/results_figures/sparse_cat_losses.png', bbox_inch
```



In [10]:
```
#check the weights
model.layers[0].trainable_weights
```

Out[10]:
```
[<tf.Variable 'C1/kernel:0' shape=(3, 1, 3) dtype=float32, numpy=
 array([[[ 0.7963775 ,  0.4599409 , -0.17777663]],

        [[ 0.53172255, -0.32887614,  0.8721396 ]],

        [[ 0.9171538 ,  1.0927715 , -0.38431445]]], dtype=float32)>,
 <tf.Variable 'C1/bias:0' shape=(3,) dtype=float32, numpy=array([-0.00
718738, -0.01124741, -0.00894083], dtype=float32)>]
```

In [11]:
```
from tensorflow.keras import backend as K
# with a Sequential model
get_1st_layer_output = K.function([model.layers[0].input],
                                  [model.layers[1].output])
# layer_output = get_1st_layer_output(x_test)
layer_output = get_1st_layer_output(X)
```

```
2022-07-12 11:49:46.322324: I tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:53] Successfully opened dynamic library libcudnn.so.
8
2022-07-12 11:49:46.555653: I tensorflow/stream_executor/cuda/cuda_dn
n.cc:359] Loaded cuDNN version 8204
```

In [12]:
```
model.layers[0].get_config()
```

Out[12]:
```
{'name': 'C1',
 'trainable': True,
 'batch_input_shape': (None, 229, 1),
 'dtype': 'float32',
 'filters': 3,
 'kernel_size': (3,),
 'strides': (1,),
 'padding': 'valid',
 'data_format': 'channels_last',
 'dilation_rate': (1,),
 'groups': 1,
 'activation': 'relu',
 'use_bias': True,
 'kernel_initializer': {'class_name': 'GlorotUniform',
  'config': {'seed': None}},
 'bias_initializer': {'class_name': 'Zeros', 'config': {}},
 'kernel_regularizer': None,
 'bias_regularizer': None,
 'activity_regularizer': None,
 'kernel_constraint': None,
 'bias_constraint': None}
```

In [13]:
```
layer_output[0].shape
```

Out[13]:
```
(1968, 113, 3)
```

In [14]:
```
layer_output[0][0].shape
```

Out[14]:
```
(113, 3)
```

training accuracies vs epoch

In [15]:
```
model.evaluate(x_test, y_test, verbose=2)
```

```
19/19 - 0s - loss: 0.0544 - SparseCatCrossentropy: 0.0550 - accuracy:
0.9966
```

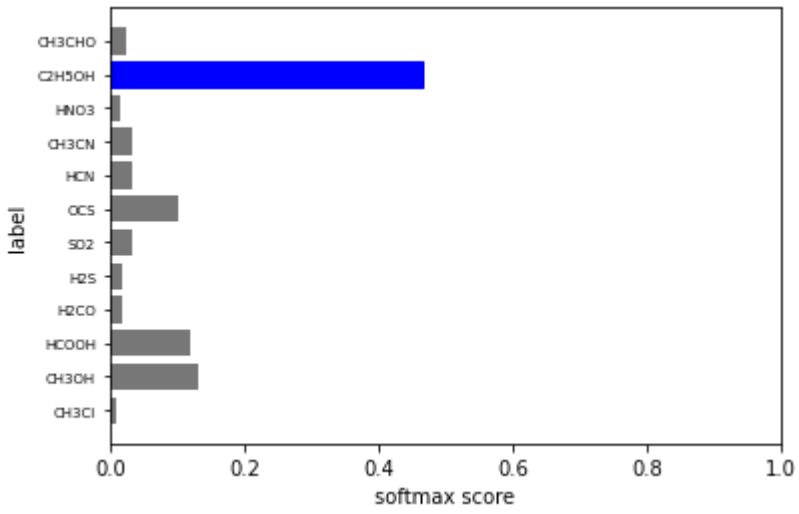
Out[15]:
```
[0.05441968888044357, 0.055005963891744614, 0.9966158866882324]
```

softmax scores

In [16]:
```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

In [17]:
```
predictions = probability_model.predict(x_test)
```

In [18]:
```
i=3
# simple_spectrum_fig(s.frequencies, x_test[i])
simple_plot_raw_scores(i, predictions, y_test,s.labels)
```



In [19]:
```
x_test.shape
```

Out[19]: (591, 229, 1)

In [20]:
```
np.squeeze(x_test).shape
```

Out[20]: (591, 229)

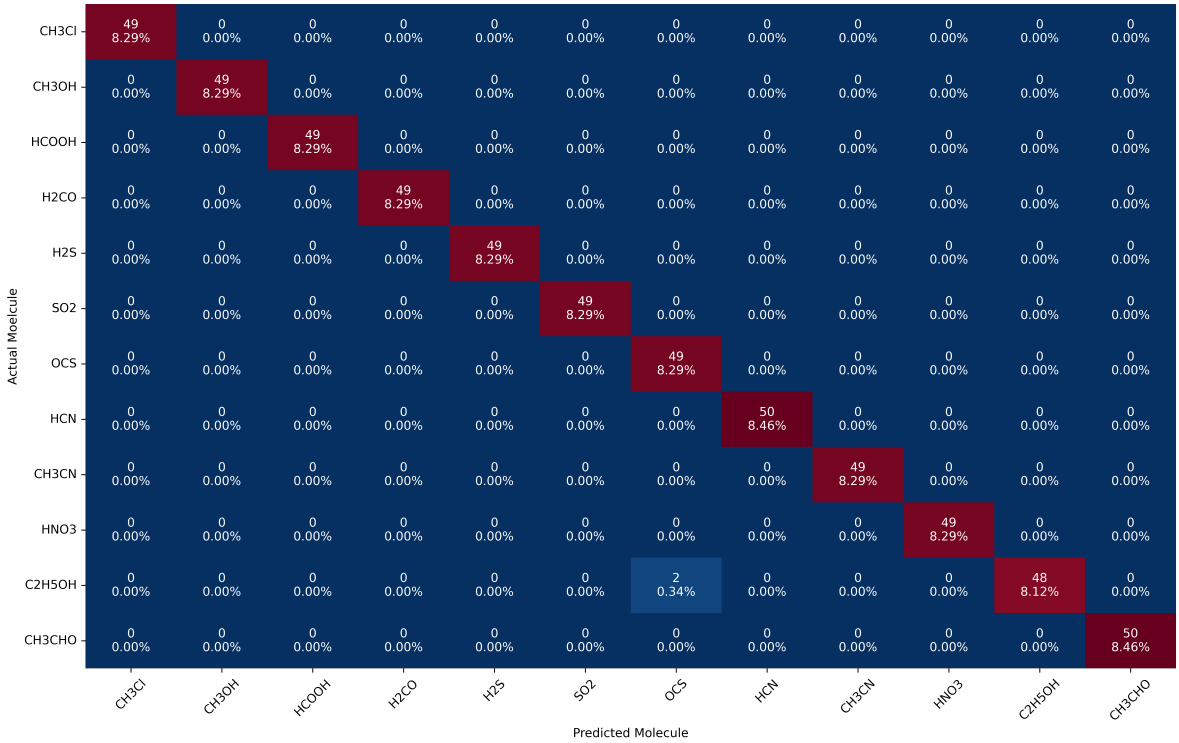
In [21]:
```
y_test.shape
```

Out[21]: (591,)

In [22]:
```
pred_y=np.argmax(model.predict(x_test), axis=-1)
```

In [23]:
```
cm = confusion_matrix(y_test, pred_y)

group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]

annot_labels = [f"{v1}\n{v2}" for v1, v2 in
          zip(group_counts,group_percentages)]

fig = plt.figure(figsize=(16,10), dpi = 600);
# plt.title('Confusion matrix);

# ax = sns.heatmap(cm, annot=True, cmap='PiYG');   #cmap='coolwarm' als
ax = sns.heatmap(cm/np.sum(cm), annot=np.asarray(annot_labels).reshape(
#ax = sns.heatmap(cm/np.sum(cm), annot=True, fmt='.2%', cmap='Blues') #
ax.set_xticklabels(s.labels);
```

```
ax.set_yticklabels(s.labels);
plt.xlabel('Predicted Molecule');
plt.ylabel('Actual Moelcule');
plt.xticks(rotation=45);
plt.yticks(rotation=0);
plt.savefig(r'RESULTS/results_figures/cm_test_data.png', bbox_inches='t
```



In [24]:
```
classifier_internals(pred_y, y_test, y_train, 'simple_CNN')
```

```
---------------------------- simple_CNN
------------------------------
Fraction Correct[Accuracy]:
0.9966159052453468
Samples Correctly Classified:
(array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  1
2,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  2
5,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  3
8,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  5
1,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  6
4,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  7
7,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  9
0,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 10
3,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 11
6,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 12
9,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 14
2,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 15
5,
       156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 16
8,
       169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 18
1,
       182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 19
4,
       195, 196, 197, 198, 199, 200, 202, 203, 204, 205, 206, 207, 20
8,
       209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 22
1,
       222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 23
4,
       235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 24
7,
       248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26
0,
       261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 27
3,
       274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 28
6,
       287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 29
9,
       300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 31
2,
       313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 32
5,
       326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 33
```

```
8,
       339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 35
1,
       352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 36
4,
       365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 37
7,
       378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 39
0,
       391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 40
3,
       404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 41
6,
       417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 42
9,
       430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 44
2,
       443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 45
5,
       456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 46
8,
       469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 48
1,
       482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 49
4,
       495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 50
7,
       508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 52
1,
       522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 53
4,
       535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 54
7,
       548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 56
0,
       561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 57
3,
       574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 58
6,
       587, 588, 589, 590]),)
Samples Incorrectly Classified:
(array([201, 520]),)
All Train y with label identifier [ 6  3  0 ...  8  0 10]

Data misidentified:

Test y with incorrect indexes label identifier [10 10]

Predicted y with incorrect indexes label identifier [6 6]

Correct data identified:

Test y with correct indexes label identifier [ 0  2  1 10  9  2  8  6
0 10  8  4 11  6  6  4  2  4  4  8  2  0  2 11
  7  5  7  5  9  3  5  8 10  9  9  9  2  1  0  3  4  9  5  1  7  0  8
8
```

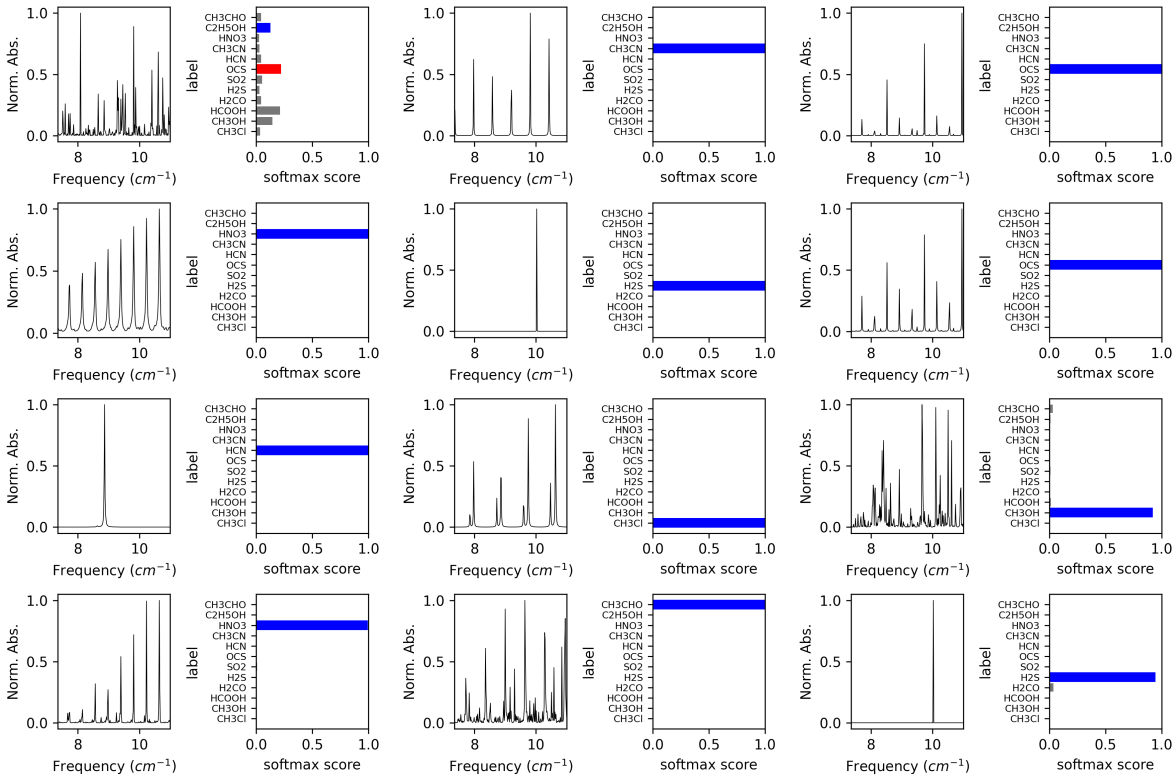
```
  2  3  0  3  7  6  6  1  5  3  2  5 10  8  9  6  5  9  4  7  9  7  2
1
 10  7 10  7  4  5  5  5  0  2  1  6  2  5  0  5 11  7 10  0 11  4  5
10
 11 11  6  6  4  9 11  2  0  6  6  1  2  3  3  5  8  9  7  8  7  0  8
8
  4  6  9 11  1  3  9  8 11  2  2  7  7  3  0 10  6  1  1  3  2 11 10
11
  8  0  2  0  5  7  9 11  8  4 11  1  3 11  2  1  4  0  2  7  6  0  2
3
  4  6  3 11 11  8  4 11  2  4  0 11  9  8  5 10  1  9  8  9  0  0  5
10
  1  9  4  4  3  5 11  6  2  8  6  9  4  6  7  0  1  9 11  4  7 11  4
7
  0  2  2  1  7  1 10  5  3  1 10  6  5  4  7 11  9  0  3  9  7  5  7
8
  8 11  6  7  7  7  4  8  8 10  8 10  1  3  9  3  1  8 11  6  3  5  9
3
  3 11  7 10  7  2 10  6  0  9  4  4  9  1  8 11  0  0  0  1 10  3  1
9
  1  5  7 11  0 10  8  6  7  3 11  1  3  3  3 10  4  2  4  4  0  0  4
9
  8  7 10  0 11  8  3  4  1  9 11  4  0  1  5  5 10  6  8  9  7  9  9
7
  6  2  9 10  3  8  3  1  4  0  2  2  4  5 11  7  2  8  2 11  6  6  0
1
  0  0 10  5 11  0 11  1  4  3  8  3  9  5  7  5  8  3  2  0  3  6  7
11
  6  5  6  7  2  6  5  2  1  9  6 10  1  2  4 11  9  0  2  3  1  0 10
6
 10 10  4  1  6  0  9  1  8  6  5  9 11  1  6  5  2  7 10  3 10  4  0
10
  3 11 10  7  6  8 11  3  6  2  5  1  8  1  4 10  8  3  8 11  3  7 11
2
  4 10  5  1  4 11 10  2  5  2 10  8  7  7  9  6  0  2  9 10  8 10  5
2
 11  8  0  9  5  3  5  1 10  4  4  2  9  4  8  0  8  4  1  5  8 10  7
6
  7  8  1  8  4  7  6  1  5  3  2  5 11  5  6  5 11  5  9  0  2  2  1
1
  9  4  2  2  0  3  6  6 10  0  6  7  9  5  4  3 11 10 11  7  3  0  6
10
 10  8 11  1  3  6  8  3  4  5  3  9  7  0 10  4  7 11  7  5  9 10  9
6
  1  4  8  9  1  7  3 10  1  3  5 11  6]

Predicted y correct indexes label identifier [ 0  2  1 10  9  2  8  6
0 10  8  4 11  6  6  4  2  4  4  8  2  0  2 11
  7  5  7  5  9  3  5  8 10  9  9  9  2  1  0  3  4  9  5  1  7  0  8
8
  2  3  0  3  7  6  6  1  5  3  2  5 10  8  9  6  5  9  4  7  9  7  2
1
 10  7 10  7  4  5  5  5  0  2  1  6  2  5  0  5 11  7 10  0 11  4  5
10
 11 11  6  6  4  9 11  2  0  6  6  1  2  3  3  5  8  9  7  8  7  0  8
8
```

```
  4  6  9 11  1  3  9  8 11  2  2  7  7  3  0 10  6  1  1  3  2 11 10
11
  8  0  2  0  5  7  9 11  8  4 11  1  3 11  2  1  4  0  2  7  6  0  2
3
  4  6  3 11 11  8  4 11  2  4  0 11  9  8  5 10  1  9  8  9  0  0  5
10
  1  9  4  4  3  5 11  6  2  8  6  9  4  6  7  0  1  9 11  4  7 11  4
7
  0  2  2  1  7  1 10  5  3  1 10  6  5  4  7 11  9  0  3  9  7  5  7
8
  8 11  6  7  7  7  4  8  8 10  8 10  1  3  9  3  1  8 11  6  3  5  9
3
  3 11  7 10  7  2 10  6  0  9  4  4  9  1  8 11  0  0  0  1 10  3  1
9
  1  5  7 11  0 10  8  6  7  3 11  1  3  3  3 10  4  2  4  4  0  0  4
9
  8  7 10  0 11  8  3  4  1  9 11  4  0  1  5  5 10  6  8  9  7  9  9
7
  6  2  9 10  3  8  3  1  4  0  2  2  4  5 11  7  2  8  2 11  6  6  0
1
  0  0 10  5 11  0 11  1  4  3  8  3  9  5  7  5  8  3  2  0  3  6  7
11
  6  5  6  7  2  6  5  2  1  9  6 10  1  2  4 11  9  0  2  3  1  0 10
6
 10 10  4  1  6  0  9  1  8  6  5  9 11  1  6  5  2  7 10  3 10  4  0
10
  3 11 10  7  6  8 11  3  6  2  5  1  8  1  4 10  8  3  8 11  3  7 11
2
  4 10  5  1  4 11 10  2  5  2 10  8  7  7  9  6  0  2  9 10  8 10  5
2
 11  8  0  9  5  3  5  1 10  4  4  2  9  4  8  0  8  4  1  5  8 10  7
6
  7  8  1  8  4  7  6  1  5  3  2  5 11  5  6  5 11  5  9  0  2  2  1
1
  9  4  2  2  0  3  6  6 10  0  6  7  9  5  4  3 11 10 11  7  3  0  6
10
 10  8 11  1  3  6  8  3  4  5  3  9  7  0 10  4  7 11  7  5  9 10  9
6
  1  4  8  9  1  7  3 10  1  3  5 11  6]

All Test y with label identifier [ 0  2  1 10  9  2  8  6  0 10  8  4
11  6  6  4  2  4  4  8  2  0  2 11
  7  5  7  5  9  3  5  8 10  9  9  9  2  1  0  3  4  9  5  1  7  0  8
8
  2  3  0  3  7  6  6  1  5  3  2  5 10  8  9  6  5  9  4  7  9  7  2
1
 10  7 10  7  4  5  5  5  0  2  1  6  2  5  0  5 11  7 10  0 11  4  5
10
 11 11  6  6  4  9 11  2  0  6  6  1  2  3  3  5  8  9  7  8  7  0  8
8
  4  6  9 11  1  3  9  8 11  2  2  7  7  3  0 10  6  1  1  3  2 11 10
11
  8  0  2  0  5  7  9 11  8  4 11  1  3 11  2  1  4  0  2  7  6  0  2
3
  4  6  3 11 11  8  4 11  2  4  0 11  9  8  5 10  1  9  8  9  0  0  5
10
```

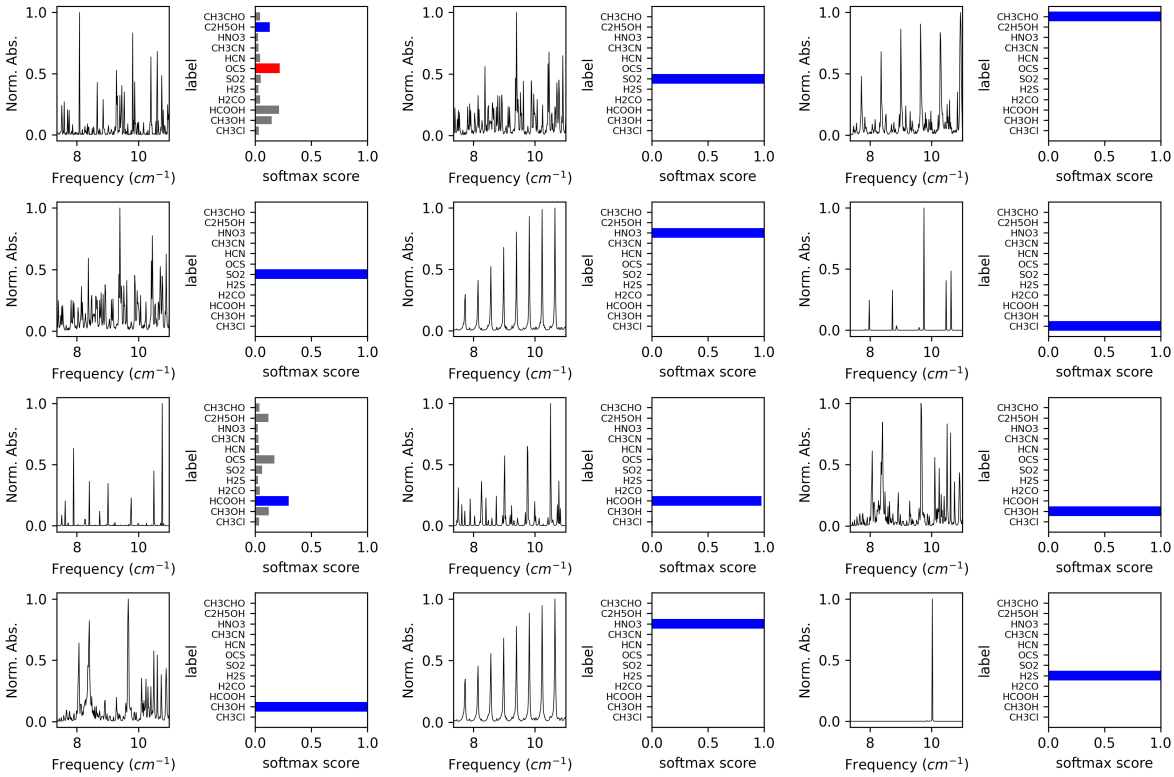
```
  1  9  4  4  3  5 11  6  2 10  8  6  9  4  6  7  0  1  9 11  4  7 11
4
  7  0  2  2  1  7  1 10  5  3  1 10  6  5  4  7 11  9  0  3  9  7  5
7
  8  8 11  6  7  7  7  4  8  8 10  8 10  1  3  9  3  1  8 11  6  3  5
9
  3  3 11  7 10  7  2 10  6  0  9  4  4  9  1  8 11  0  0  0  1 10  3
1
  9  1  5  7 11  0 10  8  6  7  3 11  1  3  3  3 10  4  2  4  4  0  0
4
  9  8  7 10  0 11  8  3  4  1  9 11  4  0  1  5  5 10  6  8  9  7  9
9
  7  6  2  9 10  3  8  3  1  4  0  2  2  4  5 11  7  2  8  2 11  6  6
0
  1  0  0 10  5 11  0 11  1  4  3  8  3  9  5  7  5  8  3  2  0  3  6
7
 11  6  5  6  7  2  6  5  2  1  9  6 10  1  2  4 11  9  0  2  3  1  0
10
  6 10 10  4  1  6  0  9  1  8  6  5  9 11  1  6  5  2  7 10  3 10  4
0
 10  3 11 10  7  6  8 11  3  6  2  5  1  8  1  4 10  8  3  8 11  3  7
11
  2  4 10  5  1  4 11 10  2  5  2 10  8  7  7  9  6  0  2  9 10  8 10
5
  2 11  8  0  9  5  3  5  1 10  4  4  2  9  4  8  0  8  4  1  5  8 10
7
  6  7  8  1  8  4  7  6  1  5  3  2  5 11  5  6 10  5 11  5  9  0  2
2
  1  1  9  4  2  2  0  3  6  6 10  0  6  7  9  5  4  3 11 10 11  7  3
0
  6 10 10  8 11  1  3  6  8  3  4  5  3  9  7  0 10  4  7 11  7  5  9
10
  9  6  1  4  8  9  1  7  3 10  1  3  5 11  6]

All Pred y with label identifier [ 0  2  1 10  9  2  8  6  0 10  8  4
11  6  6  4  2  4  4  8  2  0  2 11
  7  5  7  5  9  3  5  8 10  9  9  9  2  1  0  3  4  9  5  1  7  0  8
8
  2  3  0  3  7  6  6  1  5  3  2  5 10  8  9  6  5  9  4  7  9  7  2
1
 10  7 10  7  4  5  5  5  0  2  1  6  2  5  0  5 11  7 10  0 11  4  5
10
 11 11  6  6  4  9 11  2  0  6  6  1  2  3  3  5  8  9  7  8  7  0  8
8
  4  6  9 11  1  3  9  8 11  2  2  7  7  3  0 10  6  1  1  3  2 11 10
11
  8  0  2  0  5  7  9 11  8  4 11  1  3 11  2  1  4  0  2  7  6  0  2
3
  4  6  3 11 11  8  4 11  2  4  0 11  9  8  5 10  1  9  8  9  0  0  5
10
  1  9  4  4  3  5 11  6  2  6  8  6  9  4  6  7  0  1  9 11  4  7 11
4
  7  0  2  2  1  7  1 10  5  3  1 10  6  5  4  7 11  9  0  3  9  7  5
7
  8  8 11  6  7  7  7  4  8  8 10  8 10  1  3  9  3  1  8 11  6  3  5
9
```

```
 3  3 11  7 10  7  2 10  6  0  9  4  4  9  1  8 11  0  0  0  1 10  3
```

In [25]:
```
fig = plot_sequential_group_prediction(np.squeeze(x_test), y_test, pred

fig.savefig(r'RESULTS/results_figures/softmax_figures_test_spectra_star
```



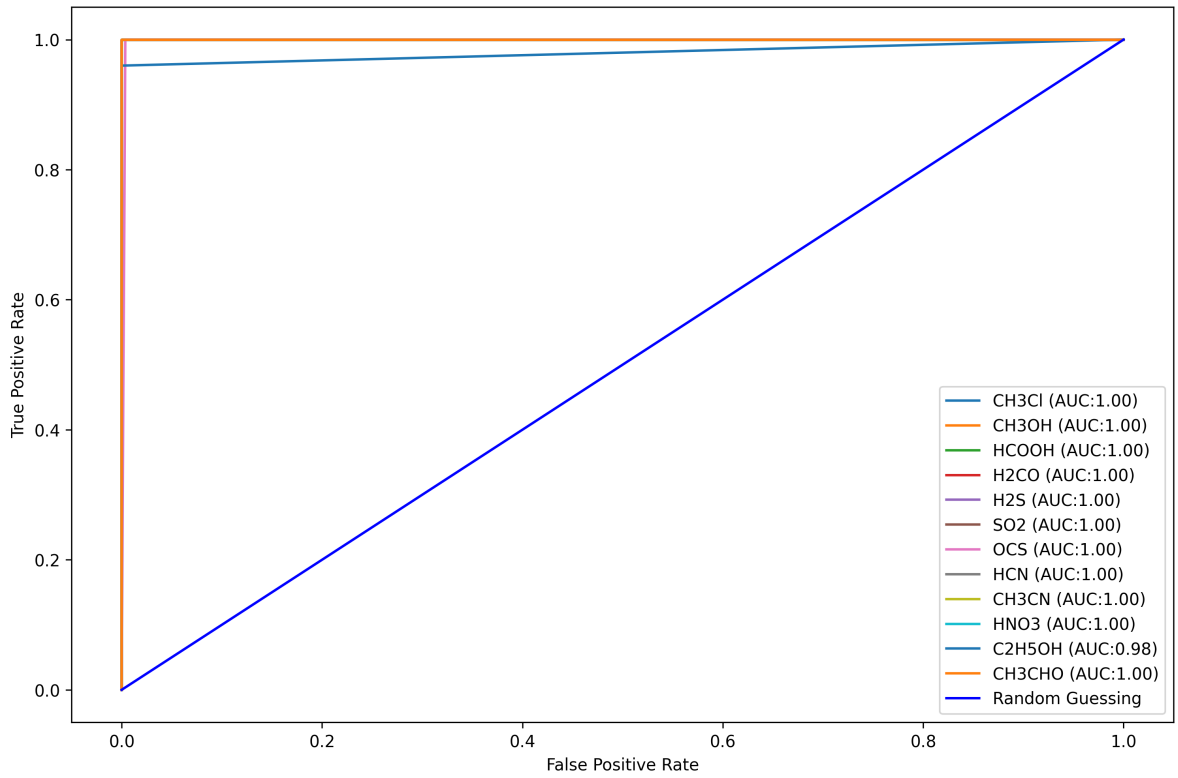
In [26]:
```
fig = plot_sequential_group_prediction(np.squeeze(x_test), y_test, pred

fig.savefig(r'RESULTS/results_figures/softmax_figures_test_spectra_star
```



In [27]:
```
print('ROC AUC score:', multiclass_roc_auc_score(y_test, pred_y, s.labe

fig = multiclass_roc_auc_score(y_test, pred_y, s.labels)[1]
fig.savefig(r'RESULTS/results_figures/ROC_test_data.png', bbox_inches='
fig
```

```
ROC AUC score: 0.998179581795818
```

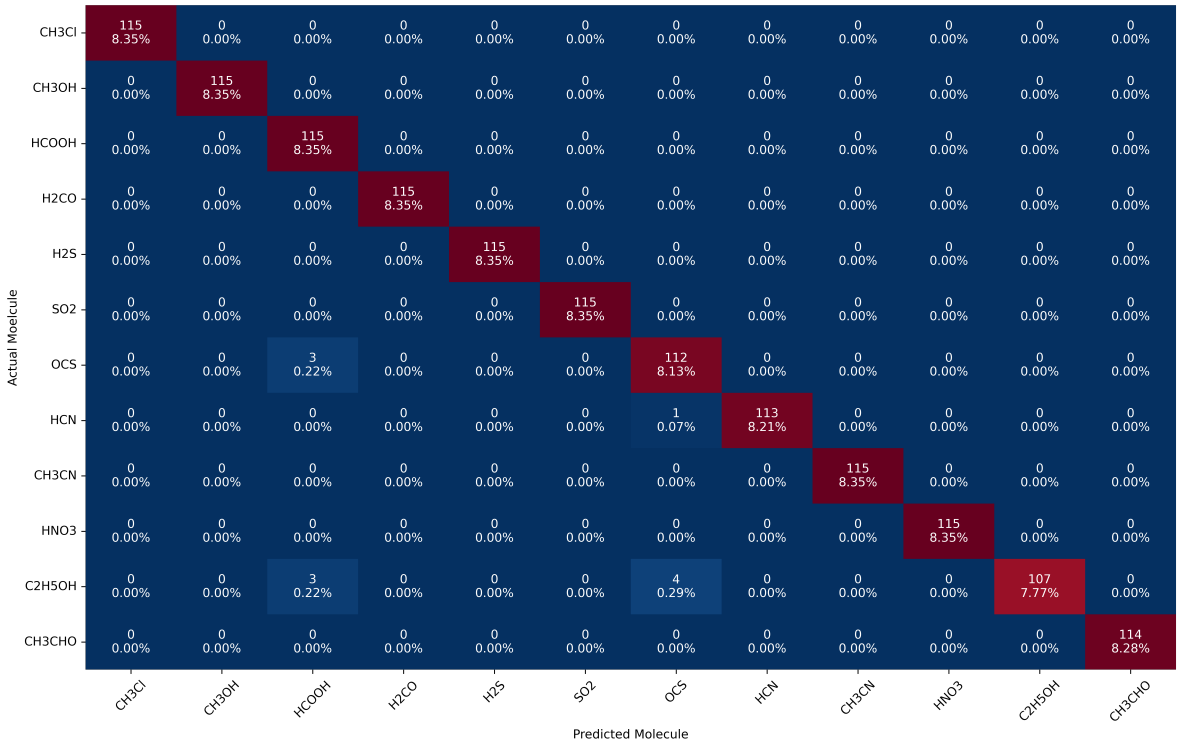
Out[27]:



In [28]:
```
pred_y_train=np.argmax(model.predict(x_train), axis= 1)
```

In [29]:
```
cm = confusion_matrix(y_train, pred_y_train)

group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]

annot_labels = [f"{v1}\n{v2}" for v1, v2 in
          zip(group_counts,group_percentages)]

fig = plt.figure(figsize=(16,10), dpi = 600);
# plt.title('Confusion matrix);

# ax = sns.heatmap(cm, annot=True, cmap='PiYG');   #cmap='coolwarm' als
ax = sns.heatmap(cm/np.sum(cm), annot=np.asarray(annot_labels).reshape(
#ax = sns.heatmap(cm/np.sum(cm), annot=True, fmt='.2%', cmap='Blues') #
ax.set_xticklabels(s.labels);
ax.set_yticklabels(s.labels);
plt.xlabel('Predicted Molecule');
plt.ylabel('Actual Moelcule');
```

```
plt.xticks(rotation=45);
plt.yticks(rotation=0);
plt.savefig(r'RESULTS/results_figures/cm_train_data.png', bbox_inches='
```



In [30]:
```
classifier_internals(pred_y_train, y_train, y_train, 'simple_CNN')
```

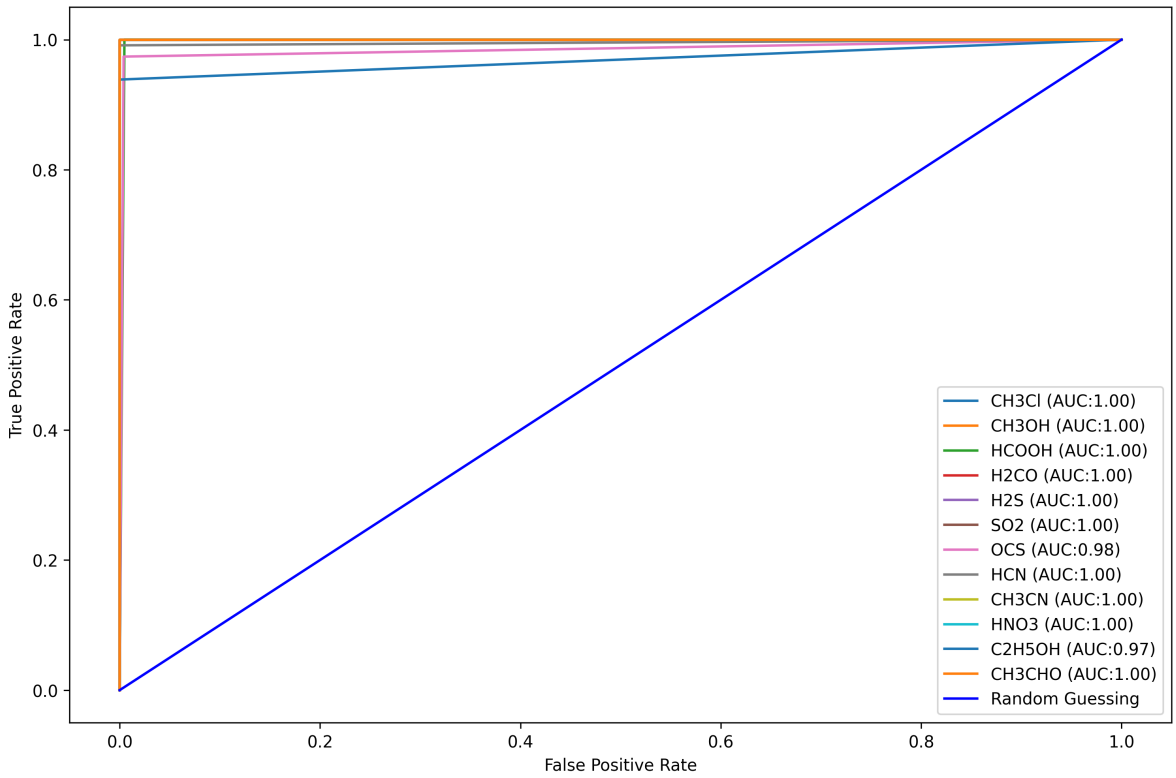
```
------------------------------ simple_CNN
--------------------------------
```

In [31]:

```
print('ROC AUC score:', multiclass_roc_auc_score(y_train, pred_y_train,

fig = multiclass_roc_auc_score(y_train, pred_y_train, s.labels)[1]
fig.savefig(r'RESULTS/results_figures/ROC_train_data.png', bbox_inches=
fig
```

```
ROC AUC score: 0.9956258867327256
```

Out[31]:

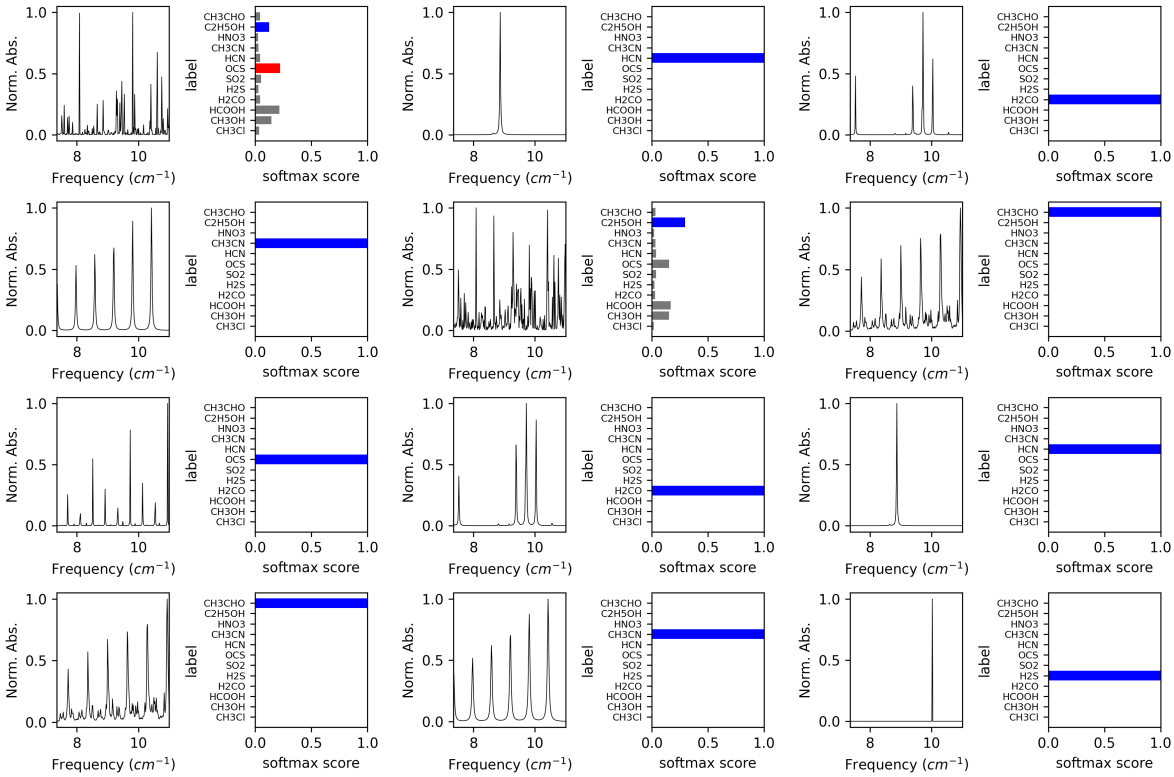


In [32]:

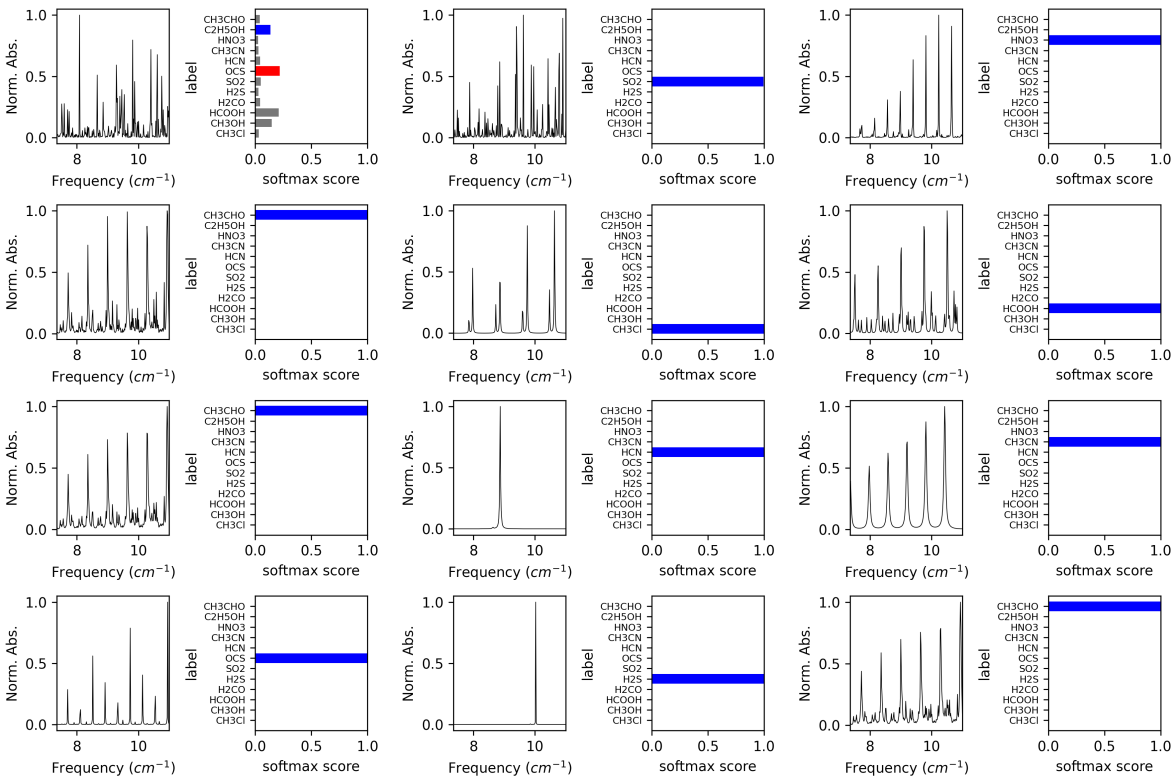
```
predictions_on_train = probability_model.predict(x_train)
```

In [ ]:

In [33]:
```
fig = plot_sequential_group_prediction(np.squeeze(x_train), y_train, pr

fig.savefig(r'RESULTS/results_figures/softmax_figures_train_spectra_sta
```



In [34]:
```
fig = plot_sequential_group_prediction(np.squeeze(x_train), y_train, pr

fig.savefig(r'RESULTS/results_figures/softmax_figures_train_spectra_sta
```

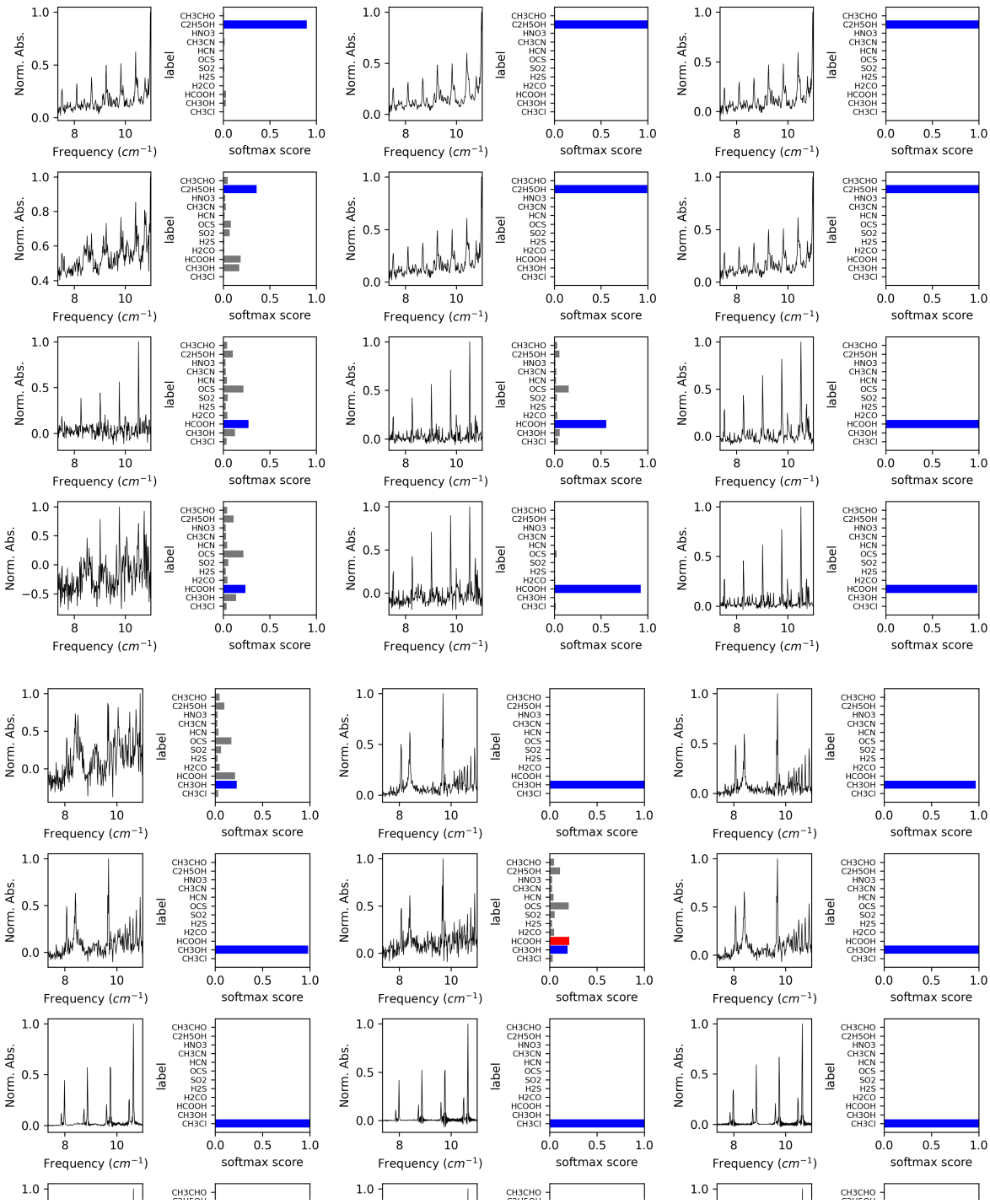


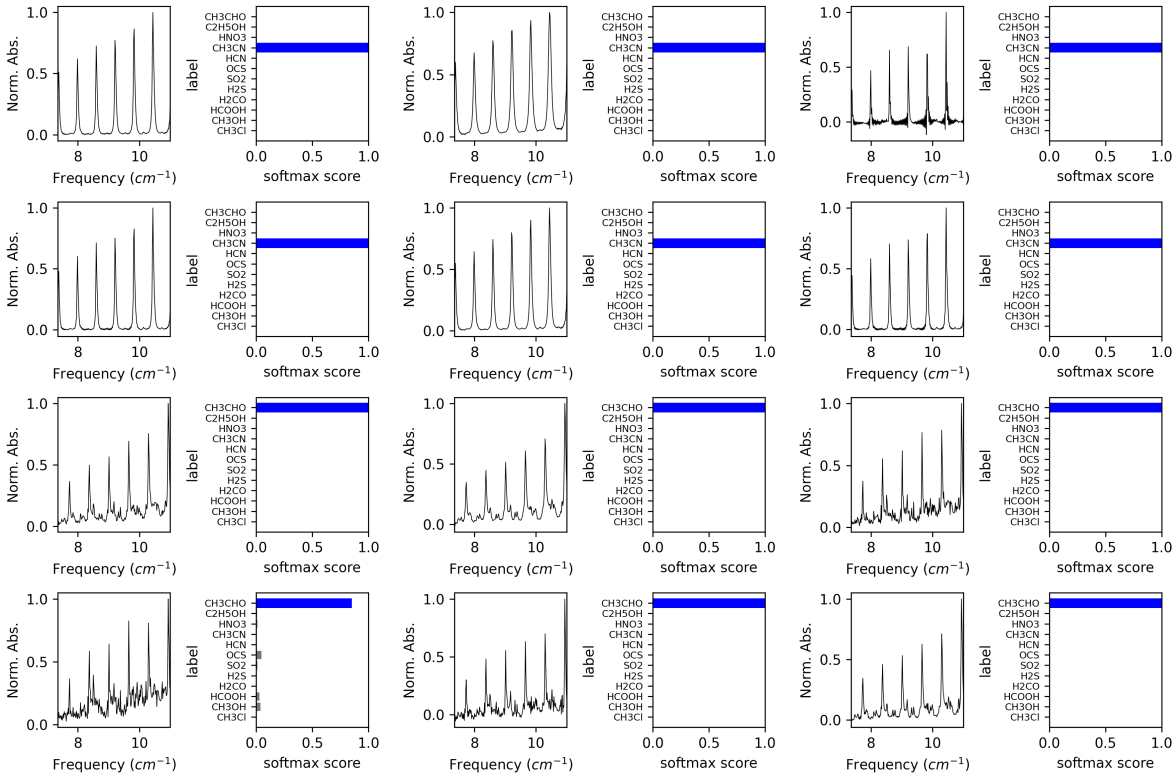
In [35]:
```
pred_y_exp=np.argmax(model.predict(Xexp), axis=-1)
print(pred_y_exp)
```

```
[10 10 10 10 10 10  2  2  2  2  2  2  1  1  1  1  2  1  0  0  0  0  0
0
  8  8  8  8  8  8 11 11 11 11 11 11]
```

In [36]:
```
predictions_exp = probability_model.predict(Xexp) # softmax scores for
```

In [37]:
```
fig = plot_sequential_group_prediction(Xexp, yexp, predictions_exp, s.f
                                       s.labels, 0)

fig.savefig(r'RESULTS/results_figures/softmax_figures_exp_spectra_start

fig = plot_sequential_group_prediction(Xexp, yexp, predictions_exp, s.f
                                       s.labels, 12)

fig.savefig(r'RESULTS/results_figures/softmax_figures_exp_spectra_start

fig = plot_sequential_group_prediction(Xexp, yexp, predictions_exp, s.f
                                       s.labels, 24)

fig.savefig(r'RESULTS/results_figures/softmax_figures_exp_spectra_start
```





In [38]:
```
print([s.labels[i] for i in s.exp_targets])
```

```
['C2H5OH', 'C2H5OH', 'C2H5OH', 'C2H5OH', 'C2H5OH', 'C2H5OH', 'HCOOH',
'HCOOH', 'HCOOH', 'HCOOH', 'HCOOH', 'HCOOH', 'CH3OH', 'CH3OH', 'CH3OH
', 'CH3OH', 'CH3OH', 'CH3OH', 'CH3Cl', 'CH3Cl', 'CH3Cl', 'CH3Cl', 'CH3
Cl', 'CH3Cl', 'CH3CN', 'CH3CN', 'CH3CN', 'CH3CN', 'CH3CN', 'CH3CN', 'C
H3CHO', 'CH3CHO', 'CH3CHO', 'CH3CHO', 'CH3CHO', 'CH3CHO']
```

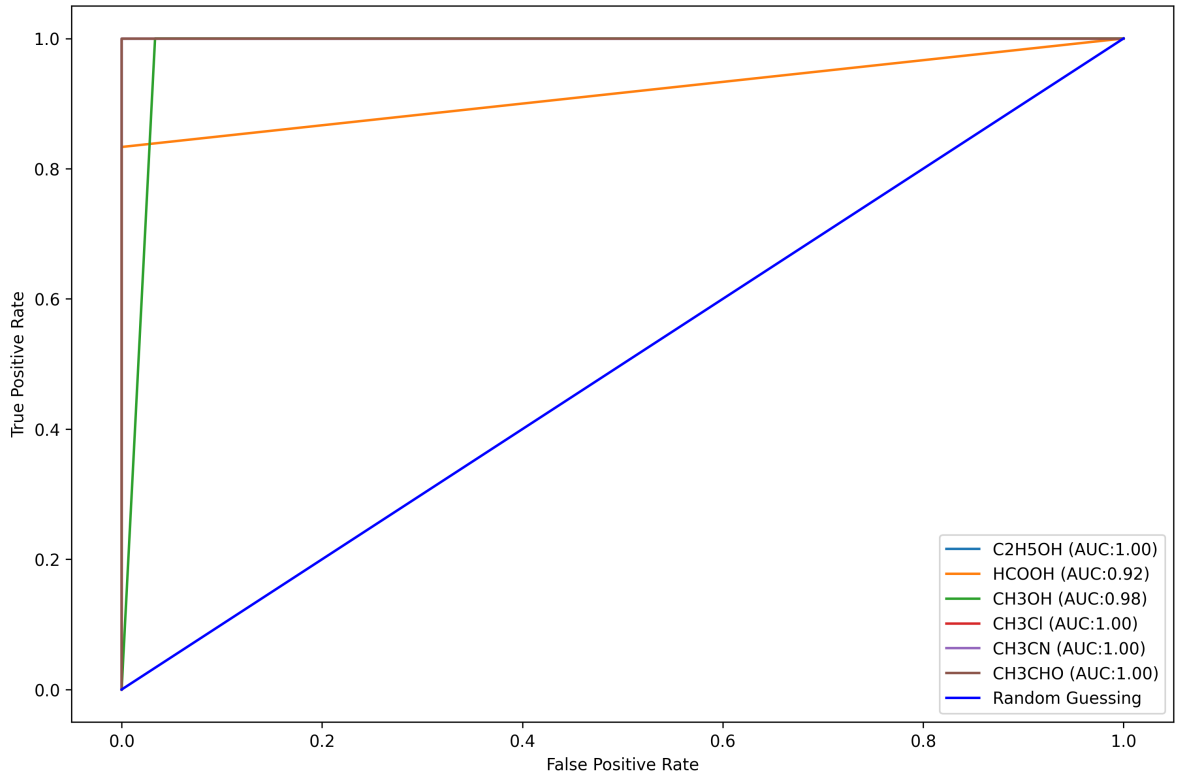
In [39]:
```
cm = confusion_matrix(yexp, pred_y_exp)

# group_counts = ["{0:0.0f}".format(value) for value in
#                 cm.flatten()]

# group_percentages = ["{0:.2%}".format(value) for value in
#                      cm.flatten()/np.sum(cm)]

# annot_labels = [f"{v1}\n{v2}" for v1, v2 in
#           zip(group_counts,group_percentages)]

# fig = plt.figure(figsize=(16,10), dpi = 600);
# # plt.title('Confusion matrix);

# # ax = sns.heatmap(cm, annot=True, cmap='PiYG');   #cmap='coolwarm' a
# ax = sns.heatmap(cm/np.sum(cm), annot=np.asarray(annot_labels).reshap
# #ax = sns.heatmap(cm/np.sum(cm), annot=True, fmt='.2%', cmap='Blues')
# ax.set_xticklabels(np.unique([s.labels[i] for i in s.exp_targets]));
# ax.set_yticklabels(np.unique([s.labels[i] for i in s.exp_targets]));
# plt.xlabel('Predicted Molecule');
# plt.ylabel('Actual Moelcule');
# plt.xticks(rotation=45);
# plt.yticks(rotation=0);
```

```
# plt.savefig(r'RESULTS/results_figures/cm_exp_data.png', bbox_inches='
```

In [40]:
```
print('ROC AUC score:', multiclass_roc_auc_score(yexp, pred_y_exp, s.la

fig = multiclass_roc_auc_score(yexp, pred_y_exp, s.labels_exp)[1]
fig.savefig(r'RESULTS/results_figures/ROC_exp_data.png', bbox_inches='t
fig
```

```
ROC AUC score: 0.9833333333333334
```

Out[40]:



GRAD-CAM

PR Curve for test and exp_data

In [41]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.preprocessing import label_binarize
```

In [42]:
```
s.label_id
```

Out[42]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [43]:
```
Y_test = label_binarize(y_test, classes=s.label_id)
```

In [ ]:
```

```

In [44]:
```
Y_test_exp = label_binarize(yexp, classes=s.label_id)
```

In [45]:
```
Y_test_exp
```

Out[45]:
```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

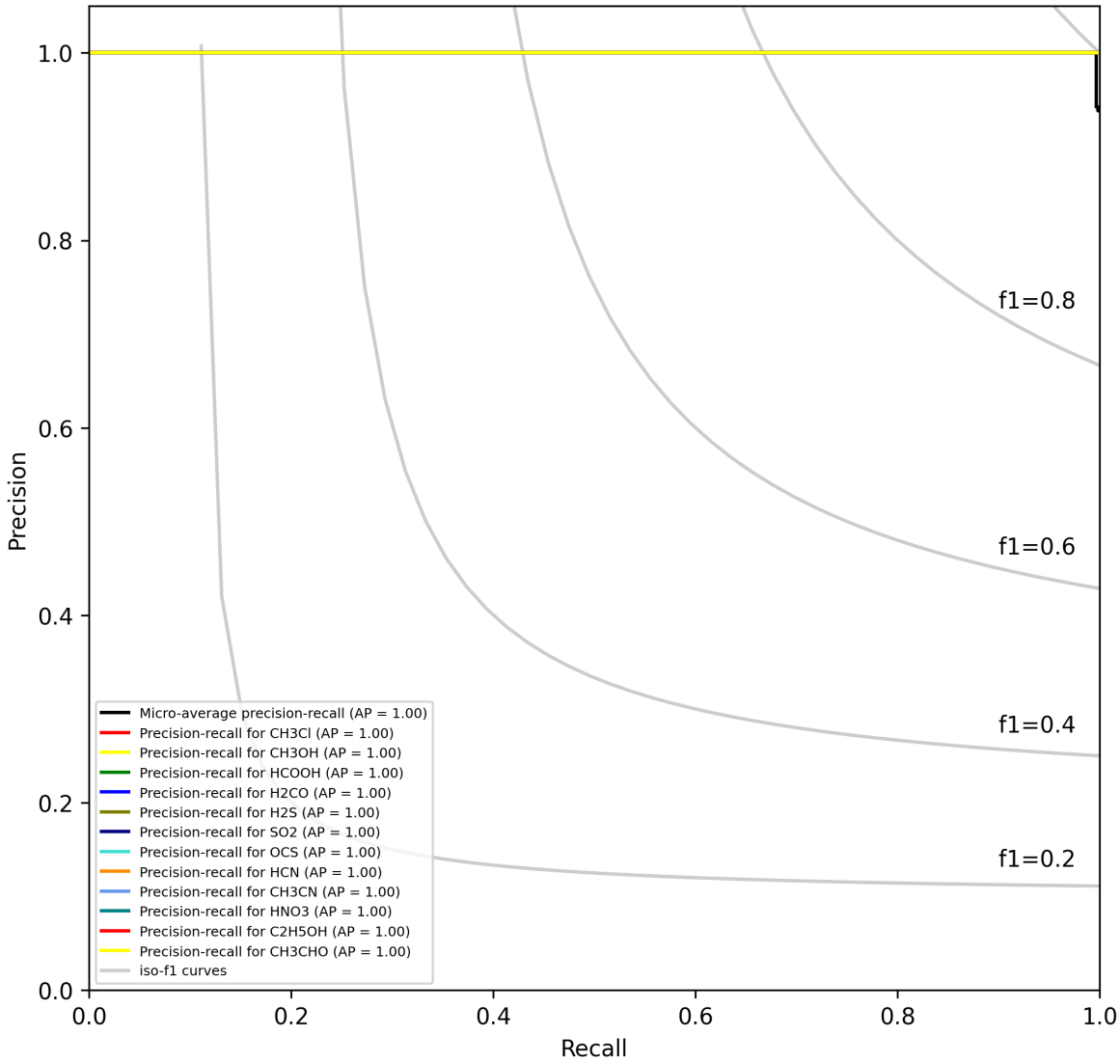
In [46]:
```
from sklearn.metrics import PrecisionRecallDisplay
from itertools import cycle


def voc_net_pr_curve(n_classes,scores, Y_test, all_unique_labels):

    '''
    scores:raw score values from classifier
    Y_test: label binarized values

    '''

    n_classes =n_classes
    y_score = scores

    precision = dict()
```

```
    recall = dict()
    average_precision = dict()
    for i in range(n_classes):
        precision[i], recall[i], _ = precision_recall_curve(Y_test[:, i
                                                             y_score[:,
        average_precision[i] = average_precision_score(Y_test[:, i], y_

    # A "micro-average": quantifying score on all classes jointly
    precision["micro"], recall["micro"], _ = precision_recall_curve(Y_t
        y_score.ravel())
    average_precision["micro"] = average_precision_score(Y_test, y_scor
                                                         average="micro

    print('Average precision score, micro-averaged over all classes: {0
          .format(average_precision["micro"]))


    # setup plot details
    colors = cycle(["red", "yellow", "green", "blue",
                    "olive", "navy", "turquoise", "darkorange",
                    "cornflowerblue", "teal"])

    _, ax = plt.subplots(figsize=(8, 8),dpi=300)

    f_scores = np.linspace(0.2, 1, num=5)
    lines, labels = [], []
    for f_score in f_scores:
        x = np.linspace(0.01, 1)
        y = f_score * x / (2 * x - f_score)
        (l,) = plt.plot(x[y >= 0], y[y >= 0], color="gray", alpha=0.4)
        plt.annotate("f1={0:0.1f}".format(f_score), xy=(0.9, y[45] + 0.

    display = PrecisionRecallDisplay(
        recall=recall["micro"],
        precision=precision["micro"],
        average_precision=average_precision["micro"],
    )
    display.plot(ax=ax, name="Micro-average precision-recall", color="b

    for i, color in zip(range(n_classes), colors):
        display = PrecisionRecallDisplay(
            recall=recall[i],
            precision=precision[i],
            average_precision=average_precision[i],
        )
        display.plot(ax=ax, name=f"Precision-recall for {all_unique_lab

    # add the legend for the iso-f1 curves
    handles, labels = display.ax_.get_legend_handles_labels()
    handles.extend([l])
    labels.extend(["iso-f1 curves"])
    # set the legend and the axes
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.legend(handles=handles, labels=labels, loc="lower left", prop={'
#     ax.set_title("Precision-Recall curve.")
```

```
    # plt.show()

    return precision, recall, average_precision
```

In [47]:
```
fig = voc_net_pr_curve(12 , predictions, Y_test, s.labels);
plt.savefig(r'RESULTS/results_figures/VOC_Net_pr_curve_test.png', bbox_i
```

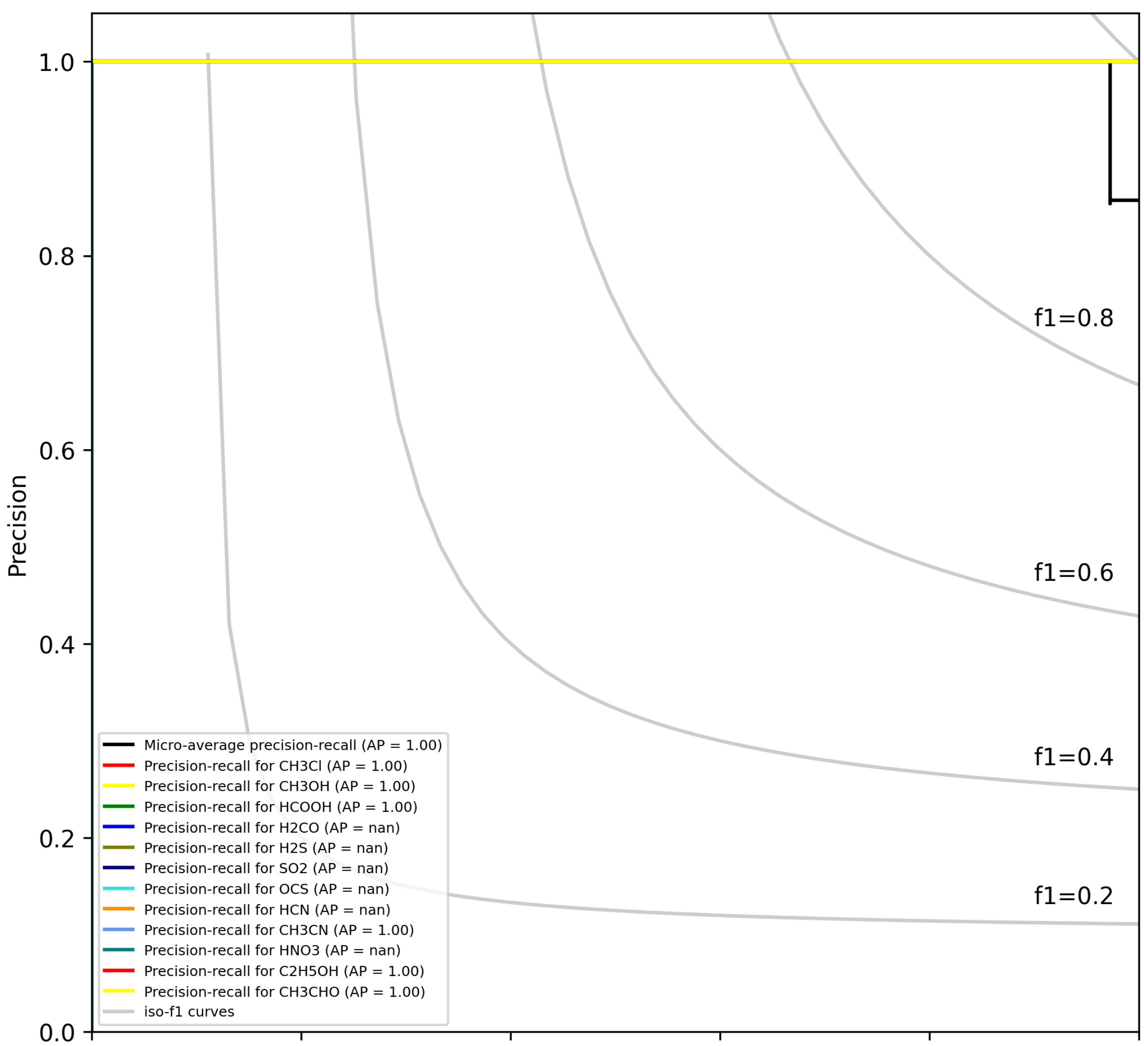
```
Average precision score, micro-averaged over all classes: 1.00
```



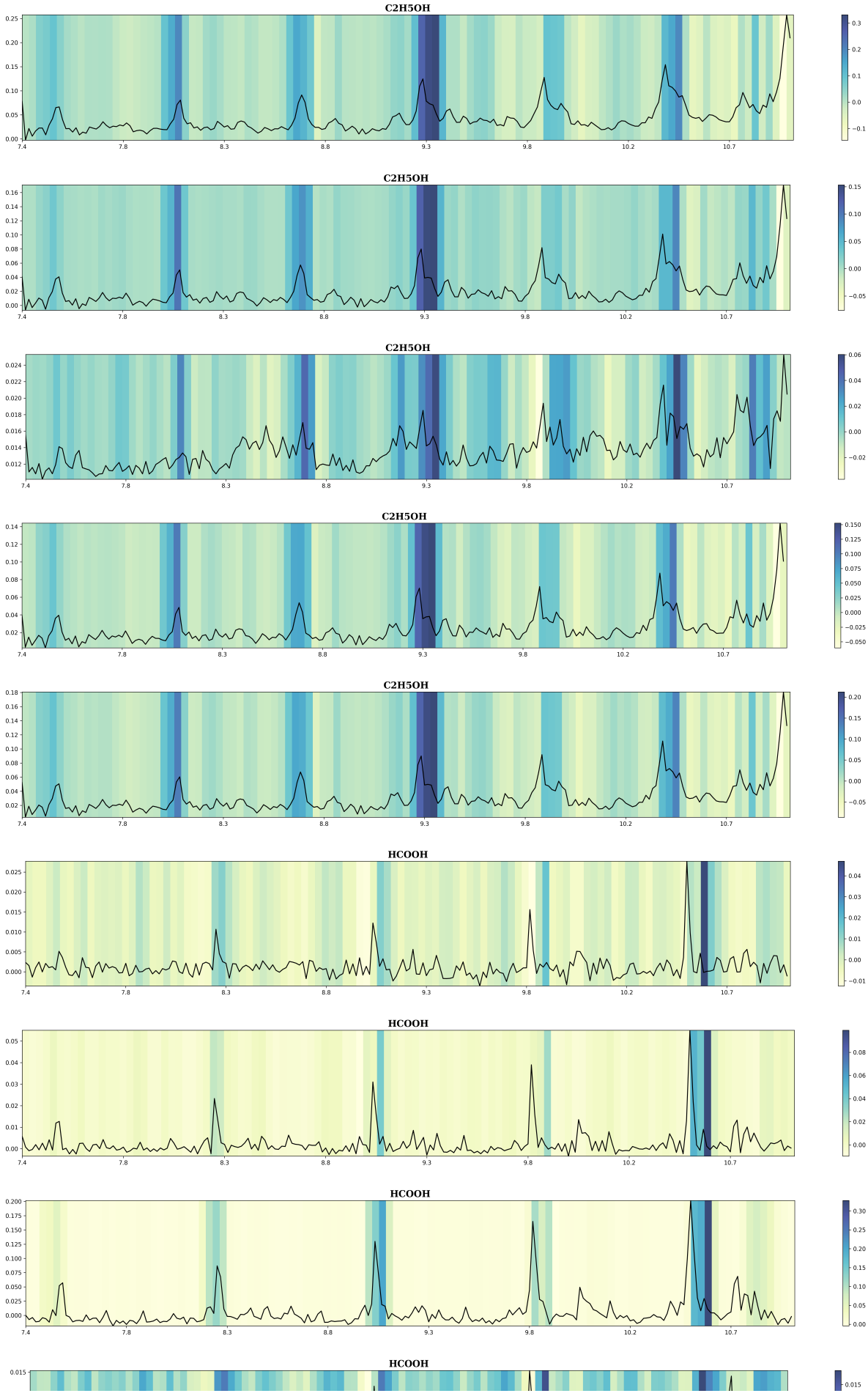
In [48]:
```
fig = voc_net_pr_curve(12 , predictions_exp, Y_test_exp, s.labels);
plt.savefig(r'RESULTS/results_figures/VOC_Net_pr_curve_exp.png', bbox_i
```

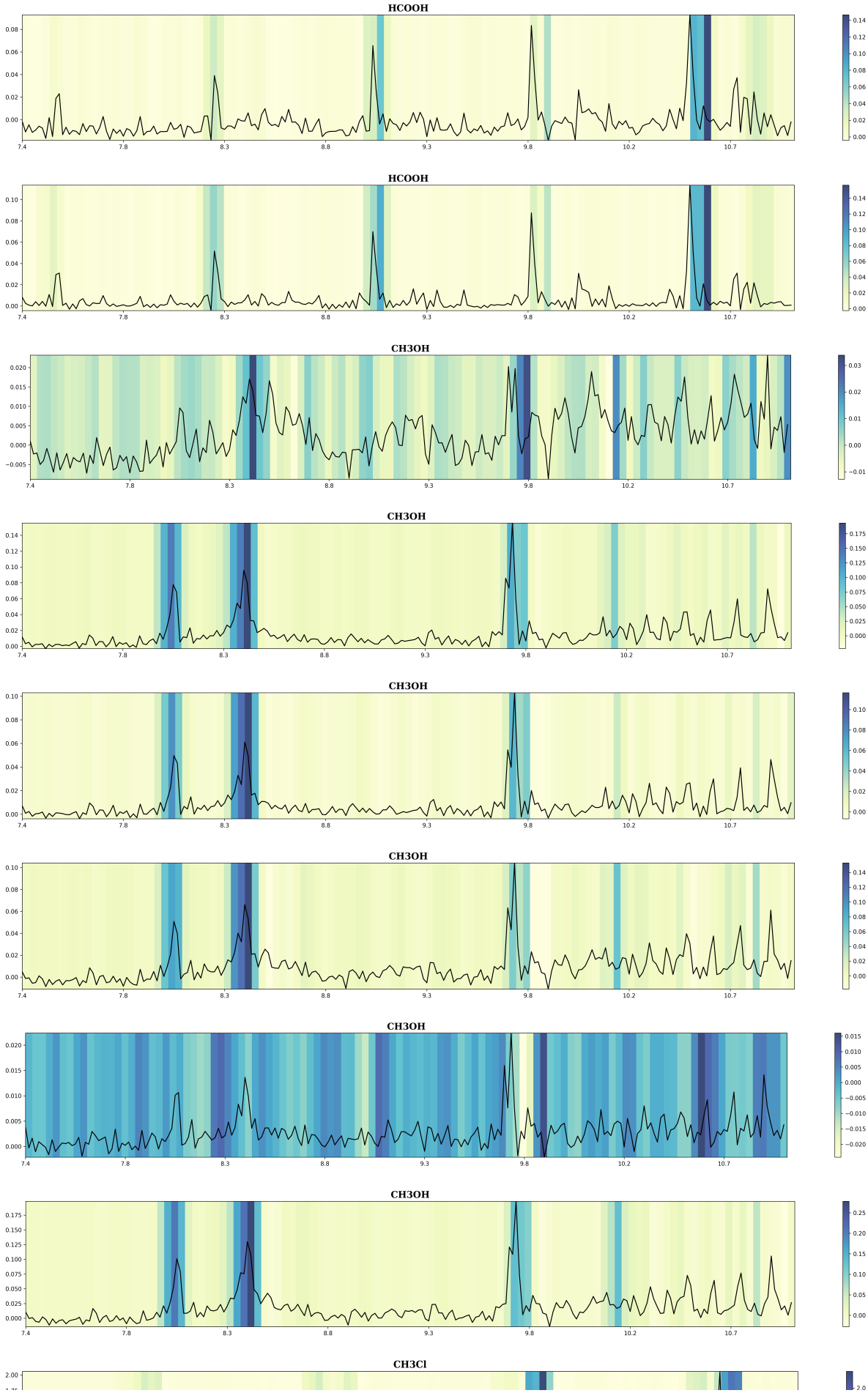
```
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
/home/reshad812/.local/lib/python3.8/site-packages/sklearn/metrics/_ra
nking.py:864: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
```

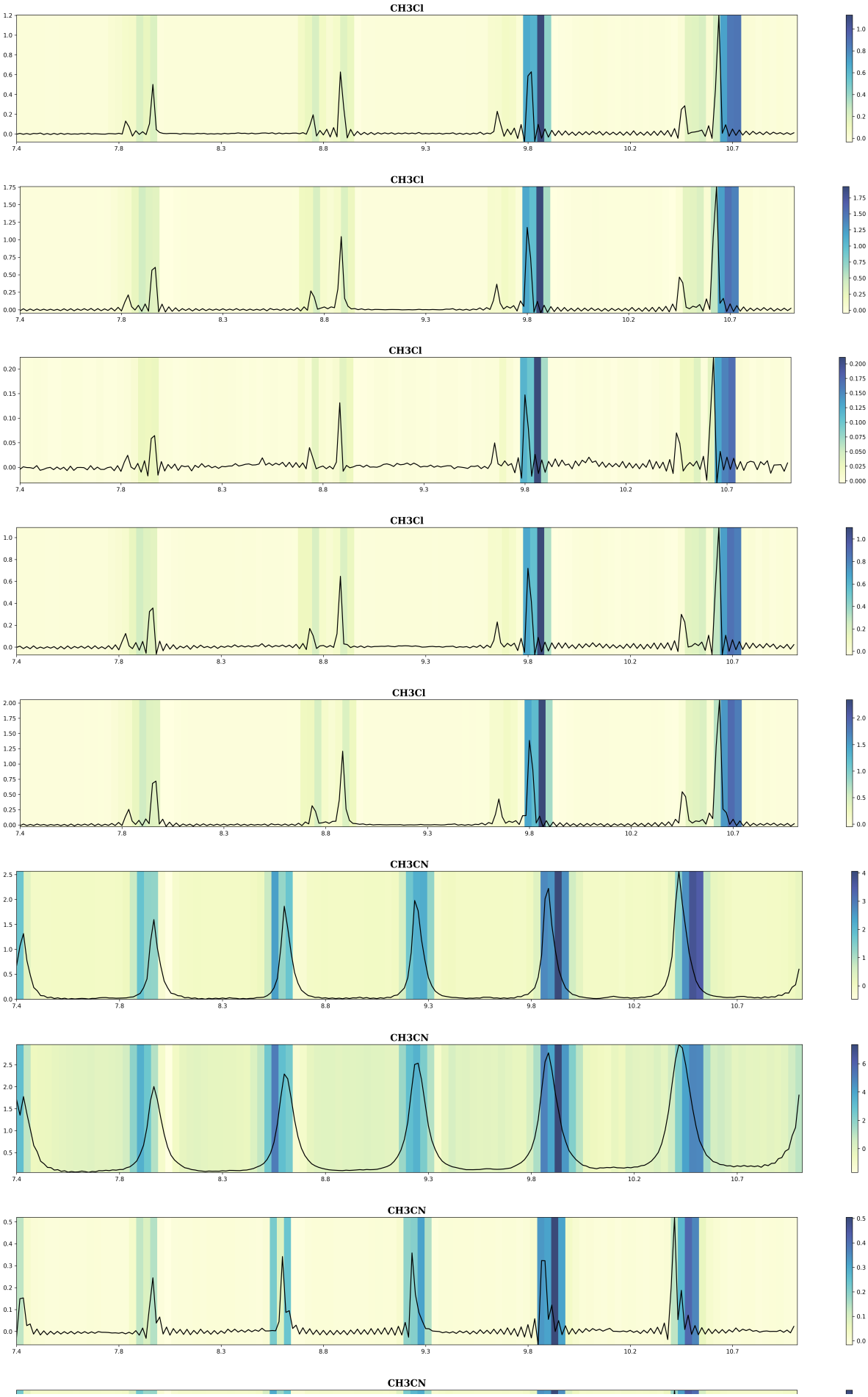
```
Average precision score, micro-averaged over all classes: 1.00
```

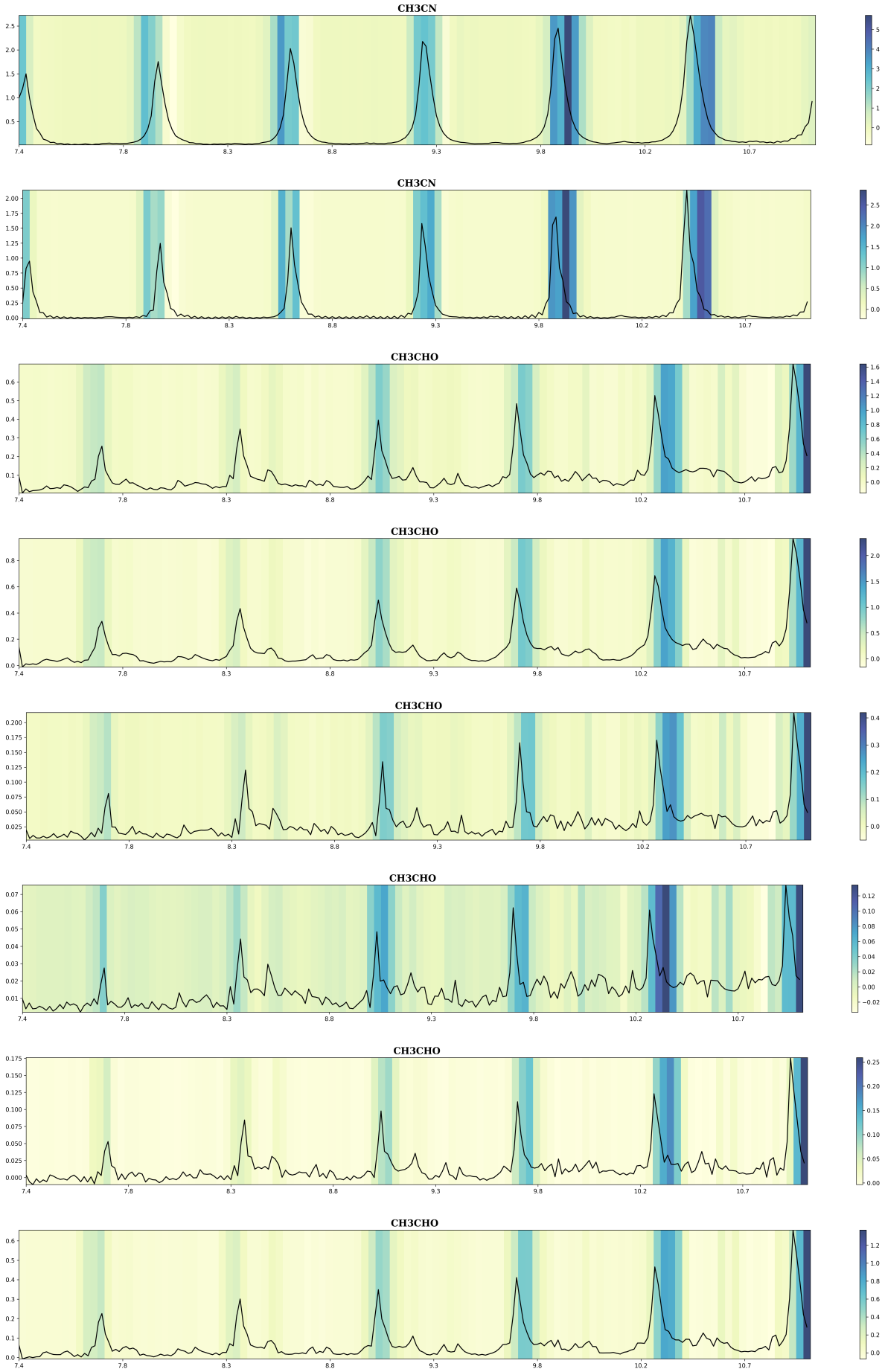


In [49]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

def grad_cam(layer_name, data):
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(layer_name).output, model.outp
    )
    last_conv_layer_output, preds = grad_model(data)

    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(data)
        pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    grads = tape.gradient(class_channel, last_conv_layer_output)

    pooled_grads = tf.reduce_mean(grads, axis=(0))

    last_conv_layer_output = last_conv_layer_output[0]

    heatmap = last_conv_layer_output * pooled_grads
    heatmap = tf.reduce_mean(heatmap, axis=(1))
    heatmap = np.expand_dims(heatmap,0)
    return heatmap

layer_name = "C3"

font = {'family': 'serif',
        'color':  'black',
        'weight': 'bold',
        'size': 16,
        }


count = 0
for i,j in zip(Xexp,yexp):

    data = np.expand_dims(i,0)
    heatmap = grad_cam(layer_name,data)

    fig = plt.figure(figsize=(30,4),dpi=300)
    plt.imshow(np.expand_dims(heatmap,axis=2),cmap='YlGnBu', aspect="au

    ticklist = range(0,229)
    plt.xticks(ticklist[::30], np.round(s.frequencies.tolist()[::30], d
    plt.plot(i,'k')
    plt.title(f'{s.labels[j]}', fontdict=font)
    plt.colorbar()
    plt.clim(np.min(heatmap),np.max(heatmap))

    plt.show()

    fig.savefig(r'RESULTS/class_activation_maps/CAM_exp' + str(count) +
    count = count + 1
```

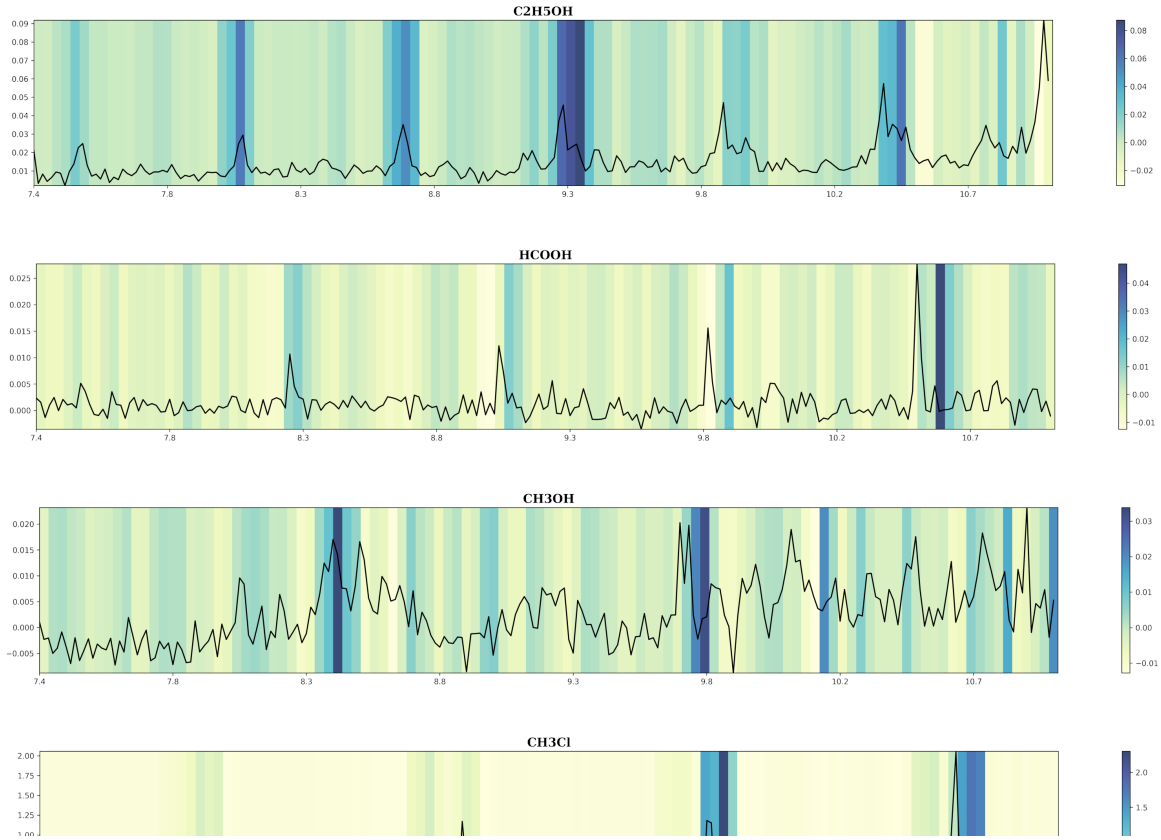

C2H5OH
C2H5OH
C2H5OH
C2H5OH
C2H5OH
HCOOH
HCOOH
HCOOH
HCOOH


HCOOH
HCOOH
CH3OH
CH3OH
CH3OH
CH3OH
CH3OH
CH3Cl

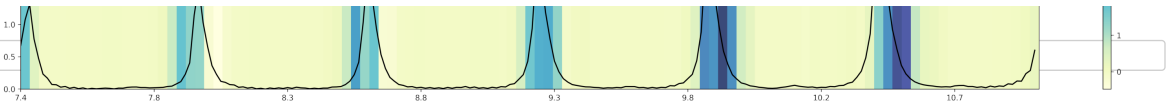

CH3Cl
CH3Cl
CH3Cl
CH3Cl
CH3Cl
CH3CN
CH3CN
CH3CN
CH3CN


CH3CN
CH3CN
CH3CHO
CH3CHO
CH3CHO
CH3CHO
CH3CHO
CH3CHO

In [50]:
```
# Imports PIL module
from PIL import Image

# open method used to open different extension image file
im0 = Image.open(r'RESULTS/class_activation_maps/CAM_exp' + str(0) + '.
im1 = Image.open(r'RESULTS/class_activation_maps/CAM_exp' + str(6) + '.
im2 = Image.open(r'RESULTS/class_activation_maps/CAM_exp' + str(12) + '
im3 = Image.open(r'RESULTS/class_activation_maps/CAM_exp' + str(18) + '
im4 = Image.open(r'RESULTS/class_activation_maps/CAM_exp' + str(24) + '
im5 = Image.open(r'RESULTS/class_activation_maps/CAM_exp' + str(30) + '


fig, ax = plt.subplots(6)
fig.set_size_inches(16, 16)
fig.set_dpi(300)
plt.axis(False)

ax[0].imshow(im0)
ax[0].axis(False)
ax[1].imshow(im1)
ax[1].axis(False)
ax[2].imshow(im2)
ax[2].axis(False)
ax[3].imshow(im3)
ax[3].axis(False)
ax[4].imshow(im4)
ax[4].axis(False)
ax[5].imshow(im5)
ax[5].axis(False)

fig.savefig(r'RESULTS/results_figures/Class_activation_maps_combined_V1
```

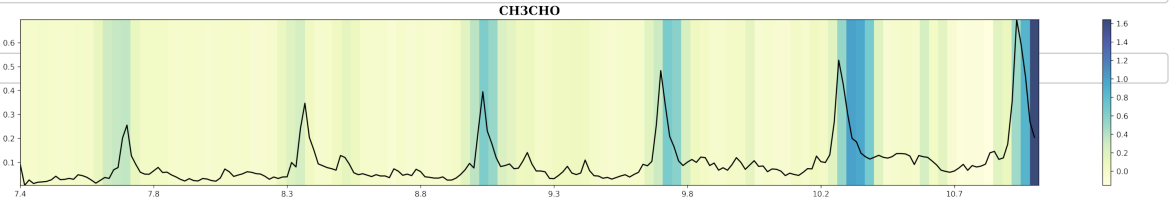


In [ ]:


1.0
0.5
0.0
7.4
7.8
8.3
8.8
9.3
9.8
10.2
10.7
1
0

In [ ]:

In [ ]:


CH3CHO
0.6
0.5
0.4
0.3
0.2
0.1
7.4
7.8
8.3
8.8
9.3
9.8
10.2
10.7
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0