Challenge Data Visualization

November 23, 2021

[11]:
```
import torch
import os
import numpy as np
import matplotlib.pyplot as plt
from random import shuffle
from torch.utils.data import IterableDataset
from torch.utils.data import ChainDataset
import seaborn as sns
from scipy import stats
import pandas as pd
from torch.utils.data._utils.collate import np_str_obj_array_pattern,␣
↪default_collate_err_msg_format

import sys
sys.path.insert(0, 'Downloads/Github/contrastive-predictive-coding/util/
↪visualize')
#import timeseries_to_image_converter
```

0.1 Implementation because imports suck

[3]:
```
class ECGChallengeDatasetBaseline(torch.utils.data.IterableDataset):
    def __init__(self, BASE_DIR, window_size, pad_to_size=None, files=None,␣
↪channels=None, return_labels=False, return_filename=False, classes=None,␣
↪normalize_fn=None, verbose=False):
        super(torch.utils.data.IterableDataset).__init__()
        self.BASE_DIR = BASE_DIR
        self.window_size = window_size
        self.pad_to_size = pad_to_size or window_size
        self.files = files or self.search_files()
        self.classes = classes or get_classes(self.files)
        if verbose:
            self.print_file_attributes()
        self.channels = channels
        self.total_length = 1 #Trying a weird approach (calculated in __iter__)
        self.return_labels = return_labels
        self.return_filename = return_filename
```

```
        self.normalize_fn = normalize_fn if not normalize_fn is None else␣
↪lambda x: x

    def generate_datasets_from_split_file(self, ttsfile='train-test-splits.
↪txt'):
        splits = load_train_test_split(os.path.join(self.BASE_DIR, ttsfile))
        return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.
↪window_size, self.pad_to_size, files=s,
                                                 channels=self.channels,␣
↪return_labels=self.return_labels,
                                                 return_filename=self.
↪return_filename, classes=self.classes,
                                                 normalize_fn=self.normalize_fn)
                     for s in splits)

    def generate_datasets_from_split_list(self, trainf, valf, testf):
        splits = [trainf, valf, testf]
        return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.
↪window_size, self.pad_to_size, files=s,
                                                 channels=self.channels,␣
↪return_labels=self.return_labels,
                                                 return_filename=self.
↪return_filename, classes=self.classes,
                                                 normalize_fn=self.normalize_fn)
                     for s in splits)

    def __iter__(self):
        file_index = 0
        shuffle(self.files)
        while file_index < len(self.files):
            current_file = self.files[file_index]
            data = self.normalize_fn(self._read_recording_file(current_file))
            if self.channels:
                data = data[:, self.channels]
            if self.return_labels:
                labels = self._read_header_labels(current_file)
            if len(data) - self.window_size > 0:
                offset = np.random.randint(len(data) - self.window_size)  #␣
↪Random offset
                data = data[offset:self.window_size+offset]
            else:
                offset = 0
                data = np.pad(data, ((max(0, self.pad_to_size-min(self.
↪window_size, len(data))), 0), (0,0)))
            if not any([self.return_filename, self.return_labels]):
                yield data
```

```
            else:
                yield [data] + ([labels] if self.return_labels else [None]) +␣
↪([current_file] if self.return_filename else [])

            file_index += 1

    def _read_recording_file(self, path_without_ext):
        fp = path_without_ext + '.mat'
        return load_recording(fp, key='val').transpose()

    def _read_header_file(self, path_without_ext):
        fp = path_without_ext + '.hea'
        return load_header(fp)

    def _read_header_labels(self, path_without_ext, onerror_class='426783006'):
        header = self._read_header_file(path_without_ext)
        return encode_header_labels(header, self.classes, onerror_class)

    def __len__(self):
        return self.total_length

    def search_files(self):
        headers, records = find_challenge_files(self.BASE_DIR)
        print(len(records), 'record files found in ', self.BASE_DIR)
        return list(map(lambda x: os.path.splitext(x)[0], records)) #remove␣
↪extension

    def random_train_split(self, train_fraction=0.7, val_fraction=0.2,␣
↪test_fraction=0.1, save=True, save_path_overwrite=None,␣
↪filename_overwrite=None):
        assert train_fraction+val_fraction+test_fraction <= 1
        N = len(self.files)
        shuffle(self.files)
        train_slice = slice(0, int(train_fraction*N))
        val_slice = slice(train_slice.stop, train_slice.
↪stop+int(val_fraction*N))
        test_slice = slice(val_slice.stop, val_slice.stop+int(test_fraction*N))
        if save:
            p = save_path_overwrite or self.BASE_DIR
            fname = filename_overwrite or 'train-test-splits.txt'
            save_train_test_split(os.path.join(p, fname), self.
↪files[train_slice], self.files[val_slice], self.files[test_slice])
        return self.files[train_slice], self.files[val_slice], self.
↪files[test_slice]
```

```
   def random_train_split_with_class_count(self, train_fraction=0.7,␣
↪val_fraction=0.2, test_fraction=0.1, save=True, save_path_overwrite=None,␣
↪filename_overwrite=None):
       assert train_fraction+val_fraction+test_fraction <= 1
       class_buckets = [[] for x in range(len(self.classes))] #Creates classes␣
↪buckets
       for i, f in enumerate(self.files):
           label = self._read_header_labels(f)
           label_idx = np.argwhere(label).flatten()
           for l in label_idx: #can return more than one (multi-label)
               class_buckets[l].append(f) #Put this file into class l bucket
       train_files, val_files, test_files = [], [], []
       sorted_idx = list(sorted(range(len(class_buckets)), key=lambda x:␣
↪len(class_buckets[x]))) #make index sorted by class count low->high
       print(sorted_idx)
       while len(sorted_idx) > 0:
           idx = sorted_idx[0]
           shuffle(class_buckets[idx])
           b = class_buckets[idx]
           c_N = len(b)
           train_slice = slice(0, int(train_fraction * c_N))
           val_slice = slice(train_slice.stop, train_slice.stop +␣
↪int(val_fraction * c_N))
           test_slice = slice(val_slice.stop, val_slice.stop +␣
↪int(test_fraction * c_N))
           train_files += b[train_slice]
           val_files += b[val_slice]
           test_files += b[test_slice]
           used_set = set(b[train_slice]+b[val_slice]+b[test_slice])
           for j in sorted_idx[1:]:
               class_buckets[j] = [x for x in class_buckets[j] if x not in␣
↪used_set] #REMOVE THIS FILE FROM ALL OTHER BUCKETS
           sorted_idx = list(sorted(sorted_idx[1:], key=lambda x:␣
↪len(class_buckets[x]))) #sort again (removal may change order)

       train_files = list(set(train_files))
       val_files = list(set(val_files))
       test_files = list(set(test_files))
       if save:
           p = save_path_overwrite or self.BASE_DIR
           fname = filename_overwrite or 'train-test-splits.txt'
           save_train_test_split(os.path.join(p, fname), train_files,␣
↪val_files, test_files)
       return train_files, val_files, test_files

   def count_classes(self):
```

```
        counts = np.zeros(len(self.classes), dtype=int)
        for i, f in enumerate(self.files):
            labels = self._read_header_labels(f).astype(float)
            counts += labels != 0.0 #Count where label isnt 0
        return counts


    def train_split_with_function(self, file_mapping_function, save_path=None):
        splits = [[], [], []]
        for f in self.files:
            i = file_mapping_function(f)
            splits[i].append(f)
        if not save_path is None:
            save_train_test_split(os.path.join(save_path, 'train-test-splits.
↪txt'), splits[0], splits[1], splits[2])
        return splits[0], splits[1], splits[2]

    def print_file_attributes(self):
        f = self.files[0]
        print('Information for file', f)
        data = self._read_recording_file(f)
        print('Data has shape:', data.shape)
        header = self._read_header_file(f)
        print('Header is:', header, end='################\n')
        print('Classes found in data folder:', self.classes)
        labels = self._read_header_labels(f)
        print('Labels have shape', labels.shape)

    def merge_and_update_classes(self, datasets):
        all_classes = set()
        for d in datasets:
            all_classes = all_classes | set(d.classes.keys())
        all_classes = dict(zip(sorted(all_classes), range(len(all_classes))))
        for d in datasets:
            d.classes = all_classes
        print('Labels for datasets set to:', all_classes)

    def remove_unknown_label_files(self):
        for f in self.files[:]:
            if self._read_header_labels(f, onerror_class=None) is None:
                print('removed', f)
                self.files.remove(f)


def filter_update_classes_by_count(datasets, min_count, add_unknown=False):
    counts, all_classes = count_merged_classes(datasets)
    filtered_classes = set()
```

```
    for k, v in all_classes.items():
        if counts[v] >= min_count:
            filtered_classes.add(k)
    if add_unknown:
        filtered_classes.add('-1')
    filtered_classes = dict(zip(sorted(filtered_classes),␣
↪range(len(filtered_classes))))
    for d in datasets:
        d.classes = filtered_classes
        d.remove_unknown_label_files()
    return filtered_classes

def count_merged_classes(datasets):
    all_classes = set()
    for d in datasets:
        all_classes = all_classes | set(d.classes.keys())
    all_classes = sorted(all_classes)
    all_classes = dict(zip(all_classes, range(len(all_classes))))
    counts = np.zeros(len(all_classes), dtype=int)
    for d in datasets:
        temp_classes = d.classes.copy() #set back later
        d.classes = all_classes
        counts += d.count_classes()
        d.classes = temp_classes #set back
    return counts, all_classes

def load_train_test_split(tts_file_path:str):
    splits = [[],[],[]]
    with open(tts_file_path, 'r') as f:
        line_count = 0
        for line in f:
            if not line.strip().startswith('#') or line == '\n': #Comment line␣
↪or empty line
                splits[line_count] = [f.strip() for f in line.split(',')]
                line_count += 1
            if line_count >= 3:
                break
    return splits[0], splits[1], splits[2]

def save_train_test_split(tts_file:str, trainf=[], valf=[], testf=[]):
    if os.path.isfile(tts_file): #make a backup just in case
        sf = os.path.split(tts_file)
        print(os.path.join(sf[0], timestamp.string_timestamp_minutes())+sf[1])
        os.rename(tts_file, os.path.join(sf[0], timestamp.
↪string_timestamp_minutes())+sf[1])

    with open(tts_file, 'w') as f:
```

```
        f.write('#train files\n')
        f.write(",".join(trainf)+'\n')
        f.write('#val files\n')
        f.write(",".join(valf) + '\n')
        f.write('#test files\n')
        f.write(",".join(testf))


def collate_fn(batch): #https://github.com/pytorch/pytorch/blob/master/torch/
 ↪utils/data/_utils/collate.py
    r"""Puts each data field into a tensor with outer dimension batch size"""

    elem = batch[0]
    elem_type = type(elem)
    if isinstance(elem, torch.Tensor):
        out = None
        if torch.utils.data.get_worker_info() is not None:
            # If we're in a background process, concatenate directly into a
            # shared memory tensor to avoid an extra copy
            numel = sum([x.numel() for x in batch])
            storage = elem.storage()._new_shared(numel)
            out = elem.new(storage)
        return torch.stack(batch, 0, out=out)
    elif elem_type.__module__ == 'numpy' and elem_type.__name__ != 'str_' \
            and elem_type.__name__ != 'string_':
        if elem_type.__name__ == 'ndarray' or elem_type.__name__ == 'memmap':
            # array of string classes and object
            if np_str_obj_array_pattern.search(elem.dtype.str) is not None:
                raise TypeError(default_collate_err_msg_format.format(elem.
 ↪dtype))

            return collate_fn([torch.as_tensor(b) for b in batch])
        elif elem.shape == ():  # scalars
            return torch.as_tensor(batch)
    elif isinstance(elem, float):
        return torch.tensor(batch)
    elif issubclass(type(elem), int):
        return torch.tensor(batch)
    elif issubclass(type(elem), str):
        return batch
    elif issubclass(type(elem), dict):
        return {key: collate_fn([d[key] for d in batch]) for key in elem}
    elif isinstance(elem, tuple) and hasattr(elem, '_fields'):  # namedtuple
        return elem_type(*(collate_fn(samples) for samples in zip(*batch)))
    elif issubclass(type(elem), list):
        # check to make sure that the elements in batch have consistent size
        transposed = zip(*batch)
```

```
        return [collate_fn(samples) for samples in transposed]

def normalize_feature_scaling(data, low:int=0, high:int=1):
    mini = np.min(data, axis=1)[:, np.newaxis]
    maxi = np.max(data, axis=1)[:, np.newaxis]
    dif = np.where(maxi-mini==0, 1, maxi-mini)
    return (data-mini)/(dif)*high-low

#!/usr/bin/env python

# These are helper variables and functions that you can use with your code.
# Do not edit this script.

import numpy as np
import os
from scipy.io import loadmat

# Define 12, 6, and 2 lead ECG sets.
twelve_leads = ('I', 'II', 'III', 'aVR', 'aVL', 'aVF', 'V1', 'V2', 'V3', 'V4',␣
 ↪'V5', 'V6')
six_leads = ('I', 'II', 'III', 'aVR', 'aVL', 'aVF')
two_leads = ('II', 'V5')

# Check if a variable is an integer or represents an integer.
def is_integer(x):
    try:
        if int(x)==float(x):
            return True
        else:
            return False
    except (ValueError, TypeError):
        return False

# Find header and recording files.
def find_challenge_files(data_directory):
    header_files = list()
    recording_files = list()
    for f in os.listdir(data_directory):
        root, extension = os.path.splitext(f)
        if not root.startswith('.') and extension=='.hea':
            header_file = os.path.join(data_directory, root + '.hea')
            recording_file = os.path.join(data_directory, root + '.mat')
            if os.path.isfile(header_file) and os.path.isfile(recording_file):
                header_files.append(header_file)
                recording_files.append(recording_file)
    return header_files, recording_files
```

```python
# Load header file as a string.
def load_header(header_file):
    with open(header_file, 'r') as f:
        header = f.read()
    return header

# Load recording file as an array.
def load_recording(recording_file, header=None, leads=None, key='val'):
    x = loadmat(recording_file)[key]
    recording = np.asarray(x, dtype=np.float32)
    return recording

# Get leads from header.
def get_leads(header):
    leads = list()
    for i, l in enumerate(header.split('\n')):
        entries = l.split(' ')
        if i==0:
            num_leads = int(entries[1])
        elif i<=num_leads:
            leads.append(entries[-1])
        else:
            break
    return leads

# Get age from header.
def get_age(header):
    age = None
    for l in header.split('\n'):
        if l.startswith('#Age'):
            try:
                age = float(l.split(': ')[1].strip())
            except:
                age = float('nan')
    return age

# Get age from header.
def get_sex(header):
    sex = None
    for l in header.split('\n'):
        if l.startswith('#Sex'):
            try:
                sex = l.split(': ')[1].strip()
            except:
                pass
    return sex
```

```python
# Get frequency from header.
def get_frequency(header):
    frequency = None
    for i, l in enumerate(header.split('\n')):
        if i==0:
            try:
                frequency = float(l.split(' ')[2])
            except:
                pass
        else:
            break
    return frequency

# Get amplitudes from header.
def get_amplitudes(header, leads):
    amplitudes = np.zeros(len(leads), dtype=np.float32)
    for i, l in enumerate(header.split('\n')):
        entries = l.split(' ')
        if i==0:
            num_leads = int(entries[1])
        elif i<=num_leads:
            current_lead = entries[-1]
            if current_lead in leads:
                j = leads.index(current_lead)
                try:
                    amplitudes[j] = float(entries[2].split('/')[0])
                except:
                    pass
        else:
            break
    return amplitudes

# Get baselines from header.
def get_baselines(header, leads):
    baselines = np.zeros(len(leads), dtype=np.float32)
    for i, l in enumerate(header.split('\n')):
        entries = l.split(' ')
        if i==0:
            num_leads = int(entries[1])
        elif i<=num_leads:
            current_lead = entries[-1]
            if current_lead in leads:
                j = leads.index(current_lead)
                try:
                    baselines[j] = float(entries[4].split('/')[0])
                except:
                    pass
```

```
        else:
            break
    return baselines

# Get labels from header.
def get_labels(header):
    labels = list()
    for l in header.split('\n'):
        if l.startswith('#Dx'):
            entries = l.split(': ')[1].split(',')
            for entry in entries:
                labels.append(entry.strip())
    return labels

# Save outputs from model.
def save_outputs(output_file, classes, labels, probabilities):
    # Extract the recording identifier from the filename.
    head, tail = os.path.split(output_file)
    root, extension = os.path.splitext(tail)
    recording_identifier = root

    # Format the model outputs.
    recording_string = '#{}'.format(recording_identifier)
    class_string = ','.join(str(c) for c in classes)
    label_string = ','.join(str(l) for l in labels)
    probabilities_string = ','.join(str(p) for p in probabilities)
    output_string = recording_string + '\n' + class_string + '\n' +␣
↪label_string + '\n' + probabilities_string + '\n'

    # Save the model outputs.
    with open(output_file, 'w') as f:
        f.write(output_string)


def get_classes(files_without_ext):
    classes = set()
    for filename in files_without_ext:
        with open(filename+'.hea', 'r') as f:
            for l in f:
                if l.startswith('#Dx'):
                    tmp = l.split(':')[1].split(',')
                    for c in tmp:
                        classes.add(c.strip())
    return dict(zip(sorted(classes), range(len(classes))))

def encode_header_labels(header, classes, onerror_class='426783006'):
    labels_act = np.zeros(len(classes))
```

```
    for l in header.split('\n'):
        if l.startswith('#Dx'):
            tmp = l.split(':')[1].split(',')
            for c in tmp:
                cl = c.strip()
                if cl in classes:
                    class_index = classes[cl]
                else:
                    if onerror_class is None:
                        return None
                    class_index = classes[onerror_class]
                labels_act[class_index] = 1
    return labels_act

def save_challenge_predictions(output_directory, filenames, classes, scores,␣
 ↪labels):
    for i in range(0, len(filenames)):
        filename = filenames[i]
        sc = scores[i]
        ls = labels[i]
        recording = os.path.splitext(filename)[0]
        new_file = filename.replace('.mat','') + '.csv'

        output_file = os.path.join(output_directory, new_file)

        # Include the filename as the recording number
        recording_string = '#{}'.format(recording)
        class_string = ','.join(classes)
        label_string = ','.join(str(i) for i in ls)
        score_string = ','.join(str(i) for i in sc)
        with open(output_file, 'w') as f:
            f.write(recording_string + '\n' + class_string + '\n' +␣
 ↪label_string + '\n' + score_string + '\n')
```

[12]:
```
crop_size = 4500


georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪georgia_challenge/',
                                                 window_size=crop_size,␣
 ↪pad_to_size=crop_size, return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
cpsc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪cps2018_challenge/',
                                                    window_size=crop_size,␣
 ↪pad_to_size=crop_size, return_labels=True,
```

```
                                                    ␣
 ↪normalize_fn=normalize_feature_scaling)
cpsc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪china_challenge', window_size=crop_size,
                                                 pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                    ␣
 ↪normalize_fn=normalize_feature_scaling)
ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪ptbxl_challenge', window_size=crop_size,
                                                  pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                    ␣
 ↪normalize_fn=normalize_feature_scaling)



ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file()
georgia_train, georgia_val, t2 = georgia_challenge.
 ↪generate_datasets_from_split_file()
cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file()
cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file()

filter_update_classes_by_count([ptbxl_train, ptbxl_val, t1, georgia_train,␣
 ↪georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4], 1)
print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.
 ↪classes)

counts_all, counted_classes_all = count_merged_classes([ptbxl_train, ptbxl_val,␣
 ↪t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,␣
 ↪cpsc2_val, t4])
PATIENTS_TRAIN_TOTAL = sum(map(lambda x: len(x.files), [ptbxl_train,␣
 ↪georgia_train, cpsc_train, cpsc2_train]))
PATIENTS_VAL_TOTAL = sum(map(lambda x: len(x.files),[ptbxl_val, georgia_val,␣
 ↪cpsc_val, cpsc2_val]))

train_set = ChainDataset([ptbxl_train, georgia_train, cpsc_train, cpsc2_train])
val_set = ChainDataset([ptbxl_val, georgia_val, cpsc_val, cpsc2_val])
test_set = ChainDataset([t1, t2, t3, t4])
```

```
10344 record files found in  /media/julian/data/data/ECG/georgia_challenge/
6877 record files found in  /media/julian/data/data/ECG/cps2018_challenge/
3453 record files found in  /media/julian/data/data/ECG/china_challenge
21837 record files found in  /media/julian/data/data/ECG/ptbxl_challenge
Classes after last update 67 {'10370003': 0, '11157007': 1, '111975006': 2,
'164861001': 3, '164865005': 4, '164867002': 5, '164873001': 6, '164884008': 7,
```

```
'164889003': 8, '164890007': 9, '164909002': 10, '164917005': 11, '164930006':
12, '164931005': 13, '164934002': 14, '164947007': 15, '164951009': 16,
'17338001': 17, '195042002': 18, '195080001': 19, '195126007': 20, '233917008':
21, '251120003': 22, '251146004': 23, '251180001': 24, '251200008': 25,
'251266004': 26, '251268003': 27, '253352002': 28, '266249003': 29, '270492004':
30, '27885002': 31, '284470004': 32, '39732003': 33, '413844008': 34,
'425419005': 35, '425623009': 36, '426177001': 37, '426434006': 38, '426627000':
39, '426761007': 40, '426783006': 41, '427084000': 42, '427172004': 43,
'427393009': 44, '428417006': 45, '428750005': 46, '429622005': 47, '445118002':
48, '445211001': 49, '446358003': 50, '446813000': 51, '47665007': 52,
'54329005': 53, '55930002': 54, '59118001': 55, '59931005': 56, '63593006': 57,
'6374002': 58, '67198005': 59, '67741000119109': 60, '698252002': 61,
'713422000': 62, '713426002': 63, '713427006': 64, '74390002': 65, '89792004':
66}
```

[5]:
```
counts = {}
counted_classes = {}
counts['ptbxl'], counted_classes['ptbxl'] = count_merged_classes([ptbxl_train,␣
↪ptbxl_val, t1])
counts['georgia'], counted_classes['georgia'] =␣
↪count_merged_classes([georgia_train, georgia_val, t2,])
counts['cpsc'], counted_classes['cpsc'] = count_merged_classes([cpsc_train,␣
↪cpsc_val, t3])
counts['cpsc2'], counted_classes['cpsc2'] = count_merged_classes([cpsc2_train,␣
↪cpsc2_val, t4])
```

[27]:
```
save_path = '/home/julian/Documents/projekt-master'
```

0.2 Class descriptions

[6]:
```
import pandas as pd
import urllib
import time
import json
```

[7]:
```
snomed_data = {}
DOWNLOAD_AGAIN = False
if not DOWNLOAD_AGAIN:
    try:
        with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
↪snomed_data.json', 'r') as f:
            snomed_data = json.load(f)
    except FileNotFoundError:
        print("File not Found, Download again")
        DOWNLOAD_AGAIN = True
if DOWNLOAD_AGAIN:
    for c_n in counted_classes_all.keys():
```

```
        with urllib.request.urlopen(f"https://browser.ihtsdotools.org/snowstorm/
↪snomed-ct/browser/MAIN/2021-01-31/concepts/{c_n}") as url:
            snomed_data[c_n] = json.loads(url.read().decode())
        time.sleep(0.5)
    with open('Downloads/Github/contrastive-predictive-coding/snomed_data.
↪json', 'w') as f:
        json.dump(snomed_data, f)
```

[8]:
```
code_names = {}
for c_n in counted_classes_all.keys():
    code_names[c_n] = {}
    code_names[c_n]['Term'] = snomed_data[c_n]['pt']['term']
    code_names[c_n]['Count'] = counts_all[counted_classes_all[c_n]]

code_df = pd.DataFrame.from_dict(code_names, orient='index')
#code_df.to_latex(os.path.join(save_path, 'tables/SnomedCodes.tex'))
display(code_df)
```

	Term	Count
10370003	Rhythm from artificial pacing	298
11157007	Ventricular bigeminy	89
111975006	Prolonged QT interval	1493
164861001	EKG myocardial ischemia	2556
164865005	EKG: myocardial infarction	5640
...	...	...
713422000	EKG: atrial tachycardia	40
713426002	EKG: Incomplete right bundle branch block	1606
713427006	EKG: complete right bundle branch block	681
74390002	Wolff-Parkinson-White pattern	82
89792004	Right ventricular hypertrophy	229

[67 rows x 2 columns]

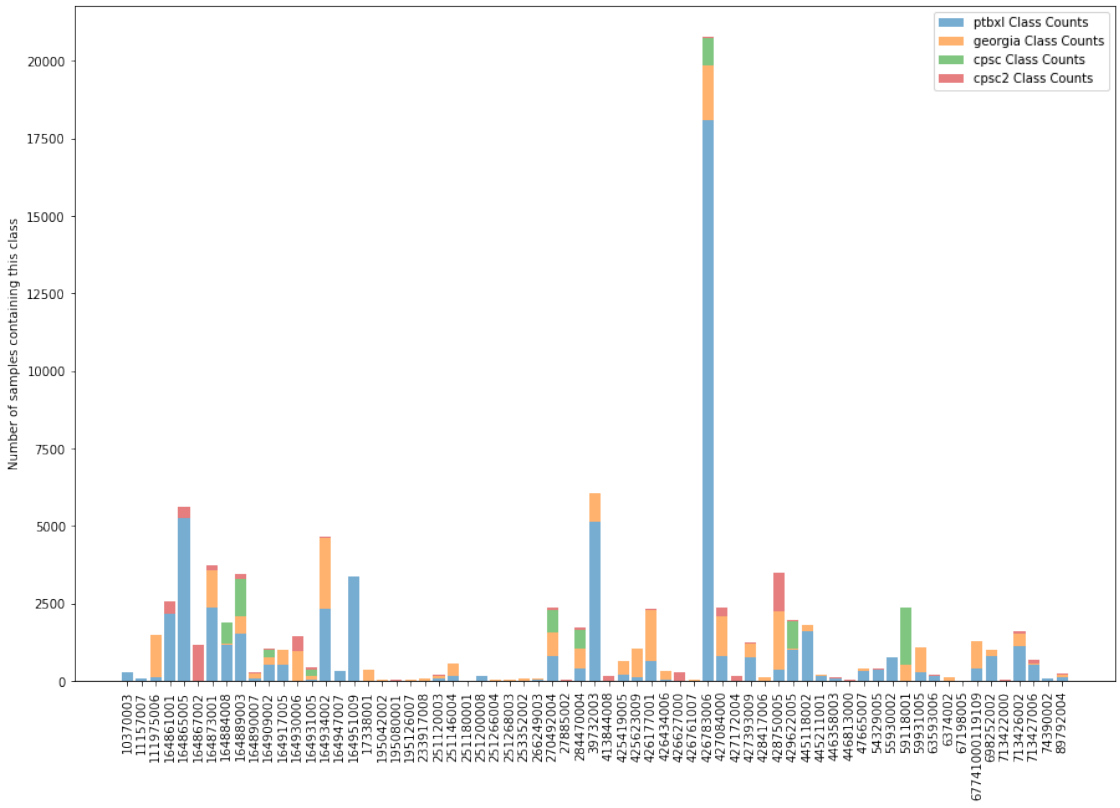
0.3 Count visualization

[6]:
```
use_names = False
```

0.3.1 all samples

[40]:
```
plt.figure(figsize=(15,10))
plt.tight_layout()
dsets = ['ptbxl', 'georgia', 'cpsc', 'cpsc2']
for i in range(0, len(dsets)):
    dset = dsets[i]
```

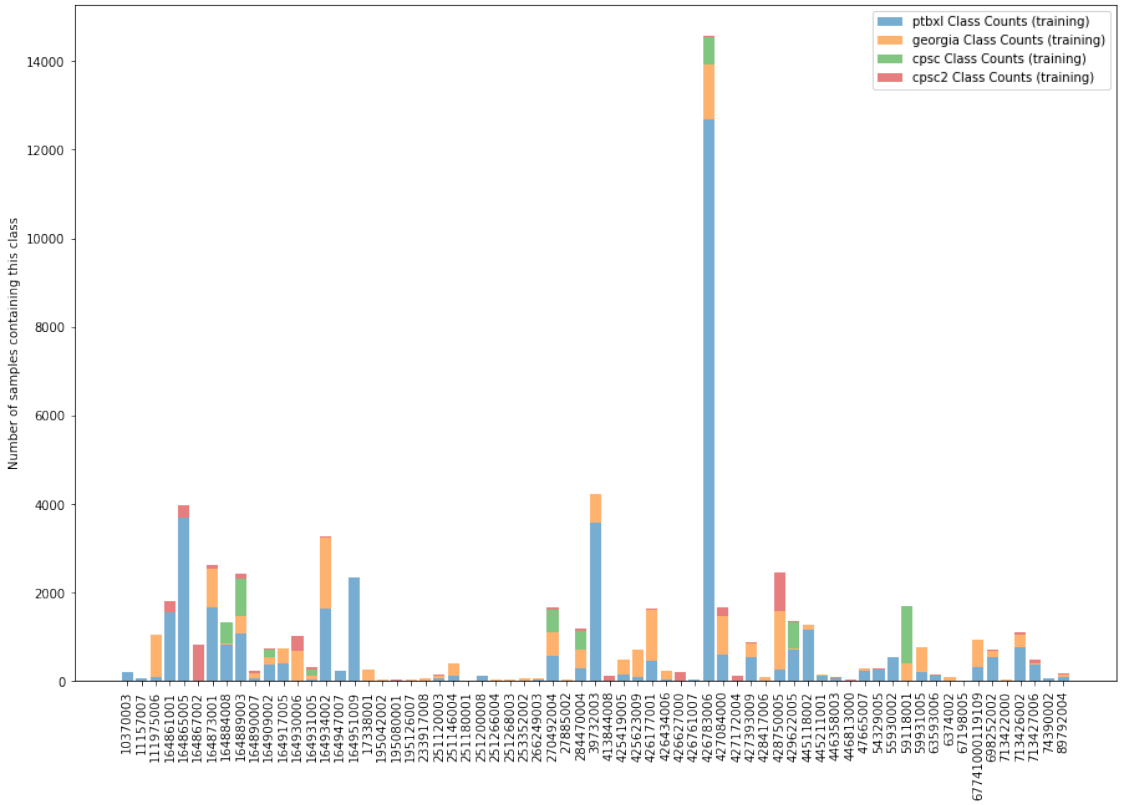
```
    cumm = [sum([counts[dset][c] for dset in dsets[0:i]]) for c in␣
 ↪counted_classes[dset].values()]
    plt.bar(counted_classes[dset].keys(), [counts[dset][c] for c in␣
 ↪counted_classes[dset].values()], bottom=cumm, label=f'{dset} Class Counts',␣
 ↪alpha=0.6)
if use_names:
    plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term']␣
 ↪for c in counted_classes_all.keys()], rotation='vertical')
else:
    plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
 ↪keys()), rotation='vertical')
plt.ylabel("Number of samples containing this class")
plt.legend()
plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetAll.png'),␣
 ↪bbox_inches = "tight")
plt.show()
```



0.3.2 train set

[41]:
```
train_counts = {}
train_counted_classes = {}
train_counts['ptbxl'], train_counted_classes['ptbxl'] =␣
↪count_merged_classes([ptbxl_train])
train_counts['georgia'], train_counted_classes['georgia'] =␣
↪count_merged_classes([georgia_train])
train_counts['cpsc'], train_counted_classes['cpsc'] =␣
↪count_merged_classes([cpsc_train])
train_counts['cpsc2'], train_counted_classes['cpsc2'] =␣
↪count_merged_classes([cpsc2_train])
```

[43]:
```
plt.figure(figsize=(15,10))
plt.tight_layout()
dsets = ['ptbxl', 'georgia', 'cpsc', 'cpsc2']
for i in range(0, len(dsets)):
    dset = dsets[i]
    cumm = [sum([train_counts[dset][c] for dset in dsets[0:i]]) for c in␣
↪train_counted_classes[dset].values()]
    plt.bar(train_counted_classes[dset].keys(), [train_counts[dset][c] for c in␣
↪train_counted_classes[dset].values()], bottom=cumm, label=f'{dset} Class␣
↪Counts (training)', alpha=0.6)
if use_names:
    plt.xticks(range(len(train_counted_classes_all.keys())),␣
↪[code_names[c]['Term'] for c in train_counted_classes_all.keys()],␣
↪rotation='vertical')
else:
    plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
↪keys()), rotation='vertical')
plt.ylabel("Number of samples containing this class")
plt.legend()
plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetTrain.png'),␣
↪bbox_inches = "tight")
plt.show()
```



0.4 Count Visualization by Dataset

[18]:
```
def dataframe_to_image(df, savepath):
    with pd.option_context('display.max_rows', 300, 'display.max_columns', 7,␣
 ↪'display.expand_frame_repr', False):
        html = df.render()
        options = {
                'zoom': 1,
                'quality': 100
            }
        imgkit.from_string(html, savepath, options=options)

def dataframe_to_latex(df, savepath):
    with open(savepath, 'w') as f:
        f.write(df.to_latex(convert_css=True))
```

[19]:
```
import seaborn as sns
import imgkit
cm = sns.light_palette("seagreen", as_cmap=True)

counts_all
```

```
counted_classes_all
fractions = {}
for i, dset in enumerate(['ptbxl', 'georgia', 'cpsc', 'cpsc2']):
    fractions[dset] = {}
    for c, cidx in counted_classes[dset].items():
        cidx_in_all = counted_classes_all[c]
        fractions[dset][f"{code_names[c]['Term']} ({counts_all[cidx_in_all]})"]␣
 ↪= counts[dset][cidx]#/counts_all[cidx_in_all]
df = pd.DataFrame.from_dict(fractions)
df.insert(loc=0, column='Class Snomed Code', value=list(counted_classes[dset].
 ↪keys()))
df.insert(loc=0, column='Nr.', value=range(len(df)))
df.index.name = 'Class Term'
df = df.reset_index().set_index(['Nr.', 'Class Snomed Code', 'Class Term'])
# df = df.set_index("Class Snomed Code", append=True)
# df = df.set_index("Nr.", append=True)
s = df.style.background_gradient(cmap=cm, axis=1)
dataframe_to_latex(s, os.path.join(save_path, 'tables/count_visualization.tex'))

s
```

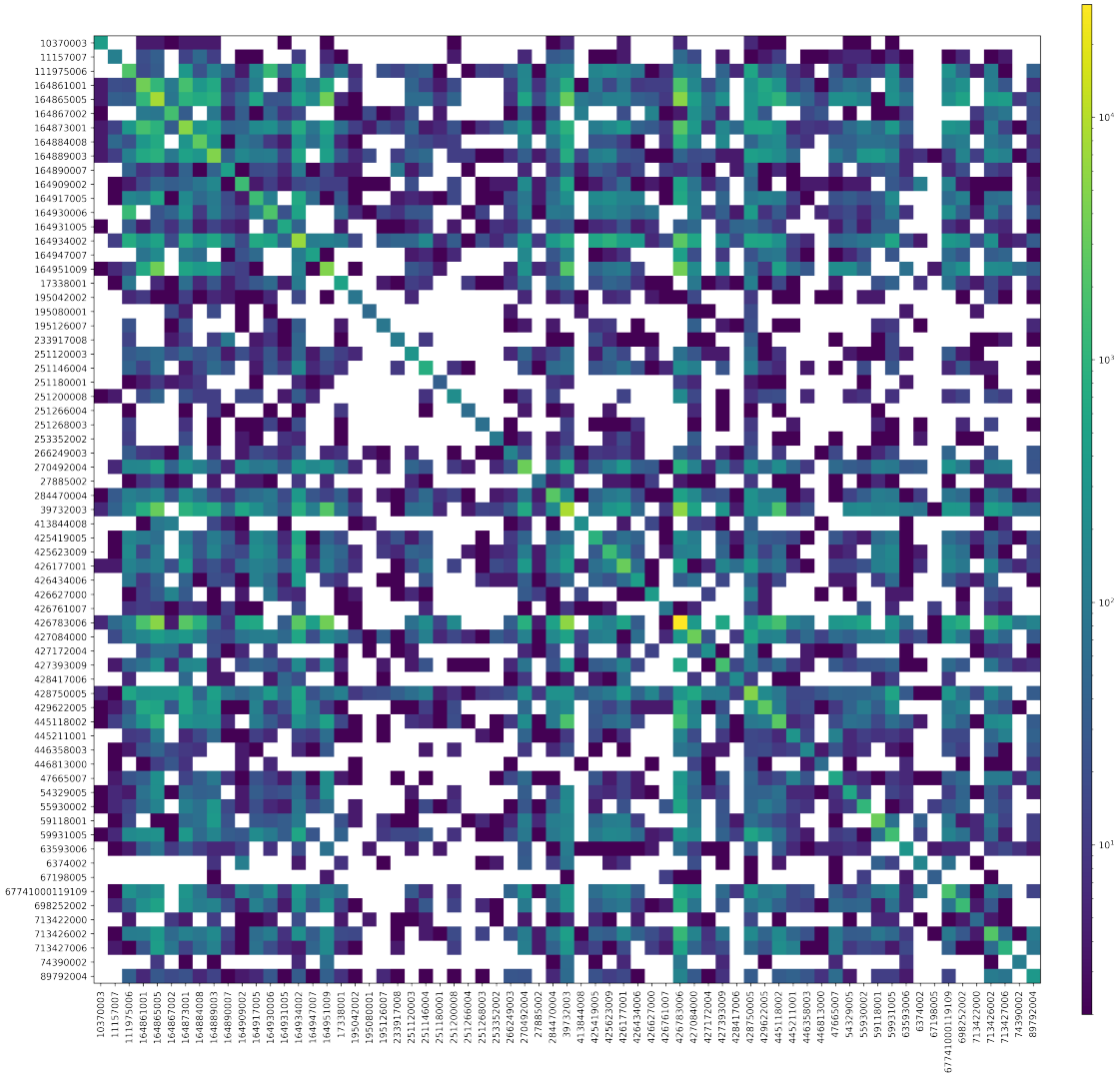
[19]: <pandas.io.formats.style.Styler at 0x7f79580098d0>

0.5 Class label visualization

[68]:
```
from torch.utils.data import DataLoader, ChainDataset
downstream_train_dataset = ChainDataset([ptbxl_train, georgia_train,␣
 ↪cpsc_train, cpsc2_train])
dl = DataLoader(downstream_train_dataset, batch_size=1, collate_fn=collate_fn)
occ = np.zeros((len(counted_classes_all.keys()), len(counted_classes_all.
 ↪keys())))
patient_count = 0
for _, labels in dl:
    labels = labels[0]
    patient_count += 1
    for l_ix in np.nonzero(labels):
        occ[l_ix, :] += labels.numpy()
        occ[:, l_ix] += labels.numpy()
```

[77]:
```
from matplotlib.colors import LogNorm
plt.figure(figsize=(20, 20), dpi=300)
#plt.title('Classes occuring together')
plt.tight_layout()
plt.imshow(occ.astype(int), norm=LogNorm())
if use_names:
```

```
    plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term']␣
 ↪for c in counted_classes_all.keys()], rotation='vertical')
    plt.yticks(range(len(counted_classes_all.keys())), [code_names[c]['Term']␣
 ↪for c in counted_classes_all.keys()])
else:
    plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
 ↪keys()), rotation='vertical')
    plt.yticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
 ↪keys()))
plt.colorbar(fraction=0.046, pad=0.04, aspect=100)
plt.savefig(os.path.join(save_path, 'bilder/ClassOccurences.png'),␣
 ↪bbox_inches='tight')
plt.show()
```



0.6 Count files for few labels

[3]:
```
crop_size = 4500
georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪georgia_challenge/',
                                                 window_size=crop_size,␣
 ↪pad_to_size=crop_size, return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
cpsc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪cps2018_challenge/',
                                                    window_size=crop_size,␣
 ↪pad_to_size=crop_size, return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
cpsc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪china_challenge', window_size=crop_size,
                                               pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪ptbxl_challenge', window_size=crop_size,
                                                pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
nature = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪nature_database', window_size=crop_size,
                                                pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file()
georgia_train, georgia_val, t2 = georgia_challenge.
 ↪generate_datasets_from_split_file()
cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file()
cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file()
total_train_file_sum = sum(map(len, [ptbxl_train.files, georgia_train.files,␣
 ↪cpsc_train.files, cpsc2_train.files]))
total_file_sum = sum(map(len, map(lambda x: x.files, [ptbxl_train, ptbxl_val,␣
 ↪t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,␣
 ↪cpsc2_val, t4])))
```

```
10344 record files found in  /media/julian/data/data/ECG/georgia_challenge/
6877 record files found in  /media/julian/data/data/ECG/cps2018_challenge/
3453 record files found in  /media/julian/data/data/ECG/china_challenge
```

```
21837 record files found in  /media/julian/data/data/ECG/ptbxl_challenge
10646 record files found in  /media/julian/data/data/ECG/nature_database
```

[4]:
```
fractions = {}
few_labels_splits_filenames = ['train-test-splits-fewer-labels0.001.txt',
    'train-test-splits-fewer-labels0.005.txt',
    'train-test-splits-fewer-labels0.05.txt',
    'train-test-splits-fewer-labels0.01.txt',
    'train-test-splits-fewer-labels10.txt',
    'train-test-splits-fewer-labels14.txt',
    'train-test-splits-fewer-labels20.txt',
    'train-test-splits-fewer-labels30.txt',
    'train-test-splits-fewer-labels40.txt',
    'train-test-splits-fewer-labels50.txt',
    'train-test-splits-fewer-labels60.txt',
    'train-test-splits.txt',
    'train-test-splits_min_cut10.txt',
    'train-test-splits_min_cut25.txt',
    'train-test-splits_min_cut50.txt',
    'train-test-splits_min_cut100.txt',
    'train-test-splits_min_cut150.txt',
    'train-test-splits_min_cut200.txt']
for few_labels_splits_filename in few_labels_splits_filenames:
    ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
    georgia_train, georgia_val, t2 = georgia_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
    cpsc_train, cpsc_val, t3 = cpsc_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
    cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)

    file_sum = sum(map(len, [ptbxl_train.files, georgia_train.files, cpsc_train.
 ↪files, cpsc2_train.files]))
    fractions[few_labels_splits_filename] = file_sum/total_file_sum
```

[5]:
```
fractions
```

[5]:
```
{'train-test-splits-fewer-labels0.001.txt': 0.0012756005952802778,
 'train-test-splits-fewer-labels0.005.txt': 0.009378026598634634,
 'train-test-splits-fewer-labels0.05.txt': 0.10032362459546926,
 'train-test-splits-fewer-labels0.01.txt': 0.02010252049228734,
 'train-test-splits-fewer-labels10.txt': 0.18834006566980843,
 'train-test-splits-fewer-labels14.txt': 0.18829282120331656,
 'train-test-splits-fewer-labels20.txt': 0.3308057543760187,
 'train-test-splits-fewer-labels30.txt': 0.44031842770415514,
 'train-test-splits-fewer-labels40.txt': 0.5257600453546878,
```

```
 'train-test-splits-fewer-labels50.txt': 0.5950912999314956,
 'train-test-splits-fewer-labels60.txt': 0.6526823045850755,
 'train-test-splits.txt': 0.7017220608036284,
 'train-test-splits_min_cut10.txt': 0.015330829376609265,
 'train-test-splits_min_cut25.txt': 0.037252261828833295,
 'train-test-splits_min_cut50.txt': 0.06798478728178962,
 'train-test-splits_min_cut100.txt': 0.116174143103489,
 'train-test-splits_min_cut150.txt': 0.15470200552760258,
 'train-test-splits_min_cut200.txt': 0.18864715470200552}
```

0.7 Fewer Labels

[67]:
```
few_labels_splits_filename = 'train-test-splits-fewer-labels40.txt'

crop_size = 4500
georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
↪georgia_challenge/',
                                                window_size=crop_size,␣
↪pad_to_size=crop_size, return_labels=True,
                                                ␣
↪normalize_fn=normalize_feature_scaling)
cpsc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
↪cps2018_challenge/',
                                                    window_size=crop_size,␣
↪pad_to_size=crop_size, return_labels=True,
                                                ␣
↪normalize_fn=normalize_feature_scaling)
cpsc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
↪china_challenge', window_size=crop_size,
                                              pad_to_size=crop_size,␣
↪return_labels=True,
                                                ␣
↪normalize_fn=normalize_feature_scaling)
ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
↪ptbxl_challenge', window_size=crop_size,
                                                pad_to_size=crop_size,␣
↪return_labels=True,
                                                ␣
↪normalize_fn=normalize_feature_scaling)
nature = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
↪nature_database', window_size=crop_size,
                                                pad_to_size=crop_size,␣
↪return_labels=True,
                                                ␣
↪normalize_fn=normalize_feature_scaling)
```

```
ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
georgia_train, georgia_val, t2 = georgia_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
cpsc_train, cpsc_val, t3 = cpsc_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
```

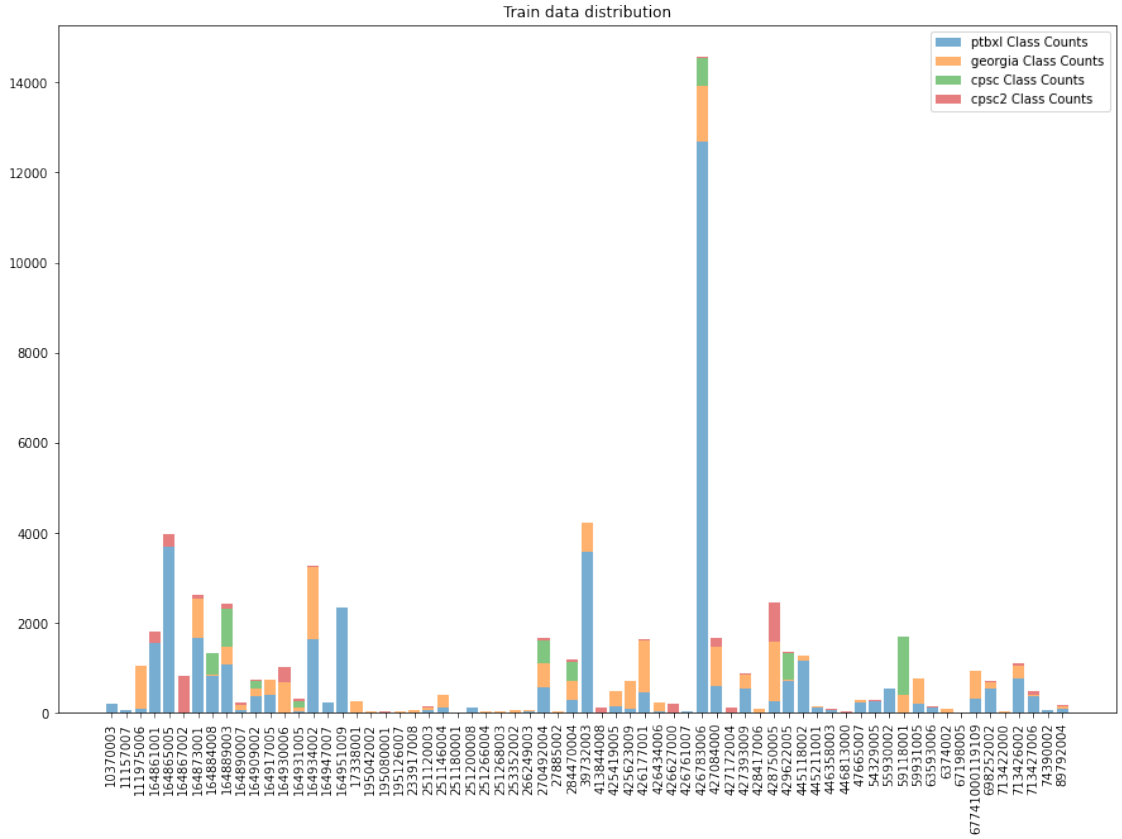
```
10344 record files found in  /media/julian/data/data/ECG/georgia_challenge/
6877 record files found in  /media/julian/data/data/ECG/cps2018_challenge/
3453 record files found in  /media/julian/data/data/ECG/china_challenge
21837 record files found in  /media/julian/data/data/ECG/ptbxl_challenge
10646 record files found in  /media/julian/data/data/ECG/nature_database
```

[30]:
```
filter_update_classes_by_count([ptbxl_train, ptbxl_val, t1, georgia_train,␣
 ↪georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4], 1)
print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.
 ↪classes)
counts_all, counted_classes_all = count_merged_classes([ptbxl_train, ptbxl_val,␣
 ↪t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,␣
 ↪cpsc2_val, t4])

counts = {}
counted_classes = {}
counts['ptbxl'], counted_classes['ptbxl'] = count_merged_classes([ptbxl_train])
counts['georgia'], counted_classes['georgia'] =␣
 ↪count_merged_classes([georgia_train])
counts['cpsc'], counted_classes['cpsc'] = count_merged_classes([cpsc_train])
counts['cpsc2'], counted_classes['cpsc2'] = count_merged_classes([cpsc2_train])
```

```
Classes after last update 67 {'10370003': 0, '11157007': 1, '111975006': 2,
'164861001': 3, '164865005': 4, '164867002': 5, '164873001': 6, '164884008': 7,
'164889003': 8, '164890007': 9, '164909002': 10, '164917005': 11, '164930006':
12, '164931005': 13, '164934002': 14, '164947007': 15, '164951009': 16,
'17338001': 17, '195042002': 18, '195080001': 19, '195126007': 20, '233917008':
21, '251120003': 22, '251146004': 23, '251180001': 24, '251200008': 25,
'251266004': 26, '251268003': 27, '253352002': 28, '266249003': 29, '270492004':
30, '27885002': 31, '284470004': 32, '39732003': 33, '413844008': 34,
'425419005': 35, '425623009': 36, '426177001': 37, '426434006': 38, '426627000':
39, '426761007': 40, '426783006': 41, '427084000': 42, '427172004': 43,
'427393009': 44, '428417006': 45, '428750005': 46, '429622005': 47, '445118002':
48, '445211001': 49, '446358003': 50, '446813000': 51, '47665007': 52,
'54329005': 53, '55930002': 54, '59118001': 55, '59931005': 56, '63593006': 57,
'6374002': 58, '67198005': 59, '67741000119109': 60, '698252002': 61,
'713422000': 62, '713426002': 63, '713427006': 64, '74390002': 65, '89792004':
66}
```

[31]:
```
plt.figure(figsize=(15,10))
plt.title("Train data distribution")
dsets = ['ptbxl', 'georgia', 'cpsc', 'cpsc2']
for i, dset in enumerate(dsets):
    cumm = [sum([counts[dset][c] for dset in dsets[0:i]]) for c in␣
 ↪counted_classes[dset].values()]
    plt.bar(counted_classes[dset].keys(), [counts[dset][c] for c in␣
 ↪counted_classes[dset].values()], bottom=cumm, label=f'{dset} Class Counts',␣
 ↪alpha=0.6)
if use_names:
    plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term']␣
 ↪for c in counted_classes_all.keys()], rotation='vertical')
else:
    plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
 ↪keys()), rotation='vertical')
plt.legend()
plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetFew.png'))
plt.show()
```



0.8 Fewer Labels (min_cut)

[75]:
```
few_labels_splits_filename = '05_07_21-15-17train-test-splits_min_cut100.txt'

crop_size = 4500
georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪georgia_challenge/',
                                                 window_size=crop_size,␣
 ↪pad_to_size=crop_size, return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
cpsc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪cps2018_challenge/',
                                                    window_size=crop_size,␣
 ↪pad_to_size=crop_size, return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
cpsc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪china_challenge', window_size=crop_size,
                                               pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪ptbxl_challenge', window_size=crop_size,
                                                pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)
nature = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/
 ↪nature_database', window_size=crop_size,
                                                pad_to_size=crop_size,␣
 ↪return_labels=True,
                                                ␣
 ↪normalize_fn=normalize_feature_scaling)

ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
georgia_train, georgia_val, t2 = georgia_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
cpsc_train, cpsc_val, t3 = cpsc_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.
 ↪generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
```

```
10344 record files found in  /media/julian/data/data/ECG/georgia_challenge/
```

```
6877 record files found in  /media/julian/data/data/ECG/cps2018_challenge/
3453 record files found in  /media/julian/data/data/ECG/china_challenge
21837 record files found in  /media/julian/data/data/ECG/ptbxl_challenge
10646 record files found in  /media/julian/data/data/ECG/nature_database
```

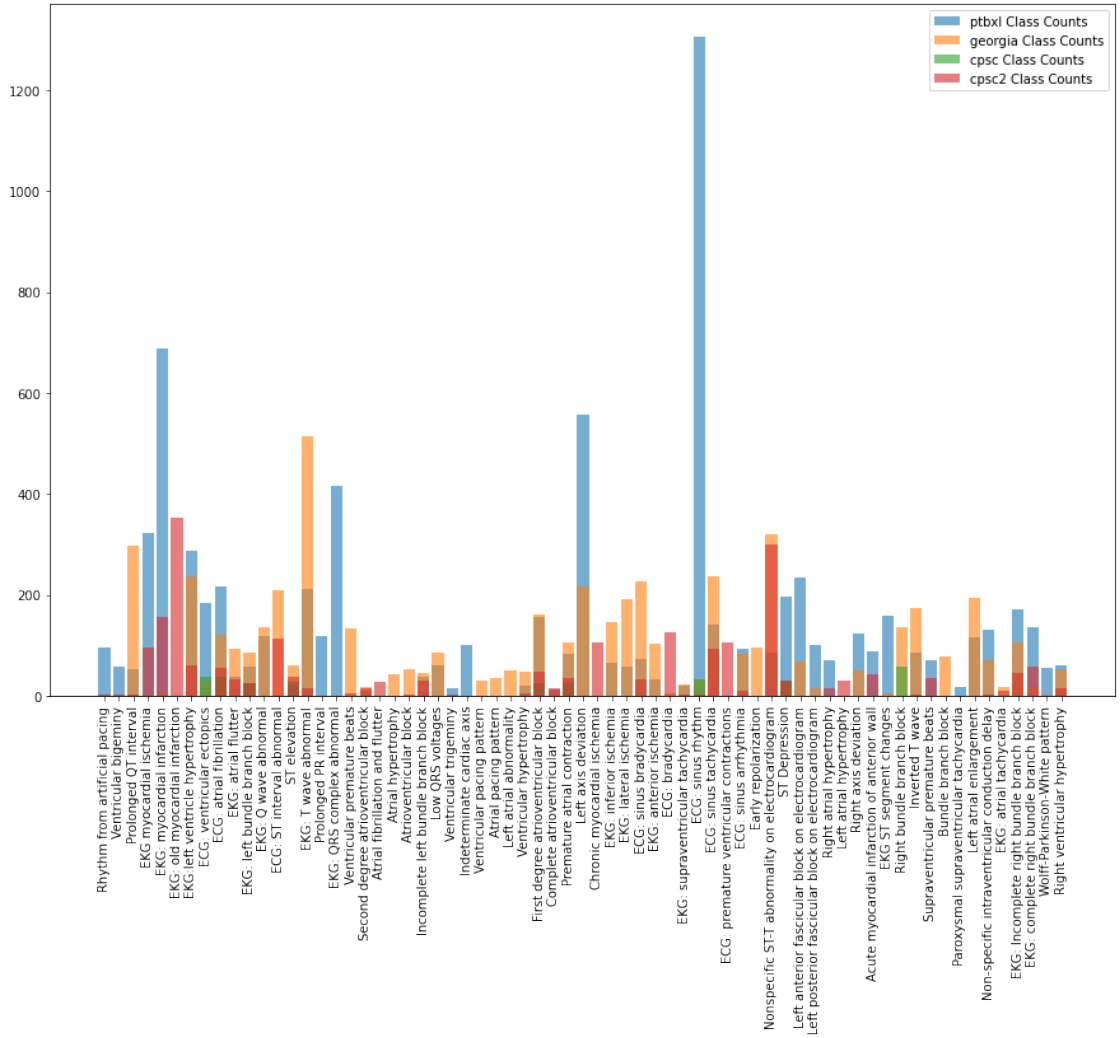
[76]:
```
filter_update_classes_by_count([ptbxl_train, ptbxl_val, t1, georgia_train,␣
 ↪georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4], 1)
print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.
 ↪classes)
counts_all, counted_classes_all = count_merged_classes([ptbxl_train, ptbxl_val,␣
 ↪t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,␣
 ↪cpsc2_val, t4])

counts = {}
counted_classes = {}
counts['ptbxl'], counted_classes['ptbxl'] = count_merged_classes([ptbxl_train])
counts['georgia'], counted_classes['georgia'] =␣
 ↪count_merged_classes([georgia_train])
counts['cpsc'], counted_classes['cpsc'] = count_merged_classes([cpsc_train])
counts['cpsc2'], counted_classes['cpsc2'] = count_merged_classes([cpsc2_train])
```

```
Classes after last update 67 {'10370003': 0, '11157007': 1, '111975006': 2,
'164861001': 3, '164865005': 4, '164867002': 5, '164873001': 6, '164884008': 7,
'164889003': 8, '164890007': 9, '164909002': 10, '164917005': 11, '164930006':
12, '164931005': 13, '164934002': 14, '164947007': 15, '164951009': 16,
'17338001': 17, '195042002': 18, '195080001': 19, '195126007': 20, '233917008':
21, '251120003': 22, '251146004': 23, '251180001': 24, '251200008': 25,
'251266004': 26, '251268003': 27, '253352002': 28, '266249003': 29, '270492004':
30, '27885002': 31, '284470004': 32, '39732003': 33, '413844008': 34,
'425419005': 35, '425623009': 36, '426177001': 37, '426434006': 38, '426627000':
39, '426761007': 40, '426783006': 41, '427084000': 42, '427172004': 43,
'427393009': 44, '428417006': 45, '428750005': 46, '429622005': 47, '445118002':
48, '445211001': 49, '446358003': 50, '446813000': 51, '47665007': 52,
'54329005': 53, '55930002': 54, '59118001': 55, '59931005': 56, '63593006': 57,
'6374002': 58, '67198005': 59, '67741000119109': 60, '698252002': 61,
'713422000': 62, '713426002': 63, '713427006': 64, '74390002': 65, '89792004':
66}
```

[63]:
```
plt.figure(figsize=(15,10))
plt.title("Distribution of training data")
for i, dset in enumerate(['ptbxl', 'georgia', 'cpsc', 'cpsc2']):
    plt.bar(counted_classes[dset].keys(), [counts[dset][c] for c in␣
 ↪counted_classes[dset].values()], label=f'{dset} Class Counts', alpha=0.6)
if use_names:
    plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term']␣
 ↪for c in counted_classes_all.keys()], rotation='vertical')
else:
```

```
    plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
 ↪keys()), rotation='vertical')
plt.legend()
plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetFew.png'))
plt.show()
```



[77]:
```
patient_counts = sum(map(lambda x: len(x.files), [ptbxl_train, georgia_train,␣
 ↪cpsc_train, cpsc2_train]))


patient_counts/PATIENTS_TRAIN_TOTAL
```

[77]: 0.16538746381202452

[48]:
```
DESIRED_SIZE = class_total*0.1
print(DESIRED_SIZE)
sort_idx = np.argsort(class_counts)
sorted_class_counts = class_counts[sort_idx]
total = 0
last_size = 0
i = 0
extracted_counts = np.zeros(len(sorted_class_counts)).astype(int)
if DESIRED_SIZE/sorted_class_counts[0] > len(sorted_class_counts):
    while total + (len(class_counts)-i)*last_size < DESIRED_SIZE:
        last_size = sorted_class_counts[i]
        total += last_size
        extracted_counts[sort_idx[i]] = last_size
        i += 1
rest_needed = int((DESIRED_SIZE-sum(extracted_counts))/(len(class_counts)-i))
extracted_counts[np.where(extracted_counts==0)[0]] = rest_needed
print(extracted_counts)
```

```
5511.0
[ 97  55  97  97  97  97  97  97  97  97  97  97  97  97  97  97  97  97
  39  20  40  52  97  97  18 101  22  31  43  72  97  29  97  97  94  97
  97  97  97  97  42  97  97  97  97  86  97  97  97  97  80  28  97  97
  97  97  97  97  74  14  97  97  27  97  97  50  97]
```

[ ]:
```
from collections import defaultdict
class_counts_dset = np.empty((4, len(class_counts))).astype(int)
counted_classes_dset = np.empty((4, len(class_counts)))
for i, dset in enumerate([ptbxl_train, georgia_train, cpsc_train, cpsc2_train]):
    c_count, counted_c = count_merged_classes([dset])
    class_counts_dset[i] = np.array(c_count)
class_counts_dset
dsets = [ptbxl_train, georgia_train, cpsc_train, cpsc2_train]
buckets = np.zeros((4, len(class_counts))).astype(int)
sort_idx = np.argsort(class_counts)

for ix in sort_idx:
    #find this label in files.
    dset_order = np.argsort(class_counts_dset[:, ix])
    n_per_dataset = 0
    have = 0
    for i, dset_i in enumerate(dset_order):
        n_per_dataset = int(np.ceil((extracted_counts[ix] - have)/
↪(len(dsets)-i)))
        cc_in_dset = class_counts_dset[dset_i, ix]
        take_n_files = min(cc_in_dset, n_per_dataset)
        buckets[dset_i, ix] = take_n_files
        have += take_n_files
```

```
sum(buckets), buckets
```

[50]:
```
np.sum(buckets, axis=0) - extracted_counts
```

[50]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0])
```

[130]:
```
np.sum(class_counts_dset, axis=0)
```

[130]:
```
array([  208,    62,  1043,  1805,  3959,   817,  2631,  1331,  2412,
         216,   722,   725,  1019,   319,  3256,   242,  2352,   250,
          43,    28,    44,    57,   145,   388,    18,   110,    31,
          35,    50,    74,  1669,    34,  1193,  4234,   115,   474,
         719,  1626,   226,   196,    46, 14556,  1671,   118,   867,
          97,  2464,  1368,  1272,   141,    84,    30,   288,   288,
         542,  1684,   771,   148,    79,    18,   935,   694,    29,
        1105,   479,    58,   163])
```

[ ]:
```
def min_splits(DESIRED_SIZE):
    print(DESIRED_SIZE)
    sort_idx = np.argsort(class_counts)
    sorted_class_counts = class_counts[sort_idx]
    total = 0
    last_size = 0
    i = 0
    extracted_counts = []
    if DESIRED_SIZE/sorted_class_counts[0] > len(sorted_class_counts):
        while total + (len(class_counts)-i)*last_size < DESIRED_SIZE:
            last_size = sorted_class_counts[i]
            total += last_size
            extracted_counts.append(last_size)
            i += 1
    rest_needed = int((DESIRED_SIZE-sum(extracted_counts))/
↪(len(class_counts)-len(extracted_counts)))
    extracted_counts = extracted_counts + [rest_needed] * (len(class_counts) -␣
↪len(extracted_counts))
    extracted_counts
    assert train_fraction+val_fraction+test_fraction <= 1
    class_buckets = [[] for x in range(len(self.classes))] #Creates classes␣
↪buckets
    for i, f in enumerate(self.files):
        label = self._read_header_labels(f)
        label_idx = np.argwhere(label).flatten()
        for l in label_idx: #can return more than one (multi-label)
```

```
            class_buckets[l].append(f) #Put this file into class l bucket
    train_files, val_files, test_files = [], [], []
    sorted_idx = list(sorted(range(len(class_buckets)), key=lambda x:␣
↪len(class_buckets[x]))) #make index sorted by class count low->high
    print(sorted_idx)
    while len(sorted_idx) > 0:
        idx = sorted_idx[0]
        shuffle(class_buckets[idx])
        b = class_buckets[idx]
        c_N = len(b)
        train_slice = slice(0, int(train_fraction * c_N))
        val_slice = slice(train_slice.stop, train_slice.stop + int(val_fraction␣
↪* c_N))
        test_slice = slice(val_slice.stop, val_slice.stop + int(test_fraction *␣
↪c_N))
        train_files += b[train_slice]
        val_files += b[val_slice]
        test_files += b[test_slice]
        used_set = set(b[train_slice]+b[val_slice]+b[test_slice])
        for j in sorted_idx[1:]:
            class_buckets[j] = [x for x in class_buckets[j] if x not in␣
↪used_set] #REMOVE THIS FILE FROM ALL OTHER BUCKETS
        sorted_idx = list(sorted(sorted_idx[1:], key=lambda x:␣
↪len(class_buckets[x]))) #sort again (removal may change order)

    train_files = list(set(train_files))
    val_files = list(set(val_files))
    test_files = list(set(test_files))
    if save:
        p = save_path_overwrite or self.BASE_DIR
        fname = filename_overwrite or 'train-test-splits.txt'
        save_train_test_split(os.path.join(p, fname), train_files, val_files,␣
↪test_files)
    return train_files, val_files, test_files
```

[45]:
```
from torch.utils.data import DataLoader, ChainDataset
whole_dataset = ChainDataset([ptbxl_train, ptbxl_val, t1, georgia_train,␣
 ↪georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4])
dl = DataLoader(whole_dataset, batch_size=1, collate_fn=collate_fn)
occ = np.zeros((len(counted_classes_all.keys()), len(counted_classes_all.
 ↪keys())))
patient_count = 0
all_labels = []
for _, labels in dl:
    labels = labels[0]
    patient_count += 1
```

```
    all_labels.append(labels)
all_labels = np.concatenate(all_labels).reshape((-1, 67))
```

[53]:
```
pearson = np.zeros((67, 67))
for cix1 in range(67):
    for cix2 in range(67):
        pearson[cix1, cix2] = stats.pearsonr(all_labels[:, cix1], all_labels[:,␣
↪cix2])[0]
        #pearson[cix1, cix2] = np.cov(all_labels[:, cix1], all_labels[:,␣
↪cix2])[0, 1]/(np.std(all_labels[:, cix1])*np.std(all_labels[:, cix2]))
```

[54]:
```
pd.DataFrame(pearson, columns=ptbxl_train.classes, index=ptbxl_train.classes)
```

[54]:

	10370003	11157007	111975006	164861001	164865005	164867002	\
10370003	1.000000	-0.003865	-0.016099	-0.018971	-0.028853	-0.012402	
11157007	-0.003865	1.000000	-0.008776	0.014348	0.016910	-0.004545	
111975006	-0.016099	-0.008776	1.000000	-0.042553	-0.066670	-0.032093	
164861001	-0.018971	0.014348	-0.042553	1.000000	0.135552	-0.001239	
164865005	-0.028853	0.016910	-0.066670	0.135552	1.000000	-0.040258	
…	…	…	…	…	…	…	
713422000	-0.002589	-0.001412	0.006624	-0.007796	-0.012057	0.018386	
713426002	-0.013762	0.001683	0.012977	-0.009327	0.016022	-0.002278	
713427006	-0.010766	0.014627	-0.023430	-0.017435	0.110656	0.050997	
74390002	-0.003709	-0.002022	-0.008423	-0.011167	-0.007786	-0.007395	
89792004	-0.002358	-0.003385	0.029547	-0.010582	-0.008064	0.011294	

	164873001	164884008	164889003	164890007	…	63593006	\
10370003	-0.024199	-0.015488	-0.020909	-0.007149	…	-0.005916	
11157007	0.005723	0.092348	0.003332	-0.003897	…	0.018902	
111975006	0.043470	-0.032087	-0.010009	0.023264	…	-0.009772	
164861001	0.349488	0.046842	0.088639	-0.014464	…	0.016228	
164865005	0.046907	0.074197	0.037281	-0.025042	…	0.037070	
…	…	…	…	…	…	…	
713422000	-0.001434	-0.002937	0.016174	-0.002611	…	-0.002161	
713426002	-0.021665	-0.000510	0.002922	-0.000726	…	0.003729	
713427006	-0.031832	0.008657	0.033424	0.002507	…	0.028609	
74390002	-0.011810	-0.009534	-0.009172	-0.003741	…	0.035321	
89792004	-0.002503	-0.009729	-0.011330	-0.002441	…	0.008635	

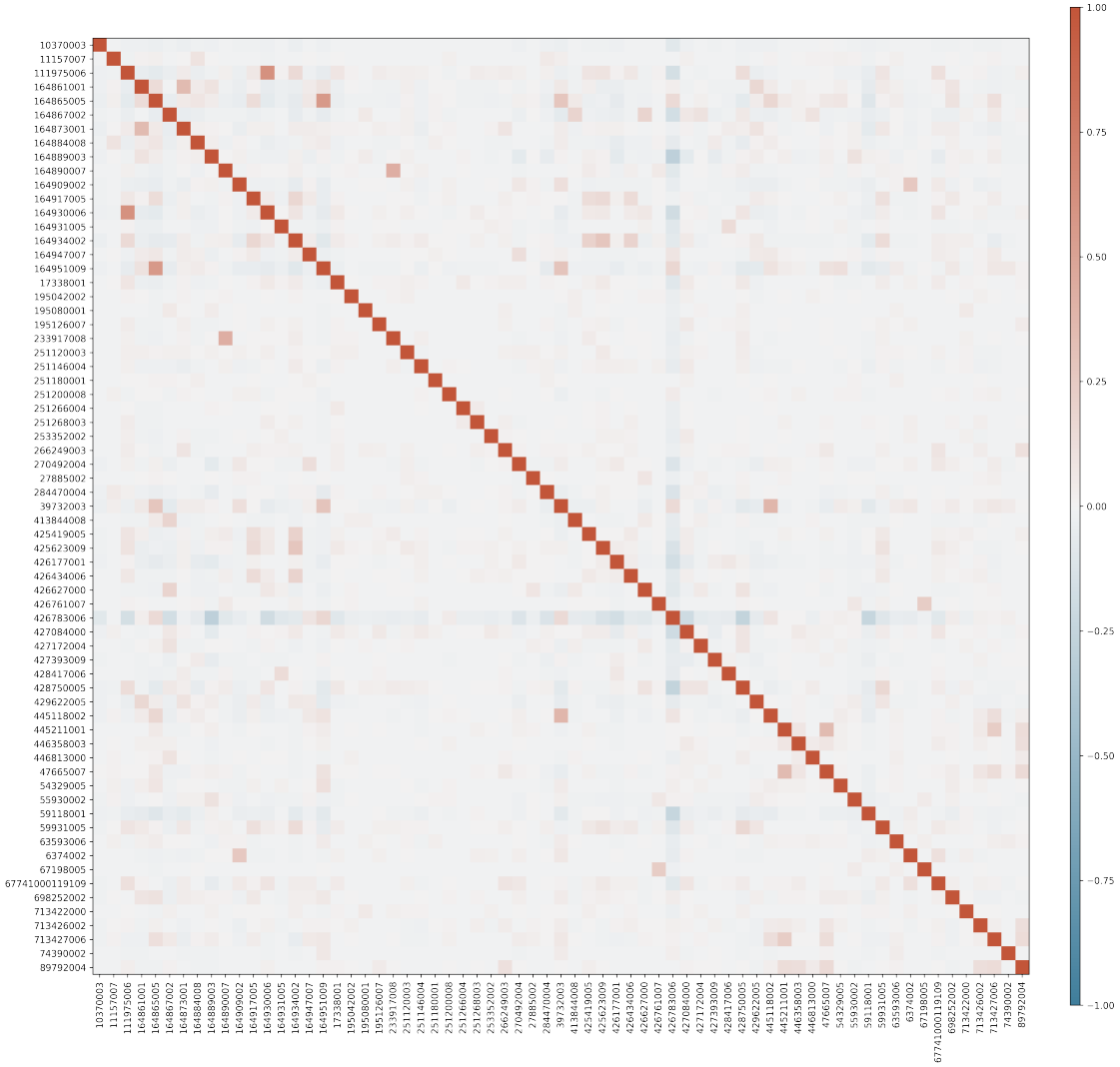
	6374002	67198005	67741000119109	698252002	713422000	\
10370003	-0.004394	-0.002005	-0.014957	-0.011190	-0.002589	
11157007	-0.002396	-0.001093	-0.005159	0.006506	-0.001412	
111975006	-0.007519	-0.004554	0.104357	-0.000048	0.006624	
164861001	-0.013230	-0.006037	0.010833	0.078584	-0.007796	
164865005	-0.020462	-0.009338	-0.004251	0.100797	-0.012057	
…	…	…	…	…	…	
713422000	0.027937	-0.000732	-0.000998	0.005386	1.000000	

```
713426002 -0.010364  0.000465        0.018576  -0.019361   0.001942
713427006 -0.006674 -0.003045        0.009996  -0.018588   0.002179
74390002  -0.002299 -0.001049       -0.007826  -0.006831  -0.001355
89792004  -0.003849 -0.001756        0.003730  -0.009309  -0.002268

           713426002  713427006  74390002  89792004
10370003   -0.013762  -0.010766 -0.003709 -0.002358
11157007    0.001683   0.014627 -0.002022 -0.003385
111975006   0.012977  -0.023430 -0.008423  0.029547
164861001  -0.009327  -0.017435 -0.011167 -0.010582
164865005   0.016022   0.110656 -0.007786 -0.008064
…                …          …        …         …
713422000   0.001942   0.002179 -0.001355 -0.002268
713426002   1.000000   0.026698 -0.000312  0.120213
713427006   0.026698   1.000000 -0.005633  0.133919
74390002   -0.000312  -0.005633  1.000000 -0.003249
89792004    0.120213   0.133919 -0.003249  1.000000

[67 rows x 67 columns]
```

[78]:
```
import matplotlib as mpl
plt.figure(figsize=(20, 20), dpi=300)
#plt.title('Classes occuring together')
plt.tight_layout()
plt.imshow(pearson, cmap=cmap, norm=mpl.colors.Normalize(vmin=-1, vmax=1))
if use_names:
    plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term']␣
 ↪for c in counted_classes_all.keys()], rotation='vertical')
    plt.yticks(range(len(counted_classes_all.keys())), [code_names[c]['Term']␣
 ↪for c in counted_classes_all.keys()])
else:
    plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
 ↪keys()), rotation='vertical')
    plt.yticks(range(len(counted_classes_all.keys())), list(counted_classes_all.
 ↪keys()))
plt.colorbar(fraction=0.046, pad=0.04, aspect=100)
plt.savefig(os.path.join(save_path, 'bilder/ClassOccurencesCorrelation.png'),␣
 ↪bbox_inches='tight')
plt.show()
```


10370003
11157007
111975006
164861001
164865005
164867002
164873001
164884008
164889003
164890007
164909002
164917005
164930006
164931005
164934002
164947007
164951009
17338001
195042002
195080001
195126007
233917008
251120003
251146004
251180001
251200008
251266004
251268003
253352002
266249003
270492004
27885002
284470004
39732003
413844008
425419005
425623009
426177001
426434006
426627000
426761007
426783006
427084000
427172004
427393009
428417006
428750005
429622005
445118002
445211001
446358003
446813000
47665007
54329005
55930002
59118001
59931005
63593006
6374002
67198005
67741000119109
698252002
713422000
713426002
713427006
74390002
89792004
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00