Nina Stawski's (group 90) final project report [DRAFT]

Illinois ID: ninas2

GitHub repo link

In [1]:
```
%%script false --no-raise-error
# disabling the cell since I am not using it, but keeping in the notebook in 

from google.colab import drive
drive.mount('/content/drive')
```

Introduction

Background of the problem

Type of problem

This is a data preparation and processing problem. The authors of the article are testing a common belief that adding more data improves the resulting model performance. Their main hypothesis, which they subsequently prove, is that incorporating more data does not necessary improve the model performance. It can introduce spurious correlations, and hurt the resulting model performance rather than helping it.

What is the importance/meaning of solving the problem

The paper is challenging a common belief, meaning a lot of researchers are likely trying to incorporate as much data as they can expecting it would improve the performance of their models. The outcome of this research would provide guidance on the possible pitfalls and the cases where you wouldn't want to add external data - so it could set a new standard of processing and incorporating data for everyone in the field.

The difficulty of the problem

The problem is non-obvious and the paper is challenging the common belief held in the industry. The authors are putting a lot of state-of-the-art approaches to the test, and attempt to quantify the results as well as provide new standards and explanations. This is

extremely hard to do so I believe the problem is difficult.

The state of the art methods and effectiveness

The "industry standard" way of improving model performance is adding more data from additional datasets, which the authors of this article prove to not be effective, and even being harmful in many cases.

One of the main issues causing the model performance decrease when adding more data from other sources is spurious correlations, which in case of x-rays could be coming even from the scanner artifacts, or other hospital-specific data. One of the state-of-the-art ways to mitigate this is balancing a dataset to reduce the influence of hospital-specific factors. While balancing definitely improved the situation, the resulting model performance was still in many cases worse than with a single-hospital dataset.

Paper explanation

What did the paper propose

The paper used four most-used chest x-ray datasets - MIMIC-CXR-JPG, CheXpert, PadChest, ChestXray8 - to disprove a popular belief that adding more data always would improve the performance of your model. They postulate that, for the specific x-ray data, even the scanners themselves, the way hospitals produce data, or send specific patients to specific places to do their scan, can introduce spurious correlations which, in many cases, significantly affect the worst group performance.

What is the innovations of the method

Existing research (for example, John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. PLoS medicine, 15(11): e1002683, 2018.) proves that adding a second dataset improves the average per-group accuracy. In contrast, the paper I am reproducing focuses on the worst per-group accuracy.

How well the proposed method work (in its own metrics)

According to the article authors, their method works really well and proves that in nearly 50% of cases adding a second dataset, and even balancing it to reduce spurious

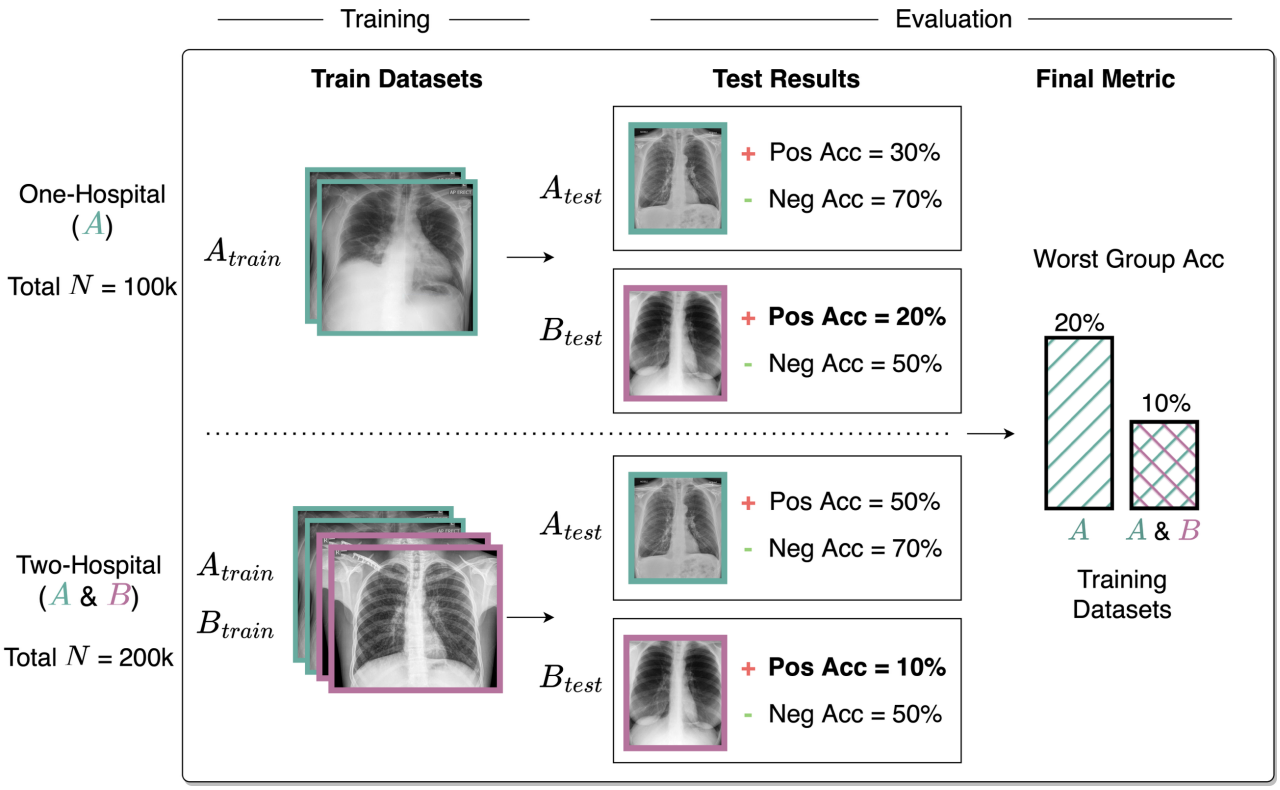
correllations doesn't get the model to perform better than without that additional dataset. The models pick up on hospital-specific features even if those features weren't explicitly defined in the original data. They postulate that every CNN model, regardless of training disease or datasets, learns embeddings that can distinguish any of the hospital sources with near-perfect accuracy, even if the embeddings were trained via one or two hospitals' data.

What is the contribution to the reasearch regime (referring the Background above, how important the paper is to the problem).

The article cautions against blindly adding more datasets, and provides a number of approaches you can take if you still decide to do so. The conclusion is adding more data shouldn't be done blindly. The authors of the article definitely discourage the researchers from the most common approach of throwing data at the problem to improve model performance.

Scope of Reproducibility:

List hypotheses from the paper you will test and the corresponding experiments you will run.



Hypothesis 1

In 43% of training dataset/disease tasks, adding data from an external source hurts worst-group performance.

Hypothesis 2

Balancing the dataset to reduce spurious correlations is often beneficial, but in the scenarios where adding an additional data source hurts generalization performance, it does not always improve generalization; in some cases, training on a balanced dataset achieves lower worst-group accuracy than training on datasets from one or two hospitals.

In [2]:

```
%%script false --no-raise-error
# disabling the cell since I am not using it, but keeping in the notebook in 

# no code is required for this section
'''
if you want to use an image outside this notebook for explanaition,
you can upload it to your google drive and show it with OpenCV or matplotlib
'''
# mount this notebook to your google drive
drive.mount('/content/gdrive')

# define dirs to workspace and data
img_dir = '/content/gdrive/My Drive/Colab Notebooks/<path-to-your-image>'

import cv2
img = cv2.imread(img_dir)
cv2.imshow("Title", img)
```

Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

In [3]:
```
# import  packages you need
import numpy as np
import pandas as pd
from pathlib import Path
import os
from os.path import exists
import sys
import matplotlib.pyplot as plt
from PIL import Image
import json
import random
from IPython.display import display
# from google.colab import drive

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset
from torchvision import datasets, models, transforms

import Data_Constants as Constants

#making sure all referenced files are reloaded
import importlib
importlib.reload(Constants)
```

Out[3]:
```
<module 'Data_Constants' from '/Users/noemi/dlh-final-project/Data_Constants.py'>
```

Data

The study is using four datasets: MIMIC-CXR-JPG, CheXpert, PadChest, ChestXray8

The datasets are being filtered to include only frontal (PA/AP) images. Instances are labeled with one or more pathologies. Each dataset has a different set of diseases but they are preprocessed using code derived from ClinicalDG2 (Zhang et al., 2021) to extract the eight common labels and homogenize the datasets. Additionally, authors of the article created the Any label which indicates a positive label for any of the seven common disease labels, resulting in nine different binary labels. All experiments use the labels in a binary manner; a pathology is chosen as the target label, with an instance labeled 1 if the pathology of interest is present and 0 otherwise.

The autors apply an 80%/10%/10% subject-wise train/val/test split, with the same split used across seeds.

MIMIC-CXR

1. Obtain access to the MIMIC-CXR-JPG Database Database on PhysioNet and download the dataset. The best option is downloading from the GCP bucket:

```
gcloud auth login
mkdir MIMIC-CXR-JPG
gsutil -m rsync -d -r gs://mimic-cxr-jpg-2.0.0.physionet.org MIMIC-CXR-JPG
```

1. In order to obtain gender information for each patient, you will need to obtain access to MIMIC-IV. Download `core/patients.csv.gz` and place the file in the `MIMIC-CXR-JPG` directory.

CheXpert

1. Sign up with your email address here.
2. Download either the original or the downsampled dataset (we recommend the downsampled version - `CheXpert-v1.0-small.zip`) and extract it.

ChestX-ray8

1. Download the `images` folder and `Data_Entry_2017_v2020.csv` from the NIH website.
2. Unzip all of the files in the `images` folder.

PadChest

1. The paper uses a resized version of PadChest, which can be downloaded here.
2. Unzip `images-224.tar` .

    - Statistics: include basic descriptive statistics of the dataset like size, cross validation split, label distribution, etc.
    - Data process: how do you munipulate the data, e.g., change the class labels, split the dataset to train/valid/test, refining the dataset.
    - Illustration: printing results, plotting figures for illustration.
    - You can upload your raw dataset to Google Drive and mount this Colab to the same directory. If your raw dataset is too large, you can upload the processed dataset and have a code to load the processed dataset.

Data Processing

The original pre-processing for the article was done using the scripts outside of the Jupyter Notebook. Some of them didnt' work for me, and the installation process didn't succeed despite multiple attempts either. Instead, I have adapted some of the original scripts to run in the notebook (with some modifications so they actually work with my data), using the external "Constants.py" file that points to the location of the datasets.

1. In `./Data_Constants.py`, update `image_paths` to point to each of the four directories that you downloaded.
2. Run the next two cells to pre-process the data

Validating

I am using the validation and pre-processing code provided by the authors of the article, with some modifications to make it run as expected.

In [4]:
```
#making sure constants are up to date if they were changed
importlib.reload(Constants)

def validate_mimic():
    img_dir = Path(Constants.image_paths['MIMIC'])
    meta_dir = Path(Constants.meta_paths['MIMIC'])

    assert (meta_dir/'mimic-cxr-2.0.0-metadata.csv').is_file()
    assert (meta_dir/'mimic-cxr-2.0.0-negbio.csv').is_file()
    assert (meta_dir/'patients.csv').is_file()
    # modified the file that's being checked since I don't have the full MIMI
    # in the original script, the file in p19 was being checked.
    assert (img_dir/'p10/p10000032/s50414267/02aa804e-bde0afdd-112c0b34-7bc16

def validate_cxp():
    img_dir = Path(Constants.image_paths['CXP'])
    if (img_dir/'CheXpert-v1.0').is_dir():
        cxp_subfolder = 'CheXpert-v1.0'
    else:
        cxp_subfolder = 'CheXpert-v1.0-small'
    assert (img_dir/cxp_subfolder/'train.csv').is_file()
    assert (img_dir/cxp_subfolder/'train/patient48822/study1/view1_frontal.jp
    assert (img_dir/cxp_subfolder/'valid/patient64636/study1/view1_frontal.jp

def validate_pad():
    img_dir = Path(Constants.image_paths['PAD'])
    meta_dir = Path(Constants.meta_paths['PAD'])
    assert (meta_dir/'PADCHEST_chest_x_ray_images_labels_160K_01.02.19.csv').
    assert (img_dir/'185566798805711692534207714722577525271_qb3lyn.png').is_

def validate_nih():
    img_dir = Path(Constants.image_paths['NIH'])
    meta_dir = Path(Constants.meta_paths['NIH'])
    assert (meta_dir/'Data_Entry_2017.csv').is_file()
    assert (img_dir/'images/00002072_003.png').is_file()

def validate_splits():
    for dataset in Constants.df_paths:
        for split in Constants.df_paths[dataset]:
            assert Path(Constants.df_paths[dataset][split]).is_file()


def validate_all():
    validate_mimic()
    validate_cxp()
    validate_nih()
    validate_pad()
```

Data pre-processing setup

In [5]:

In [5]:

```
# making sure constants are up to date if they were changed after running thi
importlib.reload(Constants)

def preprocess_mimic():
    img_dir = Path(Constants.image_paths['MIMIC'])
    meta_dir = Path(Constants.meta_paths['MIMIC'])
    out_folder = meta_dir/'clinicaldg'
    out_folder.mkdir(parents = True, exist_ok = True)

    patients = pd.read_csv(meta_dir/'patients.csv')
    labels = pd.read_csv(meta_dir/'mimic-cxr-2.0.0-negbio.csv')
    meta = pd.read_csv(meta_dir/'mimic-cxr-2.0.0-metadata.csv')

    df = meta.merge(patients, on = 'subject_id').merge(labels, on = ['subject
    df['age_decile'] = pd.cut(df['anchor_age'], bins = list(range(0, 100, 10)
    df['frontal'] = df.ViewPosition.isin(['AP', 'PA'])

    df['path'] = df.apply(lambda x: os.path.join(f'p{str(x["subject_id"])[:2]
    df.to_csv(out_folder/"preprocessed.csv", index=False)

def preprocess_pad():
    # I have modified this function from the original one, because I was gett
    img_dir = Path(Constants.image_paths['PAD'])
    meta_dir = Path(Constants.meta_paths['PAD'])
    out_folder = meta_dir/'clinicaldg'
    out_folder.mkdir(parents=True, exist_ok=True)

    dtype_spec = {
        'ImageID': str,
        'StudyID': str,
        'PatientID': str,
        'PatientBirth': str, # converting this to the integer later to avoid
        'PatientSex_DICOM': str,
        'ViewPosition_DICOM': str,
        'Projection': str,
        'Labels': str,
        'WindowCenter_DICOM': str,
        'WindowWidth_DICOM': str
    }

    df = pd.read_csv(meta_dir/'PADCHEST_chest_x_ray_images_labels_160K_01.02.
    df = df[['ImageID', 'StudyID', 'PatientID', 'PatientBirth', 'PatientSex_D
    df = df[~df["Labels"].isnull()]
    df = df[df["ImageID"].apply(lambda x: os.path.exists(os.path.join(img_dir
    df = df[df.Projection.isin(['PA', 'L', 'AP_horizontal', 'AP'])]

    df['frontal'] = ~(df['Projection'] == 'L')
    df = df[~df['Labels'].apply(lambda x: 'exclude' in x or 'unchanged' in x)

    mapping = dict()
    mapping['Effusion'] = ['hydropneumothorax', 'empyema', 'hemothorax']
    mapping["Consolidation"] = ["air bronchogram"]
    mapping['No Finding'] = ['normal']
```

```
    for pathology in Constants.take_labels:
        mask = df["Labels"].str.contains(pathology.lower())
        if pathology in mapping:
            for syn in mapping[pathology]:
                mask |= df["Labels"].str.contains(syn.lower())
        df[pathology] = mask.astype(int)

    df['PatientBirth'] = df['PatientBirth'].dropna().astype(float).astype(int
    df['Age'] = 2017 - df['PatientBirth']
    df.reset_index(drop=True).to_csv(out_folder/"preprocessed.csv", index=Fal


def preprocess_cxp():
    img_dir = Path(Constants.image_paths['CXP'])
    out_folder = img_dir/'clinicaldg'
    if (img_dir/'CheXpert-v1.0'/'train.csv').is_file():
        df = pd.concat([pd.read_csv(img_dir/'CheXpert-v1.0'/'train.csv'),
                        pd.read_csv(img_dir/'CheXpert-v1.0'/'valid.csv')],
                        ignore_index = True)
    elif (img_dir/'CheXpert-v1.0-small'/'train.csv').is_file():
        df = pd.concat([pd.read_csv(img_dir/'CheXpert-v1.0-small'/'train.csv'
                        pd.read_csv(img_dir/'CheXpert-v1.0-small'/'valid.csv'
                        ignore_index = True)
    elif (img_dir/'train.csv').is_file():
        raise ValueError('Please set Constants.image_paths["CXP"] to be the P
                ' directory and rerun this script.')
    else:
        raise ValueError("CheXpert files not found!")

    out_folder.mkdir(parents = True, exist_ok = True)

    df['subject_id'] = df['Path'].apply(lambda x: int(Path(x).parent.parent.n
    df['Path'] = df['Path'].apply(lambda x: str(x).replace("CheXpert-v1.0/",
    df.reset_index(drop = True).to_csv(out_folder/"preprocessed.csv", index=F

def preprocess_nih():
    img_dir = Path(Constants.image_paths['NIH'])
    meta_dir = Path(Constants.meta_paths['NIH'])
    out_folder = meta_dir/'clinicaldg'
    out_folder.mkdir(parents = True, exist_ok = True)
    df = pd.read_csv(meta_dir/"Data_Entry_2017.csv")
    df['labels'] = df['Finding Labels'].apply(lambda x: x.split('|'))

    for label in Constants.take_labels:
        df[label] = df['labels'].apply(lambda x: label in x)
    df.reset_index(drop = True).to_csv(out_folder/"preprocessed.csv", index=F


if __name__ == '__main__':
    print("Validating paths...")
    validate_all()
    print("Preprocessing MIMIC-CXR...")
```

```
    preprocess_mimic()
    print("Preprocessing CheXpert...")
    preprocess_cxp()
    print("Preprocessing ChestX-ray8...")
    preprocess_nih()
    print("Preprocessing PadChest... This might take a few minutes...")
    preprocess_pad()
    print("Done.")
```

```
Validating paths...
Preprocessing MIMIC-CXR...
Preprocessing CheXpert...
Preprocessing ChestX-ray8...
Preprocessing PadChest... This might take a few minutes...
Done.
```

Next, we need to resize and process the data.

I am using the code provided by the authors of the article to do this.

In [6]:

```
def process_MIMIC(split, only_frontal):
    copy_subjectid = split['subject_id']
    split = split.drop(columns = ['subject_id']).replace(
            [[None], -1, "[False]", "[True]", "[ True]", 'UNABLE TO OBTAIN',
             'DIVORCED', 'SEPARATED', '0-10', '10-20', '20-30', '30-40', '40-
             '>=90'],
            [0, 0, 0, 1, 1, 0, 0, 'MARRIED/LIFE PARTNER', 'MARRIED/LIFE PARTN
             'DIVORCED/SEPARATED', '0-20', '0-20', '20-40', '20-40', '40-60',

    split['subject_id'] = copy_subjectid.astype(str)
    split['study_id'] = split['study_id'].astype(str)
    split['Age'] = split["age_decile"]
    split['Sex'] = split["gender"]
    split = split.rename(
        columns = {
            'Pleural Effusion':'Effusion',
        })
    split['path'] = split['path'].astype(str).apply(lambda x: os.path.join(Co
    if only_frontal:
        split = split[split.frontal]

    split['env'] = 'MIMIC'
    split.loc[split.Age == 0, 'Age'] = '0-20'

    return split[['subject_id','path','Sex',"Age", 'env', 'frontal', 'study_i

def process_NIH(split, only_frontal = True):
    split['Patient Age'] = np.where(split['Patient Age'].between(0,19), 19, s
    split['Patient Age'] = np.where(split['Patient Age'].between(20,39), 39,
    split['Patient Age'] = np.where(split['Patient Age'].between(40,59), 59,
    split['Patient Age'] = np.where(split['Patient Age'].between(60,79), 79,
    split['Patient Age'] = np.where(split['Patient Age']>=80, 81, split['Pati
```

```
    copy_subjectid = split['Patient ID']

    split = split.drop(columns = ['Patient ID']).replace([[None], -1, "[False
                            [0, 0, 0, 1, 1, "0-20", "20-40", "40-60", "60-80"

    split['subject_id'] = copy_subjectid.astype(str)
    split['Sex'] = split['Patient Gender']
    split['Age'] = split['Patient Age']
    split = split.drop(columns=["Patient Gender", 'Patient Age'])
    split['path'] = split['Image Index'].astype(str).apply(lambda x: os.path.
    split['env'] = 'NIH'
    split['frontal'] = True
    split['study_id'] = split['subject_id'].astype(str)
    return split[['subject_id','path','Sex',"Age", 'env', 'frontal','study_id


def process_CXP(split, only_frontal):
    split['Age'] = np.where(split['Age'].between(0,19), 19, split['Age'])
    split['Age'] = np.where(split['Age'].between(20,39), 39, split['Age'])
    split['Age'] = np.where(split['Age'].between(40,59), 59, split['Age'])
    split['Age'] = np.where(split['Age'].between(60,79), 79, split['Age'])
    split['Age'] = np.where(split['Age']>=80, 81, split['Age'])

    copy_subjectid = split['subject_id']
    split = split.drop(columns = ['subject_id']).replace([[None], -1, "[False
                            [0, 0, 0, 1, 1, "0-20", "20-40", "40-60", "60-80"

    split['subject_id'] = copy_subjectid.astype(str)
    split['Sex'] = np.where(split['Sex']=='Female', 'F', split['Sex'])
    split['Sex'] = np.where(split['Sex']=='Male', 'M', split['Sex'])
    split = split.rename(
        columns = {
            'Pleural Effusion':'Effusion',
            'Lung Opacity': 'Airspace Opacity'
        })
    split['path'] = split['Path'].astype(str).apply(lambda x: os.path.join(Co
    split['frontal'] = (split['Frontal/Lateral'] == 'Frontal')
    if only_frontal:
        split = split[split['frontal']]
    split['env'] = 'CXP'
    split['study_id'] = split['path'].apply(lambda x: x[x.index('patient'):x.
    return split[['subject_id','path','Sex',"Age", 'env', 'frontal','study_id


def process_PAD(split, only_frontal):
    split['Age'] = np.where(split['Age'].between(0,19), 19, split['Age'])
    split['Age'] = np.where(split['Age'].between(20,39), 39, split['Age'])
    split['Age'] = np.where(split['Age'].between(40,59), 59, split['Age'])
    split['Age'] = np.where(split['Age'].between(60,79), 79, split['Age'])
    split['Age'] = np.where(split['Age']>=80, 81, split['Age'])

    split = split.replace([[None], -1, "[False]", "[True]", "[ True]", 19, 39
```

```
                                [0, 0, 0, 1, 1, "0-20", "20-40", "40-60", "60-80"

    split.loc[split['Age'] == 0.0, 'Age'] = '0-20'
    split.loc[split['Age'].isnull(), 'Age'] = '0-20'
    split = split.rename(columns = {
        'PatientID': 'subject_id',
        'StudyID': 'study_id',
        'PatientSex_DICOM' :'Sex'
    })

    split.loc[~split['Sex'].isin(['M', 'F', 'O']), 'Sex'] = 'O'
    split['path'] =  split['ImageID'].astype(str).apply(lambda x: os.path.joi
    if only_frontal:
        split = split[split['frontal']]
    split['env'] = 'PAD'
    return split[['subject_id','path','Sex',"Age", 'env', 'frontal','study_id


def split(df, split_portions = (0.8, 0.9), seed=0):
    # We don't want the data splits to be affected by seed
    # So lets temporarily set the seed to a static value...
    rand_state = np.random.get_state()
    np.random.seed(seed)

    # Split our data (irrespective of the random seed provided in train.py)
    subject_df = pd.DataFrame({'subject_id': np.sort(df['subject_id'].unique(
    subject_df['random_number'] = np.random.uniform(size=len(subject_df))

    train_id = subject_df[subject_df['random_number'] <= split_portions[0]].d
    valid_id = subject_df[(subject_df['random_number'] > split_portions[0]) &
    test_id = subject_df[subject_df['random_number'] > split_portions[1]].dro

    train_df = df[df.subject_id.isin(train_id.subject_id)]
    valid_df = df[df.subject_id.isin(valid_id.subject_id)]
    test_df = df[df.subject_id.isin(test_id.subject_id)]

    # ...then return the random state back to what it was
    np.random.set_state(rand_state)

    return train_df, valid_df, test_df

def get_process_func(env):
    if env == 'MIMIC':
        return process_MIMIC
    elif env == 'NIH':
        return process_NIH
    elif env == 'CXP':
        return process_CXP
    elif env == 'PAD':
        return process_PAD
    else:
        raise NotImplementedError
```

In [7]:
```
# show data paths from constants
Constants.df_paths

def img_exists(path):
    return exists(path)

def is_diseased(row):
    # diseases = Constants.take_labels[1:]
    return int((row[Constants.take_labels[1:]]).sum() > 0)
```

The following cell is pre-processing the data and will take a long time to run

The cell below needs to run once, after that everything is saved into the CSV file and can be loaded from there. this block of code needs to re-run only if the data changed.

In [8]:
```
# loads data with random splits
print('This might take a while.')

for data_env in Constants.df_paths:
    print('Processing:', data_env)
    func = get_process_func(data_env)
    print('Got processing function, filtering by only frontal...')
    df_env = func(pd.read_csv(Constants.df_paths[data_env]), only_frontal = T
    print('Filtering out the data without images...')
    df_env["img_exists"] = df_env["path"].apply(img_exists)
    print(df_env["img_exists"].value_counts())
    df_env = df_env[df_env["img_exists"]]

    df_env = df_env.fillna(0)

    print('Adding "All" column...')
    df_env["All"] = df_env.apply(is_diseased, axis=1)

    print('Saving results...')
    df_env.to_csv(f"{Constants.base_path}/processed/{data_env}.csv", index=Fa

    display(df_env)

print("Done.")
```

```
This might take a while.
Processing: MIMIC
Got processing function, filtering by only frontal...
Filtering out the data without images...
False    167664
True      63047
Name: img_exists, dtype: int64
Adding "All" column...
```

```
Saving results...
```

	subject_id	path	Sex	Age	env	frontal	study_id	No Finding	Ate
0	10000032	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	40-60	MIMIC	True	50414267	1.0	
2	10000032	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	40-60	MIMIC	True	53189527	1.0	
4	10000032	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	40-60	MIMIC	True	53911762	1.0	
5	10000032	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	40-60	MIMIC	True	53911762	1.0	
6	10000032	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	40-60	MIMIC	True	56699142	1.0	
...	...	...	...	...	...	...	...	...	
97453	12742782	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	40-60	MIMIC	True	54917116	1.0	
97456	12742898	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	20-40	MIMIC	True	53339588	0.0	
97461	12743572	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	60-80	MIMIC	True	51989892	1.0	
97463	12743572	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	60-80	MIMIC	True	52648347	1.0	
97467	12743572	/Volumes/Passport-2TB 2/data/mimic/physionet.o...	F	60-80	MIMIC	True	52929291	1.0	

63047 rows × 17 columns

```
Processing: CXP
Got processing function, filtering by only frontal...
Filtering out the data without images...
True    191229
Name: img_exists, dtype: int64
Adding "All" column...
Saving results...
```

	subject_id	path	Sex	Age	env	frontal	study_id
0	1	/Users/noemi/ood-generalization/data/CheXpert/...	F	60-80	CXP	True	patient00001/study1
1	2	/Users/noemi/ood-generalization/data/CheXpert/...	F	80-	CXP	True	patient00002/study2
2	2	/Users/noemi/ood-generalization/data/CheXpert/...	F	80-	CXP	True	patient00002/study1
4	3	/Users/noemi/ood-generalization/data/CheXpert/...	M	40-60	CXP	True	patient00003/study1
5	4	/Users/noemi/ood-generalization/data/CheXpert/...	F	20-40	CXP	True	patient00004/study1
...	...	...	...	...	...	...	...
223643	64736	/Users/noemi/ood-generalization/data/CheXpert/...	F	40-60	CXP	True	patient64736/study1
223644	64737	/Users/noemi/ood-generalization/data/CheXpert/...	M	60-80	CXP	True	patient64737/study1
223645	64738	/Users/noemi/ood-generalization/data/CheXpert/...	M	60-80	CXP	True	patient64738/study1
223646	64739	/Users/noemi/ood-generalization/data/CheXpert/...	F	40-60	CXP	True	patient64739/study1
223647	64740	/Users/noemi/ood-generalization/data/CheXpert/...	M	80-	CXP	True	patient64740/study1

191229 rows × 17 columns

```
Processing: NIH
Got processing function, filtering by only frontal...
Filtering out the data without images...
True    112120
Name: img_exists, dtype: int64
Adding "All" column...
Saving results...
```

	subject_id	path	Sex	Age	env	frontal	study_id	No Finding	At
0	1	/Users/noemi/ood-generalization/data/chestxray...	M	40-60	NIH	True	1	False	
1	1	/Users/noemi/ood-generalization/data/chestxray...	M	40-60	NIH	True	1	False	
2	1	/Users/noemi/ood-generalization/data/chestxray...	M	40-60	NIH	True	1	False	
3	2	/Users/noemi/ood-generalization/data/chestxray...	M	80-	NIH	True	2	True	
4	3	/Users/noemi/ood-generalization/data/chestxray...	F	60-80	NIH	True	3	False	
...	...	...	...	...	...	...	...	...	
112115	30801	/Users/noemi/ood-generalization/data/chestxray...	M	20-40	NIH	True	30801	False	
112116	30802	/Users/noemi/ood-generalization/data/chestxray...	M	20-40	NIH	True	30802	True	
112117	30803	/Users/noemi/ood-generalization/data/chestxray...	F	40-60	NIH	True	30803	True	
112118	30804	/Users/noemi/ood-generalization/data/chestxray...	F	20-40	NIH	True	30804	True	
112119	30805	/Users/noemi/ood-generalization/data/chestxray...	M	20-40	NIH	True	30805	True	

112120 rows × 17 columns

```
Processing: PAD
Got processing function, filtering by only frontal...
Filtering out the data without images...
True    99827
Name: img_exists, dtype: int64
Adding "All" column...
Saving results...
```

	subject_id	path	Sex	Age
0	83986048869429233163798823568146098 7	/Users/noemi/ood-generalization/data/PadChest/...	F	80-
2	31357275043099734750293265431938987 5966	/Users/noemi/ood-generalization/data/PadChest/...	M	80-
3	50783093527901818115346441867348318648	/Users/noemi/ood-generalization/data/PadChest/...	F	80-
6	93535126770783451980359712286922420997	/Users/noemi/ood-generalization/data/PadChest/...	M	60-80
7	93535126770783451980359712286922420997	/Users/noemi/ood-generalization/data/PadChest/...	M	60-80
...	...	...	...	...
144479	11293095241607406037137101459949649 3673	/Users/noemi/ood-generalization/data/PadChest/...	M	60-80
144480	28274372997142335870605673189051060 0934	/Users/noemi/ood-generalization/data/PadChest/...	F	60-80
144481	52648743308541843883453242716226652771	/Users/noemi/ood-generalization/data/PadChest/...	M	40-60
144482	22864613059315293381194899663415420 1216	/Users/noemi/ood-generalization/data/PadChest/...	F	60-80
144483	13742404723030361060208041028458882 5286	/Users/noemi/ood-generalization/data/PadChest/...	M	60-80

99827 rows × 17 columns

```
Done.
```

Resample data

In [9]:
```
dfs = {}
print('Processing the data, splitting to all, train, val and test...')
for env in Constants.df_paths:
    func = get_process_func(env)
    df_env = pd.read_csv(f"{Constants.base_path}/processed/{env}.csv")

    print('Source:', env)
    print('Data length:', len(df_env))

    train_df, valid_df, test_df = split(df_env)
    dfs[env] = {
        'all': df_env,
        'train': train_df,
        'val': valid_df,
        'test': test_df
    }
    print(f'{env}: done.')

print('All done.')
```

```
Processing the data, splitting to all, train, val and test...
Source: MIMIC
Data length: 63047
MIMIC: done.
Source: CXP
Data length: 191229
CXP: done.
Source: NIH
Data length: 112120
NIH: done.
Source: PAD
Data length: 99827
PAD: done.
All done.
```

Balancing the dataset

In [10]:
```
def get_prop(df, column="Pneumonia"):
    num_instances = len(df)
    num_diseased = df[df[column] == 1][column].count()
    return num_diseased / (num_instances - num_diseased)

def get_resample_class(orig_prop, new_prop, resample_method):
    if new_prop > orig_prop:
        if resample_method == "over":
            return 1
        else:
            return 0
    if new_prop < orig_prop:
        if resample_method == "under":
```

```
            return 1
        else:
            return 0

def calculate_num_resample(df, orig_prop, new_prop, resample_method):
    pass

from imblearn.over_sampling import RandomOverSampler

def balance_df_label(df, sampler, label_bal=0.05154780337262089, invert=False
    target = df["Pneumonia"] == 1
    rus = sampler(random_state=0, sampling_strategy=label_bal if not invert e
    res_df, _ = rus.fit_resample(df, target)

    print(f"Previous pneumonia prop: {get_pneumonia_prop(df)} with {len(df)}
    print(f"Resampled pneumonia prop: {get_pneumonia_prop(res_df)} with {len(

    return res_df

def balance_proportion(orig_df, new_df, resample_method="over", column="Pneum
    orig_df = orig_df.fillna(0.0)
    orig_prop = get_prop(orig_df, column)
    new_prop = get_prop(new_df, column)
    assert resample_method in ["over", "under"]
    resample_class = get_resample_class(orig_prop, new_prop, resample_method)
    print(f"Resampling '{column}' via '{resample_method}' on class {resample_

    # Estimate the number of items we'll need to resample
    df_diseased = orig_df[orig_df[column] == 1.0]
    df_normal = orig_df[orig_df[column] == 0.0]
    num_diseased = len(df_diseased)
    num_normal = len(df_normal)
    assert num_diseased + num_normal == len(orig_df)

    if resample_method == "over":
        if resample_class == 0:
            new_num_normal = int(num_diseased / new_prop)
            print(f"Resampling normal samples from {num_normal} to {new_num_n
            df_normal_rs = df_normal.sample(new_num_normal, replace=True, ran
            resampled_df = pd.concat([df_normal_rs, df_diseased])
        else:
            # Resample the pneumonia class
            # new_num_diseased = int(new_prop * num_normal)
            # print(f"Resampling diseased samples from {num_diseased} to {new
            # df_diseased_rs = df_diseased.sample(new_num_diseased, replace=T
            # resampled_df = pd.concat([df_normal, df_diseased_rs])
            target = df["Pneumonia"] == 1
            rus = RandomOverSampler(random_state=0, sampling_strategy=new_pro
            resampled_df, _ = rus.fit_resample(df, target)

    resampled_df.sort_index(inplace=True)
    print(f"New df proportion: {get_prop(resampled_df, column)}")
    return resampled_df
```

```
# balance_proportion(dfs["MIMIC"]["train"], dfs["MIMIC"]["test"])
```

In [11]:
```
dfs["CXP"]["train"]
```

Out[11]:

	subject_id	path	Sex	Age	env	frontal	study_id
0	1	/Users/noemi/ood-generalization/data/CheXpert/...	F	60-80	CXP	True	patient00001/study1
1	2	/Users/noemi/ood-generalization/data/CheXpert/...	F	80-	CXP	True	patient00002/study2
2	2	/Users/noemi/ood-generalization/data/CheXpert/...	F	80-	CXP	True	patient00002/study1
3	3	/Users/noemi/ood-generalization/data/CheXpert/...	M	40-60	CXP	True	patient00003/study1
4	4	/Users/noemi/ood-generalization/data/CheXpert/...	F	20-40	CXP	True	patient00004/study1
...	...	...	...	...	...	...	...
191222	64734	/Users/noemi/ood-generalization/data/CheXpert/...	M	40-60	CXP	True	patient64734/study1
191223	64735	/Users/noemi/ood-generalization/data/CheXpert/...	F	60-80	CXP	True	patient64735/study1
191225	64737	/Users/noemi/ood-generalization/data/CheXpert/...	M	60-80	CXP	True	patient64737/study1
191227	64739	/Users/noemi/ood-generalization/data/CheXpert/...	F	40-60	CXP	True	patient64739/study1
191228	64740	/Users/noemi/ood-generalization/data/CheXpert/...	M	80-	CXP	True	patient64740/study1

153411 rows × 17 columns

Metrics to evaluate my model

Similar to the original paper, for each base hospital I plan to choose one additional hospital to include in evaluation (for example, evaluate a model trained on MIMIC data using MIMIC and PAD data).

- analyse accuracies within each class for each hospital - the result is a group for the disease class from hospital A, the non-disease class from hospital A, the disease class from hospital B, and the non-disease class from hospital B
- Track the worst accuracy of the four groups
- Compute AUROC

I plan to plot the results and compare them to the results provided in the paper.

Since I wasn't yet able to fully complete the previous steps, and instead am stuck on the training portion, this section is a ToDo. I plan to complete it by week of April 21.

There are two alternative approaches I can take, depending on the situation:

- If I manage to run the original paper code, then this is what I will do, since it should be closest to the original paper
- If I won't be able to run the original paper training and validation code on my machine, I will update the code I wrote for training and validation to take it as close as possible to the intent of the original researchers.

In [12]:

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

def balance_df_label(df, sampler, label_bal=0.05154780337262089, invert=False
    target = df["Pneumonia"] == (1 if not invert else 0)
    rus = sampler(random_state=42, sampling_strategy=label_bal if not invert
    res_df, _ = rus.fit_resample(df, target)

    print(f"Previous pneumonia prop: {get_prop(df)} with {len(df)} instances"
    print(f"Resampled pneumonia prop: {get_prop(res_df)} with {len(res_df)} i

    return res_df

print('Balancing...')
mimic_balanced = balance_df_label(dfs["MIMIC"]["train"], RandomOverSampler, i
cxp_balanced = balance_df_label(dfs["CXP"]["train"], RandomOverSampler, inver
print('Done.')

# # Balance the size of the two datasets
# n = len(cxp_balanced)
# mimic_balanced = mimic_balanced.sample(n)
```

```
Balancing...
Previous pneumonia prop: 0.07020832445788779 with 50242 instances
Resampled pneumonia prop: 0.7184424658117837 with 80674 instances
Previous pneumonia prop: 0.02491281516815649 with 153411 instances
Resampled pneumonia prop: 0.051542603653077855 with 157397 instances
Done.
```

Calculating stats

In [13]:

```
stat_rows = []
num_instances = []

disease_labels = ["Pneumonia", "Cardiomegaly", "Edema", "Effusion", "Atelecta
target_labels = disease_labels + ["Any", "No Finding"]
all_labels = target_labels + ["Num Instances"]

for env in dfs:
    df = dfs[env]['all']
    df['Any'] = (df[disease_labels] > 0).any(axis=1).astype(int)
    totals = {}
    totals['Dataset'] = env
    totals['Num Instances'] = len(df)
    num_instances.append(totals['Num Instances'])

    for label in target_labels:
        if label in df.columns:
            totals[label] = df[label].sum() / len(df)
        else:
            totals[label] = 0.0

    stat_rows.append(totals)

stat_df = pd.DataFrame(stat_rows)
stat_df.set_index('Dataset', inplace=True)

ordered_cols = all_labels
stat_df = stat_df[ordered_cols]

transposed_stat_df = stat_df.T

# styled_stat_df = stat_df.style.background_gradient(cmap='Blues', subset=tar
#     .format({label: "{:.2%}" for label in target_labels})

styled_transposed_stat_df = transposed_stat_df.style.apply(
    lambda x: ["background-color: lightblue" if x.name != 'Num Instances' els
    axis=1
).background_gradient(cmap='Blues', subset=pd.IndexSlice[target_labels, :])
styled_transposed_stat_df = styled_transposed_stat_df.format("{:.2%}", subset
styled_transposed_stat_df = styled_transposed_stat_df.format("{:,.0f}", subse

styled_transposed_stat_df
```

Out[13]:

Dataset	MIMIC	CXP	NIH	PAD
Pneumonia	6.60%	2.45%	1.28%	4.90%
Cardiomegaly	17.64%	12.26%	2.48%	9.08%
Edema	12.05%	26.00%	2.05%	1.20%
Effusion	23.46%	40.25%	11.88%	6.01%
Atelectasis	20.30%	15.58%	10.31%	5.49%
Pneumothorax	3.99%	9.26%	4.73%	0.35%
Consolidation	4.64%	6.81%	4.16%	1.56%
Any	51.21%	70.35%	28.02%	23.01%
No Finding	34.60%	8.89%	53.84%	36.19%
Num Instances	63,047	191,229	112,120	99,827

Here is the table from the article for comparison:

Table 1: Total number of instances and disease prevalence in each dataset.

Target Label	MIMIC	CXP	NIH	PAD
Pneumonia	6.82%	2.43%	1.31%	4.84%
Cardiomegaly	17.05%	12.38%	2.51%	9.15%
Edema	11.83%	26.01%	2.11%	1.23%
Effusion	23.18%	40.28%	11.94%	5.99%
Atelectasis	20.11%	15.47%	10.33%	5.50%
Pneumothorax	4.19%	9.25%	4.66%	0.31%
Consolidation	4.67%	6.81%	4.19%	1.56%
Any	50.73%	70.35%	28.04%	23.03%
No Finding	34.76%	8.98%	53.65%	36.12%
Num Instances	243k	192k	113k	100k

Looks like the distribution of the labels in the original dataset, while not the same, still is close enough.

Model

The model includes the model definitation which usually is a class, model training, and other necessary parts.

Model architecture

In the article, the authors use the same model architecture as Zhang et al. (2021): a **DenseNet-121** network (Huang et al., 2017) **initialized with pre-trained weights from ImageNet** (Deng et al., 2009). The final layer is replaced with a **two-output linear layer** (for binary classification). For simplicity, the authors only consider binary disease classification.

Model Training

For training the network, all images are resized to **224 × 224** and normalized to the ImageNet (Deng et al., 2009) mean and standard deviation.

During training, the following image augmentations are applied:

- random horizontal flip
- random rotation up to 10 degrees
- a crop of random size (75% - 100%) and aspect ratio (3/4 to 4/3)

All runs use **Adam** with **lr = 1e-5** and **batch size = 128**, which was found to be a performant configuration in early tuning ((Zhang et al., 2021) use lr = 5e-4 and batch size = 32).

[This part I haven't implemented yet] Training runs for **a maximum of 20k steps**, with validation occurring every 500 steps and an early stopping patience of 10 validations.

All test results are obtained using the optimal model found during training as measured by the highest validation macro-F1 score (following (Fiorillo et al., 2021; Berenguer et al., 2022)) as it gives a robust ranking of model performance under imbalanced labels.

Details for model info:

- Model architecture: layer number/size/type, activation function, etc
- Training objectives: loss function, optimizer, weight of each loss term, etc
- Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc.
- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

In [14]:

```
# This is the model defined and provided by the autors of the article.
# While they are using densenet 121 for the article, the provided model code

class EmbModel(nn.Module):
    # I had to add the num_labels parameter to reduce the resulting response
    def __init__(self, emb_type, feature_size_override, pretrain, concat_feat
        super().__init__()
        self.emb_type = emb_type
        self.pretrain = pretrain
        self.concat_features = concat_features
        self.num_labels = num_labels

        assert emb_type in ["densenet121", "densenet201", "resnet"], f"Invali

        if emb_type == 'densenet121':
            model = models.densenet121()
            self.encoder = nn.Sequential(*list(model.children())[:-1]) #https
            self.emb_dim = model.classifier.in_features
        elif emb_type == 'densenet201':
            model = models.densenet201()
            self.encoder = nn.Sequential(*list(model.children())[:-1]) #https
            self.emb_dim = model.classifier.in_features
        elif emb_type == 'resnet':
            model = models.resnet50()
            self.encoder = nn.Sequential(*list(model.children())[:-1])
            self.emb_dim = list(model.children())[-1].in_features

        print("\nEmb Dim:")
        print(self.emb_dim)

        if feature_size_override:
            print(f"Manually setting output dim to {feature_size_override}")
            self.emb_dim = feature_size_override
            print(self.emb_dim)

        self.n_outputs = self.emb_dim + concat_features
        self.final_layer = nn.Linear(self.n_outputs, self.num_labels)
```

```
        nn.init.kaiming_normal_(self.final_layer.weight, mode='fan_out', nonl

    def forward(self, inp):
        if isinstance(inp, dict): # dict with image and additional feature(s)
            x = inp['img']
            concat = inp['concat']
            assert(concat.shape[-1] == self.concat_features)
        else: # tensor image
            assert(self.concat_features == 0)
            x = inp

        x = self.encoder(x).squeeze(-1).squeeze(-1)
        if "densenet" in self.emb_type:
            x = F.relu(x)
            x = F.avg_pool2d(x, kernel_size = 7).view(x.size(0), -1)

        if isinstance(inp, dict):
            x = torch.cat([x, concat], dim = -1)

        x = self.final_layer(x)
        return x
```

Training

Haven't figured out how to make the training from the supplied code work yet, so I am writing my own training code using the standard approach learned in class and homeworks.

Creating a data loader

The authors of the article have a script to load the data in different configurations. I am reusing it partially but wasn't able to make it work yet because of the errors, so I am creating my own Dataset class and a data loader that can deal with it.

In [15]:
```
class MultiEnvDataset(Dataset):
    def __init__(self, dataframes, subset='train', envs=None, transform=None)
        """
        Initializes the dataset with data from multiple environments and a sp
        :param dataframes: A dictionary with environment keys, each containin
        :param subset: The subset to load ('train', 'val', or 'test').
        :param envs: A list of environment names to include. If None, include
        :param transform: PyTorch transforms to apply to the images.
        """
        if envs is None:
            envs = list(dataframes.keys())

        self.data = pd.concat([dataframes[env][subset] for env in envs if env

        self.label_columns = ["No Finding", "Atelectasis", "Cardiomegaly", "E
                              "Pneumothorax", "Consolidation", "Edema"]
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        img_path = self.data.iloc[idx]['path']
        image = Image.open(img_path).convert('RGB')  # Convert to RGB to hand

        if self.transform:
            image = self.transform(image)

        labels = torch.tensor(self.data.iloc[idx][self.label_columns].values.
        if np.isnan(labels).any():
            raise ValueError("NaN values found in labels")

        return image, labels
```

In [16]:
```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225
])

train_dataset = MultiEnvDataset(dfs, subset='val', envs=['MIMIC', 'CXP'], tra
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shu

val_dataset = MultiEnvDataset(dfs, subset='test', envs=['MIMIC', 'CXP'], tran
val_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuff


# for images, labels in train_loader:
#     print(images.shape, labels.shape)
```

I have not been able to make the training work yet, see the issue below. The original paper provides separate scripts to do the training, which require some packages that seem to be not compatible with my platform. Still figuring out how to either make the original scripts work, or write my own training in a way that it provides results similar to the article.

In [18]:

```
model = EmbModel(emb_type="densenet121", feature_size_override=1024, pretrain
loss_func = nn.BCEWithLogitsLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)

def train_model_one_epoch(model, train_loader, loss_func, optimizer):
    model.train()
    running_loss = 0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)

        if torch.isnan(outputs).any():
            raise ValueError("NaN detected in model outputs")

        loss = loss_func(outputs, labels)
        if torch.isnan(loss).any():
            raise ValueError("NaN detected in loss computation")

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
        print('running loss', running_loss)

    epoch_loss = running_loss / len(train_loader.dataset)
    return epoch_loss

def validate_model(model, val_loader, loss_func):
    model.eval()
    running_loss = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            outputs = model(inputs)
            loss = loss_func(outputs, labels)
            running_loss += loss.item() * inputs.size(0)

    epoch_loss = running_loss / len(val_loader.dataset)
    return epoch_loss

num_epoch = 10
# model training loop: it is better to print the training/validation losses d
for i in range(num_epoch):
    train_loss = train_model_one_epoch(model, train_loader, loss_func, optimi
    valid_loss = validate_model(model, val_loader, loss_func)
    print("Epoch: %.2f, Train Loss: %.2f, Validation Loss: %.2f" % (i+1, trai
```

```
Emb Dim:
1024
Manually setting output dim to 1024
1024
Epoch: 1.00, Train Loss: 0.66, Validation Loss: 0.37
Epoch: 2.00, Train Loss: 0.36, Validation Loss: 0.35
Epoch: 3.00, Train Loss: 0.34, Validation Loss: 0.33
Epoch: 4.00, Train Loss: 0.33, Validation Loss: 0.32
Epoch: 5.00, Train Loss: 0.32, Validation Loss: 0.31
Epoch: 6.00, Train Loss: 0.31, Validation Loss: 0.30
Epoch: 7.00, Train Loss: 0.30, Validation Loss: 0.29
Epoch: 8.00, Train Loss: 0.30, Validation Loss: 0.28
Epoch: 9.00, Train Loss: 0.29, Validation Loss: 0.27
Epoch: 10.00, Train Loss: 0.28, Validation Loss: 0.26
```

Saving model

In [21]:

```
from datetime import datetime

now = datetime.now()
dt_string = now.strftime("%d-%m-%Y-%H-%M-%S")

torch.save(model.state_dict(), "model/model-snapshot" + dt_string + ".pth")
```

In [ ]:

```
# This block of code doesn't work yet, I am currently figuring out how to mak

from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
import wandb
import json

def train_models(df, pred_col, pred_vals, fix_col, fix_vals, results_dict, ta
    for fix_val in tqdm(fix_vals):
        # Subset the dataframe to just have this val in the column
        df_fix = df[df[fix_col] == fix_val]
        # Drop this column to avoid any oddities during training
        df_fix = df_fix.drop(columns=fix_col)

        # Get just the 2/4 classes that we're trying to
        for pred_val in tqdm(pred_vals):
            df_pred = df_fix[df_fix[pred_col].isin(pred_val)]

            # We have the final dataframe, but we need to create a perfectly
            # version of it
            grouped = df_pred.groupby(pred_col)
            # print("Count per class:", grouped["emb0"].count())
            min_group_size = grouped.count()["emb0"].min()
```

```
            df_bal = grouped.sample(n=min_group_size, random_state=0)
            # print("Count per class after balancing:", df_bal.groupby(pred_c

            # Note that we may have a single class remaining in our dataset (
            # CXP vs CXP prediction, for example). We need to check that and
            # If that is the case
            df_bal = df_bal.sample(frac=1, random_state=0).reset_index(drop=T

            if len(pred_val) == 1:
                print(f"INFO: SINGLE PRED VAL: {pred_val} for col: {pred_col}
                mid_val = len(df_bal) // 2

                df_bal.loc[:mid_val, pred_col] = "0"
                df_bal.loc[mid_val:, pred_col] = "1"

            # Now lets pass this dataframe into our train method
            acc = train_model(df_bal, pred_col)

            # Store the results in our global dictionary
            results_dict[task_type].append({
                "fix_val": fix_val,
                "pred_val": pred_val,
                "min_group_size": min_group_size,
                "df_size": len(df_bal),
                "acc": acc,
            })

def train_model(df, pred_col, max_iter=5000):
    X, y = df.drop(columns=pred_col), df[pred_col]

    model = make_pipeline(StandardScaler(), LinearSVC(random_state=0, tol=1e-

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

    model.fit(X_train, y_train)

    preds = model.predict(X_test)

    acc = accuracy_score(y_test, preds)
    return acc

def main():
#     wandb.init(project="ood-generalization",
#             job_type="emb_train",
#             entity="basedrhys",
#               name=f"row {row_idx}")

    results_dict = {}
    results_dict["env_pred"] = []
    results_dict["label_pred"] = []

    # Environment Prediction Task
    env_fix_col = "targets"
```

```
    env_fix_vals = [0, 1]

    env_pred_col = "env"
    env_pred_vals = [("CXP", ), ("MIMIC", ), ("NIH", ), ("PAD", ), ("CXP","NI

    train_models(df=ml_df,
                 pred_col=env_pred_col,
                 pred_vals=env_pred_vals,
                 fix_col=env_fix_col,
                 fix_vals=env_fix_vals,
                 results_dict=results_dict,
                 task_type="env_pred")

    # Label prediction task
    label_fix_col = "env"
    label_fix_vals = ["CXP", "MIMIC", "NIH", "PAD"]

    label_pred_col = "targets"
    label_pred_vals = [(0, 1), (0, ), (1, )]

    train_models(df=ml_df,
                 pred_col=label_pred_col,
                 pred_vals=label_pred_vals,
                 fix_col=label_fix_col,
                 fix_vals=label_fix_vals,
                 results_dict=results_dict,
                 task_type="label_pred")

#     wandb.log(results_dict)

    output_dir = row["output_dir"]

    print("Outputting JSON to", output_dir)

    with open(f"{output_dir}/emb_test_results.json", mode="w") as f:
        json.dump(results_dict, f, indent=True)

    return results_dict
row_idx = 2
main()
```

Results

Model comparison

You don't need to re-run all other experiments, instead, you can directly refer the metrics/numbers in the paper

Once available, I plan to compare my model performance using different datasets with the results posted in the paper.

Discussion

Is the paper reproducible?

It is too early to tell right now, but at least a portion of the code provided is runnable with minimal updates. I was able to reproduce the initial dataset statistics, so at least that portion is definitely reproducible. The rest will depend on whether I am able to solve and run the training.

If the paper is *not* reproducible, explain the results

TBD depending on whether the paper results will be reproducible or not.

What was easy and what was difficult

The authors did a great job documenting some parts of the project, for example, access to data. Following the instructions was very easy, and while MIMIC-CXR-JPG dataset access took some time to get, overall the process was a breeze.

Downloading the datasets is a hassle though, I ran out of space on my laptop, had to buy an external drive and restart the download process for MIMIC-CXR-JPG a few times.

There are a few notebooks and standalone scripts provided to process the data. While it is possible to figure out what steps need to be done in what order, many of the parts of the process are not documented. 'pyproject.toml' did not run successfully for me, and I've been stuck trying to figure out why and how to run it (I have a suspicion my processor architecture is not supported, but not enough experience to tell for sure yet).

In parallel, I opted to re-implement the training and model validation myself. There is code for training and validation in the project, which has a lot of comments (great!), but the process itself is not well documented, so the reproducer is left figuring out which steps in the code are needed and which are not, and how to adapt it to use for their experiment. The code is very general and there is a lot of it. There are some pointers in the readme, but they

are at this point not sufficient for reproducing things successfully without additional modification.

wandb isn't really working for me either yet, and I am yet to figure out why it is needed and whether it is necessary to reproduce the results.

The data is not processed evenly / equally for each dataset, there are different values for the same labels (NaN, True/False, 1/0, 1.1/0.0). I had to write some processing code to make sure we mitigate those differences.

Additional complication is due to the fact that the amount of data is very large. Any training or processing takes a long time, the notebook kernel dies frequently and the overall process is frustrating.

I tried to avoid multiple separate files and scripts, and pulled many of the data preprocessing into my notebook. However, this increased the runtime of the notebook significantly. Additional factor affecting the runtime is the size of the input data, even when working on one dataset. I doubt it would be possible to achieve the 8 minute runtime, but will try to do so.

Suggestions for the author

Trim the codebase leaving only relevant parts. Add documentation for the training and validation process. Add some background on why wandb is used and how to use it for this project correctly. Provide a suggested order of execution for the notebooks.

Plans for the next phase

In the remaining time until the final submission May 7 deadline, my plans are:

- Further update the data processing functions so they are producing similar type results (right now while compatible, it's a mix of *int*, *float* and *True/False*, I would like to homogenize the resulting dataset further)
- Finalize the training for the model and compute worst per-group accuracy for all data combinations listed in the article (so far I had the most issues with training, as the code supplied with the article didn't work and I had to come up with my own in which I try to replicate the experiment as close to the article description as possible)
- Plot the results and finalize the writeup (compute both worst per-group accuracy, and AUROC and compare to the results of the article)
- Prepare a subset of data and optimize the notebook to run under 8 minutes if at all

possible (as per the original requirements) - this might be complicated as the main focus of this article is dealing with more data and all datasets are quite large. Randomly picking samples from each dataset might further introduce some unintended spurious correlations
- Check the main hypothesis on both balanced and unbalanced datasets, time permitting
- Prepare a video presentation

References

1. Rhys Compton; Lily Zhang; Aahlad Puli; Rajesh Ranganath, When More is Less: Incorporating Additional Datasets Can Hurt Performance By Introducing Spurious Correlations, arXiv preprint, 2023-08-09, Accepted at MLHC 2023, doi: 10.48550/arXiv.2308.04431
2. Haoran Zhang, Natalie Dullerud, Laleh Seyyed-Kalantari, Quaid Morris, Shalmali Joshi, and Marzyeh Ghassemi. An empirical framework for domain generalization in clinical settings. In Proceedings of the Conference on Health, Inference, and Learning, pages 279–290, 2021, doi: 10.48550/arXiv.2103.11163
3. Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017, doi: 10.48550/arXiv.1608.06993
4. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009, doi: 10.1109/CVPR.2009.5206848
5. John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. PLoS medicine, 15(11): e1002683, 2018, doi: 10.1371/journal.pmed.1002683

In [ ]: