Nina Stawski's (group 90) final project report

Illinois ID: ninas2

GitHub repo link (https://github.com/nstawski/dlh-final-project)

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)

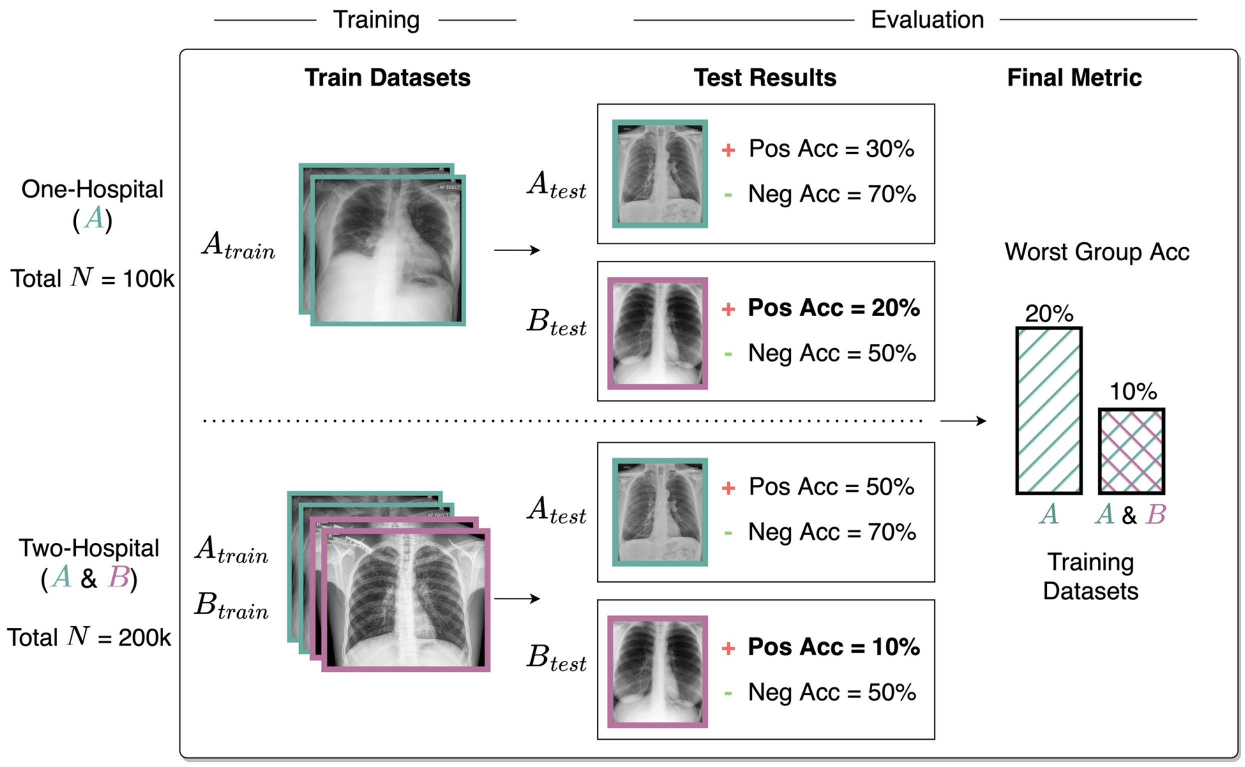
According to the article authors, their method works really well and proves that in nearly 50% of cases adding a second dataset, and even balancing it to reduce spurious correllations doesn't get the model to perform better than without that additional dataset. The models pick up on hospital-specific 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.

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.

Python environment and package versions

I have originally set up the Jupyter Notebook and was running the code on my laptop. However, processing of the data took days and the training was promising to take months. My husband had a gaming computer with a powerful video card and Cuda available, so I ended up using his machine.

To keep the environment isolated, I installed `Anaconda` and created a separate environment for all my packages. I then ran `Jupyter` with access to local network and opened my notebook remotly from my laptop:

```
jupyter notebook --ip 192.168.x.xxx --port 8888
```

Original requirements listed a bit different versions, but due to package compatibility I had to update them. Most significant changes were: install a version of torch that supports Cuda, and upgrade `torchvision to 0.9.1`

In [98]:
```
env_df = pd.read_csv('env_packages.csv', header=None, names=['Package'
styled_env_df = env_df.style.set_table_styles([
    {'selector': 'th, td',
     'props': [('min-width', '200px')]},
    {'selector': 'table',
     'props': [('width', '70%')]}
]).set_properties(**{
    'background-color': 'white',
    'color': 'black',
    'border-color': 'black',
    'border-style': 'solid',
    'border-width': '1px'
}).hide_index()

styled_env_df
```

Out[98]:

Package	Version
python	3.6.13
pip	21.2.2
jupyter_core	4.8.1
imbalanced-learn	0.8.1
jupyter	1.0.0
matplotlib	2.2.2
numpy	1.19.5
pandas	1.1.0
pillow	8.4.0
scikit-learn	0.24.2
scipy	1.5.1
seaborn	0.11.2
torch	1.8.1+cu111
torchvision	0.9.1

In [44]:

```
# !pip install importlib
# !pip install torch
# !pip install torchvision
# !pip install pandas
# !pip install matplotlib
# !pip install imblearn
```

In [99]:
```
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
import seaborn as sns
from PIL import Image, ImageFile

import json
import random
from IPython.display import display
from datetime import datetime
from sklearn.metrics import f1_score, classification_report, accuracy_



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

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

import Data_Constants as Constants

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

Out[99]:
```
<module 'Data_Constants' from 'C:\\Users\\Stan\\Documents\\GitHub\\dl
h-final-project\\Data_Constants.py'>
```

It was a lot of debugging to make sure Cuda is available in the notebook, and it took me a few days to finally make it work. It made all processing and training code run a lot faster.

The next challenge I encountered was that the full source data did not fit into my laptop and I had to get an external storage to hold it - however, that external storage's speed wasn't keeping up. So instead, my husband got an upgrade to his internal storage with a very fast ssd.

In [100]:
```
# os.environ['KMP_DUPLICATE_LIB_OK']='True'
# torch.set_default_device('cuda')

torch.set_default_tensor_type('torch.cuda.FloatTensor')
Tensor = torch.cuda.FloatTensor if torch.cuda.is_available() else torc

print("Cuda is available:", torch.cuda.is_available())
```

```
Cuda is available: True
```

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 (https://mimic-cxr.mit.edu/about/access/) to the MIMIC-CXR-JPG Database Database on PhysioNet and download the dataset (https://physionet.org/content/mimic-cxr-jpg/2.0.0/). 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 M
   IMIC-CXR-JPG
   ```

2. In order to obtain gender information for each patient, you will need to obtain access to MIMIC-IV (https://physionet.org/content/mimiciv/0.4/). Download `core/patients.csv.gz` and place the file in the `MIMIC-CXR-JPG` directory.

CheXpert

1. Sign up with your email address [here (https://stanfordmlgroup.github.io/competitions/chexpert/)](https://stanfordmlgroup.github.io/competitions/chexpert/).
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 (https://nihcc.app.box.com/v/ChestXray-NIHCC)](https://nihcc.app.box.com/v/ChestXray-NIHCC).
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 (https://academictorrents.com/details/96ebb4f92b85929eadfb16761f310a6d04105797)](https://academictorrents.com/details/96ebb4f92b85929eadfb16761f310a6d04105797).
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 [101]: #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'])

    print('meta_dir', meta_dir, os.getcwd())
    print('meta_dir', meta_dir/'mimic-cxr-2.0.0-metadata.csv')
    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 fu
    # in the original script, the file in p19 was being checked.
    assert (img_dir/'p10/p10000032/s50414267/02aa804e-bde0afdd-112c0b3

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_fro
    assert (img_dir/cxp_subfolder/'valid/patient64636/study1/view1_fro

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
    assert (img_dir/'185566798805711692534207714722577525271_qb3lyn.pn

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 [102]:
```
# making sure constants are up to date if they were changed after runn
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 = ['
    df['age_decile'] = pd.cut(df['anchor_age'], bins = list(range(0, 1
    df['frontal'] = df.ViewPosition.isin(['AP', 'PA'])

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

def preprocess_pad():
    # I have modified this function from the original one, because I w
    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
        '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
    df = df[['ImageID', 'StudyID', 'PatientID', 'PatientBirth', 'Patie
    df = df[~df["Labels"].isnull()]
    df = df[df["ImageID"].apply(lambda x: os.path.exists(os.path.join(
    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

    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).ast
    df['Age'] = 2017 - df['PatientBirth']
    df.reset_index(drop=True).to_csv(out_folder/"preprocessed.csv", in


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'/'tra
                        pd.read_csv(img_dir/'CheXpert-v1.0-small'/'val
                        ignore_index = True)
    elif (img_dir/'train.csv').is_file():
        raise ValueError('Please set Constants.image_paths["CXP"] to b
                ' 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.p
    df['Path'] = df['Path'].apply(lambda x: str(x).replace("CheXpert-v
    df.reset_index(drop = True).to_csv(out_folder/"preprocessed.csv",
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",
```

In [103]:

```
%%script false --no-raise-error
# skipping this cell since I already ran this.

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...
meta_dir C:\Nina\e-root\data\mimic\physionet.org\files\mimic-cxr-jpg\
2.0.0 C:\Users\Stan\Documents\GitHub\dlh-final-project
meta_dir C:\Nina\e-root\data\mimic\physionet.org\files\mimic-cxr-jpg\
2.0.0\mimic-cxr-2.0.0-metadata.csv
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 [104]:

```
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 OB
             'DIVORCED', 'SEPARATED', '0-10', '10-20', '20-30', '30-40
             '>=90'],
            [0, 0, 0, 1, 1, 0, 0, 'MARRIED/LIFE PARTNER', 'MARRIED/LIF
             'DIVORCED/SEPARATED', '0-20', '0-20', '20-40', '20-40', '

    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.
    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', '

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

    copy_subjectid = split['Patient ID']

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

    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: o
    split['env'] = 'NIH'
    split['frontal'] = True
    split['study_id'] = split['subject_id'].astype(str)
    return split[['subject_id','path','Sex',"Age", 'env', 'frontal','s


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['Ag
    split['Age'] = np.where(split['Age'].between(40,59), 59, split['Ag
    split['Age'] = np.where(split['Age'].between(60,79), 79, split['Ag
    split['Age'] = np.where(split['Age']>=80, 81, split['Age'])
    copy_subjectid = split['subject_id']
    split = split.drop(columns = ['subject_id']).replace([[None], -1,
                            [0, 0, 0, 1, 1, "0-20", "20-40", "40-60",

    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.
    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('patie
    return split[['subject_id','path','Sex',"Age", 'env', 'frontal','s


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['Ag
    split['Age'] = np.where(split['Age'].between(40,59), 59, split['Ag
    split['Age'] = np.where(split['Age'].between(60,79), 79, split['Ag
    split['Age'] = np.where(split['Age']>=80, 81, split['Age'])

    split = split.replace([[None], -1, "[False]", "[True]", "[ True]",
                            [0, 0, 0, 1, 1, "0-20", "20-40", "40-60",

    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.p
    if only_frontal:
        split = split[split['frontal']]
    split['env'] = 'PAD'
    return split[['subject_id','path','Sex',"Age", 'env', 'frontal','s


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 trai
    subject_df = pd.DataFrame({'subject_id': np.sort(df['subject_id'].
    subject_df['random_number'] = np.random.uniform(size=len(subject_d

    train_id = subject_df[subject_df['random_number'] <= split_portion
    valid_id = subject_df[(subject_df['random_number'] > split_portion
    test_id = subject_df[subject_df['random_number'] > split_portions[

    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 [105]:
```
# 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 [106]:
```
%%script false --no-raise-error
# skipping this cell since I already ran this.

# 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_fror
    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",

    display(df_env)

print("Done.")
```

144479	112930952416074060371371014599496493673	C:\Nina\e-root\data\PadChest\images-224\128401...
144480	282743729971423358706056731890510600934	C:\Nina\e-root\data\PadChest\images-224\128401...
144481	52648743308541843883453242716226652771	C:\Nina\e-root\data\PadChest\images-224\128401...
144482	228646130593152933811948996634154201216	C:\Nina\e-root\data\PadChest\images-224\128401...
144483	137424047230303610602080410284588825286	C:\Nina\e-root\data\PadChest\images-224\128401...

99827 rows × 17 columns

```
Done.
```

Resample data

In [107]:
```
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: 230693
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 [108]:
```
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

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

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

    return res_df

def balance_proportion(orig_df, new_df, resample_method="over", column
    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_
    print(f"Resampling '{column}' via '{resample_method}' on class {re

    # 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 {ne
            df_normal_rs = df_normal.sample(new_num_normal, replace=Tr
            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}
            # df_diseased_rs = df_diseased.sample(new_num_diseased, re
            # resampled_df = pd.concat([df_normal, df_diseased_rs])
            target = df["Pneumonia"] == 1
            rus = RandomOverSampler(random_state=0, sampling_strategy=
            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 [109]: dfs["CXP"]["train"]
```

Out[109]:

	subject_id	path
0	1	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
1	2	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
2	2	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
3	3	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
4	4	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
...	...	...
191222	64734	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
191223	64735	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
191225	64737	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
191227	64739	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...
191228	64740	C:\Nina\e-root\data\CheXpert\CheXpert-v1.0-sma...

153411 rows × 17 columns

In [119]:
```
def balance_df_label(df, sampler, label_bal=0.05154780337262089, inver
    target = df["Pneumonia"] == (1 if not invert else 0)
    rus = sampler(random_state=42, sampling_strategy=label_bal if not
    res_df, _ = rus.fit_resample(df, target)

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

    return res_df

# # uncomment this code if you want a balanced dataset
# print('Balancing...')
# mimic_balanced = balance_df_label(dfs["MIMIC"]["train"], RandomOverS
# cxp_balanced = balance_df_label(dfs["CXP"]["train"], RandomOverSampl
# print('Done.')

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

In [117]:
```
import warnings
warnings.filterwarnings('ignore')
```

Calculating stats

In [120]:
```
stat_rows = []
num_instances = []

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

dfs2 = {}

for env in dfs:
    df = dfs[env]['all']
    df['Any'] = (df[disease_labels] > 0).any(axis=1).astype(int)

    # keep only every 30th sample for the dataset to reduce the size o
    # I am keeping the full dataset coe commented out to be able to ea
    df2 = df[df.index % 30 == 0]
    dfs2[env] = {}
    dfs2[env]['all'] = df2

    train_df, valid_df, test_df = split(df2)
    dfs2[env] = {
        'all': df_env,
        'train': train_df,
        'val': valid_df,
        'test': test_df
    }

    totals = {}
    totals['Dataset'] = env
#     totals['Num Instances'] = len(df)
    totals['Num Instances'] = len(df2)
    num_instances.append(totals['Num Instances'])

    for label in target_labels:
#         if label in df.columns:
#             totals[label] = df[label].sum() / len(df)
        if label in df2.columns:
            totals[label] = df2[label].sum() / len(df2)
        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_transposed_stat_df = transposed_stat_df.style.apply(
    lambda x: ["background-color: lightblue" if x.name != 'Num Instanc
    axis=1
).background_gradient(cmap='Blues', subset=pd.IndexSlice[target_labels
styled_transposed_stat_df = styled_transposed_stat_df.format("{:.2%}",
styled_transposed_stat_df = styled_transposed_stat_df.format("{:,.0f}"

styled_transposed_stat_df
```

Out[120]:

Dataset	MIMIC	CXP
Pneumonia	6.87%	2.68%
Cardiomegaly	16.62%	12.27%
Edema	11.70%	25.77%
Effusion	22.94%	40.20%
Atelectasis	20.42%	16.02%
Pneumothorax	4.23%	9.33%

Consolidation	4.68%	6.59%
Any	50.51%	71.14%
No Finding	35.01%	8.69%
Num Instances	7,690	6,375

Here is the table from the article for comparison:

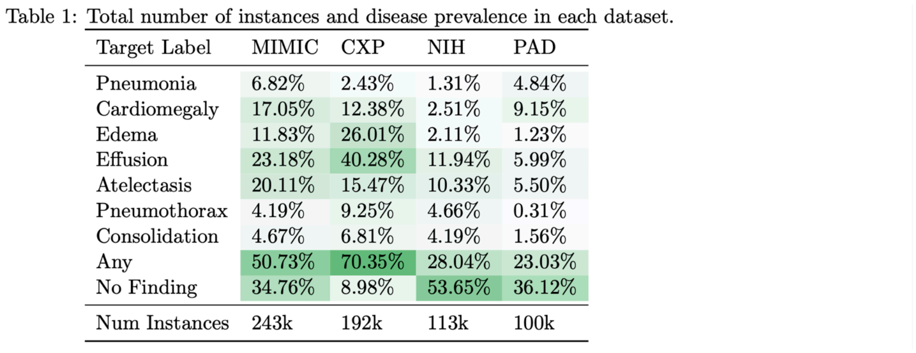


Table 1: Total number of instances and disease prevalence in each dataset.

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.

Citation to the original paper

- 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 (https://doi.org/10.48550/arXiv.2308.04431)

Original paper repo

- ood-generalization (https://github.com/basedrhys/ood-generalization/tree/master)

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).

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.

In [121]:
```
# This is the model defined and provided by the autors of the article.
# While they are using densenet 121 for the article, the provided mode

class EmbModel(nn.Module):
    # I had to add the num_labels parameter to reduce the resulting re
    def __init__(self, emb_type, feature_size_override, pretrain, conc
        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

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

    def forward(self, inp):
        if isinstance(inp, dict): # dict with image and additional fea
            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

I wasn't able to run the training code provided by the authors of the article - the setup didn't work for me neither on my MacBook Pro laptop, nor on my husband's Windows 10 gaming computer.

To proceed, I instead wrote my own training code using the standard approach learned in class and homeworks.

Hyperparameters used

- Model: `densenet121`
- Number of epochs for each model trainig: `10`
- Hidden size: `1024` since I am setting the `feature_size_override` to `1024`
- Batch size: `128`
- Learning rate: `1e-5`
- Optimizer: `Adam`

Computational requirements

It is possible to run this code on a CPU with minor modifications. Hovewer, since I moved to another computer with GPU, some portions of this notebook send the computation to Cuda directly (todo: rewrite so it checks for Cuda and sends to the appropriate device).

Hardware and software

- AMD Ryozen 7 7800X4D 8-Core Processor (4.20 GHz)
- 64 GB RAM
- NVMe Samsung SSD 970 SCSI - 1TB
- Windows 10 64-bit

Training requirements

I was not able to train on the full dataset since even on GPU one epoch of one model was running for 2-4 hours depending on the number of batches. This would require roughly 16 days to finish the whole training.

Initially, I attempted to run the training on the full dataset, but a number of circumstances (out of memory, kernel panic, random automatic Windows updates, power down, kids getting to the computer and switching the power supply off) proved that the expectation to run the training continuously for days to be completely unrealistic.

Instead, I modified my dataset to pick every 30th entry and discard the rest. As a result, I was able to run the training multiple times with different parameters when needed, both on balanced and unbalanced datasets.

- Average epoch running time: `6min`
- Average time to complete all training: `12h`
- Total number of attempts: `200+`

Creating a data loader

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

In [122]:
```



ImageFile.LOAD_TRUNCATED_IMAGES = True # I was getting errors during t
```

In [123]:
```
class MultiEnvDataset(Dataset):
    def __init__(self, dataframes, subset='train', envs=None, transfor
        """
        Initializes the dataset with data from multiple environments a
        :param dataframes: A dictionary with environment keys, each co
        :param subset: The subset to load ('train', 'val', or 'test').
        :param envs: A list of environment names to include. If None, 
        :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

        self.label_columns = ["No Finding", "Atelectasis", "Cardiomega
                              "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')  # Converts to RGE

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


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

        return image, labels
```

In [124]:
```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224
])
```

In [125]:
```
envs_list = [["CXP"], ["MIMIC"], ["NIH"], ["PAD"], ["CXP","NIH"], ["CX
env_list_map = {
    "cxp": 0,
    "mimic": 1,
    "nih": 2,
    "pad": 3,
    "cxp_nih": 4,
    "cxp_pad": 5,
    "mimic_cxp": 6,
    "mimic_nih": 7,
    "mimic_pad": 8,
    "nih_pad": 9,
    "cxp_mimic_nih_pad": 10,
}

# a few functions to simplify getting the right names of the dataset c
def get_dataset_index(env_name):
    return env_list_map[env_name]

def get_env_from_list(env_name):
    return envs_list[env_list_map[env_name]]
```

In [126]:
```
datasets = []
for env in envs_list:
    elem = {
        "env": env
    }
    for subset in ["train", "val"]:
        elem[subset] = {}
#         elem[subset]["dataset"] = MultiEnvDataset(dfs, subset=subset
        elem[subset]["dataset"] = MultiEnvDataset(dfs2, subset=subset,
        elem[subset]["loader"] = torch.utils.data.DataLoader(elem[subs
    datasets.append(elem)
print("Done.")
```

```
Done.
```

Metrics

In [127]:
```
loss_func = nn.BCEWithLogitsLoss()

max_batches = 10

def calculate_accuracies(outputs, labels):
    predictions = torch.sigmoid(outputs) > 0.5
    predictions = predictions.to(labels.device)
    correct_pred = (predictions == labels)
    accuracies = correct_pred.float().mean(axis=0)
    return accuracies

def calculate_f1(outputs, labels):
    predictions = torch.sigmoid(outputs) > 0.5
    predictions = predictions.to(labels.device)

    predictions = predictions.detach().cpu().numpy()
    labels = labels.detach().cpu().numpy()

    f1 = f1_score(labels, predictions, average=None)
    return f1

def train_model_one_epoch(model, train_loader, loss_func, optimizer):
    print("Starting training...")
    start = datetime.now()
    prev = start
    model.train()
    running_loss = 0
    total_accuracy = []
    total_f1_scores = []

    print('number of batches:', len(train_loader))
    for batch, (inputs, labels) in enumerate(train_loader):
        inputs = inputs.cuda()

        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.
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)

        accuracies = calculate_accuracies(outputs, labels)
        f1_scores = calculate_f1(outputs, labels)
        total_accuracy.append(accuracies)
        total_f1_scores.append(f1_scores)

        if batch % 100 == 0:
            mid = datetime.now()
            print("time passed from the beginning", mid-start)
            print('batch', batch + 1, 'time passed:', mid-prev)
            prev = mid

    epoch_loss = running_loss / len(train_loader.dataset)
    end = datetime.now()
    print("epoch done in", end-start, "number of batches:", batch)
    epoch_accuracy = torch.stack(total_accuracy).mean(dim=0)
    epoch_f1 = torch.tensor(total_f1_scores).mean(dim=0)
    return epoch_loss, epoch_accuracy, epoch_f1

def validate_model(model, val_loader, loss_func):
    model.eval()
    running_loss = 0
    total_accuracy = []
    total_f1_scores = []
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs = inputs.cuda()
            outputs = model(inputs)
            loss = loss_func(outputs, labels)
            running_loss += loss.item() * inputs.size(0)

            accuracies = calculate_accuracies(outputs, labels)
            f1_scores = calculate_f1(outputs, labels)
            total_accuracy.append(accuracies)
            total_f1_scores.append(f1_scores)

    epoch_loss = running_loss / len(val_loader.dataset)
    epoch_accuracy = torch.stack(total_accuracy).mean(dim=0)
    epoch_f1 = torch.tensor(total_f1_scores).mean(dim=0)
    return epoch_loss, epoch_accuracy, epoch_f1
```

In [128]:
```
def saveModel(model, env=None):
    now = datetime.now()
    dt_string = now.strftime("%d-%m-%Y-%H-%M-%S")

    model_file_name = "model/model-snapshot-"
    if (env):
        model_file_name += "env_" + "_".join(env) + "_"
    model_file_name += dt_string + ".pth"

    torch.save(model.state_dict(), model_file_name)
```

In [19]:

```
%%script false --no-raise-error
# skipping the training since I already ran it in different variations

num_epoch = 10

metrics_df = pd.DataFrame(columns=["env", "epoch", "train_loss", "vali

for dataset in datasets:
    model = EmbModel(emb_type="densenet121", feature_size_override=102
    model.cuda()
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)
    print("Processing dataset env:", dataset["env"])
    for i in range(num_epoch):
        train_loss, train_accuracy, train_f1 = train_model_one_epoch(m
        valid_loss, valid_accuracy, valid_f1 = validate_model(model, c

        print("Epoch: %.2f, Train Loss: %.2f, Validation Loss: %.2f" %

        # Convert tensors to CPU for DataFrame update
        train_accuracy = train_accuracy.cpu().numpy()
        valid_accuracy = valid_accuracy.cpu().numpy()
        train_f1 = train_f1.cpu().numpy()
        valid_f1 = valid_f1.cpu().numpy()

        # Append metrics to DataFrame
        metrics_df = metrics_df.append({
            "env": dataset["env"],
            "epoch": i + 1,
            "train_loss": train_loss,
            "valid_loss": valid_loss,
            "train_accuracy": np.mean(train_accuracy),
            "valid_accuracy": np.mean(valid_accuracy),
            "worst_train_accuracy": np.min(train_accuracy),
            "worst_valid_accuracy": np.min(valid_accuracy),
            "train_f1": np.mean(train_f1),
            "valid_f1": np.mean(valid_f1)
        }, ignore_index=True)

    saveModel(model, env=dataset["env"])
print("All done.")
```

```
Emb Dim:
1024
Manually setting output dim to 1024
1024
Processing dataset env: ['CXP']
Starting training...
number of batches: 40
```

```
time passed from the beginning 0:00:02.636734
batch 1 time passed: 0:00:02.636734
epoch done in 0:00:26.762730 number of batches: 39
Epoch: 1.00, Train Loss: 0.90, Validation Loss: 0.62
Starting training...
number of batches: 40
time passed from the beginning 0:00:00.602461
batch 1 time passed: 0:00:00.602461
epoch done in 0:00:24.425553 number of batches: 39

C:\Users\Stan\anaconda3\envs\gpu\lib\site-packages\sklearn\metrics\_c
lassification.py:1496: UndefinedMetricWarning: F-score is ill-defined
```

In [22]:

```
%%script false --no-raise-error
# skipping this cell since I already ran this.

# Save DataFrame to CSV

df_now = datetime.now()
df_dt_string = df_now.strftime("%d-%m-%Y-%H-%M-%S")
metrics_df.to_csv(f"stats/{df_dt_string}_training_metrics.csv", index=
```

Validating the saved models and visualizing results

In [129]:
```
def predict(model, val_loader, device='cuda'):
    model.eval()
    model.to(device)
    all_preds = []
    all_preds_raw = []
    all_labels = []

    print('Started prediction validation')
    print('Number of batches:', len(val_loader))
    predict_start_time = datetime.now()
    with torch.no_grad():
        for batch, (images, labels) in enumerate(val_loader):
            print("Batch number:", batch+1, "of", len(val_loader))
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            probabilities = torch.sigmoid(outputs)

            preds = (probabilities > 0.5)

            any_disease = torch.any(preds[:, 1:], dim=1, keepdim=True)
            any_probability = torch.max(probabilities[:, 1:], dim=1, k
            any_label = torch.any(labels[:, 1:], dim=1, keepdim=True)

            preds = torch.cat((preds, any_disease), dim=1)
            probabilities = torch.cat((probabilities, any_probability)
            labels = torch.cat((labels, any_label), dim=1)

            all_preds_raw.append(probabilities.cpu().numpy())
            all_preds.append(preds.cpu().numpy())
            all_labels.append(labels.cpu().numpy())

    all_preds = np.vstack(all_preds)
    all_preds_raw = np.vstack(all_preds_raw)
    all_labels = np.vstack(all_labels)

    predict_end_time = datetime.now()

    print('Done.')
    print('Prediction took:', predict_end_time-predict_start_time)

    return all_preds, all_preds_raw, all_labels
```

In [130]:
```
def calculate_per_label_accuracy(predictions, labels):
    accuracies = {}
    num_labels = labels.shape[1]

    for i in range(num_labels):
        label_preds = predictions[:, i]
        label_true = labels[:, i]
        accuracies[target_labels[i]] = accuracy_score(label_true, labe
    return accuracies

def calculate_stats(predictions, probabilities, labels, source):

    # Calculate overall accuracy
    accuracy = accuracy_score(labels, predictions)
    print(f"Overall Accuracy: {accuracy:.2%}")

    # Detailed classification report for each disease label
    report = classification_report(labels, predictions, target_names=t
    report_df = pd.DataFrame(report).transpose()
    report_df['source'] = source
    print("Classification Report:")
    print(report_df)

    label_accuracies = calculate_per_label_accuracy(predictions, label
    worst_label = min(label_accuracies, key=label_accuracies.get)
    worst_accuracy = label_accuracies[worst_label]

    accuracy_df = pd.DataFrame(list(label_accuracies.items()), columns
    accuracy_df['Accuracy'] = accuracy_df['Accuracy'].apply(lambda x:
    accuracy_df['source'] = source

    print(accuracy_df)
    print(f"Worst Performing Label: {worst_label} with an accuracy of
    return report_df, accuracy_df
```

In [131]:
```
# repeating the labels code so I don't have to re-run the cell way abo
target_labels = ["No Finding", "Atelectasis", "Cardiomegaly", "Effusic

device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Loading trained models

I separated the loading code into different cells rather than had a cycle in one, to be able to pick and choose which parts I run. The model names are hardcoded with the ones currently in the repo - if you are running the training code above, the new snapshots will be created and the file names should be updated below.

MIMIC only

In [133]:
```
print("Model trained on MIMIC")
model_MIMIC = EmbModel(emb_type="densenet121", feature_size_override=1
model_MIMIC.load_state_dict(torch.load("model/balanced/model-snapshot-
model_MIMIC.eval()
```

```
Model trained on MIMIC

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[133]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
```

In [134]:
```
data_loader_MIMIC = datasets[get_dataset_index("mimic")]["val"]["loade
mimic_predictions, mimic_probabilities, mimic_labels = predict(model_M
print('labels', mimic_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 6
Batch number: 1 of 6
Batch number: 2 of 6
Batch number: 3 of 6
Batch number: 4 of 6
Batch number: 5 of 6
Batch number: 6 of 6
Done.
Prediction took: 0:00:35.871596
labels [[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]]
Done
```

In [135]:
```
mimic_report, mimic_accuracy = calculate_stats(mimic_predictions, mimi
```

```
Overall Accuracy: 25.81%
Classification Report:
               precision    recall  f1-score  support source
No Finding      0.715190  0.436293  0.541966    259.0  mimic
Atelectasis     0.269231  0.088050  0.132701    159.0  mimic
Cardiomegaly    0.153846  0.016393  0.029630    122.0  mimic
Effusion        0.471910  0.280000  0.351464    150.0  mimic
Pneumonia       0.000000  0.000000  0.000000     42.0  mimic
Pneumothorax    0.000000  0.000000  0.000000     29.0  mimic
Consolidation   0.000000  0.000000  0.000000     36.0  mimic
Edema           0.156250  0.070423  0.097087     71.0  mimic
Any             0.701389  0.274457  0.394531    368.0  mimic
micro avg       0.561866  0.224110  0.320416   1236.0  mimic
macro avg       0.274202  0.129513  0.171931   1236.0  mimic
weighted avg    0.474759  0.224110  0.299259   1236.0  mimic
samples avg     0.248288  0.231509  0.235901   1236.0  mimic
           Label Accuracy source
0     No Finding   74.19%  mimic
1    Atelectasis   75.27%  mimic
2   Cardiomegaly   82.30%  mimic
3       Effusion   79.05%  mimic
4      Pneumonia   93.65%  mimic
5   Pneumothorax   96.08%  mimic
6  Consolidation   95.14%  mimic
7          Edema   87.43%  mimic
8            Any   58.11%  mimic
Worst Performing Label: Any with an accuracy of 58.11%
```

In [136]:
```
def combine_with_existing(combined_accuracy, combined_report, add_accu
    combined_accuracy = pd.concat([combined_accuracy, add_accuracy], i
    combined_report = pd.concat([combined_report, add_report], ignore_

    print(combined_accuracy)
    print(combined_report)

    return combined_accuracy, combined_report
```

PAD only

In [137]:
```
print("Model trained on PAD")
model_PAD = EmbModel(emb_type="densenet121", feature_size_override=102
model_PAD.load_state_dict(torch.load("model/balanced/model-snapshot-er
model_PAD.eval()
```

```
Model trained on PAD

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[137]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
```

In [138]:
```
data_loader_PAD = datasets[get_dataset_index("pad")]["val"]["loader"]
pad_predictions, pad_probabilities, pad_labels = predict(model_PAD, da
print('labels', pad_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 3
Batch number: 1 of 3
Batch number: 2 of 3
Batch number: 3 of 3
Done.
Prediction took: 0:00:02.932136
labels [[1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 0. 1. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 1.]]
Done
```

```
In [139]: pad_report, pad_accuracy = calculate_stats(pad_predictions, pad_probab
```

```
Overall Accuracy: 46.04%
Classification Report:
               precision    recall  f1-score  support source
No Finding      0.656250  0.338710  0.446809    124.0    pad
Atelectasis     0.000000  0.000000  0.000000     16.0    pad
Cardiomegaly    0.000000  0.000000  0.000000     30.0    pad
Effusion        0.200000  0.043478  0.071429     23.0    pad
Pneumonia       0.000000  0.000000  0.000000     19.0    pad
Pneumothorax    0.000000  0.000000  0.000000      0.0    pad
Consolidation   0.000000  0.000000  0.000000      3.0    pad
Edema           0.000000  0.000000  0.000000      3.0    pad
Any             0.416667  0.065789  0.113636     76.0    pad
micro avg       0.533333  0.163265  0.250000    294.0    pad
macro avg       0.141435  0.049775  0.070208    294.0    pad
weighted avg    0.400142  0.163265  0.223413    294.0    pad
samples avg     0.135671  0.137195  0.136179    294.0    pad
           Label Accuracy source
0     No Finding   68.29%    pad
1    Atelectasis   94.21%    pad
2   Cardiomegaly   90.55%    pad
3       Effusion   92.07%    pad
4      Pneumonia   92.99%    pad
5   Pneumothorax  100.00%    pad
6  Consolidation   99.09%    pad
7          Edema   98.78%    pad
8            Any   76.22%    pad
Worst Performing Label: No Finding with an accuracy of 68.29%
```

In [140]:
```
combined_accuracy, combined_report = combine_with_existing(mimic_repor
```

```
    precision    recall  f1-score  support source
0    0.715190  0.436293  0.541966    259.0  mimic
1    0.269231  0.088050  0.132701    159.0  mimic
2    0.153846  0.016393  0.029630    122.0  mimic
3    0.471910  0.280000  0.351464    150.0  mimic
4    0.000000  0.000000  0.000000     42.0  mimic
5    0.000000  0.000000  0.000000     29.0  mimic
6    0.000000  0.000000  0.000000     36.0  mimic
7    0.156250  0.070423  0.097087     71.0  mimic
8    0.701389  0.274457  0.394531    368.0  mimic
9    0.561866  0.224110  0.320416   1236.0  mimic
10   0.274202  0.129513  0.171931   1236.0  mimic
11   0.474759  0.224110  0.299259   1236.0  mimic
12   0.248288  0.231509  0.235901   1236.0  mimic
13   0.656250  0.338710  0.446809    124.0    pad
14   0.000000  0.000000  0.000000     16.0    pad
15   0.000000  0.000000  0.000000     30.0    pad
16   0.200000  0.043478  0.071429     23.0    pad
17   0.000000  0.000000  0.000000     19.0    pad
18   0.000000  0.000000  0.000000      0.0    pad
19   0.000000  0.000000  0.000000      3.0    pad
20   0.000000  0.000000  0.000000      3.0    pad
21   0.416667  0.065789  0.113636     76.0    pad
22   0.533333  0.163265  0.250000    294.0    pad
23   0.141435  0.049775  0.070208    294.0    pad
24   0.400142  0.163265  0.223413    294.0    pad
25   0.135671  0.137195  0.136179    294.0    pad
            Label Accuracy source
0      No Finding   74.19%  mimic
1     Atelectasis   75.27%  mimic
2    Cardiomegaly   82.30%  mimic
3        Effusion   79.05%  mimic
4       Pneumonia   93.65%  mimic
5    Pneumothorax   96.08%  mimic
6   Consolidation   95.14%  mimic
7           Edema   87.43%  mimic
8             Any   58.11%  mimic
9      No Finding   68.29%    pad
10    Atelectasis   94.21%    pad
11   Cardiomegaly   90.55%    pad
12       Effusion   92.07%    pad
13      Pneumonia   92.99%    pad
14   Pneumothorax  100.00%    pad
15  Consolidation   99.09%    pad
16          Edema   98.78%    pad
17            Any   76.22%    pad
```

CXP only

```
In [141]: print("Model trained on CXP")
          model_CXP = EmbModel(emb_type="densenet121", feature_size_override=102
          model_CXP.load_state_dict(torch.load("model/balanced/model-snapshot-en
          model_CXP.eval()
```

```
Model trained on CXP

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

```
Out[141]: EmbModel(
            (encoder): Sequential(
              (0): Sequential(
                (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
          ng=(3, 3), bias=False)
                (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
                (relu0): ReLU(inplace=True)
                (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
          =1, ceil_mode=False)
                (denseblock1): _DenseBlock(
                  (denselayer1): _DenseLayer(
                    (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
```

In [142]:
```
data_loader_CXP = datasets[get_dataset_index("cxp")]["val"]["loader"]
cxp_predictions, cxp_probabilities, cxp_labels = predict(model_CXP, da

print('predictions', cxp_predictions)
print('labels', cxp_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 5
Batch number: 1 of 5
Batch number: 2 of 5
Batch number: 3 of 5
Batch number: 4 of 5
Batch number: 5 of 5
Done.
Prediction took: 0:00:07.939821
predictions [[False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]
 ...
 [False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]]
labels [[0. 0. 0. ... 0. 0. 1.]
 [0. 1. 0. ... 0. 1. 1.]
 [0. 0. 0. ... 0. 1. 1.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 1.]]
Done
```

In [143]:
```
cxp_report, cxp_accuracy = calculate_stats(cxp_predictions, cxp_probab
```

```
Overall Accuracy: 22.86%
Classification Report:
               precision    recall  f1-score  support source
No Finding      0.000000  0.000000  0.000000     60.0    cxp
Atelectasis     0.000000  0.000000  0.000000     97.0    cxp
Cardiomegaly    0.000000  0.000000  0.000000     74.0    cxp
Effusion        0.631016  0.460938  0.532731    256.0    cxp
Pneumonia       0.000000  0.000000  0.000000     12.0    cxp
Pneumothorax    0.000000  0.000000  0.000000     55.0    cxp
Consolidation   0.000000  0.000000  0.000000     43.0    cxp
Edema           0.432203  0.356643  0.390805    143.0    cxp
Any             0.836207  0.437923  0.574815    443.0    cxp
micro avg       0.663620  0.306847  0.419653   1183.0    cxp
macro avg       0.211047  0.139500  0.166483   1183.0    cxp
weighted avg    0.501931  0.306847  0.377775   1183.0    cxp
samples avg     0.248677  0.221534  0.228012   1183.0    cxp
           Label Accuracy source
0     No Finding   89.84%    cxp
1    Atelectasis   83.81%    cxp
2   Cardiomegaly   88.10%    cxp
3       Effusion   67.14%    cxp
4      Pneumonia   98.10%    cxp
5   Pneumothorax   91.27%    cxp
6  Consolidation   93.17%    cxp
7          Edema   74.76%    cxp
8            Any   54.44%    cxp
Worst Performing Label: Any with an accuracy of 54.44%
```

In [144]:
```
combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
    precision    recall  f1-score  support source
0    0.715190  0.436293  0.541966    259.0  mimic
1    0.269231  0.088050  0.132701    159.0  mimic
2    0.153846  0.016393  0.029630    122.0  mimic
3    0.471910  0.280000  0.351464    150.0  mimic
4    0.000000  0.000000  0.000000     42.0  mimic
5    0.000000  0.000000  0.000000     29.0  mimic
6    0.000000  0.000000  0.000000     36.0  mimic
7    0.156250  0.070423  0.097087     71.0  mimic
8    0.701389  0.274457  0.394531    368.0  mimic
9    0.561866  0.224110  0.320416   1236.0  mimic
10   0.274202  0.129513  0.171931   1236.0  mimic
11   0.474759  0.224110  0.299259   1236.0  mimic
12   0.248288  0.231509  0.235901   1236.0  mimic
13   0.656250  0.338710  0.446809    124.0    pad
14   0.000000  0.000000  0.000000     16.0    pad
15   0.000000  0.000000  0.000000     30.0    pad
```

```
16   0.200000  0.043478  0.071429     23.0    pad
17   0.000000  0.000000  0.000000     19.0    pad
18   0.000000  0.000000  0.000000      0.0    pad
19   0.000000  0.000000  0.000000      3.0    pad
20   0.000000  0.000000  0.000000      3.0    pad
21   0.416667  0.065789  0.113636     76.0    pad
22   0.533333  0.163265  0.250000    294.0    pad
23   0.141435  0.049775  0.070208    294.0    pad
24   0.400142  0.163265  0.223413    294.0    pad
25   0.135671  0.137195  0.136179    294.0    pad
26   0.000000  0.000000  0.000000     60.0    cxp
27   0.000000  0.000000  0.000000     97.0    cxp
28   0.000000  0.000000  0.000000     74.0    cxp
29   0.631016  0.460938  0.532731    256.0    cxp
30   0.000000  0.000000  0.000000     12.0    cxp
31   0.000000  0.000000  0.000000     55.0    cxp
32   0.000000  0.000000  0.000000     43.0    cxp
33   0.432203  0.356643  0.390805    143.0    cxp
34   0.836207  0.437923  0.574815    443.0    cxp
35   0.663620  0.306847  0.419653   1183.0    cxp
36   0.211047  0.139500  0.166483   1183.0    cxp
37   0.501931  0.306847  0.377775   1183.0    cxp
38   0.248677  0.221534  0.228012   1183.0    cxp
            Label Accuracy source
0      No Finding   74.19%  mimic
1     Atelectasis   75.27%  mimic
2    Cardiomegaly   82.30%  mimic
3        Effusion   79.05%  mimic
4       Pneumonia   93.65%  mimic
5    Pneumothorax   96.08%  mimic
6   Consolidation   95.14%  mimic
7           Edema   87.43%  mimic
8             Any   58.11%  mimic
9      No Finding   68.29%    pad
10    Atelectasis   94.21%    pad
11   Cardiomegaly   90.55%    pad
12       Effusion   92.07%    pad
13      Pneumonia   92.99%    pad
14   Pneumothorax  100.00%    pad
15  Consolidation   99.09%    pad
16          Edema   98.78%    pad
17            Any   76.22%    pad
18     No Finding   89.84%    cxp
19    Atelectasis   83.81%    cxp
20   Cardiomegaly   88.10%    cxp
21       Effusion   67.14%    cxp
22      Pneumonia   98.10%    cxp
23   Pneumothorax   91.27%    cxp
24  Consolidation   93.17%    cxp
25          Edema   74.76%    cxp
```

```
26             Any    54.44%    cxp
```

NIH only

In [145]:
```
print("Model trained on NIH")
model_NIH = EmbModel(emb_type="densenet121", feature_size_override=102
model_NIH.load_state_dict(torch.load("model/balanced/model-snapshot-er
model_NIH.eval()
```

```
1024
```

Out[145]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
ue, track_running_stats=True)
          (relu1): ReLU(inplace=True)
          (conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=T
```

In [146]:
```
data_loader_NIH = datasets[get_dataset_index("nih")]["val"]["loader"]
nih_predictions, nih_probabilities, nih_labels = predict(model_NIH, da

print('predictions', nih_predictions)
print('labels', nih_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 3
Batch number: 1 of 3
Batch number: 2 of 3
Batch number: 3 of 3
Done.
Prediction took: 0:00:07.337274
predictions [[False False False ... False False False]
 [ True False False ... False False False]
 [ True False False ... False False False]
 ...
 [ True False False ... False False False]
 [False False False ... False False False]
 [ True False False ... False False False]]
labels [[0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
Done
```

In [147]:
```
nih_report, nih_accuracy = calculate_stats(nih_predictions, nih_probab
```

```
Overall Accuracy: 47.95%
Classification Report:
               precision    recall  f1-score  support source
No Finding      0.594697  0.813472  0.687090    193.0    nih
Atelectasis     0.000000  0.000000  0.000000     34.0    nih
Cardiomegaly    0.000000  0.000000  0.000000      9.0    nih
Effusion        0.600000  0.061224  0.111111     49.0    nih
Pneumonia       0.000000  0.000000  0.000000      2.0    nih
Pneumothorax    0.000000  0.000000  0.000000      6.0    nih
Consolidation   0.000000  0.000000  0.000000     23.0    nih
Edema           0.000000  0.000000  0.000000      9.0    nih
Any             0.500000  0.037736  0.070175    106.0    nih
micro avg       0.585714  0.380510  0.461322    431.0    nih
macro avg       0.188300  0.101381  0.096486    431.0    nih
weighted avg    0.457486  0.380510  0.337567    431.0    nih
samples avg     0.435160  0.439726  0.436438    431.0    nih
           Label Accuracy source
0     No Finding   60.82%    nih
1    Atelectasis   90.14%    nih
2   Cardiomegaly   97.53%    nih
3       Effusion   86.85%    nih
4      Pneumonia   99.45%    nih
5   Pneumothorax   98.36%    nih
6  Consolidation   93.42%    nih
7          Edema   97.53%    nih
8            Any   70.96%    nih
Worst Performing Label: No Finding with an accuracy of 60.82%
```

In [148]:
```
combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
    precision    recall  f1-score  support source
0    0.715190  0.436293  0.541966    259.0  mimic
1    0.269231  0.088050  0.132701    159.0  mimic
2    0.153846  0.016393  0.029630    122.0  mimic
3    0.471910  0.280000  0.351464    150.0  mimic
4    0.000000  0.000000  0.000000     42.0  mimic
5    0.000000  0.000000  0.000000     29.0  mimic
6    0.000000  0.000000  0.000000     36.0  mimic
7    0.156250  0.070423  0.097087     71.0  mimic
8    0.701389  0.274457  0.394531    368.0  mimic
9    0.561866  0.224110  0.320416   1236.0  mimic
10   0.274202  0.129513  0.171931   1236.0  mimic
11   0.474759  0.224110  0.299259   1236.0  mimic
12   0.248288  0.231509  0.235901   1236.0  mimic
13   0.656250  0.338710  0.446809    124.0    pad
14   0.000000  0.000000  0.000000     16.0    pad
15   0.000000  0.000000  0.000000     30.0    pad
```

```
16   0.200000  0.043478  0.071429     23.0    pad
17   0.000000  0.000000  0.000000     19.0    pad
18   0.000000  0.000000  0.000000      0.0    pad
19   0.000000  0.000000  0.000000      3.0    pad
20   0.000000  0.000000  0.000000      3.0    pad
21   0.416667  0.065789  0.113636     76.0    pad
22   0.533333  0.163265  0.250000    294.0    pad
23   0.141435  0.049775  0.070208    294.0    pad
24   0.400142  0.163265  0.223413    294.0    pad
25   0.135671  0.137195  0.136179    294.0    pad
26   0.000000  0.000000  0.000000     60.0    cxp
27   0.000000  0.000000  0.000000     97.0    cxp
28   0.000000  0.000000  0.000000     74.0    cxp
29   0.631016  0.460938  0.532731    256.0    cxp
30   0.000000  0.000000  0.000000     12.0    cxp
31   0.000000  0.000000  0.000000     55.0    cxp
32   0.000000  0.000000  0.000000     43.0    cxp
33   0.432203  0.356643  0.390805    143.0    cxp
34   0.836207  0.437923  0.574815    443.0    cxp
35   0.663620  0.306847  0.419653   1183.0    cxp
36   0.211047  0.139500  0.166483   1183.0    cxp
37   0.501931  0.306847  0.377775   1183.0    cxp
38   0.248677  0.221534  0.228012   1183.0    cxp
39   0.594697  0.813472  0.687090    193.0    nih
40   0.000000  0.000000  0.000000     34.0    nih
41   0.000000  0.000000  0.000000      9.0    nih
42   0.600000  0.061224  0.111111     49.0    nih
43   0.000000  0.000000  0.000000      2.0    nih
44   0.000000  0.000000  0.000000      6.0    nih
45   0.000000  0.000000  0.000000     23.0    nih
46   0.000000  0.000000  0.000000      9.0    nih
47   0.500000  0.037736  0.070175    106.0    nih
48   0.585714  0.380510  0.461322    431.0    nih
49   0.188300  0.101381  0.096486    431.0    nih
50   0.457486  0.380510  0.337567    431.0    nih
51   0.435160  0.439726  0.436438    431.0    nih
            Label Accuracy source
0      No Finding   74.19%  mimic
1     Atelectasis   75.27%  mimic
2    Cardiomegaly   82.30%  mimic
3        Effusion   79.05%  mimic
4       Pneumonia   93.65%  mimic
5    Pneumothorax   96.08%  mimic
6   Consolidation   95.14%  mimic
7           Edema   87.43%  mimic
8             Any   58.11%  mimic
9      No Finding   68.29%    pad
10    Atelectasis   94.21%    pad
11   Cardiomegaly   90.55%    pad
12       Effusion   92.07%    pad
```

```
13      Pneumonia   92.99%    pad
14   Pneumothorax  100.00%    pad
15  Consolidation   99.09%    pad
16          Edema   98.78%    pad
17            Any   76.22%    pad
18     No Finding   89.84%    cxp
19    Atelectasis   83.81%    cxp
20   Cardiomegaly   88.10%    cxp
21       Effusion   67.14%    cxp
22      Pneumonia   98.10%    cxp
23   Pneumothorax   91.27%    cxp
24  Consolidation   93.17%    cxp
25          Edema   74.76%    cxp
26            Any   54.44%    cxp
27     No Finding   60.82%    nih
28    Atelectasis   90.14%    nih
29   Cardiomegaly   97.53%    nih
30       Effusion   86.85%    nih
31      Pneumonia   99.45%    nih
32   Pneumothorax   98.36%    nih
33  Consolidation   93.42%    nih
34          Edema   97.53%    nih
35            Any   70.96%    nih
```

CXP and NIH

In [153]:
```
print("Loading a model trained on both CXP and NIH")

model_CXP_NIH = EmbModel(emb_type="densenet121", feature_size_override
model_CXP_NIH.load_state_dict(torch.load("model/balanced/model-snapsho
model_CXP_NIH.eval()
```

```
Loading a model trained on both CXP and NIH

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[153]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
```

In [154]:
```
data_loader_CXP_NIH = datasets[get_dataset_index("cxp_nih")]["val"]["l
cxp_nih_predictions, cxp_nih_probabilities, cxp_nih_labels = predict(m

print('predictions', cxp_nih_predictions)
print('labels', cxp_nih_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 5
Batch number: 1 of 5
Batch number: 2 of 5
Batch number: 3 of 5
Batch number: 4 of 5
Batch number: 5 of 5
Done.
Prediction took: 0:00:05.094706
predictions [[False False False ... False False False]
 [False False False ... False  True  True]
 [False False False ... False False False]
 ...
 [False False False ... False False  True]
 [False False False ... False False False]
 [False False False ... False False False]]
labels [[0. 1. 0. ... 0. 1. 1.]
 [0. 0. 1. ... 0. 0. 1.]
 [0. 1. 1. ... 0. 0. 1.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]]
Done
```

In [155]:
```
cxp_nih_report, cxp_nih_accuracy = calculate_stats(cxp_nih_predictions
```

```
Overall Accuracy: 24.29%
Classification Report:
               precision    recall  f1-score  support   source
No Finding      0.000000  0.000000  0.000000     60.0  cxp+nih
Atelectasis     0.000000  0.000000  0.000000     97.0  cxp+nih
Cardiomegaly    0.000000  0.000000  0.000000     74.0  cxp+nih
Effusion        0.600000  0.539062  0.567901    256.0  cxp+nih
Pneumonia       0.000000  0.000000  0.000000     12.0  cxp+nih
Pneumothorax    0.000000  0.000000  0.000000     55.0  cxp+nih
Consolidation   0.000000  0.000000  0.000000     43.0  cxp+nih
Edema           0.433962  0.321678  0.369478    143.0  cxp+nih
Any             0.843750  0.487585  0.618026    443.0  cxp+nih
micro avg       0.662252  0.338123  0.447678   1183.0  cxp+nih
macro avg       0.208635  0.149814  0.172823   1183.0  cxp+nih
weighted avg    0.498257  0.338123  0.398989   1183.0  cxp+nih
samples avg     0.270503  0.247460  0.251977   1183.0  cxp+nih
           Label Accuracy   source
0     No Finding   90.32%  cxp+nih
1    Atelectasis   84.44%  cxp+nih
```

In [156]:
```
combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
    precision    recall  f1-score  support   source
0    0.715190  0.436293  0.541966    259.0    mimic
1    0.269231  0.088050  0.132701    159.0    mimic
2    0.153846  0.016393  0.029630    122.0    mimic
3    0.471910  0.280000  0.351464    150.0    mimic
4    0.000000  0.000000  0.000000     42.0    mimic
..        ...       ...       ...      ...      ...
60   0.843750  0.487585  0.618026    443.0  cxp+nih
61   0.662252  0.338123  0.447678   1183.0  cxp+nih
62   0.208635  0.149814  0.172823   1183.0  cxp+nih
63   0.498257  0.338123  0.398989   1183.0  cxp+nih
64   0.270503  0.247460  0.251977   1183.0  cxp+nih

[65 rows x 5 columns]
            Label Accuracy   source
0      No Finding   74.19%    mimic
1     Atelectasis   75.27%    mimic
2    Cardiomegaly   82.30%    mimic
3        Effusion   79.05%    mimic
4       Pneumonia   93.65%    mimic
5    Pneumothorax   96.08%    mimic
6   Consolidation   95.14%    mimic
7           Edema   87.43%    mimic
8             Any   58.11%    mimic
9      No Finding   68.29%      pad
```

```
10     Atelectasis   94.21%      pad
11    Cardiomegaly   90.55%      pad
12        Effusion   92.07%      pad
13       Pneumonia   92.99%      pad
14    Pneumothorax  100.00%      pad
15   Consolidation   99.09%      pad
16           Edema   98.78%      pad
17             Any   76.22%      pad
18      No Finding   89.84%      cxp
19     Atelectasis   83.81%      cxp
20    Cardiomegaly   88.10%      cxp
21        Effusion   67.14%      cxp
22       Pneumonia   98.10%      cxp
23    Pneumothorax   91.27%      cxp
24   Consolidation   93.17%      cxp
25           Edema   74.76%      cxp
26             Any   54.44%      cxp
27      No Finding   60.82%      nih
28     Atelectasis   90.14%      nih
29    Cardiomegaly   97.53%      nih
30        Effusion   86.85%      nih
31       Pneumonia   99.45%      nih
32    Pneumothorax   98.36%      nih
33   Consolidation   93.42%      nih
34           Edema   97.53%      nih
35             Any   70.96%      nih
36      No Finding   90.32%  cxp+nih
37     Atelectasis   84.44%  cxp+nih
38    Cardiomegaly   87.14%  cxp+nih
39        Effusion   66.67%  cxp+nih
40       Pneumonia   98.10%  cxp+nih
41    Pneumothorax   90.79%  cxp+nih
42   Consolidation   93.17%  cxp+nih
43           Edema   75.08%  cxp+nih
44             Any   57.62%  cxp+nih
```

In [178]:
```
# nih_cxp_report, nih_cxp_accuracy = calculate_stats(cxp_nih_predictic
# combined_accuracy, combined_report = combine_with_existing(combined_
```

CXP and PAD

In [157]:
```
print("Loading a model trained on both CXP and PAD")

model_CXP_PAD = EmbModel(emb_type="densenet121", feature_size_override
model_CXP_PAD.load_state_dict(torch.load("model/balanced/model-snapsho
model_CXP_PAD.eval()
```

```
Loading a model trained on both CXP and PAD

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[157]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
In [158]: data_loader_CXP_PAD = datasets[get_dataset_index("cxp_pad")]["val"]["l
cxp_pad_predictions, cxp_pad_probabilities, cxp_pad_labels = predict(m

print('predictions', cxp_pad_predictions)
print('labels', cxp_pad_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 5
Batch number: 1 of 5
Batch number: 2 of 5
Batch number: 3 of 5
Batch number: 4 of 5
Batch number: 5 of 5
Done.
Prediction took: 0:00:05.910238
predictions [[False False False ... False False  True]
 [False False False ... False  True  True]
 [False False False ... False False False]
 ...
 [False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False  True]]
labels [[0. 0. 0. ... 0. 1. 1.]
 [0. 0. 0. ... 0. 1. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]]
Done
```

```
In [159]: cxp_pad_report, cxp_pad_accuracy = calculate_stats(cxp_pad_predictions
```

```
Overall Accuracy: 24.44%
Classification Report:
               precision    recall  f1-score  support   source
No Finding      1.000000  0.016667  0.032787     60.0  cxp+pad
Atelectasis     0.000000  0.000000  0.000000     97.0  cxp+pad
Cardiomegaly    0.000000  0.000000  0.000000     74.0  cxp+pad
Effusion        0.630682  0.433594  0.513889    256.0  cxp+pad
Pneumonia       0.000000  0.000000  0.000000     12.0  cxp+pad
Pneumothorax    0.000000  0.000000  0.000000     55.0  cxp+pad
Consolidation   0.000000  0.000000  0.000000     43.0  cxp+pad
Edema           0.465347  0.328671  0.385246    143.0  cxp+pad
Any             0.842593  0.410835  0.552352    443.0  cxp+pad
micro avg       0.687500  0.288250  0.406194   1183.0  cxp+pad
macro avg       0.326513  0.132196  0.164919   1183.0  cxp+pad
weighted avg    0.558975  0.288250  0.366276   1183.0  cxp+pad
samples avg     0.237831  0.208757  0.217861   1183.0  cxp+pad
           Label Accuracy   source
0     No Finding   90.63%  cxp+pad
1    Atelectasis   84.44%  cxp+pad
2   Cardiomegaly   88.10%  cxp+pad
3       Effusion   66.67%  cxp+pad
4      Pneumonia   98.10%  cxp+pad
5   Pneumothorax   91.27%  cxp+pad
6  Consolidation   93.17%  cxp+pad
7          Edema   76.19%  cxp+pad
8            Any   53.17%  cxp+pad
Worst Performing Label: Any with an accuracy of 53.17%
```

```
In [52]: combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
    precision    recall  f1-score  support   source
0    0.715190  0.436293  0.541966    259.0    mimic
1    0.269231  0.088050  0.132701    159.0    mimic
2    0.153846  0.016393  0.029630    122.0    mimic
3    0.471910  0.280000  0.351464    150.0    mimic
4    0.000000  0.000000  0.000000     42.0    mimic
..        ...       ...       ...      ...      ...
73   0.841410  0.368015  0.512064    519.0  cxp+pad
74   0.679702  0.247123  0.362463   1477.0  cxp+pad
75   0.342983  0.131250  0.171620   1477.0  cxp+pad
76   0.581334  0.247123  0.328035   1477.0  cxp+pad
77   0.174322  0.156072  0.161537   1477.0  cxp+pad

[78 rows x 5 columns]
            Label Accuracy   source
0      No Finding   74.19%    mimic
1     Atelectasis   75.27%    mimic
```

```
2    Cardiomegaly   82.30%    mimic
3        Effusion   79.05%    mimic
4       Pneumonia   93.65%    mimic
5    Pneumothorax   96.08%    mimic
6   Consolidation   95.14%    mimic
7           Edema   87.43%    mimic
8             Any   58.11%    mimic
9      No Finding   68.29%      pad
10    Atelectasis   94.21%      pad
11   Cardiomegaly   90.55%      pad
12       Effusion   92.07%      pad
13      Pneumonia   92.99%      pad
14   Pneumothorax  100.00%      pad
15  Consolidation   99.09%      pad
16          Edema   98.78%      pad
17            Any   76.22%      pad
18     No Finding   89.84%      cxp
19    Atelectasis   83.81%      cxp
20   Cardiomegaly   88.10%      cxp
21       Effusion   67.14%      cxp
22      Pneumonia   98.10%      cxp
23   Pneumothorax   91.27%      cxp
24  Consolidation   93.17%      cxp
25          Edema   74.76%      cxp
26            Any   54.44%      cxp
27     No Finding   61.10%      nih
28    Atelectasis   90.14%      nih
29   Cardiomegaly   97.53%      nih
30       Effusion   86.85%      nih
31      Pneumonia   99.45%      nih
32   Pneumothorax   98.36%      nih
33  Consolidation   93.42%      nih
34          Edema   97.53%      nih
35            Any   70.96%      nih
36     No Finding   79.70%  cxp+nih
37    Atelectasis   86.73%  cxp+nih
38   Cardiomegaly   90.95%  cxp+nih
39       Effusion   73.57%  cxp+nih
40      Pneumonia   98.59%  cxp+nih
41   Pneumothorax   93.57%  cxp+nih
42  Consolidation   93.27%  cxp+nih
43          Edema   83.32%  cxp+nih
44            Any   62.51%  cxp+nih
45     No Finding   81.73%  cxp+pad
46    Atelectasis   88.00%  cxp+pad
47   Cardiomegaly   89.04%  cxp+pad
48       Effusion   75.26%  cxp+pad
49      Pneumonia   96.45%  cxp+pad
50   Pneumothorax   94.26%  cxp+pad
51  Consolidation   95.20%  cxp+pad
```

```
52          Edema   84.03%  cxp+pad
53            Any   62.00%  cxp+pad
```

In [181]:
```
# pad_cxp_report, pad_cxp_accuracy = calculate_stats(cxp_pad_predictic
# combined_accuracy, combined_report = combine_with_existing(combined_
```

NIH and PAD

In [160]:
```
print("Loading a model trained on both NIH and PAD")

model_NIH_PAD = EmbModel(emb_type="densenet121", feature_size_override
model_NIH_PAD.load_state_dict(torch.load("model/balanced/model-snapshc
model_NIH_PAD.eval()
```

```
Loading a model trained on both NIH and PAD

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[160]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
In [161]: data_loader_NIH_PAD = datasets[get_dataset_index("nih_pad")]["val"]["l
nih_pad_predictions, nih_pad_probabilities, nih_pad_labels = predict(m

print('predictions', nih_pad_predictions)
print('labels', nih_pad_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 3
Batch number: 1 of 3
Batch number: 2 of 3
Batch number: 3 of 3
Done.
Prediction took: 0:00:07.671303
predictions [[False False False ... False False False]
 [False False False ... False False False]
 [ True False False ... False False False]
 ...
 [False False False ... False False  True]
 [ True False False ... False False False]
 [ True False False ... False False False]]
labels [[0. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
Done
```

In [162]:
```
nih_pad_report, nih_pad_accuracy = calculate_stats(nih_pad_predictions
```

```
Overall Accuracy: 45.48%
Classification Report:
               precision    recall  f1-score  support   source
No Finding      0.605809  0.756477  0.672811    193.0  nih+pad
Atelectasis     0.000000  0.000000  0.000000     34.0  nih+pad
Cardiomegaly    0.000000  0.000000  0.000000      9.0  nih+pad
Effusion        0.500000  0.020408  0.039216     49.0  nih+pad
Pneumonia       0.000000  0.000000  0.000000      2.0  nih+pad
Pneumothorax    0.000000  0.000000  0.000000      6.0  nih+pad
Consolidation   0.000000  0.000000  0.000000     23.0  nih+pad
Edema           0.000000  0.000000  0.000000      9.0  nih+pad
Any             0.666667  0.018868  0.036697    106.0  nih+pad
micro avg       0.603239  0.345708  0.439528    431.0  nih+pad
macro avg       0.196942  0.088417  0.083192    431.0  nih+pad
weighted avg    0.492083  0.345708  0.314766    431.0  nih+pad
samples avg     0.403653  0.404110  0.403836    431.0  nih+pad
           Label Accuracy   source
0     No Finding   61.10%  nih+pad
1    Atelectasis   90.68%  nih+pad
2   Cardiomegaly   97.53%  nih+pad
3       Effusion   86.58%  nih+pad
4      Pneumonia   99.45%  nih+pad
5   Pneumothorax   98.36%  nih+pad
6  Consolidation   93.42%  nih+pad
7          Edema   97.53%  nih+pad
8            Any   71.23%  nih+pad
Worst Performing Label: No Finding with an accuracy of 61.10%
```

In [163]:
```
combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
    precision    recall  f1-score  support   source
0    0.715190  0.436293  0.541966    259.0    mimic
1    0.269231  0.088050  0.132701    159.0    mimic
2    0.153846  0.016393  0.029630    122.0    mimic
3    0.471910  0.280000  0.351464    150.0    mimic
4    0.000000  0.000000  0.000000     42.0    mimic
..        ...       ...       ...      ...      ...
73   0.666667  0.018868  0.036697    106.0  nih+pad
74   0.603239  0.345708  0.439528    431.0  nih+pad
75   0.196942  0.088417  0.083192    431.0  nih+pad
76   0.492083  0.345708  0.314766    431.0  nih+pad
77   0.403653  0.404110  0.403836    431.0  nih+pad

[78 rows x 5 columns]
           Label Accuracy   source
0      No Finding   74.19%    mimic
1     Atelectasis   75.27%    mimic
```

```
2    Cardiomegaly   82.30%    mimic
3        Effusion   79.05%    mimic
4       Pneumonia   93.65%    mimic
5    Pneumothorax   96.08%    mimic
6   Consolidation   95.14%    mimic
7           Edema   87.43%    mimic
8             Any   58.11%    mimic
9      No Finding   68.29%      pad
10    Atelectasis   94.21%      pad
11   Cardiomegaly   90.55%      pad
12       Effusion   92.07%      pad
13      Pneumonia   92.99%      pad
14   Pneumothorax  100.00%      pad
15  Consolidation   99.09%      pad
16          Edema   98.78%      pad
17            Any   76.22%      pad
18     No Finding   89.84%      cxp
19    Atelectasis   83.81%      cxp
20   Cardiomegaly   88.10%      cxp
21       Effusion   67.14%      cxp
22      Pneumonia   98.10%      cxp
23   Pneumothorax   91.27%      cxp
24  Consolidation   93.17%      cxp
25          Edema   74.76%      cxp
26            Any   54.44%      cxp
27     No Finding   60.82%      nih
28    Atelectasis   90.14%      nih
29   Cardiomegaly   97.53%      nih
30       Effusion   86.85%      nih
31      Pneumonia   99.45%      nih
32   Pneumothorax   98.36%      nih
33  Consolidation   93.42%      nih
34          Edema   97.53%      nih
35            Any   70.96%      nih
36     No Finding   90.32%  cxp+nih
37    Atelectasis   84.44%  cxp+nih
38   Cardiomegaly   87.14%  cxp+nih
39       Effusion   66.67%  cxp+nih
40      Pneumonia   98.10%  cxp+nih
41   Pneumothorax   90.79%  cxp+nih
42  Consolidation   93.17%  cxp+nih
43          Edema   75.08%  cxp+nih
44            Any   57.62%  cxp+nih
45     No Finding   61.10%  nih+pad
46    Atelectasis   90.68%  nih+pad
47   Cardiomegaly   97.53%  nih+pad
48       Effusion   86.58%  nih+pad
49      Pneumonia   99.45%  nih+pad
50   Pneumothorax   98.36%  nih+pad
51  Consolidation   93.42%  nih+pad
```

```
52          Edema   97.53%  nih+pad
53            Any   71.23%  nih+pad
```

In [184]:
```
# pad_nih_report, pad_nih_accuracy = calculate_stats(nih_pad_predictio
# combined_accuracy, combined_report = combine_with_existing(combined_
```

MIMIC and PAD

In [164]:
```
print("Loading a model trained on both MIMIC and PAD")

model_MIMIC_PAD = EmbModel(emb_type="densenet121", feature_size_overri
model_MIMIC_PAD.load_state_dict(torch.load("model/balanced/model-snaps
model_MIMIC_PAD.eval()
```

```
Loading a model trained on both MIMIC and PAD

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[164]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
```

In [165]:
```
data_loader_MIMIC_PAD = datasets[get_dataset_index("mimic_pad")]["val"
mimic_pad_predictions, mimic_pad_probabilities, mimic_pad_labels = pre

print('predictions', mimic_pad_predictions)
print('labels', mimic_pad_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 6
Batch number: 1 of 6
Batch number: 2 of 6
Batch number: 3 of 6
Batch number: 4 of 6
Batch number: 5 of 6
Batch number: 6 of 6
Done.
Prediction took: 0:00:37.308962
predictions [[ True False False ... False False False]
 [ True False False ... False False False]
 [ True False False ... False False False]
 ...
 [False False  True ... False False  True]
 [ True False False ... False False False]
 [False False False ... False False  True]]
labels [[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 1. ... 0. 1. 1.]]
Done
```

In [166]:
```
mimic_pad_report, mimic_pad_accuracy = calculate_stats(mimic_pad_predi
```

```
Overall Accuracy: 31.35%
Classification Report:
               precision    recall  f1-score  support     source
No Finding      0.681223  0.602317  0.639344    259.0  mimic+pad
Atelectasis     0.000000  0.000000  0.000000    159.0  mimic+pad
Cardiomegaly    0.250000  0.049180  0.082192    122.0  mimic+pad
Effusion        0.513274  0.386667  0.441065    150.0  mimic+pad
Pneumonia       0.000000  0.000000  0.000000     42.0  mimic+pad
Pneumothorax    0.000000  0.000000  0.000000     29.0  mimic+pad
Consolidation   0.000000  0.000000  0.000000     36.0  mimic+pad
Edema           0.294118  0.070423  0.113636     71.0  mimic+pad
Any             0.785185  0.288043  0.421471    368.0  mimic+pad
micro avg       0.618692  0.267799  0.373800   1236.0  mimic+pad
macro avg       0.280422  0.155181  0.188634   1236.0  mimic+pad
weighted avg    0.480387  0.267799  0.327627   1236.0  mimic+pad
samples avg     0.316892  0.294234  0.300727   1236.0  mimic+pad
           Label Accuracy     source
0     No Finding   76.22%  mimic+pad
1    Atelectasis   76.22%  mimic+pad
2   Cardiomegaly   81.89%  mimic+pad
3       Effusion   80.14%  mimic+pad
4      Pneumonia   94.32%  mimic+pad
5   Pneumothorax   96.08%  mimic+pad
6  Consolidation   95.14%  mimic+pad
7          Edema   89.46%  mimic+pad
8            Any   60.68%  mimic+pad
Worst Performing Label: Any with an accuracy of 60.68%
```

In [167]:
```
combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
    precision    recall  f1-score  support     source
0    0.715190  0.436293  0.541966    259.0      mimic
1    0.269231  0.088050  0.132701    159.0      mimic
2    0.153846  0.016393  0.029630    122.0      mimic
3    0.471910  0.280000  0.351464    150.0      mimic
4    0.000000  0.000000  0.000000     42.0      mimic
..        ...       ...       ...      ...        ...
86   0.785185  0.288043  0.421471    368.0  mimic+pad
87   0.618692  0.267799  0.373800   1236.0  mimic+pad
88   0.280422  0.155181  0.188634   1236.0  mimic+pad
89   0.480387  0.267799  0.327627   1236.0  mimic+pad
90   0.316892  0.294234  0.300727   1236.0  mimic+pad

[91 rows x 5 columns]
            Label Accuracy     source
0      No Finding   74.19%      mimic
1     Atelectasis   75.27%      mimic
2    Cardiomegaly   82.30%      mimic
3        Effusion   79.05%      mimic
4       Pneumonia   93.65%      mimic
..            ...      ...        ...
58      Pneumonia   94.32%  mimic+pad
59   Pneumothorax   96.08%  mimic+pad
60  Consolidation   95.14%  mimic+pad
61          Edema   89.46%  mimic+pad
62            Any   60.68%  mimic+pad

[63 rows x 3 columns]
```

In [187]:
```
# pad_mimic_report, pad_mimic_accuracy = calculate_stats(mimic_pad_pre
# combined_accuracy, combined_report = combine_with_existing(combined_
```

MIMIC and NIH

In [168]:
```
print("Loading a model trained on both MIMIC and NIH")

model_MIMIC_NIH = EmbModel(emb_type="densenet121", feature_size_overri
model_MIMIC_NIH.load_state_dict(torch.load("model/balanced/model-snaps
model_MIMIC_NIH.eval()
```

```
Loading a model trained on both MIMIC and NIH

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[168]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
```

In [169]:
```
data_loader_MIMIC_NIH = datasets[get_dataset_index("mimic_nih")]["val"
mimic_nih_predictions, mimic_nih_probabilities, mimic_nih_labels = pre

print('predictions', mimic_nih_predictions)
print('labels', mimic_nih_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 6
Batch number: 1 of 6
Batch number: 2 of 6
Batch number: 3 of 6
Batch number: 4 of 6
Batch number: 5 of 6
Batch number: 6 of 6
Done.
Prediction took: 0:00:37.731515
predictions [[False  True False ... False False  True]
 [False False False ... False False False]
 [ True False False ... False False  True]
 ...
 [False False False ... False False False]
 [ True False False ... False False False]
 [False  True False ...  True False  True]]
labels [[0. 1. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 1. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 1. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 1. 1.]]
Done
```

```
In [170]: imic_nih_report, mimic_nih_accuracy = calculate_stats(mimic_nih_predic
```

```
Overall Accuracy: 26.49%
Classification Report:
               precision    recall  f1-score  support     source
No Finding      0.743243  0.424710  0.540541    259.0  mimic+nih
Atelectasis     0.314815  0.106918  0.159624    159.0  mimic+nih
Cardiomegaly    0.275862  0.065574  0.105960    122.0  mimic+nih
Effusion        0.384615  0.533333  0.446927    150.0  mimic+nih
Pneumonia       0.000000  0.000000  0.000000     42.0  mimic+nih
Pneumothorax    0.000000  0.000000  0.000000     29.0  mimic+nih
Consolidation   0.000000  0.000000  0.000000     36.0  mimic+nih
Edema           0.200000  0.056338  0.087912     71.0  mimic+nih
Any             0.727273  0.456522  0.560935    368.0  mimic+nih
micro avg       0.558442  0.313107  0.401244   1236.0  mimic+nih
macro avg       0.293979  0.182600  0.211322   1236.0  mimic+nih
weighted avg    0.498171  0.313107  0.370560   1236.0  mimic+nih
samples avg     0.307275  0.286059  0.290798   1236.0  mimic+nih
           Label Accuracy     source
0     No Finding   74.73%  mimic+nih
1    Atelectasis   75.81%  mimic+nih
2   Cardiomegaly   81.76%  mimic+nih
3       Effusion   73.24%  mimic+nih
4      Pneumonia   94.32%  mimic+nih
5   Pneumothorax   95.95%  mimic+nih
6  Consolidation   94.86%  mimic+nih
7          Edema   88.78%  mimic+nih
8            Any   64.46%  mimic+nih
Worst Performing Label: Any with an accuracy of 64.46%
```

In [171]:
```
combined_accuracy, combined_report = combine_with_existing(combined_acc
```

```
     precision    recall  f1-score  support     source
0     0.715190  0.436293  0.541966    259.0      mimic
1     0.269231  0.088050  0.132701    159.0      mimic
2     0.153846  0.016393  0.029630    122.0      mimic
3     0.471910  0.280000  0.351464    150.0      mimic
4     0.000000  0.000000  0.000000     42.0      mimic
..         ...       ...       ...      ...        ...
99    0.727273  0.456522  0.560935    368.0  mimic+nih
100   0.558442  0.313107  0.401244   1236.0  mimic+nih
101   0.293979  0.182600  0.211322   1236.0  mimic+nih
102   0.498171  0.313107  0.370560   1236.0  mimic+nih
103   0.307275  0.286059  0.290798   1236.0  mimic+nih

[104 rows x 5 columns]
            Label Accuracy     source
0      No Finding   74.19%      mimic
1     Atelectasis   75.27%      mimic
2    Cardiomegaly   82.30%      mimic
3        Effusion   79.05%      mimic
4       Pneumonia   93.65%      mimic
..            ...      ...        ...
67      Pneumonia   94.32%  mimic+nih
68   Pneumothorax   95.95%  mimic+nih
69  Consolidation   94.86%  mimic+nih
70          Edema   88.78%  mimic+nih
71            Any   64.46%  mimic+nih

[72 rows x 3 columns]
```

In [172]:
```
# nih_mimic_report, nih_mimic_accuracy = calculate_stats(mimic_nih_pred
# combined_accuracy, combined_report = combine_with_existing(combined_a
```

MIMIC and CXP

In [173]:
```
print("Loading a model trained on both MIMIC and CXP")

model_MIMIC_CXP = EmbModel(emb_type="densenet121", feature_size_overri
model_MIMIC_CXP.load_state_dict(torch.load("model/balanced/model-snaps
model_MIMIC_CXP.eval()
```

```
Loading a model trained on both MIMIC and CXP

Emb Dim:
1024
Manually setting output dim to 1024
1024
```

Out[173]:
```
EmbModel(
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
```

In [174]:
```
data_loader_MIMIC_CXP = datasets[get_dataset_index("mimic_cxp")]["val"
mimic_cxp_predictions, mimic_cxp_probabilities, mimic_cxp_labels = pre

print('predictions', mimic_cxp_predictions)
print('labels', mimic_cxp_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 6
Batch number: 1 of 6
Batch number: 2 of 6
Batch number: 3 of 6
Batch number: 4 of 6
Batch number: 5 of 6
Batch number: 6 of 6
Done.
Prediction took: 0:00:35.213949
predictions [[False False False ... False False  True]
 [False False False ... False False  True]
 [False False False ... False False  True]
 ...
 [ True False False ... False False False]
 [False False False ... False False False]
 [ True False False ... False False False]]
labels [[1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 1. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
Done
```

```
In [176]: mimic_cxp_report, mimic_cxp_accuracy = calculate_stats(mimic_cxp_predi
```

```
Overall Accuracy: 25.14%
Classification Report:
               precision    recall  f1-score  support     source
No Finding      0.741259  0.409266  0.527363    259.0  mimic+cxp
Atelectasis     0.200000  0.062893  0.095694    159.0  mimic+cxp
Cardiomegaly    0.214286  0.024590  0.044118    122.0  mimic+cxp
Effusion        0.450382  0.393333  0.419929    150.0  mimic+cxp
Pneumonia       0.333333  0.023810  0.044444     42.0  mimic+cxp
Pneumothorax    0.000000  0.000000  0.000000     29.0  mimic+cxp
Consolidation   0.000000  0.000000  0.000000     36.0  mimic+cxp
Edema           0.176471  0.042254  0.068182     71.0  mimic+cxp
Any             0.767857  0.350543  0.481343    368.0  mimic+cxp
micro avg       0.590133  0.251618  0.352808   1236.0  mimic+cxp
macro avg       0.320399  0.145188  0.186786   1236.0  mimic+cxp
weighted avg    0.506947  0.251618  0.326874   1236.0  mimic+cxp
samples avg     0.265518  0.244730  0.250898   1236.0  mimic+cxp
           Label Accuracy     source
0     No Finding   74.32%  mimic+cxp
1    Atelectasis   74.46%  mimic+cxp
2   Cardiomegaly   82.43%  mimic+cxp
3       Effusion   77.97%  mimic+cxp
4      Pneumonia   94.19%  mimic+cxp
5   Pneumothorax   96.08%  mimic+cxp
6  Consolidation   95.00%  mimic+cxp
7          Edema   88.92%  mimic+cxp
8            Any   62.43%  mimic+cxp
Worst Performing Label: Any with an accuracy of 62.43%
```

In [177]:
```
combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
     precision    recall  f1-score  support     source
0     0.715190  0.436293  0.541966    259.0      mimic
1     0.269231  0.088050  0.132701    159.0      mimic
2     0.153846  0.016393  0.029630    122.0      mimic
3     0.471910  0.280000  0.351464    150.0      mimic
4     0.000000  0.000000  0.000000     42.0      mimic
..         ...       ...       ...      ...        ...
112   0.767857  0.350543  0.481343    368.0  mimic+cxp
113   0.590133  0.251618  0.352808   1236.0  mimic+cxp
114   0.320399  0.145188  0.186786   1236.0  mimic+cxp
115   0.506947  0.251618  0.326874   1236.0  mimic+cxp
116   0.265518  0.244730  0.250898   1236.0  mimic+cxp

[117 rows x 5 columns]
            Label Accuracy     source
0      No Finding   74.19%      mimic
1     Atelectasis   75.27%      mimic
2    Cardiomegaly   82.30%      mimic
3        Effusion   79.05%      mimic
4       Pneumonia   93.65%      mimic
..            ...      ...        ...
76      Pneumonia   94.19%  mimic+cxp
77   Pneumothorax   96.08%  mimic+cxp
78  Consolidation   95.00%  mimic+cxp
79          Edema   88.92%  mimic+cxp
80            Any   62.43%  mimic+cxp

[81 rows x 3 columns]
```

In [192]:
```
# cxp_mimic_report, cxp_mimic_accuracy = calculate_stats(mimic_cxp_pre
# combined_accuracy, combined_report = combine_with_existing(combined_
```

All four datasets together: MIMIC, CXP, NIH and PAD

In [178]:
```
print("Loading a model trained on all four datasets: MIMIC, CXP, NIH a

model_CXP_MIMIC_NIH_PAD = EmbModel(emb_type="densenet121", feature_siz
model_CXP_MIMIC_NIH_PAD.load_state_dict(torch.load("model/balanced/moc
model_CXP_MIMIC_NIH_PAD.eval()
```

```
  (encoder): Sequential(
    (0): Sequential(
      (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddi
ng=(3, 3), bias=False)
      (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu0): ReLU(inplace=True)
      (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation
=1, ceil_mode=False)
      (denseblock1): _DenseBlock(
        (denselayer1): _DenseLayer(
          (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=Tr
ue, track_running_stats=True)
          (relu1): ReLU(inplace=True)
          (conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=T
rue, track_running_stats=True)
          (relu2): ReLU(inplace=True)
          (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
```

In [179]:
```
data_loader_CXP_MIMIC_NIH_PAD = datasets[get_dataset_index("cxp_mimic_
cxp_mimic_nih_pad_predictions, cxp_mimic_nih_pad_probabilities, cxp_mi

print('predictions', cxp_mimic_nih_pad_predictions)
print('labels', cxp_mimic_nih_pad_labels)
print("Done")
```

```
Started prediction validation
Number of batches: 17
Batch number: 1 of 17
Batch number: 2 of 17
Batch number: 3 of 17
Batch number: 4 of 17
Batch number: 5 of 17
Batch number: 6 of 17
Batch number: 7 of 17
Batch number: 8 of 17
Batch number: 9 of 17
Batch number: 10 of 17
Batch number: 11 of 17
Batch number: 12 of 17
Batch number: 13 of 17
Batch number: 14 of 17
Batch number: 15 of 17
Batch number: 16 of 17
Batch number: 17 of 17
Done.
Prediction took: 0:00:51.106543
predictions [[False False False ... False False False]
 [False False False ... False False  True]
 [ True False False ... False False False]
 ...
 [False False False ... False False False]
 [ True False False ... False False False]
 [False False False ... False False False]]
labels [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 1. ... 0. 0. 1.]]
Done
```

```
In [180]: cxp_mimic_nih_pad_report, cxp_mimic_nih_pad_accuracy = calculate_stats
```

```
Overall Accuracy: 33.11%
Classification Report:
               precision    recall  f1-score  support               sou
rce
No Finding      0.639155  0.523585  0.575627    636.0  cxp+mimic+nih+
pad
Atelectasis     0.297297  0.035948  0.064140    306.0  cxp+mimic+nih+
pad
Cardiomegaly    0.285714  0.008511  0.016529    235.0  cxp+mimic+nih+
pad
Effusion        0.577558  0.366109  0.448143    478.0  cxp+mimic+nih+
pad
Pneumonia       0.142857  0.040000  0.062500     75.0  cxp+mimic+nih+
pad
Pneumothorax    0.000000  0.000000  0.000000     90.0  cxp+mimic+nih+
pad
Consolidation   0.000000  0.000000  0.000000    105.0  cxp+mimic+nih+
pad
Edema           0.398551  0.243363  0.302198    226.0  cxp+mimic+nih+
pad
Any             0.798969  0.312185  0.448950    993.0  cxp+mimic+nih+
pad
micro avg       0.628269  0.282761  0.389998   3144.0  cxp+mimic+nih+
pad
macro avg       0.348900  0.169967  0.213121   3144.0  cxp+mimic+nih+
pad
weighted avg    0.551798  0.282761  0.357065   3144.0  cxp+mimic+nih+
pad
samples avg     0.276903  0.262579  0.266328   3144.0  cxp+mimic+nih+
pad
           Label Accuracy             source
0     No Finding   76.20%  cxp+mimic+nih+pad
1    Atelectasis   84.44%  cxp+mimic+nih+pad
2   Cardiomegaly   88.46%  cxp+mimic+nih+pad
3       Effusion   79.11%  cxp+mimic+nih+pad
4      Pneumonia   95.64%  cxp+mimic+nih+pad
5   Pneumothorax   95.64%  cxp+mimic+nih+pad
6  Consolidation   94.91%  cxp+mimic+nih+pad
7          Edema   87.69%  cxp+mimic+nih+pad
8            Any   63.11%  cxp+mimic+nih+pad
Worst Performing Label: Any with an accuracy of 63.11%
```

In [181]:
```
combined_accuracy, combined_report = combine_with_existing(combined_ac
```

```
     precision    recall  f1-score  support              source
0     0.715190  0.436293  0.541966    259.0               mimic
1     0.269231  0.088050  0.132701    159.0               mimic
2     0.153846  0.016393  0.029630    122.0               mimic
3     0.471910  0.280000  0.351464    150.0               mimic
4     0.000000  0.000000  0.000000     42.0               mimic
..         ...       ...       ...      ...                 ...
125   0.798969  0.312185  0.448950    993.0  cxp+mimic+nih+pad
126   0.628269  0.282761  0.389998   3144.0  cxp+mimic+nih+pad
127   0.348900  0.169967  0.213121   3144.0  cxp+mimic+nih+pad
128   0.551798  0.282761  0.357065   3144.0  cxp+mimic+nih+pad
129   0.276903  0.262579  0.266328   3144.0  cxp+mimic+nih+pad

[130 rows x 5 columns]
            Label Accuracy             source
0      No Finding   74.19%              mimic
1     Atelectasis   75.27%              mimic
2    Cardiomegaly   82.30%              mimic
3        Effusion   79.05%              mimic
4       Pneumonia   93.65%              mimic
..            ...      ...                ...
85      Pneumonia   95.64%  cxp+mimic+nih+pad
86   Pneumothorax   95.64%  cxp+mimic+nih+pad
87  Consolidation   94.91%  cxp+mimic+nih+pad
88          Edema   87.69%  cxp+mimic+nih+pad
89            Any   63.11%  cxp+mimic+nih+pad

[90 rows x 3 columns]
```

In [ ]:

Saving all data to data frame and to file

In [182]:
```
print("Saving the evaluation stats to CSV")
combined_df_now = datetime.now()
combined_df_dt_string = combined_df_now.strftime("%d-%m-%Y-%H-%M-%S")
combined_accuracy.to_csv(f"stats/{combined_df_dt_string}_combined_accu
combined_report.to_csv(f"stats/{combined_df_dt_string}_combined_accura
print("Done.")

print("File names:")
print(f"stats/{combined_df_dt_string}_combined_accuracy_breakdown.csv"
print(f"stats/{combined_df_dt_string}_combined_accuracy_totals.csv")
```

```
Saving the evaluation stats to CSV
Done.
File names:
stats/05-05-2024-23-11-28_combined_accuracy_breakdown.csv
stats/05-05-2024-23-11-28_combined_accuracy_totals.csv
```

Visualizing data

In [183]:

```
timestamp_to_load = "04-05-2024-23-34-18"
two_dataset_eval_timestamp = "04-05-2024-23-34-18"
one_dataset_eval_timestamp = "05-05-2024-16-23-37"
balanced_dataset_timestamp = "05-05-2024-22-03-40"
balanced_single_dataset_timestamp = "05-05-2024-23-11-28"

two_dataset_accuracy_breakdown_path = f"stats/{two_dataset_eval_timest
two_dataset_accuracy_totals_path = f"stats/{two_dataset_eval_timestamp

one_dataset_accuracy_breakdown_path = f"stats/{one_dataset_eval_timest
one_dataset_accuracy_totals_path = f"stats/{one_dataset_eval_timestamp

balanced_dataset_accuracy_breakdown_path = f"stats/{balanced_dataset_t
balanced_dataset_accuracy_totals_path = f"stats/{balanced_dataset_time

balanced_single_dataset_accuracy_breakdown_path = f"stats/{balanced_da
balanced_single_dataset_accuracy_totals_path = f"stats/{balanced_datas

# Loading the DataFrames
two_dataset_combined_accuracy = pd.read_csv(two_dataset_accuracy_break
two_dataset_combined_report = pd.read_csv(two_dataset_accuracy_totals_

one_dataset_combined_accuracy = pd.read_csv(one_dataset_accuracy_break
one_dataset_combined_report = pd.read_csv(one_dataset_accuracy_totals_

balanced_dataset_combined_accuracy = pd.read_csv(one_dataset_accuracy_
balanced_dataset_combined_report = pd.read_csv(one_dataset_accuracy_to

balanced_single_dataset_combined_accuracy = pd.read_csv(one_dataset_ac
balanced_single_dataset_combined_report = pd.read_csv(one_dataset_accu


print(two_dataset_combined_accuracy.head())
print(two_dataset_combined_report.head())

print(one_dataset_combined_accuracy.head())
print(one_dataset_combined_report.head())

print(balanced_dataset_combined_accuracy.head())
print(balanced_dataset_combined_report.head())

print(balanced_single_dataset_combined_accuracy.head())
print(balanced_single_dataset_combined_report.head())
```

```
   precision    recall  f1-score  support source
0   0.654206  0.540541  0.591966    259.0  mimic
1   0.142857  0.006289  0.012048    159.0  mimic
2   0.200000  0.024590  0.043796    122.0  mimic
3   0.432692  0.300000  0.354331    150.0  mimic
4   0.000000  0.000000  0.000000     42.0  mimic
```

```
           Label Accuracy source
0    No Finding   73.92%  mimic
1   Atelectasis   77.84%  mimic
2  Cardiomegaly   82.30%  mimic
3      Effusion   77.84%  mimic
4     Pneumonia   93.92%  mimic
   precision    recall  f1-score  support source
0   0.654206  0.540541  0.591966    259.0  mimic
1   0.142857  0.006289  0.012048    159.0  mimic
2   0.200000  0.024590  0.043796    122.0  mimic
3   0.432692  0.300000  0.354331    150.0  mimic
4   0.000000  0.000000  0.000000     42.0  mimic
           Label Accuracy source
0    No Finding   73.92%  mimic
1   Atelectasis   77.84%  mimic
2  Cardiomegaly   82.30%  mimic
3      Effusion   77.84%  mimic
4     Pneumonia   93.92%  mimic
   precision    recall  f1-score  support source
0   0.654206  0.540541  0.591966    259.0  mimic
1   0.142857  0.006289  0.012048    159.0  mimic
2   0.200000  0.024590  0.043796    122.0  mimic

3   0.432692  0.300000  0.354331    150.0  mimic
4   0.000000  0.000000  0.000000     42.0  mimic
           Label Accuracy source
0    No Finding   73.92%  mimic
1   Atelectasis   77.84%  mimic
2  Cardiomegaly   82.30%  mimic
3      Effusion   77.84%  mimic
4     Pneumonia   93.92%  mimic
   precision    recall  f1-score  support source
0   0.654206  0.540541  0.591966    259.0  mimic
1   0.142857  0.006289  0.012048    159.0  mimic
2   0.200000  0.024590  0.043796    122.0  mimic
3   0.432692  0.300000  0.354331    150.0  mimic
4   0.000000  0.000000  0.000000     42.0  mimic
           Label Accuracy source
0    No Finding   73.92%  mimic
1   Atelectasis   77.84%  mimic
2  Cardiomegaly   82.30%  mimic
3      Effusion   77.84%  mimic
4     Pneumonia   93.92%  mimic
```

In [184]:
```
def get_accuracy_df(report):
    report['Accuracy'] = report['Accuracy'].replace('%', '', regex=Tru

    baseline_accuracies = {}
    for source in report['source'].unique():
        if '+' not in source:  # Only single sources
            source_data = report[report['source'] == source]
            for label in source_data['Label'].unique():
                key = (source, label)
                baseline_accuracies[key] = source_data[source_data['La

    # Calculate the changes for combinations involving these sources
    accuracy_changes = []
    for source in report['source'].unique():
        if '+' in source:
            parts = source.split('+')
            source_data = report[report['source'] == source]
            for label in source_data['Label'].unique():
                current_acc = source_data[source_data['Label'] == labe
                for part in parts:
                    base_key = (part, label)
                    base_acc = baseline_accuracies.get(base_key, 0)  #
                    change = current_acc - base_acc
                    names_without_base = [x for x in parts if x != par
                    name_base_first = [part] + names_without_base
                    if len(parts) > 2 and not(source.startswith(part))
                        pass
                    else:
                        accuracy_changes.append({
                            'source_combination': "+".join(name_base_f
                            'part_source': part,
                            'label': label,
                            'change_in_accuracy': change
                        })

    # Create a DataFrame from the changes
    accuracy_changes_df = pd.DataFrame(accuracy_changes)
    print(accuracy_changes_df)

    return accuracy_changes_df
```

In [190]:

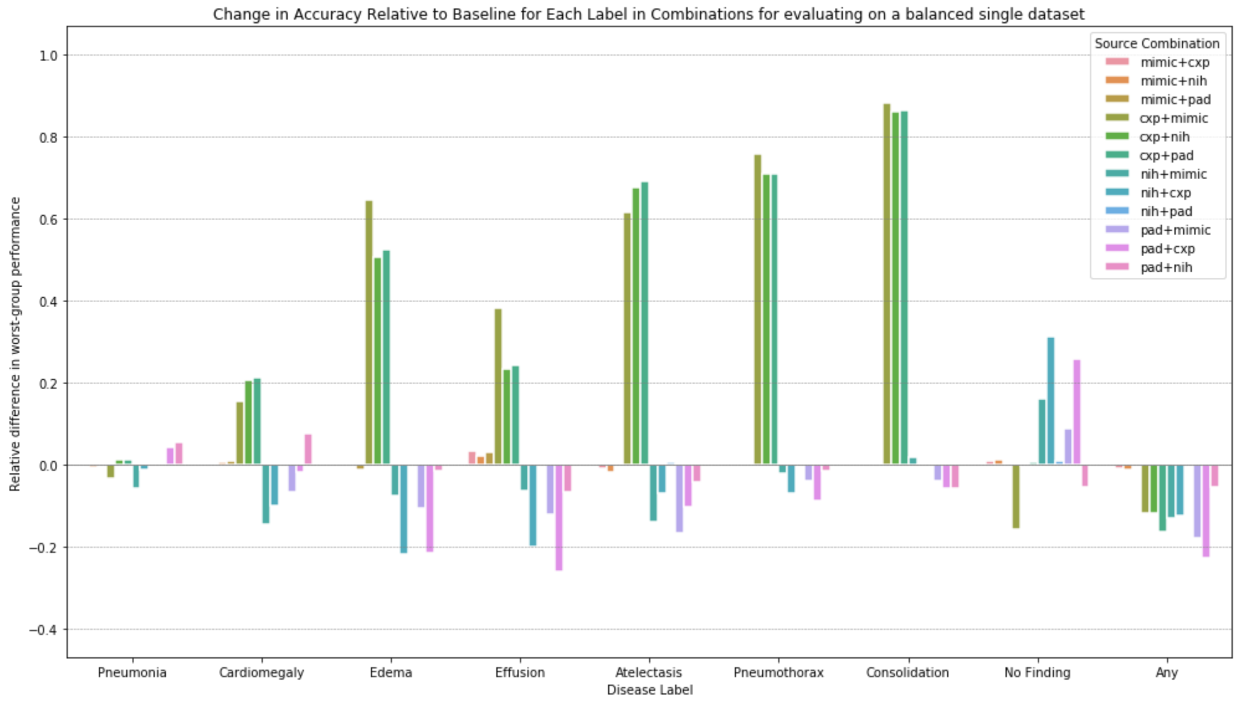
```
two_datasets_accuracy_changes_df = get_accuracy_df(two_dataset_combine

one_datasets_accuracy_changes_df = get_accuracy_df(one_dataset_combine

balanced_dataset_accuracy_changes_df = get_accuracy_df(balanced_datase

balanced_single_dataset_accuracy_changes_df = get_accuracy_df(balanced
```

```
    source_combination part_source          label  change_in_accuracy
0              cxp+nih         cxp     No Finding           -0.001169
1              nih+cxp         nih     No Finding            0.001989
2              cxp+nih         cxp    Atelectasis            0.007013
3              nih+cxp         nih    Atelectasis           -0.000461
4              cxp+nih         cxp   Cardiomegaly            0.002418
..                 ...         ...            ...                 ...
112  cxp+mimic+nih+pad         cxp      Pneumonia           -0.000071
113  cxp+mimic+nih+pad         cxp   Pneumothorax            0.007548
114  cxp+mimic+nih+pad         cxp  Consolidation            0.008808
115  cxp+mimic+nih+pad         cxp          Edema            0.006382
116  cxp+mimic+nih+pad         cxp            Any           -0.000871

[117 rows x 4 columns]
    source_combination part_source          label  change_in_accuracy
0              cxp+nih         cxp     No Finding           -0.000032
1              nih+cxp         nih     No Finding            0.003126
2              cxp+nih         cxp    Atelectasis            0.006777
3              nih+cxp         nih    Atelectasis           -0.000697
4              cxp+nih         cxp   Cardiomegaly            0.002048
..                 ...         ...            ...                 ...
112  cxp+mimic+nih+pad         cxp      Pneumonia           -0.000071
113  cxp+mimic+nih+pad         cxp   Pneumothorax            0.007548
114  cxp+mimic+nih+pad         cxp  Consolidation            0.008808
115  cxp+mimic+nih+pad         cxp          Edema            0.006382
116  cxp+mimic+nih+pad         cxp            Any           -0.000871

[117 rows x 4 columns]
    source_combination part_source          label  change_in_accuracy
0              cxp+nih         cxp     No Finding       -3.200000e-07
1              nih+cxp         nih     No Finding        3.126000e-05
2              cxp+nih         cxp    Atelectasis        6.777000e-05
3              nih+cxp         nih    Atelectasis       -6.970000e-06
4              cxp+nih         cxp   Cardiomegaly        2.048000e-05
..                 ...         ...            ...                 ...
112  cxp+mimic+nih+pad         cxp      Pneumonia       -7.100000e-07
113  cxp+mimic+nih+pad         cxp   Pneumothorax        7.548000e-05
114  cxp+mimic+nih+pad         cxp  Consolidation        8.808000e-05
115  cxp+mimic+nih+pad         cxp          Edema        6.382000e-05
116  cxp+mimic+nih+pad         cxp            Any       -8.710000e-06

[117 rows x 4 columns]
```

```
    source_combination part_source         label  change_in_accuracy
0              cxp+nih         cxp    No Finding             -0.0032
1              nih+cxp         nih    No Finding              0.3126
2              cxp+nih         cxp   Atelectasis              0.6777
3              nih+cxp         nih   Atelectasis             -0.0697
4              cxp+nih         cxp  Cardiomegaly              0.2048
..                 ...         ...           ...                 ...
112  cxp+mimic+nih+pad         cxp     Pneumonia             -0.0071
113  cxp+mimic+nih+pad         cxp  Pneumothorax              0.7548
114  cxp+mimic+nih+pad         cxp Consolidation              0.8808
115  cxp+mimic+nih+pad         cxp         Edema              0.6382
116  cxp+mimic+nih+pad         cxp           Any             -0.0871

[117 rows x 4 columns]
```

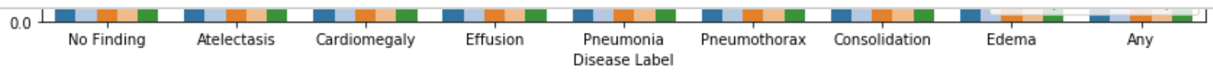
In [191]:
```
def plot_breakdown_accuracy_changes(accuracy_changes_df, chart_name=""
    accuracy_changes_df['log_change'] = np.sign(accuracy_changes_df['c

    plt.figure(figsize=(14, 8))
    # Using seaborn's barplot to visualize the data, now using 'label'
    label_order = ['Pneumonia', 'Cardiomegaly', 'Edema', 'Effusion', '
    source_order = [
        'mimic+cxp', 'mimic+nih', 'mimic+pad',
        'cxp+mimic', 'cxp+nih', 'cxp+pad',
        'nih+mimic', 'nih+cxp', 'nih+pad',
        'pad+mimic', 'pad+cxp', 'pad+nih'
    ]

    ax = sns.barplot(data=accuracy_changes_df, y='change_in_accuracy',
                linewidth=1.5)

    for y in ax.get_yticks():
        plt.axhline(y, color='gray', linewidth=0.5, linestyle='--')

    # Add a horizontal line at zero to indicate no change
    plt.axhline(0, color='gray', linewidth=0.8)

    plt.title(f'Change in Accuracy Relative to Baseline for Each Label
    plt.ylabel('Relative difference in worst-group performance')
    plt.xlabel('Disease Label')
    plt.xticks()
    plt.legend(title='Source Combination', loc='upper right')
    plt.tight_layout()
    plt.show()
```

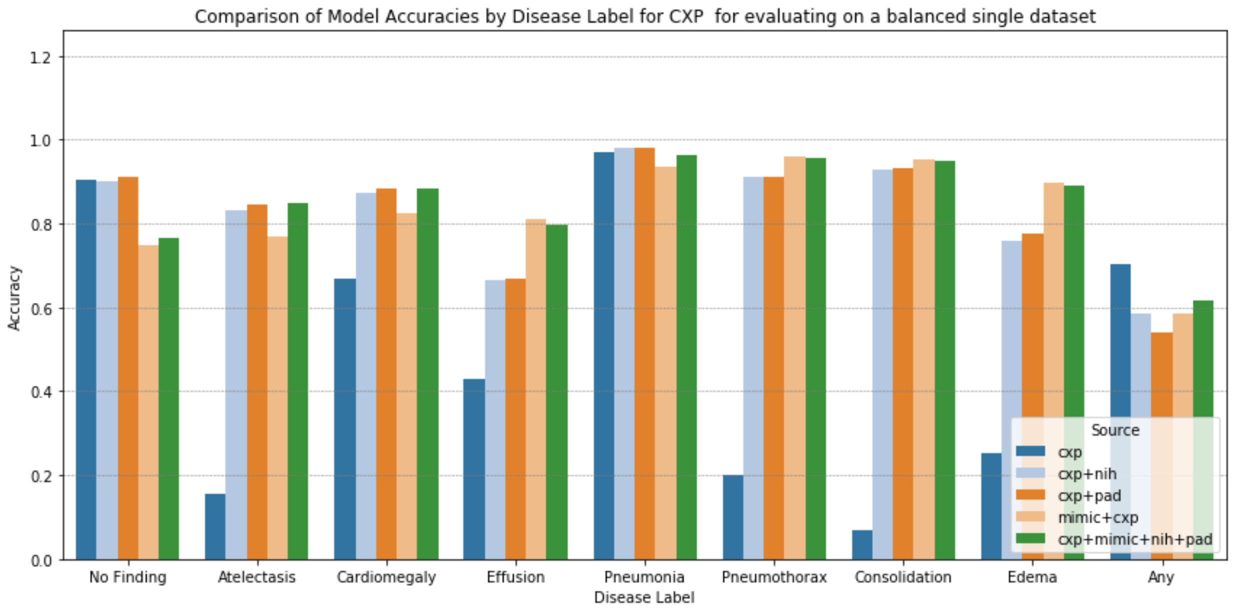
In [192]:
```
t_breakdown_accuracy_changes(two_datasets_accuracy_changes_df, "evalua
t_breakdown_accuracy_changes(one_datasets_accuracy_changes_df, "evalua
t_breakdown_accuracy_changes(balanced_dataset_accuracy_changes_df, "ev
t_breakdown_accuracy_changes(balanced_single_dataset_accuracy_changes_
```



In [193]:
```
def plot_accuracy_per_dataset(report, chart_name=""):
    dataset_names = ['nih', 'mimic', 'pad', 'cxp']

    for dataset_name in dataset_names:
        dataset_specific = report[report['source'].str.contains(datase

        sorted_sources = sorted(dataset_specific['source'].unique(), k
        dataset_specific = dataset_specific[dataset_specific['source']]

        plt.figure(figsize=(12, 6))

        ax = sns.barplot(data=dataset_specific, x='Label', y='Accuracy

        for y in ax.get_yticks():
            plt.axhline(y, color='gray', linewidth=0.5, linestyle='--'

        plt.title(f'Comparison of Model Accuracies by Disease Label fo
        plt.ylabel('Accuracy')
        plt.xlabel('Disease Label')
        plt.xticks()
        plt.legend(title='Source', loc='lower right')

        plt.tight_layout()

        # Show the plot
        plt.show()
```

In [194]:
```
plot_accuracy_per_dataset(two_dataset_combined_report, "evaluating on
plot_accuracy_per_dataset(one_dataset_combined_report, "evaluating on
plot_accuracy_per_dataset(balanced_dataset_combined_report, "evaluatin
plot_accuracy_per_dataset(balanced_single_dataset_combined_report, "ev
```

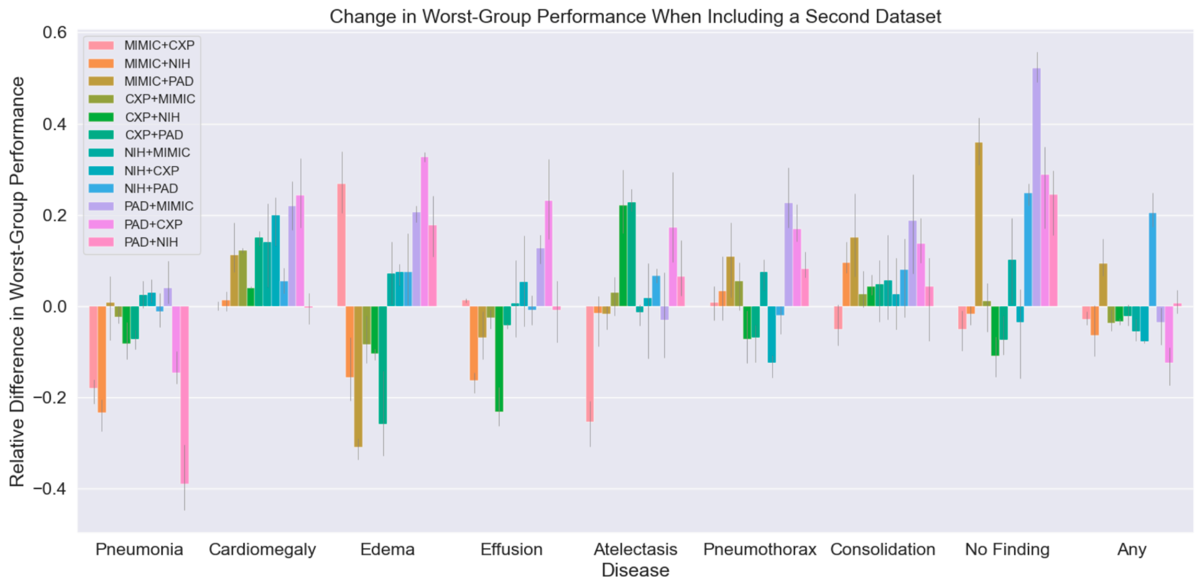




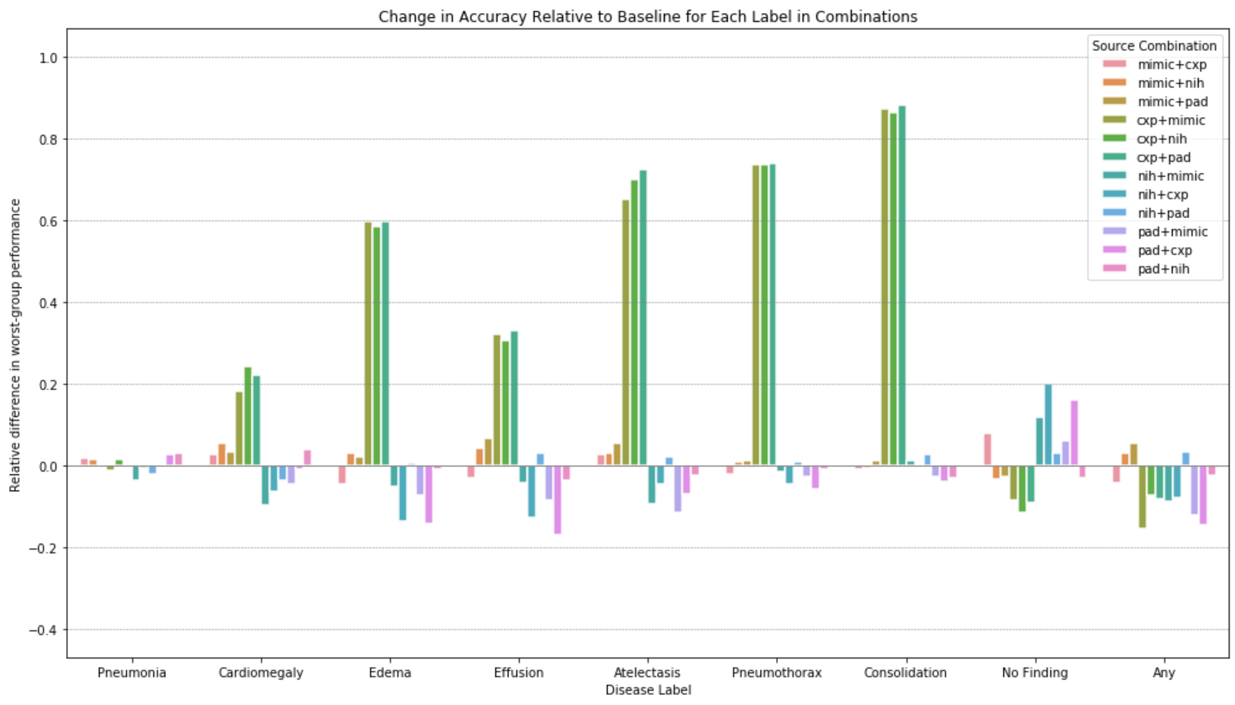
Results

I had a mixed bag results as is seen from the graphs above. Let's dive in.

Original paper results:



My results:



As it is seen from the tables above, I got a significantly different result than the artcile authors. Most accuracy changes from incorporating a second dataset were positive (most impacting in the CXP case), and the negative ones were not as bad as the article stated.

Ablation study

To make sure I am not missing anything, I have repeated the training and the study on the following dataset combinations for training, test and validation:

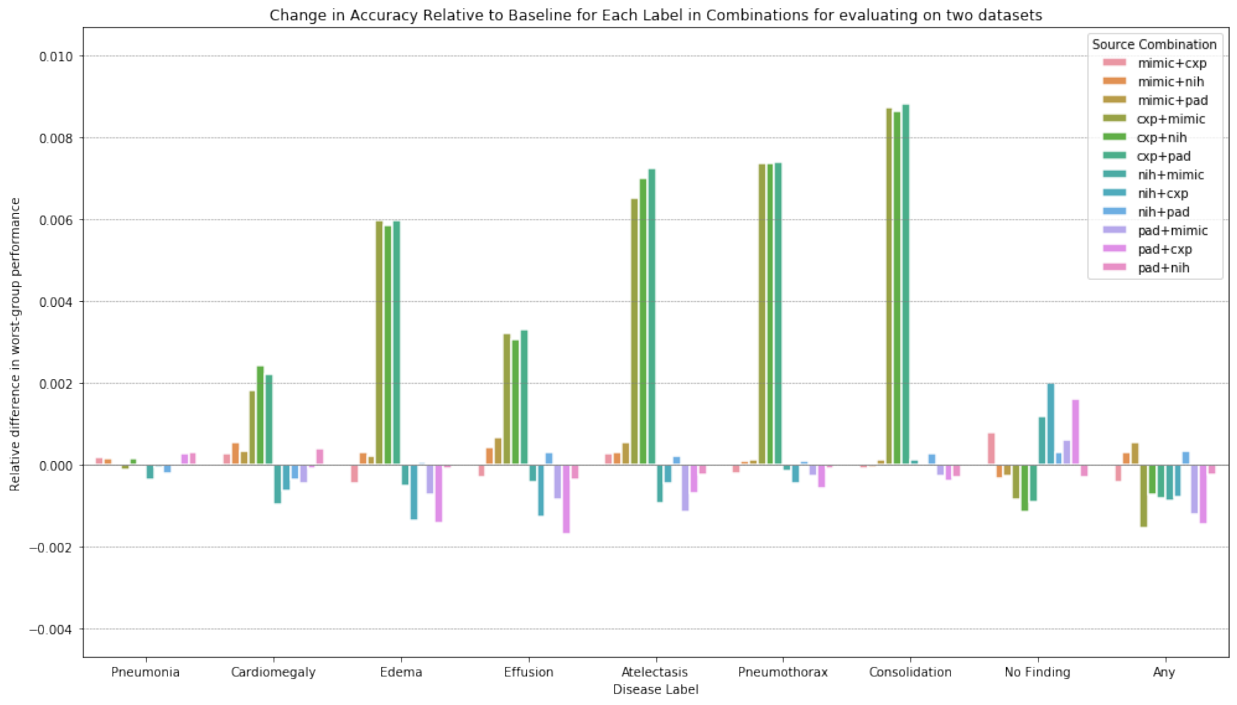
- Unbalanced dataset
  - train on combined (for example, CXP+NIH), validate on combined dataset (for example, CXP+NIH)
  - train on combined (for example, CXP+NIH), validate on single dataset (for example, just CXP)
- Balanced dataset
  - train on combined (for example, CXP+NIH), validate on combined dataset (for example, CXP+NIH)
  - train on combined (for example, CXP+NIH), validate on single dataset (for example, just CXP)

I also re-run the training on different sized results (skip every 20th entry vs every 30th), and in one case ran the study on the full amount of data for one combination.

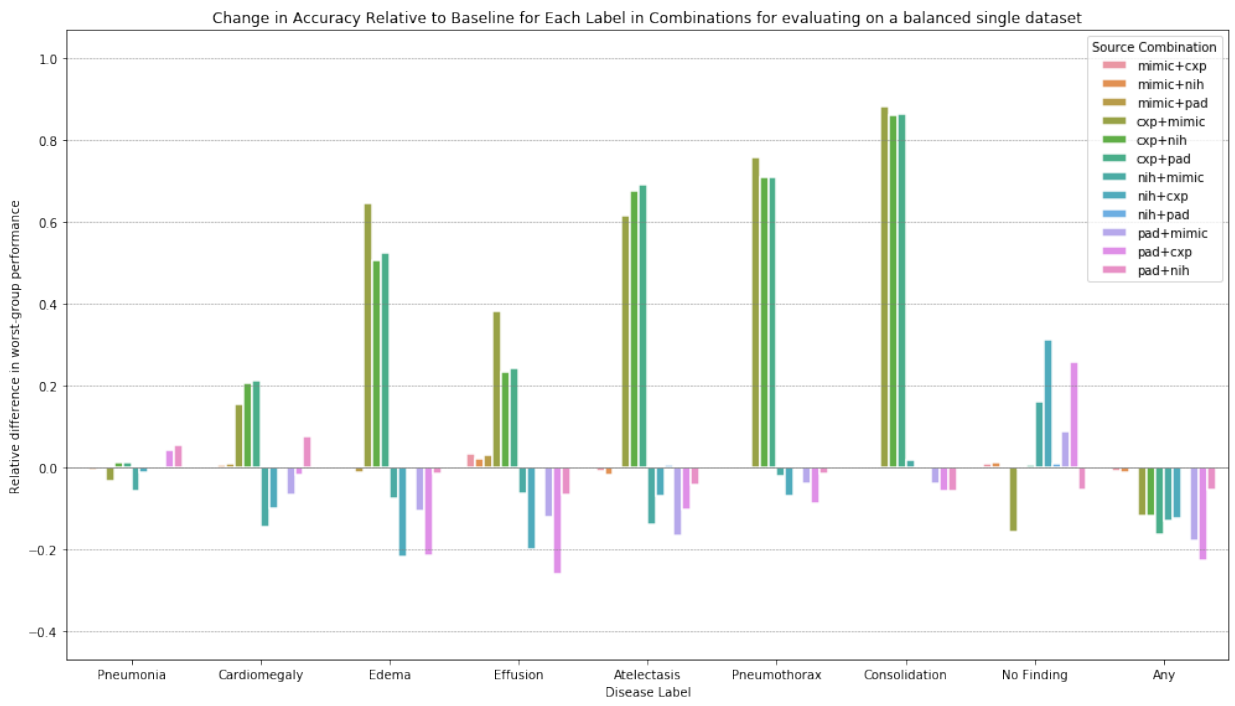
Balancing affected the results slightly, but I still didn't get the picture that the authors of the article got.

When comparing the balanced vs unbalanced dataset validation, the pattern stays the same.

Unbalanced dataset:



Balanced dataset:



When both training and validating on two datasets, the margin of error on accuracy was very small.

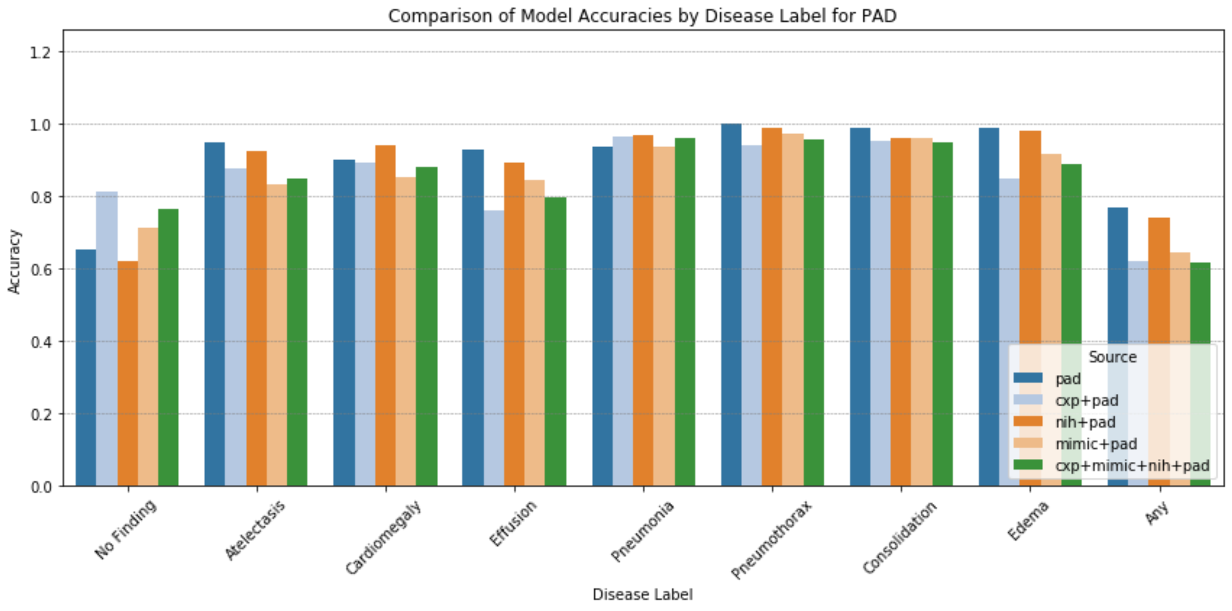
Model comparison

Discussion

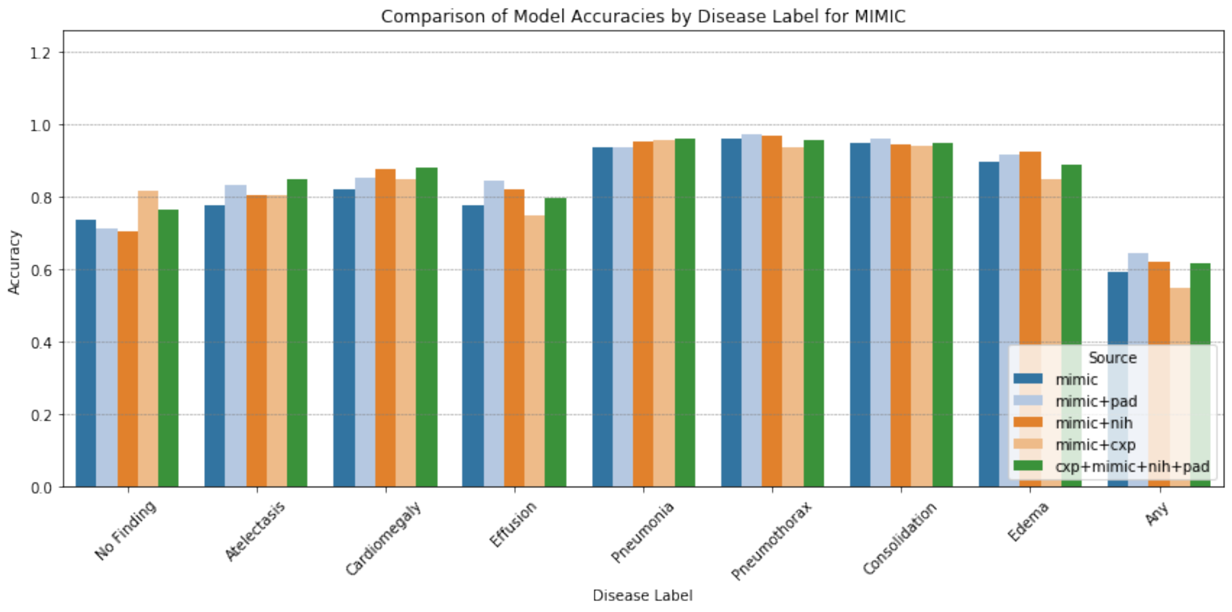
Is the paper reproducible?

Partially. My resulting visualization ended up being different from the one authors used in the article. However, looking at the results, especially when they are broken down by dataset, one can get to pretty much the same response as the authors of the article did: "It depends", and it looks like balancing the dataset, just as the article stated, doesn't always improve the outcome.

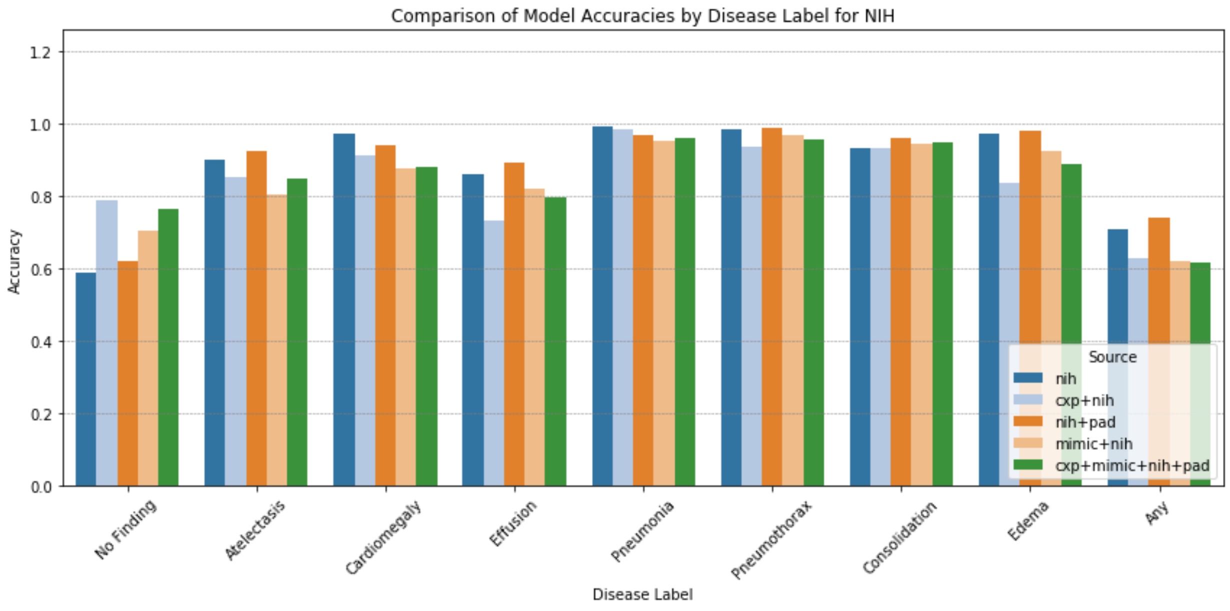
When you look at the results for individual datasets and the resulting combinations, the answer to the main question is very different depending on the dataset:



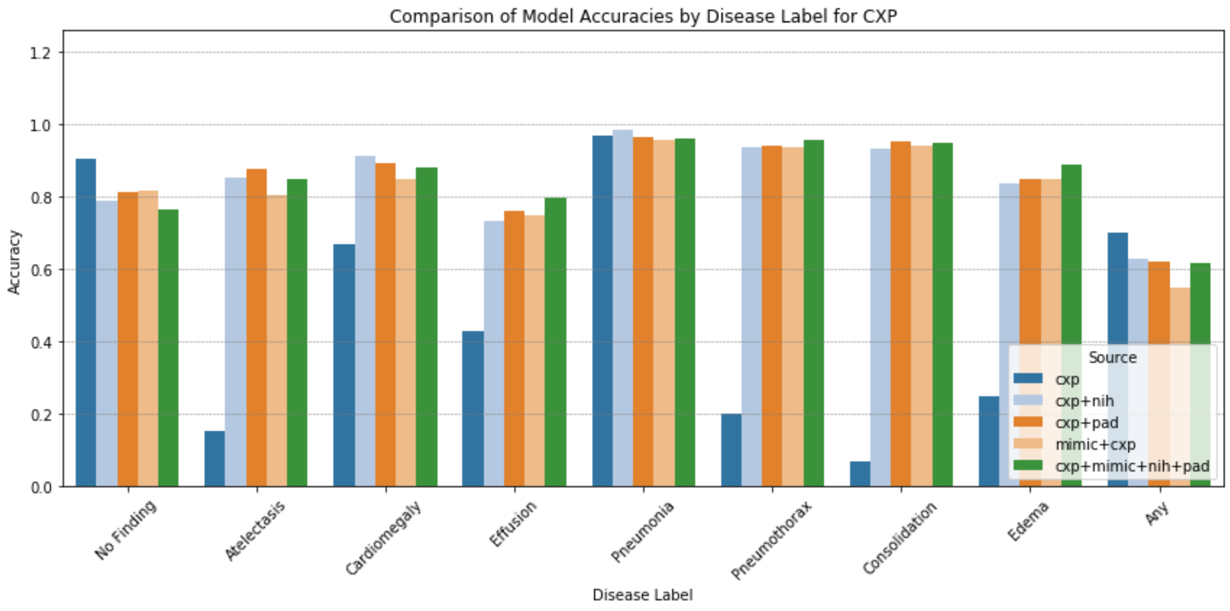
For PAD, the improvements from adding the second dataset were mainly for the "No finding" category, and the rest generally, even though very slightly, performed worse.



For MIMIC, the answer is "more-or-less", although we can see the very slight positive improvement for most labels.



For NIH, again, the performance mostly decreased.



One interesting exception, CXP almost always significantly benefitted from adding more datasets.

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

A portion of the code provided by the article authors was runnable with minimal updates. I was able to reproduce the initial dataset statistics, but wasn't able to run the training code at all.

As a result, I followed the general guidelines given by the authors of the article and wrote my own code - so this may have affected the resulting data I got.

Another factor that might have influenced the resulting data was in data pre-processing. While trying to adapt the authors' code, I discovered an inconsistency in the dataset processing: in some cases the values for certain labels were `True/False` , in some cases numeric `0/1` , and in some cases other numeric value or even a `NaN` .

When adapting the code, I corrected the processing functions so they yield similar-looking results for ease of the combination in a single dataset.

What was easy

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.

What was difficult

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. In the end, I got a message from my provider that my namely "unlimited" internet for the months was used 100% and I will be charged for each extra GB I use.

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 got stuck for a while, trying to figure out why and how to run it (I have a suspicion my processor architecture is not supported).

As a result, 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 were not sufficient for reproducing things successfully without additional modification. wandb wasn't working for me either.

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 to mitigate those differences.

Additional complication was 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. Downloading MIMIC took a week and ate all my provider's internet allowance for a month. Running the training on a full dataset proved it to be difficult due to the whole different set of circumstances: I encountered out of memory issues, kernel panic, random automatic Windows updates, power outage, kids getting to the computer and switching the power supply off. In the end, the estimation to run the training on the full dataset was circa 16+ days, so I opted for running the code on the subset of the data instead.

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 was never able to achieve the 8-minute runtime, in fact, my record was around 10 hours.

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. Provide the instructions and code to plot the results.

Future plans

I am still very interested in answering the question why my results, even though providing a similar answer, looked so different from the article authors'. I plan to do a few more things to try and figure this out:

- Try different models (for example, `LinearSVC` and other models tried by the authors of the article before arriving on their final model), and hyperparameters
- Try and incorporate more of the training and validation code provided by the article authors, and/or triple check mine
- Run the training on the full dataset (ensuring the computer is on backup power supply and is unreachable to anyone trying to switch it off)

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 (https://doi.org/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 (https://doi.org/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 (https://doi.org/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 (https://doi.org/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 (https://doi.org/10.1371/journal.pmed.1002683)

```
In [ ]:
```