Introduction

The paper we have selected is focused on Cross Architectural Self-Supervision for Healthcare Applications.

The relevant code can be found in this repository: https://github.com/dl4h/project-team-58

The Colab notebook and relevant datasets and checkpoints can be found here: https://drive.google.com/drive/folders/1bD-T5_J6JfuNw8e0VWO74tLt1LPa9aw8?usp=sharing

Background

- **Type of Problem** The major type of problem being solved by CASS is related to representation learning and data processing:
  - Existing self-supervised learning models often require expensive computational resources not widely available.
  - Medical imaging and Artificial Intelligence is often limited by a scarcity in prelabeled training data.

  To mitigate these issues, the authors propose *CASS* which allows for training CNN's and Transformers which can reduce pretraining time using less data and less computing resources.

  Importance: Representation learning allows us to use self-supervision to learn useful priors by pretraining unlabeled images. This is crucially important because the medical imaging field suffers from a lack of available labeled data due to a variety of factors, such as the high cost of labeling data at scale because it generally requires domain-specific knowledge. In fields with limited data or high cost to produce labels, we can use self-supervision to help with the downstream learning process without the need for labels. Solving these issues is important because it might allow for the expansion of machine learning and artificial intelligence research into more scenarios which might have previously been hindered by a lack of data or computing resources.

  Difficulty: Existing state-of-the-art self-supervised learning methods have extreme computational requirements that make them inaccessible to most practitioners. Additionally, the limited amount of data makes it infeasible to run smaller epochs with larger batch size to achieve the effectiveness outlined in these self-supervised learning methods. Overcoming the problem is difficult in that it is often a matter of logistics and practicality. For example, gaining labeled data might be impossible for a relatively new disease like COVID-19, as the authors describe in their paper.

State of the art methods and effectiveness: As detailed above, existing state of the art methods for representation learning face challenges due to significant computational requirements and limited data availability. Traditionally, contrastive self-supervision methods use different augmentations of images to create positive pairs. Because of this, augmentations are applied twice which increases time complexity overall. Additionally, parameter sharing between the two architectures increases the time complexity when re-initializing architectures with lagging parameters. Additionally, smaller epochs and batch sizes tend to hurt performance.The current "state of the art" in terms of broad use is *Transfer Learning*, where a model developed for one task is reused as the starting point for a model on a second task. This is practical for scenarios in which there is a lack of data. The authors note that this is common in medical research due to issues such as patient privact or disease prevalence. However, CASS represents a different approach known as *Self-Supervised Learning*.

Paper Explanation

The Proposal

In the paper, the authors acknowledge that self-supervisied learning is generally superior to Transfer Learning, but they require multiple advanced graphical processing units (GPU's) running over the course of several days, something many researchers might consider a luxury both in terms of time and money. Additionally, many self-supervised models suffer in terms of performance when run with small batch sizes.

To this end, the authors propose combining a convolutional neural network (CNN) with a transformer in a "response-based siamese contrastive method".

The Innovations

CASS helps solve our general problem while mitigating these challenges by leveraging CNN and Transformer methods for efficient learning in a siamese contrastive method. CASS leverages architecture invariance instead of using the augmentation invariance approach of existing methods. Extracted representations of each input image are compared across two branches representing each respective architecture. By contrasting their extracted features, they can learn from each other on patterns they would generally miss. This helps provide more useful pre-trained data for the downstream learning method.

Metrics (How Well it Worked)

CASS's approach helps reduce the time complexity of pre-training in two major ways. First, augmentations are only applied once in CASS in comparison to twice in augmentation invariance approaches. Therefore per application CASS uses less augmentations overall. Second, there is no scope for parameter sharing in CASS because the two architectures used are different. A large portion of time is saved in updating the two architectures each epoch as opposed to re-initializing architectures with lagging parameters. CASS has also been proven to handle smaller epochs and batch sizes with better performance overall.

Contribution to Research Regime

The contribution is extremely important to the research regime. Without CASS certain representation learning problems would not be feasible to solve because of computational requirements and data availability. CASS overcomes those challenges while achieving even better performance. CASS is recognized as a cutting-edged self-supervision learning method with accolades advertised on its' Github page such as:

- *State of the Art*: Partial Label Learning on Autoimmune Dataset
- *State of the Art*: Classification on Brain Tumor MRI Dataset
- *State of the Art*: State of the Art: Classification on ISIC 2019

In [ ]:
```
# code comment is used as inline annotations for your coding
```

Scope of Reproducibility

1. **Hypothesis 1**: Leveraging the CASS self-supervised learning approach will significantly improve the efficiency of representation learning in healthcare applications in scenarios which involve a lack of data or computing resources

2. **Hypothesis 2 (Ablation study)**: Reducing the number of pre-training epochs and batch sizes for the CASS model will still allow for strong model performance in comparison with existing methods. **Note 1: As of the draft submission, this is unlikely to be accomplished.**

Mount Notebook to Google Drive

Upload the data, pretrianed model, figures, etc to your Google Drive, then mount this notebook to Google Drive. After that, you can access the resources freely.

Instruction: https://colab.research.google.com/notebooks/io.ipynb

Example:
https://colab.research.google.com/drive/1srw_HFWQ2SMgmWIawucXfusGzrj1_U0q

Video: https://www.youtube.com/watch?v=zc8g8lGcwQU

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

Methodology

The original CASS authors utilized the following datasets in their paper:

- DERMOFIT
- Brain MRI Classification
- SIIM-ISIC 2019 Dataset

The paper involved training a CNN and Transformer using CASS, then evaluating their performance vs. DINO (an alternative state-of-the-art self-supervision model).

However, in our approach we found that only the Brain MRI Classification dataset was available (with further confusing findings). The official CASS Github repository contains the code required for training CASS on a given CNN and attention model specifically in the case of a fourth dataset: MedMNIST. Specifically, PathMNIST which is a multi-class dataset for Colon pathology.

Based on these findings, we opted to attempt to recreate the CASS paper using the Brain Tumor dataset and MedMNIST.

We recognized that the MedMNIST dataset was not originally used in the dataset, but in our proposal we originally suggested that we may be able to leverage this dataset to test our hypothesis (*Leveraging CASS self-supervised learning will significantly improve efficiency of representation learning in scenarios which involve a lack of healthcare data or computing resources*).

Required Installs

In [ ]:
```
# install necessary libraries (from requirements.txt) TODO Note errors in ou
%pip install einops~=0.4.1
%pip install matplotlib~=3.5.2
%pip install matplotlib-inline~=0.1.2
%pip install numpy~=1.23.1
%pip install pandas~=1.4.3
%pip install Pillow~=9.2.0
%pip install scikit-learn~=1.1.1
%pip install scipy~=1.8.1
```

```
%pip install tensorboard~=2.9.1
%pip install timm~=0.5.4
%pip install torch~=1.11.0+cu113 -f https://download.pytorch.org/whl/cu113/t
%pip install torchtext~=0.12.0
%pip install torchaudio~=0.11.0+cu113 -f https://download.pytorch.org/whl/cu
%pip install torchcontrib~=0.0.2
%pip install torchmetrics~=0.9.2
%pip install torchvision~=0.12.0+cu113 -f https://download.pytorch.org/whl/c
%pip install vit-pytorch~=0.35.8
%pip install pytorch-lightning~=1.6.5
%pip install tqdm~=4.64.0
%pip install h5py # for loading brain tumor HDF5 dataset
%pip install medmnist

# import packages you need
# from google.colab import drive

# imports for CASS
import math
import os
import numpy as np
import pandas as pd
import pytorch_lightning as pl
import timm
import torch
import torch.nn as nn

from PIL import Image
from torch.utils.data import Dataset, DataLoader
from pytorch_lightning import Trainer, seed_everything
from pytorch_lightning.loggers import CSVLogger
from pytorch_lightning.callbacks import ModelCheckpoint, EarlyStopping
from torch.utils.tensorboard import SummaryWriter
from torchcontrib.optim import SWA
from torchmetrics import Metric
from torchvision import transforms as tsfm
from tqdm import tqdm
```

Brain Tumor Dataset

Data

The authors cite the brain tumor dataset from their official Github:

1. https://figshare.com/articles/dataset/brain_tumor_dataset/1512427
2. https://www.hindawi.com/journals/cin/2022/3236305/

Where #1 links to a dataset which hosts the images in zipped matlab format. Link #2 references another research paper, which itself references the Brain Tumor dataset located on Kaggle.

The Kaggle version of the dataset includes a reference to the Github version of the same dataset.

For our experiment, we opted to use the Github version of the dataset which provided the below starter code for loading the dataset as well as defining train and test data. It is important to note that the CASS authors also describe this data as having been split (train/test) in this same way by the data curators. As such, we did not change these definitions.

- The methods for cloning and loading a train/test dataset were re-used from that Github repository, however the attempted CASS implementation and exploratory data analysis (EDA) remains the work of this team.

In [ ]:
```
# The code below is from the brain tumor notebook associated with its' offic
# retrieving and setting up train/test data.
# The CASS authors mention that the brain tumor dataset they used (and this

import cv2
import random
import pickle
import tqdm
import os
import numpy as np

!git clone https://github.com/SartajBhuvaji/Brain-Tumor-Classification-DataS

# Define necessary constants
IS_LOCAL = True
BASE_PATH = './' if IS_LOCAL else '/content'
TEST_DIR = f'{BASE_PATH}Brain-Tumor-Classification-DataSet/Testing'
TRAIN_DIR = f'{BASE_PATH}Brain-Tumor-Classification-DataSet/Training'
IMG_SIZE = 224
CATEGORIES = ["glioma_tumor","meningioma_tumor","no_tumor","pituitary_tumor"

# Creating training dataset
training_data = []

def create_training_data():
    for category in CATEGORIES:
        path = os.path.join(TRAIN_DIR,category)
        class_num = CATEGORIES.index(category)
        for img in tqdm.tqdm(os.listdir(path)):
          img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_COLOR)
          new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
          training_data.append([new_array, class_num])

    random.shuffle(training_data)

create_training_data()
#np.save('train_data.npy', training_data)
print(f'training dataset size: {len(training_data)}')

X_train = np.array([i[0] for i in training_data]).reshape(-1,IMG_SIZE,IMG_SI
Y_train = [i[1] for i in training_data]

pickle_out = open("X_train.pickle","wb")
pickle.dump(X_train, pickle_out)
pickle_out.close()

pickle_out = open("Y_train.pickle","wb")
pickle.dump(Y_train, pickle_out)
pickle_out.close()


# Creating testing dataset
testing_data = []

def create_testing_data():
    for category in CATEGORIES:
        path = os.path.join(TEST_DIR,category)
        class_num = CATEGORIES.index(category)

        for img in tqdm.tqdm(os.listdir(path)):
          img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_COLOR)
          new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
          testing_data.append([new_array, class_num])

    random.shuffle(testing_data)

create_testing_data()
#np.save('testing_data.npy', testing_data)
print(f'testing dataset size: {len(testing_data)}')
X_test= np.array([i[0] for i in testing_data]).reshape(-1,IMG_SIZE,IMG_SIZE,
Y_test = [i[1] for i in testing_data]

pickle_out = open("X_test.pickle","wb")
pickle.dump(X_test, pickle_out)
pickle_out.close()

pickle_out = open("Y_test.pickle","wb")
pickle.dump(Y_test, pickle_out)
pickle_out.close()
```

Exploratory Data Analysis (Brain Tumor Dataset)

Here, we analyze the data to understand it's format and most importantly, to understand if it is the same data the authors describe.

In [ ]:

```
import matplotlib.pyplot as plt

# Load the training or testing images and labels
pickle_in = open("X_train.pickle", "rb")
X_train = pickle.load(pickle_in)
pickle_in.close()

pickle_in = open("Y_train.pickle", "rb")
Y_train = pickle.load(pickle_in)
pickle_in.close()
```

```
# Choose a sample image and label, for example, the first one in the dataset
sample_image = X_train[0]
sample_label = Y_train[0]

# Since the images are stored in BGR format by OpenCV, convert them to RGB f
sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

# Display the image
plt.imshow(sample_image)
plt.title(f'Tumor Type: {sample_label}')
plt.show()
```

In [ ]:

```
# understand distribution of labels

with open("Y_train.pickle", "rb") as f:
    Y_train = pickle.load(f)

with open("Y_test.pickle", "rb") as f:
    Y_test = pickle.load(f)

all_labels = Y_train + Y_test

tumor_types = {'glioma_tumor': 0, 'meningioma_tumor': 0, 'no_tumor': 0, 'pit

for label in all_labels:
    category = CATEGORIES[label]
    if category in tumor_types:
        tumor_types[category] += 1

# visualization
plt.bar(tumor_types.keys(), tumor_types.values())
plt.xlabel('Tumor Type')
plt.ylabel('Frequency')
plt.title('Distribution of Tumor Types')
plt.xticks(rotation=45)  # Rotate category names for better visibility
plt.show()
```

Statement Regarding Brain Tumor Dataset

The authors describe in their paper a brain tumor dataset which has 5,712 training images and 1,310 test images which results in approximately 18.66% train data in their experiment, with the remaining percentage allocated to test. This is something close to a 80/20 train test split.

However, the authors of this project found multiple instances of the brain tumor dataset (both referenced in the official CASS Github), and in both cases the number of total images (as well as train vs. test is different).

In our case, we have 3,264 total images, with 2,870 for training and 394 for testing. Applying the same understanding as above, this means we have a test set ratio of ~ 12.07% to training data of 87.93%.

Because we are not operating with the same total data (or ratios), we cannot accurately replicate the study as intended using this dataset. We attempted to load this data into the CNN/Attention modle using the CASS-supplied code but ultimately were unsuccessful. These efforts will be explained in the Results section.

MedMNIST

Data

The data source for our project is the PathMNIST dataset located within the larger MedMNIST database. The MedMNIST data is curated by researchers from several universities, such as: Shanghai Jiao Tong University, RWTH Aachen University, and Harvard University.

PathMNIST in particular is a collection of images corresponding to Colon Pathology with regard to (9) different classes or "labels".

The dataset is located here: https://zenodo.org/records/10519652 but for our purposes, we chose to utilize the Python package (https://pypi.org/project/medmnist/).

- 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.

In [ ]:
```
import numpy as np
import torchvision.transforms as transforms
import torch.utils.data as data
import medmnist
from medmnist import INFO

data_flag = "pathmnist"
download = True

# BATCH_SIZE = 128 --> this causes Colab machines to run out of memory, ever
BATCH_SIZE = 16


# Load dataset information
info = INFO[data_flag]
n_channels = info["n_channels"]
n_classes = len(info["label"])

DataClass = getattr(medmnist, info["python_class"])

# NOTE switched code below to what is used in the CASS.ipynb medmnist exampl
# Define transformations
# possibly overkill for EDA
"""
Define train & valid image transformation
"""
DATASET_IMAGE_MEAN = (0.485, 0.456, 0.406)
DATASET_IMAGE_STD = (0.229, 0.224, 0.225)

train_transform = tsfm.Compose(
    [
        tsfm.Resize((384, 384)),
        tsfm.RandomApply(
            [
                tsfm.ColorJitter(0.2, 0.2, 0.2),
                tsfm.RandomPerspective(distortion_scale=0.2),
            ],
            p=0.3,
        ),
        tsfm.RandomApply(
            [
                tsfm.RandomAffine(degrees=10),
            ],
            p=0.3,
        ),
        tsfm.RandomVerticalFlip(p=0.3),
        tsfm.RandomHorizontalFlip(p=0.3),
        tsfm.ToTensor(),
        tsfm.Normalize(DATASET_IMAGE_MEAN, DATASET_IMAGE_STD),
    ]
)

valid_transform = tsfm.Compose(
    [
        tsfm.Resize((384, 384)),
        tsfm.ToTensor(),
        tsfm.Normalize(DATASET_IMAGE_MEAN, DATASET_IMAGE_STD),
    ]
)

# Load the data
train_dataset = DataClass(split="train", transform=train_transform, downloac
val_dataset = DataClass(split="val", transform=valid_transform, download=dow
test_dataset = DataClass(split="test", transform=valid_transform, download=c

pil_dataset = DataClass(split="train", download=download)

# encapsulate data into dataloader form
train_loader = data.DataLoader(
    dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True
)
valid_loader = data.DataLoader(dataset=val_dataset, batch_size=BATCH_SIZE, s
train_loader_at_eval = data.DataLoader(
    dataset=train_dataset, batch_size=2 * BATCH_SIZE, shuffle=False
)
test_loader = data.DataLoader(
    dataset=test_dataset, batch_size=2 * BATCH_SIZE, shuffle=False
)
```

```python
print(train_dataset)
print("===================")
print(test_dataset)
```

In [ ]:
```python
# summary statistics
import pandas as pd

# create a dataframe with summary statistics
data = {
    "Dataset": ["train", "validation", "test"],
    "Number of images": [len(train_dataset), len(val_dataset), len(test_data
}

# Include total sum
total_images = sum(data["Number of images"])
data["Dataset"].append("Total")
data["Number of images"].append(total_images)

summary_df = pd.DataFrame(data)
summary_df
```

In [ ]:
```python
# Show sample images in this cell
print('Sample training images')
train_dataset.montage()
```

In [ ]:
```python
# Show class distribution of labels
import matplotlib.pyplot as plt

# get the labels
train_labels = train_dataset.labels
val_labels = val_dataset.labels
test_labels = test_dataset.labels

# Map label indices to descriptive names
label_names = info["label"]

# create a dataframe with the labels
train_labels_df = pd.DataFrame(train_labels, columns=["label"])
train_labels_df["label_name"] = train_labels_df["label"].map(label_names)
val_labels_df = pd.DataFrame(val_labels, columns=["label"])
val_labels_df["label_name"] = val_labels_df["label"].map(label_names)
test_labels_df = pd.DataFrame(test_labels, columns=["label"])
test_labels_df["label_name"] = test_labels_df["label"].map(label_names)

# plot the class distribution
fig, ax = plt.subplots(1, 3, figsize=(20, 5))
train_labels_df["label"].value_counts().plot(kind="bar", ax=ax[0], title="Tr
val_labels_df["label"].value_counts().plot(
    kind="bar", ax=ax[1], title="Validation Labels"
)
test_labels_df["label"].value_counts().plot(kind="bar", ax=ax[2], title="Tes

for i in range(3):
    # Append x-axis labels with descriptions
    ax[i].set_xticklabels(
        [
            f"{label_names[t.get_text()]} ({t.get_text()})"
            for t in ax[i].get_xticklabels()
        ]
    )

    # Slighly rotate x-axis labels for better readability
    plt.setp(ax[i].get_xticklabels(), rotation=45, ha="right")


plt.show()
```

In [ ]:
```
# import os


# # datasets_path = '/content/drive/My Drive/Shared with me/598-58/datasets'


# # Or, if you are the owner (brianib2) and shared it with others:
# datasets_path = '/content/drive/My Drive/598-58/datasets/brain-tumor-datas


# import os
# import numpy as np
# import scipy.io
# import matplotlib.pyplot as plt

# base_path = datasets_path

# # Function to load .mat files and extract data
# def load_data(folder):
#     files = os.listdir(folder)
#     data_list = []
#     for file in files:
#         mat = scipy.io.loadmat(os.path.join(folder, file))
#         data = mat['cjdata']
#         data_list.append(data)
#     return data_list

# # Load data from each subset
# data_parts = ['brainTumorDataPublic_1766', 'brainTumorDataPublic_7671532',
#               'brainTumorDataPublic_15332298', 'brainTumorDataPublic_22993
# all_data = []
# for part in data_parts:
#     part_data = load_data(os.path.join(base_path, part))
#     all_data.extend(part_data)
```

Model

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

- Model architecture: layer number/size/type, activation function, etc
- The "CFG" class in the code below references both a CNN (ResNet) and a Transformer Model (Vision Transformer (ViT)), characteristic of the CASS Model
- Training objectives: loss function, optimizer, weight of each loss term, etc Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc.
- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

In [ ]:
```
# Define a the number to string maps for labels and the device

label_num2str = {
    0: "adipose",
    1: "background",
    2: "debris",
    3: "lymphocytes",
    4: "mucus",
    5: "smooth muscle",
    6: "normal colon mucosa",
    7: "cancer-associated stroma",
    8: "colorectal adenocarcinoma epithelium",
}

print(f"label_num2str: {label_num2str}")

label_str2num = {}
for i in label_num2str:
    label_str2num[label_num2str[i]] = i

print(f"label_str2num: {label_str2num}")
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

print("device:", device)
```

In [ ]:
```
class CFG:
    # We dont need to give the path to CSV and images as medMNIST provides c
    # Check MNIST Get-started-DEDL.ipynb for details on how to get the label
    # class weights for the MedMNIST dataset.
    label_num2str = label_num2str
    label_str2num = label_str2num
    fl_alpha = 1.0  # alpha of focal_loss
    fl_gamma = 2.0  # gamma of focal_loss
    cls_weight = [
        0.4368473694738948,
        0.4597319463892779,
        0.5959191838367675,
        0.6024804960992198,
        0.21920384076815363,
        0.8874974994999001,
        0.2,
        0.4424484896979396,
        1.0,
    ]
    cnn_name = "resnet50"
    vit_name = "vit_base_patch16_384"
    seed = 77
    num_classes = 9
    batch_size = 16
    t_max = 16
    lr = 1e-3
    min_lr = 1e-6
    n_fold = 6
    num_workers = 8
    gpu_idx = 0
    device = torch.device(f"cuda:{gpu_idx}" if torch.cuda.is_available() els
    gpu_list = [gpu_idx]
```

In [ ]:

```
"""
Define Focal-Loss

cls_weights Configuration: Adjust the CFG.cls_weight array to match the freq
This should reflect the inverse frequency of each class or other heuristics
"""


class FocalLoss(nn.Module):
    """
    The focal loss for fighting against class-imbalance
    """

    def __init__(self, alpha=1, gamma=2):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = 1e-12  # prevent training from Nan-loss error
        self.cls_weights = torch.tensor(
            [CFG.cls_weight], dtype=torch.float, requires_grad=False, device
        )

    def forward(self, logits, target):
        """
        logits & target should be tensors with shape [batch_size, num_classe
        """
        probs = torch.sigmoid(logits)
        one_subtract_probs = 1.0 - probs
        # add epsilon
        probs_new = probs + self.epsilon
        one_subtract_probs_new = one_subtract_probs + self.epsilon
        # calculate focal loss
        log_pt = target * torch.log(probs_new) + (1.0 - target) * torch.log(
            one_subtract_probs_new
        )
        pt = torch.exp(log_pt)
        focal_loss = -1.0 * (self.alpha * (1 - pt) ** self.gamma) * log_pt
        focal_loss = focal_loss * self.cls_weights
        return torch.mean(focal_loss)
```

In [ ]:
```
"""
Define F1 score metric
"""


class MyF1Score(Metric):
    def __init__(self, cfg, threshold: float = 0.5, dist_sync_on_step=False)
        super().__init__(dist_sync_on_step=dist_sync_on_step)
        self.cfg = cfg
        self.threshold = threshold
        self.add_state("tp", default=torch.tensor(0), dist_reduce_fx="sum")
        self.add_state("fp", default=torch.tensor(0), dist_reduce_fx="sum")
        self.add_state("fn", default=torch.tensor(0), dist_reduce_fx="sum")

    def update(self, preds: torch.Tensor, target: torch.Tensor):
        # assert preds.shape == target.shape
        preds_str_batch = self.num_to_str(torch.sigmoid(preds))
        target_str_batch = self.num_to_str(target)
        tp, fp, fn = 0, 0, 0
        for pred_str_list, target_str_list in zip(preds_str_batch, target_st
            for pred_str in pred_str_list:
                if pred_str in target_str_list:
                    tp += 1
                if pred_str not in target_str_list:
                    fp += 1

            for target_str in target_str_list:
                if target_str not in pred_str_list:
                    fn += 1
        self.tp += tp
        self.fp += fp
        self.fn += fn

    def compute(self):
        # To switch between F1 score and recall.
        f1 = 2.0 * self.tp / (2.0 * self.tp + self.fn + self.fp)
        rec = self.tp / (self.tp + self.fn)
        return f1

    def num_to_str(self, ts: torch.Tensor) -> list:
        batch_bool_list = (ts > self.threshold).detach().cpu().numpy().tolis
        batch_str_list = []
        for one_sample_bool in batch_bool_list:
            # print(self.cfg.label_num2str)
            lb_str_list = [
                self.cfg.label_num2str[lb_idx]
                for lb_idx, bool_val in enumerate(one_sample_bool)
                if bool_val
            ]
            batch_str_list.append(lb_str_list)
        return batch_str_list
```

In [ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

cfg=CFG()
```

```
model_cnn = timm.create_model(cfg.cnn_name, pretrained=True)
model_vit = timm.create_model(cfg.vit_name, pretrained=True)
model_cnn.to(device)
model_vit.to(device)
```

In [ ]:

```
# Brian Approach 4/11
# The authors had 2 parameters that weren't used in the model definition so
# this wasn't used in either of their CASS MEDNIST notebooks

import torch
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm
import torch
import torch.cuda as cuda


def ssl_train_model(
    train_loader,
    model_vit,
    optimizer_vit,
    scheduler_vit,
    model_cnn,
    optimizer_cnn,
    scheduler_cnn,
    num_epochs,
):
    writer = SummaryWriter()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    phase = "train"
    model_cnn.train()
    model_vit.train()

    # print(cuda.memory_summary(device=device, abbreviated=False))

    for i in tqdm(range(num_epochs)):
        with torch.set_grad_enabled(phase == "train"):
            for img, _ in tqdm(train_loader):
                img = img.to(device)

                # use for debugging
                # print(f"Start of loop - {cuda.memory_summary(device=device
                pred_vit = model_vit(img)
                pred_cnn = model_cnn(img)

                model_sim_loss = loss_fn(pred_vit, pred_cnn)
                loss = model_sim_loss.mean()

                loss.backward()

                optimizer_cnn.step()
                optimizer_vit.step()
                scheduler_cnn.step()
                scheduler_vit.step()

                print("For -", i, "Loss:", loss.item())
                writer.add_scalar("Self-Supervised Loss/train", loss.item(),
                del img, pred_vit, pred_cnn, model_sim_loss, loss

    writer.flush()


def loss_fn(x, y):
    x = torch.nn.functional.normalize(x, dim=-1, p=2)
    y = torch.nn.functional.normalize(y, dim=-1, p=2)
    return 2 - 2 * (x * y).sum(dim=-1)
```

In [ ]:

```
optimizer_cnn = SWA(torch.optim.Adam(model_cnn.parameters(), lr=1e-3))
optimizer_vit = SWA(torch.optim.Adam(model_vit.parameters(), lr=1e-3))
scheduler_cnn = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer_cnn, T_max=16, eta_min=1e-6
)
scheduler_vit = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer_vit, T_max=16, eta_min=1e-6
)

fl_alpha = 1.0  # alpha of focal_loss
fl_gamma = 2.0  # gamma of focal_loss
cls_weight = [0.9475164011246484, 0.4934395501405811, 0.5029053420805999, 0.

# these are unused (brian 4/11)
# criterion_vit = FocalLoss(fl_alpha, fl_gamma)
# criterion_cnn = FocalLoss(fl_alpha, fl_gamma)
```

Training

- Below we perform both self=supervised training (CASS) and supervised training via CNN and ViT
- Computational Requirements: We are able to get through a clean run of self supervised training using the V100 GPU on the Google Colab Notebook (this required a paid Colab Pro account). We are also able to run through the supervised CNN training although this part of the code still needs modifications to work as expected. Finally, when training the ViT model we run into a CUDA memory error. It is worth noting that many adjustments had to be made to even get to this point where we could successfully complete self supervised training. We sampled a subset of the medMNIST dataset, only trained for 1 epoch, and lowered batch size to 8.

In [ ]:

```
# Train using self-supervised learning
import os

print(f'Number of training samples: {len(train_loader)}')

# Check if checkpoints exist
# These are the location of the paths from the shared drive
checkpoints_directory = "/content/drive/MyDrive/598-58/checkpoints"
checkpoint_files = os.listdir(checkpoints_directory)
```

```
print(f'checkpoint files: {checkpoint_files}')

failed_to_load = False
if "cass-r50-med-mnist-pathmnist.pt" in checkpoint_files:
    print("Loading ResNet50 model")
    model_cnn = torch.load(f'{checkpoints_directory}/cass-r50-med-mnist-path
else:
    failed_to_load = True

if "cass-vit-med-mnist-pathmnist.pt" in checkpoint_files:
    print("Loading ViT model")
    model_vit = torch.load(f'{checkpoints_directory}/cass-vit-med-mnist-path
else:
    failed_to_load = True

if failed_to_load:
    print('Running self-')
    ssl_train_model(
        train_loader,
        model_vit,
        optimizer_vit,
        scheduler_vit,
        model_cnn,
        optimizer_cnn,
        scheduler_cnn,
        num_epochs=1,
    )

  # Saving SSL Models
  print("Saving Cov-T")

  torch.save(model_cnn, "./cass-r50-med-mnist-pathmnist.pt")
  torch.save(model_vit, "./cass-vit-med-mnist-pathmnist.pt")
```

In [ ]:
```
# Train using supervised learning (need labels)

import math
from tqdm import tqdm

# TODO Commented out line below because indenting was off for a for loop and
# for fold_idx, (train_indices, valid_indices) in enumerate(k_fold.split(all
model_vit = torch.load("./cass-vit-med-mnist-pathmnist.pt")
model_cnn = torch.load("./cass-r50-med-mnist-pathmnist.pt")
last_loss = math.inf
val_loss_arr = []
train_loss_arr = []
counter = 0

model_cnn.to(device)
model_vit.to(device)
print("*" * 10)


# Train Correspong Supervised CNN
print("Fine tunning Cov-T")
```

```
writer = SummaryWriter()
model_cnn.fc = nn.Linear(in_features=2048, out_features=9, bias=True)
criterion = FocalLoss(cfg.fl_alpha, cfg.fl_gamma)
metric = MyF1Score(cfg)
val_metric = MyF1Score(cfg)
optimizer = torch.optim.Adam(model_cnn.parameters(), lr=3e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer, T_max=cfg.t_max, eta_min=cfg.min_lr
)
model_cnn.train()
from torch.autograd import Variable

best = 0
best_val = 0
for epoch in tqdm(range(1)):
    total_loss = 0
    for images, label in tqdm(train_loader):
        model_cnn.train()
        images = images.to(device)
        label = label.to(device)
        model_cnn.to(device)
        pred_ts = model_cnn(images)
        loss = criterion(pred_ts, label)
        score = metric(pred_ts, label)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        scheduler.step()
        total_loss += loss.detach()
    avg_loss = total_loss / len(train_loader)
    train_score = metric.compute()
    logs = {
        "train_loss": avg_loss,
        "train_f1": train_score,
        "lr": optimizer.param_groups[0]["lr"],
    }
    writer.add_scalar("CNN Supervised Loss/train", loss, epoch)
    writer.add_scalar("CNN Supervised F1/train", train_score, epoch)
    print(logs)

    if best < train_score:
        best = train_score
        model_cnn.eval()
        total_loss = 0
        for images, label in valid_loader:
            images = images.to(device)
            label = label.to(device)
            model_cnn.to(device)
            pred_ts = model_cnn(images)
            score_val = val_metric(pred_ts, label)
            val_loss = criterion(pred_ts, label)
            total_loss += val_loss.detach()
        avg_loss = total_loss / len(train_loader)
        print("Val Loss:", avg_loss)
        val_score = val_metric.compute()
        print("CNN Validation Score:", val_score)
        writer.add_scalar("CNN Supervised F1/Validation", val_score, epoch)
        if avg_loss > last_loss:
            counter += 1
        else:
            counter = 0

        last_loss = avg_loss
        if counter > 5:
            print("Early Stopping!")
            break
        else:
            if val_score > best_val:
                best_val = val_score
                print("Saving")
                torch.save(model_cnn, "./cass-r50-med-mnist-pathmnist-label.
writer.flush()
last_loss = 999999999
val_loss_arr = []
train_loss_arr = []
counter = 0


# Training the Corresponding ViT
model_vit.head = nn.Linear(in_features=768, out_features=9, bias=True)
criterion = FocalLoss(cfg.fl_alpha, cfg.fl_gamma)
metric = MyF1Score(cfg)
optimizer = torch.optim.Adam(model_vit.parameters(), lr=3e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer, T_max=cfg.t_max, eta_min=cfg.min_lr
)
model_vit.train()
val_metric = MyF1Score(cfg)
writer = SummaryWriter()

from torch.autograd import Variable

best = 0
best_val = 0
for epoch in tqdm(range(1)):
    total_loss = 0
    for images, label in tqdm(train_loader):
        model_vit.train()
        images = images.to(device)
        label = label.to(device)
        model_vit.to(device)
        pred_ts = model_vit(images)
        loss = criterion(pred_ts, label)
        score = metric(pred_ts, label)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        scheduler.step()
        total_loss += loss.detach()
    avg_loss = total_loss / len(train_loader)
    train_score = metric.compute()
    logs = {
        "train_loss": loss,
        "train_f1": train_score,
        "lr": optimizer.param_groups[0]["lr"],
    }
    writer.add_scalar("ViT Supervised Loss/train", loss, epoch)
    writer.add_scalar("ViT Supervised F1/train", train_score, epoch)
    print(logs)
    if best < train_score:
        best = train_score
        model_vit.eval()
        total_loss = 0
        for images, label in valid_loader:
            images = images.to(device)
            label = label.to(device)
            model_vit.to(device)
            pred_ts = model_vit(images)
            score_val = val_metric(pred_ts, label)
            val_loss = criterion(pred_ts, label)
            total_loss += val_loss.detach()
        avg_loss = total_loss / len(train_loader)
        val_score = val_metric.compute()
        print("ViT Validation Score:", val_score)
        print("Val Loss:", avg_loss)
        writer.add_scalar("ViT Supervised F1/Validation", val_score, epoch)
        if avg_loss > last_loss:
            counter += 1
        else:
            counter = 0

        last_loss = avg_loss
        if counter > 5:
            print("Early Stopping!")
            break
        else:
            if val_score > best_val:
                best_val = val_score
                print("Saving")
                torch.save(model_vit, "./cass-vit-med-mnist-pathmnist.pt")

    writer.flush()
    print("*" * 10)
```

Evaluation

As of this draft, we have note been able to successfully train a model due to the challenges we ran into. We will continue working to get a model trained so that we can run evaluations and present some data to discuss.

In [ ]:
```
class my_model():
  # use this class to define your model
  pass

model = my_model()
loss_func = None
```

```
optimizer = None

def train_model_one_iter(model, loss_func, optimizer):
  pass

num_epoch = 10
# model training loop: it is better to print the training/validation losses
for i in range(num_epoch):
  train_model_one_iter(model, loss_func, optimizer)
  train_loss, valid_loss = None, None
  print("Train Loss: %.2f, Validation Loss: %.2f" % (train_loss, valid_loss)
```

Results

This section will contain statistical results from running CASS using the MedMNIST dataset.

Because the Brain Tumor dataset was previously the only dataset the authors used which we could find, and because this dataset ultimately proved to somehow not be the same data that the authors describe, we were not able to implement CASS-trained CNN and Transformers using this dataset and there are no results to present for this dataset.

In [ ]:
```
# metrics to evaluate my model

# plot figures to better show the results

# it is better to save the numbers and figures for your presentation.
```

Metric	Your Model	Author's Model	Dataset Sample Size
Accuracy	XX.X%	XX.X%	N
AUC	X.XX	X.XX	N
RMSE	X.XX	X.XX	N
Precision	XX.X%	XX.X%	N
Recall	XX.X%	XX.X%	N
F1 Score	X.XX	X.XX	N
Sensitivity	XX.X%	XX.X%	N
Specificity	XX.X%	XX.X%	N

Notes:

- `N` is the total number of samples in the dataset used for testing.
- `XX.X%` and `X.XX` represent placeholder values for the actual percentages and scores respectively.

Model comparison

```
In [ ]: # compare you model with others
        # you don't need to re-run all other experiments, instead, you can directly
```

Discussion

Generally, Team 58 found recreating the CASS experiments from the research paper to be extremely difficult for a multitude of reasons which we will explain in this section. As such, we were not able to replicate the study, much less experiment with our proposed ablations.

In this section,you should discuss your work and make future plan. The discussion should address the following questions:

- Make assessment that the paper is reproducible or not.
- Explain why it is not reproducible if your results are kind negative.
- Describe “What was easy” and “What was difficult” during the reproduction.
- Make suggestions to the author or other reproducers on how to improve the reproducibility.
- What will you do in next phase.

Our results demonstrate that the paper is/is not reproducible. This conclusion is based on the following key observations:

1. This is a key observation
2. This is a key observation
3. This is a key observation

It is important to note here that here is a bunch of filler text to swap out with unique relevancy of observations. It is reproducible in that here are the ways we demonstrated its' reproducibility, however here are some other reasons why some unexpected things happened which we could not have anticipated. In the future, we would do some of these other things differently given appropriate resources in terms of time/money/knowledge/people.

The easiest part of reproducing this paper was this things right here and some other stuff there. This was or was not easy based on our initial assessment of the required tasks.

The most difficult part was this thing here. We did or did not anticipate the difficulty of this, and would advise other groups in the future to consider this carefully. Here's how it might be mitigated.

Data Availability (or lack thereof)

As described in our `Methodology` and `Data` sections, of the datasets referenced by the authors, only one (Brain Tumor dataset) was publicly available.

Given that, upon exploratory data analysis (EDA) we discovered that the dataset did not contain the same number of total records as the one the researchers used, and further given that this dataset included an even further skewed class imbalance (closer to 90/10 than the 80/20 described by the CASS authors), we determined that we could not faithfully recreate the research experiments using this dataset.

This left us to use the MedMNIST dataset which was not described in the paper, but was referenced in the authors' Github starter code and which we proposed might be suitable for our hypothesis.

Starter Code from Researchers

We found the code available to use to be poorly documented. There were two different CASS.ipynb notebooks to use, and the "Getting Started" notebook provided insufficient details for getting CASS up and running using the authors' own code and provided dataset (MedMNIST), let alone tailoring this implementation to other datasets.

In the case of both the MedMNIST and brain tumor datasets, we found ourselves spending in excess of 50 hours attempting to troubleshoot bugs in the given code, or attempting to understand the authors' particular approach to implementing a CNN and Transformer trained with CASS. Factoring into all of this that the brain tumor dataset is in greyscale while the MedMNIST is in color, we found ourselves consumed by "learning on the fly" and attempting to troubleshoot errors rather than recreating a study.

Computing Limitations

One of the benefits of CASS espoused by the authors in their paper is the ability to overcome a lack of computing resources. However, we found this "advantage" to be ironic in that all three of our team members resorted to cloning our test notebook to personal Google accounts and upgrading to paid Google Collab subscriptions in order to buy higher-capacity computing resources than those offered for free with our educational accounts.

The authors utilized an NVIDIA RTX8000 graphical processing unit (GPU) with 48 GB video RAM, 2 PU cores, and 64 GB system RAM.

Team 58 utilized at any given time [10]:

- NVIDIA Tesla T4 GPU with 16 GB RAM, unknown CPU (system) RAM
- NVIDA T100 GPU with 16GB RAM, unknown CPU (system) RAM

- NVIDIA A100 GPU with 80GB RAM, unknown CPU (system) RAM
- Mac M1 Silicon CPU

We frequently found ourselves hitting consumption limits on the Tesla T4 GPU that would boot us from the notebook for the remainder of the day. Paid GPU instances still required us to limit batch sizes in order for the models to run through a single epoch without freezing or crashing.

Suggestions for other teams

References

1. Singh, P. & Cirrone, J. (2023). Efficient Representation Learning for Healthcare with Cross-Architectural Self-Supervision. Proceedings of the 8th Machine Learning for Healthcare Conference, in Proceedings of Machine Learning Research, 219, 691-711. Available from https://proceedings.mlr.press/v219/singh23a.html.

2. Singh, P. (2022). Official PyTorch implementation of CASS. [Software]. Retrieved from https://github.com/pranavsinghps1/CASS.

3. Cheng, J. (2017, April 2). Brain tumor dataset. figshare. Retrieved from https://figshare.com/articles/dataset/brain_tumor_dataset/1512427.

4. Tschandl P., Rosendahl C., & Kittler H. (2018). The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data, 5, 180161. https://doi.org/10.1038/sdata.2018.161.

5. Codella, N. C. F., Gutman, D., Celebi, M. E., Helba, B., Marchetti, M. A., Dusza, S. W., Kalloo, A., Liopyris, K., Mishra, N., Kittler, H., & Halpern, A. (2017). Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC). arXiv:1710.05006. Retrieved from https://arxiv.org/abs/1710.05006.

6. Combalia, M., Codella, N. C. F., Rotemberg, V., Helba, B., Vilaplana, V., Reiter, O., Halpern, A. C., Puig, S., & Malvehy, J. (2019). BCN20000: Dermoscopic Lesions in the Wild. arXiv:1908.02288. Retrieved from https://arxiv.org/abs/1908.02288.

7. Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., & Ni, B. (2023). MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification. Scientific Data.

8. Yang, J., Shi, R., & Ni, B. (2021). MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis. IEEE 18th International Symposium on Biomedical Imaging (ISBI).

9. Bhuvaji, S. (2020). Brain Tumor Classification Using Deep Learning Algorithms. Retrieved from https://github.com/SartajBhuvaji/Brain-Tumor-Classification-Using-Deep-Learning-Algorithms.

10. Google. (2024). Google Cloud Compute Specifications. Retrieved from https://cloud.google.com/compute/docs/gpus.

In [ ]:

```
```