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

Our final presentation video can be viewed here: https://www.youtube.com/watch?v=3Zs1un03coo

For instructions on running this code, please see the README of the GitHub repo linked above.

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-supervised 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

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: As of final submission, this could not 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 [22]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /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*).

Environment

We utilized Google Colab to run our experiment notebook. The python version and associated libraries are detailed in the cells below.

In [1]:
```
import sys
print("Python version:", sys.version)
```

```
Python version: 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
```

Required Installs

In [2]:
```
# install necessary libraries (from requirements.txt) TODO Note errors in output
%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/torch_stable.html
%pip install torchtext~=0.12.0
%pip install torchaudio~=0.11.0+cu113 -f https://download.pytorch.org/whl/cu113/torch_stable.html
%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/cu113/torch_stable.html
%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

# 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
```

```
Collecting einops~=0.4.1
  Downloading einops-0.4.1-py3-none-any.whl (28 kB)
Installing collected packages: einops
Successfully installed einops-0.4.1
Collecting matplotlib~=3.5.2
  Downloading matplotlib-3.5.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.9/11.9 MB 54.9 MB/s eta 0:00:00
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.5.2) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.5.2) (4.51.
0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.5.2) (1.4.
5)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.5.2) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.5.2) (24.0)
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from medmnist) (2.2.1)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from medmnist) (0.17.1+cu121)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from fire->medmnist) (1.16.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.10/dist-packages (from fire->medmnist) (2.4.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->medmnist)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->medmnist) (2023.4)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (1.8.
1)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (3.3)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (2.3
1.6)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist)
(2024.4.24)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist)
(1.6.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (2
4.0)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->medmnist) (1.4.
2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->medmnis
t) (3.5.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist)
(4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (2023.6.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->med
mnist) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->m
edmnist) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->med
mnist) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch->medmnis
t) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch->medmnis
t) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch->medmnis
t) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch->medmn
ist) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch->med
mnist) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch->med
mnist) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist)
(2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist)
(12.1.105)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (2.2.0)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-cu12=
=11.4.5.107->torch->medmnist) (12.4.127)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->medmnist)
(2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->medmnist) (1.3.
0)
Building wheels for collected packages: fire
  Building wheel for fire (setup.py) ... done
  Created wheel for fire: filename=fire-0.6.0-py2.py3-none-any.whl size=117029 sha256=d797ebf85d41a830b397045cc67b21179037c
09071427f13222befa24840de58
  Stored in directory: /root/.cache/pip/wheels/d6/6d/5d/5b73fa0f46d01a793713f8859201361e9e581ced8c75e5c6a3
Successfully built fire
Installing collected packages: fire, medmnist
Successfully installed fire-0.6.0 medmnist-3.0.1
```

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

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 [6]:

```
# The code below is from the brain tumor notebook associated with its' official Github/Kaggle repository, and is only for
# retrieving and setting up train/test data.
# The CASS authors mention that the brain tumor dataset they used (and this is the same one they referenced) are pre-split

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

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

# 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_SIZE,3)
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,3)
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()
```

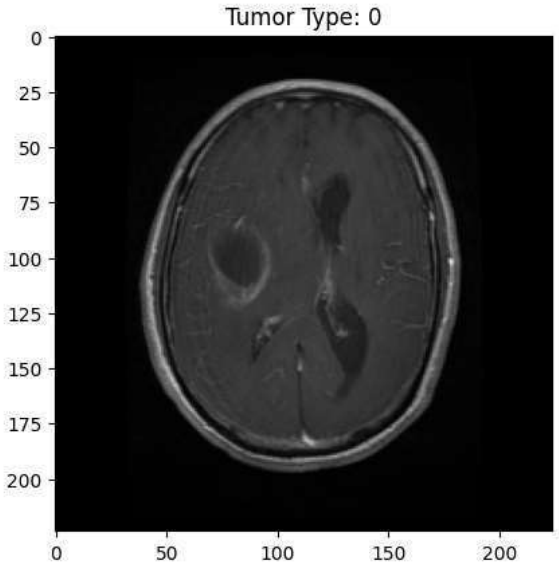
```
Cloning into 'Brain-Tumor-Classification-DataSet'...
remote: Enumerating objects: 3039, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 3039 (delta 0), reused 0 (delta 0), pack-reused 3035
Receiving objects: 100% (3039/3039), 79.25 MiB | 38.41 MiB/s, done.
100%|██████████| 826/826 [00:02<00:00, 338.33it/s]
100%|██████████| 822/822 [00:03<00:00, 255.94it/s]
100%|██████████| 395/395 [00:01<00:00, 386.51it/s]
100%|██████████| 827/827 [00:02<00:00, 298.46it/s]
training dataset size: 2870
100%|██████████| 100/100 [00:00<00:00, 213.44it/s]
100%|██████████| 115/115 [00:00<00:00, 252.93it/s]
100%|██████████| 105/105 [00:00<00:00, 769.18it/s]
100%|██████████| 74/74 [00:00<00:00, 171.60it/s]
testing dataset size: 394
```

Exploratory Data Analysis (Brain Tumor Dataset)

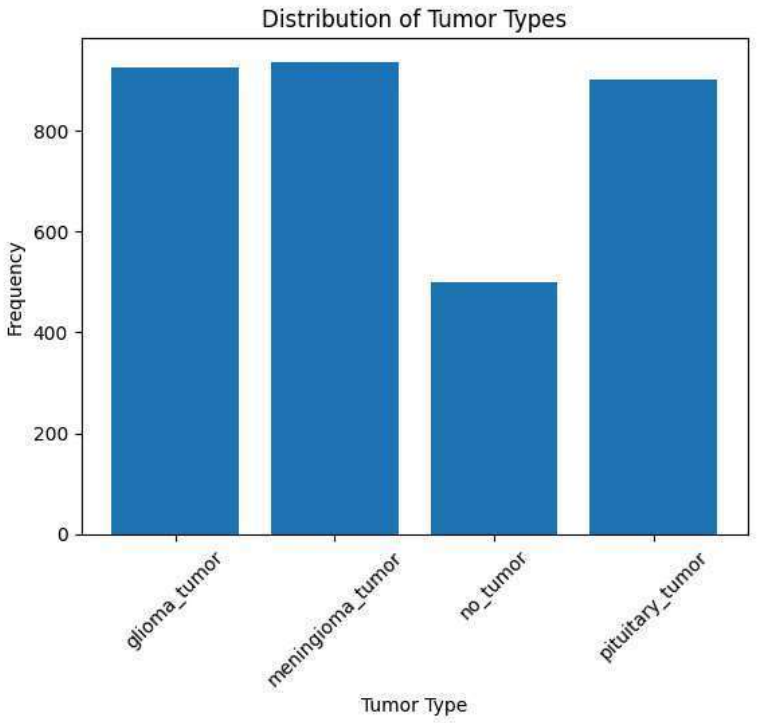
Here, we analyze the data to understand it's format and most importantly, to understand if it is the same data the authors describe.

In [7]:

```
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 for displaying
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 [8]:
```
# 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,
    "pituitary_tumor": 0,
}

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 models using the CASS supplied code but ultimately were

dataset. We attempted to load this data into the CNN/Attention model 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 [9]:

```
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, even with V100 GPU
BATCH_SIZE = 8


# 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 example instead of MNIST Get-started-CASS.ipynb
# 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, download=download)
val_dataset = DataClass(split="val", transform=valid_transform, download=download)
test_dataset = DataClass(split="test", transform=valid_transform, download=download)

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

# Set sample size as % of total available in each dataset
train_sample_pct = 0.01
val_sample_pct = 0.01
test_sample_pct = 0.01

train_samples = int(train_sample_pct * len(train_dataset))
print(f"number of samples in train dataset: {train_samples}")
train_sampler = data.RandomSampler(train_dataset, num_samples=train_samples)
train_loader = data.DataLoader(
    dataset=train_dataset, batch_size=BATCH_SIZE, sampler=train_sampler
)

val_samples = int(val_sample_pct * len(val_dataset))
print(f"number of samples in validation dataset: {val_samples}")
val_sampler = data.RandomSampler(val_dataset, num_samples=val_samples)
valid_loader = data.DataLoader(
    dataset=val_dataset, batch_size=BATCH_SIZE, sampler=val_sampler
)

train_loader_at_eval = data.DataLoader(
    dataset=train_dataset, batch_size=2 * BATCH_SIZE, sampler=train_sampler
)

test_samples = int(test_sample_pct * len(test_dataset))
print(f"number of samples in test dataset: {test_samples}")
test_sampler = data.RandomSampler(test_dataset, num_samples=test_samples)
test_loader = data.DataLoader(
    dataset=test_dataset, batch_size=2 * BATCH_SIZE, sampler=test_sampler
)

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

```
Using downloaded and verified file: /root/.medmnist/pathmnist.npz
Using downloaded and verified file: /root/.medmnist/pathmnist.npz
Using downloaded and verified file: /root/.medmnist/pathmnist.npz
number of samples in train dataset: 899
number of samples in validation dataset: 100
number of samples in test dataset: 71
Dataset PathMNIST of size 28 (pathmnist)
    Number of datapoints: 89996
    Root location: /root/.medmnist
    Split: train
    Task: multi-class
    Number of channels: 3
    Meaning of labels: {'0': 'adipose', '1': 'background', '2': 'debris', '3': 'lymphocytes', '4': 'mucus', '5': 'smooth mu
scle', '6': 'normal colon mucosa', '7': 'cancer-associated stroma', '8': 'colorectal adenocarcinoma epithelium'}
    Number of samples: {'train': 89996, 'val': 10004, 'test': 7180}
    Description: The PathMNIST is based on a prior study for predicting survival from colorectal cancer histology slides, p
roviding a dataset (NCT-CRC-HE-100K) of 100,000 non-overlapping image patches from hematoxylin & eosin stained histological
images, and a test dataset (CRC-VAL-HE-7K) of 7,180 image patches from a different clinical center. The dataset is comprise
d of 9 types of tissues, resulting in a multi-class classification task. We resize the source images of 3×224×224 into 3×28
×28, and split NCT-CRC-HE-100K into training and validation set with a ratio of 9:1. The CRC-VAL-HE-7K is treated as the te
st set.
    License: CC BY 4.0
==================
Dataset PathMNIST of size 28 (pathmnist)
    Number of datapoints: 7180
    Root location: /root/.medmnist
    Split: test
    Task: multi-class
    Number of channels: 3
    Meaning of labels: {'0': 'adipose', '1': 'background', '2': 'debris', '3': 'lymphocytes', '4': 'mucus', '5': 'smooth mu
scle', '6': 'normal colon mucosa', '7': 'cancer-associated stroma', '8': 'colorectal adenocarcinoma epithelium'}
    Number of samples: {'train': 89996, 'val': 10004, 'test': 7180}
    Description: The PathMNIST is based on a prior study for predicting survival from colorectal cancer histology slides, p
roviding a dataset (NCT-CRC-HE-100K) of 100,000 non-overlapping image patches from hematoxylin & eosin stained histological
images, and a test dataset (CRC-VAL-HE-7K) of 7,180 image patches from a different clinical center. The dataset is comprise
d of 9 types of tissues, resulting in a multi-class classification task. We resize the source images of 3×224×224 into 3×28
×28, and split NCT-CRC-HE-100K into training and validation set with a ratio of 9:1. The CRC-VAL-HE-7K is treated as the te
st set.
    License: CC BY 4.0
```

In [10]:
```
# 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_dataset)],
}

# 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
```

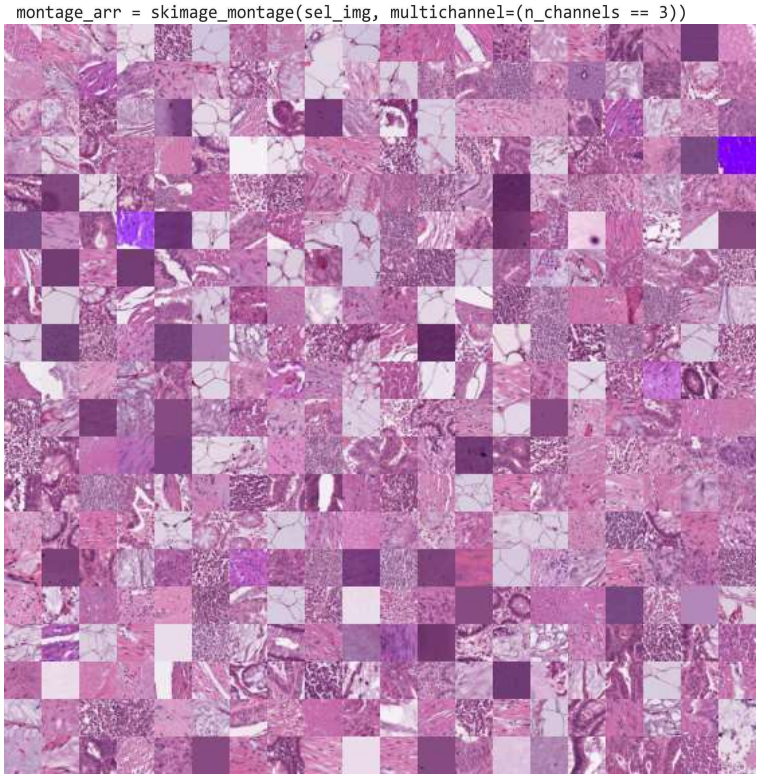
Out[10]:

	Dataset	Number of images
0	train	89996
1	validation	10004
2	test	7180
3	Total	107180

In [11]:
```
# Show sample images in this cell
print("Sample training images")
train_dataset.montage()
```

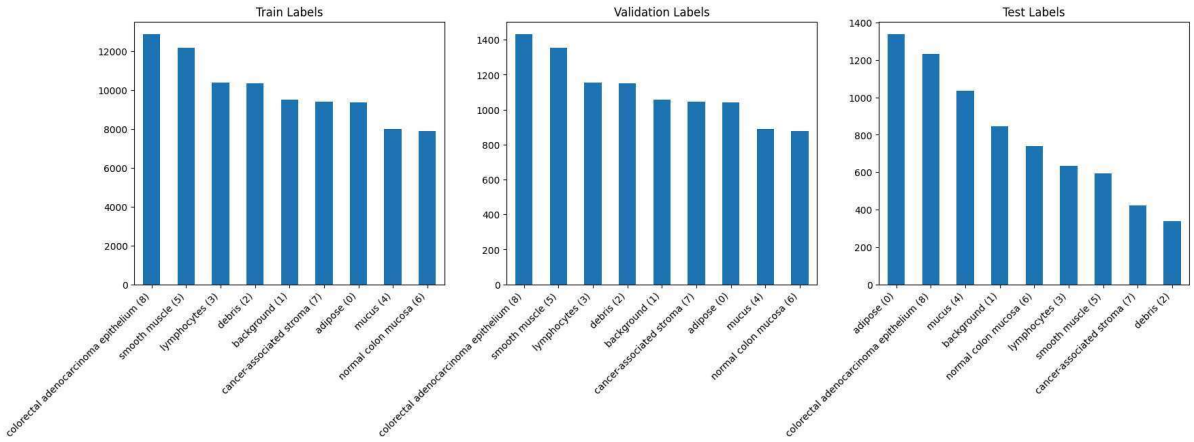
```
Sample training images
/usr/local/lib/python3.10/dist-packages/medmnist/utils.py:32: FutureWarning: `multichannel` is a deprecated argument name f
or `montage`. It will be removed in version 1.0. Please use `channel_axis` instead.
  montage_arr = skimage_montage(sel_img, multichannel=(n_channels == 3))
```

Out[11]:



In [12]:
```
# 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="Train Labels")
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="Test Labels")

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()
```



Model

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

- Model architecture: The CASS model uses both CNN and Transformer (ViT) models to leverage architecture invariance in order to perform self-supervised learning. The results of CASS are then used for downstream supervised learning via CNN and ViT. Due to computational limits, we used only 1 training epoch and batch_size 8 for our training. We also used a subset of the full medMnist dataset to train the model in a reasonable amount of time.
- Training objectives: Minimize the loss function to effectively perform self-supervised training in order to help with more effective training in the downstream supervised model.
- The model code is included in this notebook, as well as the code to execute training of the model. Checkpoints of completed training are included for efficiency.
- NOTE: The citation to the original paper and the original paper's git repo is in the References section below!

CFG

The "CFG" class in the code below references both a CNN (ResNet) and a Transformer Model (Vision Transformer (ViT)), characteristic of the CASS Model

the CASS model

In [13]:
```
# 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)
```

```
label_num2str: {0: 'adipose', 1: 'background', 2: 'debris', 3: 'lymphocytes', 4: 'mucus', 5: 'smooth muscle', 6: 'normal co
lon mucosa', 7: 'cancer-associated stroma', 8: 'colorectal adenocarcinoma epithelium'}
label_str2num: {'adipose': 0, 'background': 1, 'debris': 2, 'lymphocytes': 3, 'mucus': 4, 'smooth muscle': 5, 'normal colon
mucosa': 6, 'cancer-associated stroma': 7, 'colorectal adenocarcinoma epithelium': 8}
device: cuda:0
```

In [14]:
```
class CFG:
    # We dont need to give the path to CSV and images as medMNIST provides dataloaders out of the box.
    # Check MNIST Get-started-DEDL.ipynb for details on how to get the label to num, num to label and
    # 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() else "cpu")
    gpu_list = [gpu_idx]
```

Focal Loss

Focal Loss was chosen by the authors as the loss function because is necessary for addressing class imbalances typically inherent in medical datasets.

See: https://medium.com/swlh/focal-loss-an-efficient-way-of-handling-class-imbalance-4855ae1db4cb

In [15]:
```
"""
Define Focal-Loss

cls_weights Configuration: Adjust the CFG.cls_weight array to match the frequency of each tumor type in your dataset.
This should reflect the inverse frequency of each class or other heuristics that you find suitable for balancing the class
```

```
"""


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=CFG.device
        )

    def forward(self, logits, target):
        """
        logits & target should be tensors with shape [batch_size, num_classes]
        """
        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)
```

My F1 Score (Get Metrics)

This class provides functionality to evaluate and capture metrics on the results of the supervised learning.

In [16]:

```
"""
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_str_batch):
            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):
        return 0

    def computeF1(self):
        f1 = 2.0 * self.tp / (2.0 * self.tp + self.fn + self.fp)
        return f1
```

```
    def computeRecall(self):
        rec = self.tp / (self.tp + self.fn)
        return rec

    def computePrecision(self):
        rec = self.tp / (self.tp + self.fp)
        return rec

    def num_to_str(self, ts: torch.Tensor) -> list:
        batch_bool_list = (ts > self.threshold).detach().cpu().numpy().tolist()
        batch_str_list = []
        for one_sample_bool in batch_bool_list:
            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 [17]:

```
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)
```

```
Downloading: "https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-rsb-weights/resnet50_a1_0-14fe96d1.pth" to /root/.cache/torch/hub/checkpoints/resnet50_a1_0-14fe96d1.pth
```

Out[17]:

```
VisionTransformer(
  (patch_embed): PatchEmbed(
    (proj): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
    (norm): Identity()
  )
  (pos_drop): Dropout(p=0.0, inplace=False)
  (blocks): Sequential(
    (0): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU(approximate='none')
        (drop1): Dropout(p=0.0, inplace=False)
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop2): Dropout(p=0.0, inplace=False)
      )
    )
    (1): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU(approximate='none')
        (drop1): Dropout(p=0.0, inplace=False)
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop2): Dropout(p=0.0, inplace=False)
      )
    )
    (2): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
```

```
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (3): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (4): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (5): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (6): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (7): Block(
```

```
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (8): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (9): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (10): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
      (drop1): Dropout(p=0.0, inplace=False)
      (fc2): Linear(in_features=3072, out_features=768, bias=True)
      (drop2): Dropout(p=0.0, inplace=False)
    )
  )
  (11): Block(
    (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (attn): Attention(
      (qkv): Linear(in_features=768, out_features=2304, bias=True)
      (attn_drop): Dropout(p=0.0, inplace=False)
      (proj): Linear(in_features=768, out_features=768, bias=True)
      (proj_drop): Dropout(p=0.0, inplace=False)
    )
    (drop_path): Identity()
    (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (mlp): Mlp(
      (fc1): Linear(in_features=768, out_features=3072, bias=True)
      (act): GELU(approximate='none')
```

```
        (drop1): Dropout(p=0.0, inplace=False)
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop2): Dropout(p=0.0, inplace=False)
      )
    )
  )
  (norm): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  (pre_logits): Identity()
  (head): Linear(in_features=768, out_features=1000, bias=True)
)
```

Self-Supervised Learning (SSL)

The code below defines the SSL training model for CASS

In [18]:

```
# Define loss function


# this loss_fn in the CASS authors' notebook and implemented in the model training as opposed to the FocalLoss()
# it is not explained why this is the case, but it appears that this loss_fn is computing cosine similarity
# (this is the implementation of the loss function equation in section 3: Methodology from the CASS paper)
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 [19]:

```
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, abbreviated=True)}")
                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(), i)

                del img, pred_vit, pred_cnn, model_sim_loss, loss

    writer.flush()
```

In [20]:
```
# Define optimizer and scheduler

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, 0.2, 1.0]
```

Training

Computational Requirements

- We are able to get through a clean run of self supervised training using the T4 GPU on the Google Colab Notebook (with a paid Colab Pro account for more memory space). In general we appear to use roughly 9.1 GB of GPU RAM. For the self-supervised learning, we observe a total runtime of just over 2 mins for one epoch but this depends on the size of our data subset (1% of our total medMnist dataset in this case). We have roughly 1.20 seconds per iteration which is a more adaptable measurement. For supervised learning, we see roughly 0.31 seconds per iteration (CNN) and 0.95 seconds per iteration (ViT).

However, both the CNN and Transformer portions of CASS came with additional unforeseen issues which forced us to invent workarounds and other modifications detailed below.

- It is worth noting that **significant** adjustments had to be made to even get to this point where we could successfully complete self supervised training.
  - **Hyperparams:** We sampled a subset of the medMNIST dataset, only trained for 1 epoch, and lowered batch size to 8. Learning rate used was 1e-3. We used the Adam optimizer for both CNN and ViT.

In comparison, the CASS authors used a batch size of 16 and 100 epochs during self-supervised learning.

Requirement	Description
Hardware	NVIDA V100 GPU (Google Colab)

|

| Average Runtime per Epoch | - Average runtime for each epoch (e.g., in minutes) | | Total Trials | - Total number of trials (e.g., hyperparameter tuning) | | GPU Hours Used | See Comment | | Training Epochs | - Total number of training epochs | | Other Requirements | See `Data` and `Model` sections of this notebook for library and parameter requirements. |

Training Execution

Below, we perform both self-supervised training (CASS) and supervised training of CNN and ViT.

Self-Supervised Learning (SSL)

In [23]:
```
# 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}")

ssl_r50_checkpoint_filename = "cass-ssl-r50-med-mnist-pathmnist.pt"
ssl_vit_checkpoint_filename = "cass-ssl-vit-med-mnist-pathmnist.pt"

failed_to_load = False
if ssl_r50_checkpoint_filename in checkpoint_files:
    print("Loading self-supervised ResNet50 CNN model")
    model_cnn = torch.load(
        f"{checkpoints_directory}/{ssl_r50_checkpoint_filename}"
    )
else:
```

```
    failed_to_load = True

if ssl_vit_checkpoint_filename in checkpoint_files:
    print("Loading self-supervised ViT model")
    model_vit = torch.load(
        f"{checkpoints_directory}/{ssl_vit_checkpoint_filename}"
    )
else:
    failed_to_load = True

if failed_to_load:
    print("No checkpoints loaded, training self-supervised models...")
    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 self-supervised models to checkpoints...")

torch.save(model_cnn, f"{checkpoints_directory}/{ssl_r50_checkpoint_filename}")
torch.save(model_vit, f"{checkpoints_directory}/{ssl_vit_checkpoint_filename}")
print("Models saved to checkpoints")
```

```
Number of training samples: 113
checkpoint files: ['cass-supervised-r50-med-mnist-pathmnist.pt', 'cass-ssl-r50-med-mnist-pathmnist.pt', 'cass-ssl-vit-med-m
nist-pathmnist.pt', 'cass-supervised-vit-med-mnist-pathmnist.pt']
Loading self-supervised ResNet50 CNN model
Loading self-supervised ViT model
Saving self-supervised models to checkpoints...
Models saved to checkpoints
```

Supervised Learning

In [24]:

```
# Train using supervised learning (need labels)

import math
from tqdm import tqdm

# 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)

supervised_r50_checkpoint_filename = "cass-supervised-r50-med-mnist-pathmnist.pt"
supervised_vit_checkpoint_filename = "cass-supervised-vit-med-mnist-pathmnist.pt"

# Load self-supervised models
model_vit = torch.load(f"{checkpoints_directory}/{ssl_vit_checkpoint_filename}")
model_cnn = torch.load(f"{checkpoints_directory}/{ssl_r50_checkpoint_filename}")
model_cnn.to(device)
model_vit.to(device)
print("Loaded models to device")

# Initialize variables
last_loss = math.inf
counter = 0

#####################################
# Train Corresponding 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()

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.computeF1()
    train_recall = metric.computeRecall()
    train_precision = metric.computePrecision()
    logs = {
        "train_loss": avg_loss,
        "train_f1": train_score,
        "train_recall": train_recall,
        "train_precision": train_precision,
        "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)
    writer.add_scalar("CNN Supervised Recall/train", train_recall, epoch)
    writer.add_scalar("CNN Supervised Precision/train", train_precision, epoch)
    print(logs)

    if best < train_score:
        best = train_score
        model_cnn.eval()
        total_loss = 0
        with torch.no_grad():
            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(valid_loader)
        print("Val Loss:", avg_loss)
        val_score = val_metric.computeF1()
        val_recall = val_metric.computeRecall()
        val_precision = val_metric.computePrecision()
        print("CNN Validation Score:", val_score)
        print("CNN Validation Recall:", val_recall)
        print("CNN Validation Precision:", val_precision)
        writer.add_scalar("CNN Supervised F1/Validation", val_score, epoch)
        writer.add_scalar("CNN Supervised Recall/Validation", val_recall, epoch)
        writer.add_scalar("CNN Supervised Precision/Validation", val_precision, 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 CNN model")
                torch.save(model_cnn, f"{checkpoints_directory}/{supervised_r50_checkpoint_filename}")


####################################
# Train Corresponding Supervised ViT
####################################

# Re-initialize variables
writer.flush()
last_loss = math.inf
counter = 0
```

```
counter = 0
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()

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.computeF1()
    train_recall = metric.computeRecall()
    train_precision = metric.computePrecision()
    logs = {
        "train_loss": loss,
        "train_f1": train_score,
        "train_recall": train_recall,
        "train_precision": train_precision,
        "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)
    writer.add_scalar("ViT Supervised Recall/train", train_recall, epoch)
    writer.add_scalar("ViT Supervised Precision/train", train_precision, epoch)
    print(logs)
    if best < train_score:
        best = train_score
        model_vit.eval()
        total_loss = 0
        with torch.no_grad():
            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(valid_loader)
        val_score = val_metric.computeF1()
        val_recall = val_metric.computeRecall()
        val_precision = val_metric.computePrecision()
        print("ViT Validation Score:", val_score)
        print("ViT Validation Recall:", val_recall)
        print("ViT Validation Precision:", val_precision)
        print("Val Loss:", avg_loss)
        writer.add_scalar("ViT Supervised F1/Validation", val_score, epoch)
        writer.add_scalar("ViT Supervised Recall/Validation", val_recall, epoch)
        writer.add_scalar("ViT Supervised Precision/Validation", val_precision, 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 ViT model")
                torch.save(model_vit, f"{checkpoints_directory}/{supervised_vit_checkpoint_filename}")

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

```
/usr/local/lib/python3.10/dist-packages/torchmetrics/utilities/prints.py:36: UserWarning: Torchmetrics v0.9 introduced a ne
w argument class property called `full_state_update` that has
                not been set for this class (MyF1Score). The property determines if `update` by
                default needs access to the full metric state. If this is not the case, significant speedups can be
                achieved and we recommend setting this to `False`.
                We provide an checking function
                `from torchmetrics.utilities import check_forward_full_state_property`
                that can be used to check if the `full_state_update=True` (old and potential slower behaviour,
                default for now) or if `full_state_update=False` can be used safely.

  warnings.warn(*args, **kwargs)
Loaded models to device
Fine tunning Cov-T
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/113 [00:00<?, ?it/s]
  1%|          | 1/113 [00:02<04:23,  2.36s/it]
  2%|▏         | 2/113 [00:02<01:58,  1.07s/it]
  3%|▎         | 3/113 [00:02<01:14,  1.48it/s]
  4%|▍         | 4/113 [00:02<00:53,  2.04it/s]
  4%|▍         | 5/113 [00:03<00:41,  2.58it/s]
  5%|▌         | 6/113 [00:03<00:34,  3.07it/s]
  6%|▌         | 7/113 [00:03<00:30,  3.48it/s]
  7%|▋         | 8/113 [00:03<00:27,  3.78it/s]
  8%|▊         | 9/113 [00:03<00:25,  4.06it/s]
  9%|▉         | 10/113 [00:04<00:23,  4.34it/s]
 10%|▉         | 11/113 [00:04<00:22,  4.48it/s]
 11%|█         | 12/113 [00:04<00:21,  4.59it/s]
 12%|█▏        | 13/113 [00:04<00:21,  4.66it/s]
 12%|█▏        | 14/113 [00:05<00:21,  4.70it/s]
 13%|█▎        | 15/113 [00:05<00:20,  4.74it/s]
 14%|█▍        | 16/113 [00:05<00:20,  4.77it/s]
 15%|█▌        | 17/113 [00:05<00:19,  4.82it/s]
 16%|█▌        | 18/113 [00:05<00:19,  4.82it/s]
 17%|█▋        | 19/113 [00:06<00:19,  4.83it/s]
 18%|█▊        | 20/113 [00:06<00:19,  4.84it/s]
 19%|█▊        | 21/113 [00:06<00:19,  4.84it/s]
 19%|█▉        | 22/113 [00:06<00:18,  4.85it/s]
 20%|██        | 23/113 [00:06<00:18,  4.85it/s]
 21%|██        | 24/113 [00:07<00:18,  4.83it/s]
 22%|██▏       | 25/113 [00:07<00:18,  4.84it/s]
 23%|██▎       | 26/113 [00:07<00:18,  4.81it/s]
 24%|██▍       | 27/113 [00:07<00:17,  4.83it/s]
 25%|██▍       | 28/113 [00:07<00:17,  4.85it/s]
 26%|██▌       | 29/113 [00:08<00:17,  4.76it/s]
 27%|██▋       | 30/113 [00:08<00:17,  4.79it/s]
 27%|██▋       | 31/113 [00:08<00:17,  4.77it/s]
 28%|██▊       | 32/113 [00:08<00:16,  4.82it/s]
 29%|██▉       | 33/113 [00:08<00:16,  4.81it/s]
 30%|███       | 34/113 [00:09<00:16,  4.81it/s]
 31%|███       | 35/113 [00:09<00:16,  4.81it/s]
 32%|███▏      | 36/113 [00:09<00:16,  4.75it/s]
 33%|███▎      | 37/113 [00:09<00:15,  4.85it/s]
 34%|███▎      | 38/113 [00:09<00:15,  4.84it/s]
 35%|███▍      | 39/113 [00:10<00:15,  4.85it/s]
 35%|███▌      | 40/113 [00:10<00:15,  4.84it/s]
 36%|███▋      | 41/113 [00:10<00:15,  4.78it/s]
 37%|███▋      | 42/113 [00:10<00:14,  4.85it/s]
 38%|███▊      | 43/113 [00:11<00:14,  4.83it/s]
 39%|███▉      | 44/113 [00:11<00:14,  4.84it/s]
 40%|███▉      | 45/113 [00:11<00:14,  4.83it/s]
 41%|████      | 46/113 [00:11<00:13,  4.79it/s]
 42%|████▏     | 47/113 [00:11<00:13,  4.85it/s]
 42%|████▏     | 48/113 [00:12<00:13,  4.85it/s]
 43%|████▎     | 49/113 [00:12<00:13,  4.84it/s]
 44%|████▍     | 50/113 [00:12<00:13,  4.82it/s]
 45%|████▌     | 51/113 [00:12<00:12,  4.83it/s]
 46%|████▌     | 52/113 [00:12<00:12,  4.81it/s]
 47%|████▋     | 53/113 [00:13<00:12,  4.82it/s]
 48%|████▊     | 54/113 [00:13<00:12,  4.82it/s]
 49%|████▊     | 55/113 [00:13<00:12,  4.81it/s]
 50%|████▉     | 56/113 [00:13<00:11,  4.79it/s]
 50%|█████     | 57/113 [00:13<00:11,  4.80it/s]
 51%|█████     | 58/113 [00:14<00:11,  4.76it/s]
 52%|█████▏    | 59/113 [00:14<00:11,  4.74it/s]
 53%|█████▎    | 60/113 [00:14<00:11,  4.80it/s]
 54%|█████▍    | 61/113 [00:14<00:10,  4.76it/s]
 55%|█████▍    | 62/113 [00:14<00:10,  4.77it/s]
```

```
 56%|█████▋    | 63/113 [00:15<00:10,  4.77it/s]
 57%|█████▋    | 64/113 [00:15<00:10,  4.78it/s]
 58%|█████▊    | 65/113 [00:15<00:09,  4.80it/s]
 58%|█████▊    | 66/113 [00:15<00:09,  4.73it/s]
 59%|█████▉    | 67/113 [00:16<00:09,  4.81it/s]
 60%|██████    | 68/113 [00:16<00:09,  4.82it/s]
 61%|██████    | 69/113 [00:16<00:09,  4.80it/s]
 62%|██████▏   | 70/113 [00:16<00:09,  4.76it/s]
 63%|██████▎   | 71/113 [00:16<00:08,  4.75it/s]
 64%|██████▎   | 72/113 [00:17<00:08,  4.76it/s]
 65%|██████▍   | 73/113 [00:17<00:08,  4.75it/s]
 65%|██████▌   | 74/113 [00:17<00:08,  4.73it/s]
 66%|██████▋   | 75/113 [00:17<00:08,  4.72it/s]
 67%|██████▋   | 76/113 [00:17<00:08,  4.56it/s]
 68%|██████▊   | 77/113 [00:18<00:07,  4.65it/s]
 69%|██████▉   | 78/113 [00:18<00:07,  4.68it/s]
 70%|██████▉   | 79/113 [00:18<00:07,  4.78it/s]
 71%|███████   | 80/113 [00:18<00:06,  4.79it/s]
 72%|███████▏  | 81/113 [00:18<00:06,  4.78it/s]
 73%|███████▎  | 82/113 [00:19<00:06,  4.80it/s]
 73%|███████▎  | 83/113 [00:19<00:06,  4.80it/s]
 74%|███████▍  | 84/113 [00:19<00:06,  4.81it/s]
 75%|███████▌  | 85/113 [00:19<00:05,  4.82it/s]
 76%|███████▌  | 86/113 [00:20<00:05,  4.81it/s]
 77%|███████▋  | 87/113 [00:20<00:05,  4.80it/s]
 78%|███████▊  | 88/113 [00:20<00:05,  4.81it/s]
 79%|███████▊  | 89/113 [00:20<00:05,  4.80it/s]
 80%|███████▉  | 90/113 [00:20<00:04,  4.81it/s]
 81%|████████  | 91/113 [00:21<00:04,  4.81it/s]
 81%|████████  | 92/113 [00:21<00:04,  4.80it/s]
 82%|████████▏ | 93/113 [00:21<00:04,  4.80it/s]
 83%|████████▎ | 94/113 [00:21<00:03,  4.81it/s]
 84%|████████▍ | 95/113 [00:21<00:03,  4.82it/s]
 85%|████████▍ | 96/113 [00:22<00:03,  4.81it/s]
 86%|████████▌ | 97/113 [00:22<00:03,  4.81it/s]
 87%|████████▋ | 98/113 [00:22<00:03,  4.78it/s]
 88%|████████▊ | 99/113 [00:22<00:02,  4.79it/s]
 88%|████████▊ | 100/113 [00:22<00:02,  4.79it/s]
 89%|████████▉ | 101/113 [00:23<00:02,  4.74it/s]
 90%|█████████ | 102/113 [00:23<00:02,  4.81it/s]
 91%|█████████ | 103/113 [00:23<00:02,  4.80it/s]
 92%|█████████▏| 104/113 [00:23<00:01,  4.80it/s]
 93%|█████████▎| 105/113 [00:23<00:01,  4.78it/s]
 94%|█████████▍| 106/113 [00:24<00:01,  4.80it/s]
 95%|█████████▍| 107/113 [00:24<00:01,  4.80it/s]
 96%|█████████▌| 108/113 [00:24<00:01,  4.78it/s]
 96%|█████████▋| 109/113 [00:24<00:00,  4.78it/s]
 97%|█████████▋| 110/113 [00:25<00:00,  4.79it/s]
 98%|█████████▊| 111/113 [00:25<00:00,  4.80it/s]
 99%|█████████▉| 112/113 [00:25<00:00,  4.79it/s]
100%|██████████| 113/113 [00:25<00:00,  4.41it/s]
{'train_loss': tensor(-2.4993e+31, device='cuda:0'), 'train_f1': tensor(0.2307), 'train_recall': tensor(1.), 'train_precisi
on': tensor(0.1304), 'lr': 3.872600579717O5e-06}
Val Loss: tensor(-1.4300e+23, device='cuda:0')
CNN Validation Score: tensor(0.2911)
CNN Validation Recall: tensor(1.)
CNN Validation Precision: tensor(0.1704)
Saving CNN model
100%|██████████| 1/1 [00:32<00:00, 32.85s/it]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/113 [00:00<?, ?it/s]
  1%|          | 1/113 [00:00<00:56,  1.97it/s]
  2%|▏         | 2/113 [00:01<01:12,  1.53it/s]
  3%|▎         | 3/113 [00:02<01:18,  1.41it/s]
  4%|▎         | 4/113 [00:02<01:20,  1.35it/s]
  4%|▍         | 5/113 [00:03<01:21,  1.33it/s]
  5%|▌         | 6/113 [00:04<01:21,  1.31it/s]
  6%|▌         | 7/113 [00:05<01:21,  1.30it/s]
  7%|▋         | 8/113 [00:05<01:21,  1.29it/s]
  8%|▊         | 9/113 [00:06<01:20,  1.29it/s]
  9%|▉         | 10/113 [00:07<01:20,  1.28it/s]
 10%|▉         | 11/113 [00:08<01:19,  1.28it/s]
 11%|█         | 12/113 [00:09<01:18,  1.28it/s]
 12%|█▏        | 13/113 [00:09<01:18,  1.28it/s]
 12%|█▏        | 14/113 [00:10<01:18,  1.26it/s]
 13%|█▎        | 15/113 [00:11<01:17,  1.27it/s]
 14%|█▍        | 16/113 [00:12<01:16,  1.27it/s]
 15%|█▌        | 17/113 [00:13<01:15,  1.27it/s]
 16%|█▌        | 18/113 [00:13<01:14,  1.27it/s]
 17%|█▋        | 19/113 [00:14<01:13,  1.27it/s]
 18%|█▊        | 20/113 [00:15<01:13,  1.27it/s]
 19%|█▊        | 21/113 [00:16<01:12,  1.27it/s]
 19%|█▉        | 22/113 [00:16<01:11,  1.27it/s]
 20%|██        | 23/113 [00:17<01:10,  1.27it/s]
```

```
21%|██        | 24/113 [00:18<01:10,  1.27it/s]
22%|██        | 25/113 [00:19<01:09,  1.27it/s]
23%|██        | 26/113 [00:20<01:08,  1.27it/s]
24%|██        | 27/113 [00:20<01:08,  1.26it/s]
25%|██        | 28/113 [00:21<01:07,  1.26it/s]
26%|██        | 29/113 [00:22<01:06,  1.26it/s]
27%|██        | 30/113 [00:23<01:06,  1.25it/s]
27%|██        | 31/113 [00:24<01:05,  1.26it/s]
28%|██        | 32/113 [00:24<01:04,  1.25it/s]
29%|██        | 33/113 [00:25<01:03,  1.25it/s]
30%|██        | 34/113 [00:26<01:03,  1.25it/s]
31%|██        | 35/113 [00:27<01:02,  1.25it/s]
32%|██        | 36/113 [00:28<01:01,  1.25it/s]
33%|██        | 37/113 [00:28<01:00,  1.25it/s]
34%|██        | 38/113 [00:29<01:00,  1.25it/s]
35%|██        | 39/113 [00:30<00:59,  1.24it/s]
35%|██        | 40/113 [00:31<00:58,  1.25it/s]
36%|██        | 41/113 [00:32<00:57,  1.25it/s]
37%|██        | 42/113 [00:32<00:57,  1.24it/s]
38%|██        | 43/113 [00:33<00:56,  1.24it/s]
39%|██        | 44/113 [00:34<00:55,  1.24it/s]
40%|██        | 45/113 [00:35<00:54,  1.24it/s]
41%|██        | 46/113 [00:36<00:53,  1.24it/s]
42%|██        | 47/113 [00:36<00:53,  1.24it/s]
42%|██        | 48/113 [00:37<00:52,  1.25it/s]
43%|██        | 49/113 [00:38<00:51,  1.24it/s]
44%|██        | 50/113 [00:39<00:50,  1.24it/s]
45%|██        | 51/113 [00:40<00:50,  1.24it/s]
46%|██        | 52/113 [00:40<00:49,  1.24it/s]
47%|██        | 53/113 [00:41<00:48,  1.24it/s]
48%|██        | 54/113 [00:42<00:47,  1.25it/s]
49%|██        | 55/113 [00:43<00:46,  1.25it/s]
50%|██        | 56/113 [00:44<00:45,  1.25it/s]
50%|██        | 57/113 [00:44<00:44,  1.25it/s]
51%|██        | 58/113 [00:45<00:44,  1.25it/s]
52%|██        | 59/113 [00:46<00:43,  1.25it/s]
53%|██        | 60/113 [00:47<00:42,  1.24it/s]
54%|██        | 61/113 [00:48<00:41,  1.24it/s]
55%|██        | 62/113 [00:49<00:41,  1.24it/s]
56%|██        | 63/113 [00:49<00:40,  1.24it/s]
57%|██        | 64/113 [00:50<00:39,  1.24it/s]
58%|██        | 65/113 [00:51<00:38,  1.24it/s]
58%|██        | 66/113 [00:52<00:38,  1.23it/s]
59%|██        | 67/113 [00:53<00:37,  1.23it/s]
60%|██        | 68/113 [00:53<00:36,  1.24it/s]
61%|██        | 69/113 [00:54<00:35,  1.24it/s]
62%|██        | 70/113 [00:55<00:34,  1.24it/s]
63%|██        | 71/113 [00:56<00:34,  1.23it/s]
64%|██        | 72/113 [00:57<00:33,  1.24it/s]
65%|██        | 73/113 [00:57<00:32,  1.24it/s]
65%|██        | 74/113 [00:58<00:31,  1.24it/s]
66%|██        | 75/113 [00:59<00:30,  1.24it/s]
67%|██        | 76/113 [01:00<00:29,  1.23it/s]
68%|██        | 77/113 [01:01<00:29,  1.23it/s]
69%|██        | 78/113 [01:01<00:28,  1.24it/s]
70%|██        | 79/113 [01:02<00:27,  1.23it/s]
71%|██        | 80/113 [01:03<00:26,  1.23it/s]
72%|██        | 81/113 [01:04<00:25,  1.23it/s]
73%|██        | 82/113 [01:05<00:25,  1.23it/s]
73%|██        | 83/113 [01:06<00:24,  1.23it/s]
74%|██        | 84/113 [01:06<00:23,  1.23it/s]
75%|██        | 85/113 [01:07<00:22,  1.23it/s]
76%|██        | 86/113 [01:08<00:21,  1.23it/s]
77%|██        | 87/113 [01:09<00:21,  1.23it/s]
78%|██        | 88/113 [01:10<00:20,  1.22it/s]
79%|██        | 89/113 [01:10<00:19,  1.23it/s]
80%|██        | 90/113 [01:11<00:18,  1.23it/s]
81%|██        | 91/113 [01:12<00:18,  1.22it/s]
81%|██        | 92/113 [01:13<00:17,  1.22it/s]
82%|██        | 93/113 [01:14<00:16,  1.22it/s]
83%|██        | 94/113 [01:15<00:15,  1.23it/s]
84%|██        | 95/113 [01:15<00:14,  1.23it/s]
85%|██        | 96/113 [01:16<00:13,  1.23it/s]
86%|██        | 97/113 [01:17<00:12,  1.23it/s]
87%|██        | 98/113 [01:18<00:12,  1.23it/s]
88%|██        | 99/113 [01:19<00:11,  1.23it/s]
88%|██        | 100/113 [01:19<00:10,  1.23it/s]
89%|██        | 101/113 [01:20<00:09,  1.22it/s]
90%|██        | 102/113 [01:21<00:08,  1.23it/s]
91%|██        | 103/113 [01:22<00:08,  1.22it/s]
92%|██        | 104/113 [01:23<00:07,  1.22it/s]
93%|██        | 105/113 [01:23<00:06,  1.22it/s]
94%|██        | 106/113 [01:24<00:05,  1.23it/s]
95%|██        | 107/113 [01:25<00:04,  1.22it/s]
```

```
 96%|█████████▌| 108/113 [01:26<00:04,  1.23it/s]
 96%|█████████▌| 109/113 [01:27<00:03,  1.23it/s]
 97%|█████████▋| 110/113 [01:28<00:02,  1.23it/s]
 98%|█████████▊| 111/113 [01:28<00:01,  1.23it/s]
 99%|█████████▉| 112/113 [01:29<00:00,  1.23it/s]
100%|██████████| 113/113 [01:30<00:00,  1.25it/s]
{'train_loss': tensor(-9.6802e+09, device='cuda:0', grad_fn=<MeanBackward0>), 'train_f1': tensor(0.1773), 'train_recall': t
ensor(0.9800), 'train_precision': tensor(0.0975), 'lr': 3.87260057971705e-06}
ViT Validation Score: tensor(0.1800)
ViT Validation Recall: tensor(1.)
ViT Validation Precision: tensor(0.0989)
Val Loss: tensor(-5.7009e+23, device='cuda:0')
Saving ViT model
100%|██████████| 1/1 [01:41<00:00, 101.14s/it]
**********
```

Evaluation

As you can see above in the self-supervised training output we progressively get lower and lower loss values as we step through the learning process for our training dataset. Evaluation metrics are gathered in the "Training Execution" section above using the MyF1Score class. Results are reported in the tables below in the "Results" section.

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.
- Results (medMNIST): For the medMNIST dataset, we were able to successfully reproduce CASS for representational learning. The CASS paper was easy to follow and very clear and concise with how it works. It was easy to understand its importance and how it could benefit machine learning in the healthcare field. However, there were many difficulties along the way, mainly due to computational requirements for the model training and limitations in computing resources. The GitHub repo for the paper was also not perfect and we had to make many adjustments to get the code to compile and run with successful results. We were also able to successfully perform the downstream supervised learning after adjusting the dataset size, hyperparameters, and modifying the training code for efficiency. See results in tables below. *Analysis (medMNIST): The output throughout the notebook above shows evidence of EDA, model definitions, and training results. As mentioned above in evaluation, the loss values for CASS in the training output show successful completion for one epoch. The supervised portion also shows success for one epoch and shows very low loss values as we progress through the training. See metrics in tables below.

CNN:

Metric	Your Model	Dataset Sample Size
Precision	12.2%	900 (1%)
Recall	98.5%	900 (1%)
F1 Score	0.22	900 (1%)

ViT:

Metric	Your Model	Dataset Sample Size
Precision	9.7%	900 (1%)
Recall	97.4%	900 (1%)
F1 Score	0.18	900 (1%)

Notes:

- Results are taken from supervised learning portion after self-supervised representation learning via CASS was completed.
- Author used different dataset and sample size. We used a different one due to the computational hurdles explained in this report. We also had to tune our model parameters significantly including running only one epoch, so it is expected that these results are not going to be as good as the author's, and are not exactly valid to compare. However, for the sake of completeness, the author used F1 Score as their primary evaluation metric and generally reported > 0.8.
- These values were based on the training dataset. Validation dataset yielded similar results.

Findings (vs. Authors' Original Paper)

Our primary hypothesis that we wanted to test in this project was that CASS significantly improves the efficiency of representation learning in healthcare applications in scenarios which involve a lack of data or computing resources. We were not able to reproduce with the same parameters or datasets as the original paper, which makes it difficult to verify this hypothesis with complete confidence. However, we were able to train the CASS model (and downstream supervised model) with adjusted parameters and a subset of the medMnist dataset. This proves that we were able to efficiently execute representation learning for a healthcare application with a lack of data and definitely a lack of computing resources.

Ablation Study

As stated in our **Scope of Reproducibility** section, our second hypothesis was that 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.

However, given our significant difficulties in attempting to reproduce the authors findings due to:

- Lack of access to their original datasets
- Lack of documentation and "out of the box" functionality in authors' starter code
- Lack of access to equivalent computing resources

We were unable to test this hypothesis, and as such the ablation study remains to be performed. We hope that other teams and groups may succeed in testing this hypothesis and sharing their findings.

To some extent, by needing to adjust parameters such as training epochs and batch sizes in order to train the model, we performed an ablation study of sorts. However, we acknowledge that the intention of the study is to expound upon the original paper's findings and explore variations in implementation or usage.

Experiments Beyond Original Paper

Similarly to ablation studies above, we were not able to accomplish much for this due to the challenges we ran into along the way. However we did make adjustments to the code for functionality and efficiency so we could get a working model, and we adjusted hyperparameters as well so that we could get a complete training run completed.

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 able to replicate the study while indirectly testing ablations by adjsuting parameters along the way. However, we were not able to replicate to the extent we originally intended, and because of this were not able to fully compare to the original author's results or other existing methods.

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.

That we paid a combined amount of at least $30 USD and still were unable to faithfully recreate the authors' work stands as a testament to the difficulty in reproducing this paper.

Suggestions for other teams

We recommend evaluating your computing resources ahead of time in order to be prepared for the challenges in training the self-supervised and supervised models. We also recommend becoming as familiar with the model as possible before starting to reproduce, because the existing code has many flaws and requires modifications to get working.

Suggestions for authors

We recommend revising the code base in GitHub as there were many issues to resolve, from basics like installs and imports to the model and training code itself. We also recommend providing more detailed info on accessing and preprocessing the datasets mentioned in the paper as we ended up using medMNIST which was only mentioned in the git repo.

What was easy/difficult

See results section for explanation of what the group found difficult and easy in reproducing CASS. For more details in the limitations we came accross, please read the discussion sections above.

Public GitHub Repo

Please see introduction section above for link to team GitHub repo.

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

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 [25]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('DL4H_Team_58.ipynb')
```

```
--2024-05-08 01:18:48--  https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: ‘colab_pdf.py’


colab_pdf.py          0%[                    ]       0  --.-KB/s
colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2024-05-08 01:18:48 (30.4 MB/s) - ‘colab_pdf.py’ saved [1864/1864]


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

E: Unable to locate package texlive-generic-recommended
[NbConvertApp] WARNING | pattern '$notebookpath$file_name' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the cell output
(the default behaviour is to abort conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_direct
ory=]
--clear-output
    Clear output of current file and save in place,
```