Introduction

Team 144 bcorn2@uiuc.edu

Github Repo: https://github.com/benpcorn/DLH598-Team144 (https://github.com/benpcorn/DLH598-Team144)

Paper Repo: https://github.com/tufts-ml/SAMIL/tree/main (https://github.com/tufts-ml/SAMIL/tree/main)

Problem

Huang, Zhe, Wessler, Benjamin S., and Hughes, Michael C. (2023) – Detecting Heart Disease from Multi-View Ultrasound Images via Supervised Attention Multiple Instance Learning describes the clinical problem of under-diagnosis and under-treatment of aortic stenosis (AS), a degenerative valve condition. In clinical practice, AS is diagnosed by manual expert view of a transthoracic echocardiography (TTE) – which uses ultrasound to produce many images of the heart. AS can be treated effectively, but requires identification early on. If left untreated, "severe AS has lower 5-year survival rates than several metastatic cancers" (Huang et al., 2021). When treated, AS has a low mortality rate, but up to 2/3 of symptomatic AS pateints go undiagnosed (Huang et al., 2021). Automatic screening of AS from transthoracic echocardiography imagery can improve the rate of detection and decrease mortality.

Paper Explanation

The challenge with automatic detection is each TTE “consists of dozens of images or videos (typically 27-97 in our data) that show the heart’s complex anatomy from different acquisition angles” (Huang, Wessler, and Hughes, 2023) where a clinical expert identifies imagery where the aortic valve is clearly visible, then assesses the severity on a 3-level scale (no, early, significant disease). Traditional Deep Learning approaches classify a single image with a single result, however the clinical expert review makes a single “coherent prediction” (Huang, Wessler, and Hughes, 2023) from knowledge gathered from the set of images. Additionally, the image views produced by a TTE are often unlabeled in Electronic Health Records, further complicating any Deep Learning approaches.

The paper finds previous approaches to automatic detection such as attention-based multiple instance learning (MIL) to be insufficient based on accuracy and detection yield, and explores a novel MIL approach to improve the detection of AS from automatic detection that mimics the methodology of a clinical expert.

The paper outlines two novel contributions to automatic AS detection:

1. Supervised attention mechanism that identifies relevant TTE views (often unlabeled),

mimicking human filtering done by a clinical expert. This is accomplished by introducing a new loss term, “supervised attention (SA)”, to match attention weights to the relevance scores from a View Relevance classifier.

2. Self-supervised pretraining strategy through contrastive learning on the embedding of

the entire TTE study (i.e., a “bag of images”) – compared to traditional pretraining strategies which focus on individual images.

Paper Results

The paper uses *balanced accuracy* as the performance metric due to the class imbalance in the TMED-2 dataset -- making standard accuracy "less suitable" (Huang et al., 2023). The proposed method (SAMIL) was compared to general-purpose multi-instance algorithms and prior methods for AS diagnosis using deep neural networks.

SAMIL performed much better (76% balanced accuracy) than 4 other state-of-the-art attention-based MIL architectures tested vs. a range of 60-67% balanced accuracy for existing algorithms.

The chart below from the original paper outlines the balanced accuracy of SAMIL against other approaches dedicated to AS diagnosis (*Filter then Average* and *Weighted Average by View Relevance*), and other general approaches including ABMIL, Set Transformer, and DSMIL.

(Huang et al., 2023)

Scope of Reproducibility:

The scope of this project is to reproduce the original claims in the paper. Using the existing code provided by the authors of the paper, each model will be trained using the TMED-2 dataset and the paper's claimed Balanced Accuracy scores will be compared to the results of our training.

While the original paper compares SAMIL to ABMIL and DSMIL, only ABMIL will be compared. Additionally, only Split 1 will be trained due to the computational requirements needed to train just a single split - training all three is not feasible at this time.

Hypotheses To Be Tested

1. A supervised attention mechanism will provide significant improvements over

standard MIL approaches in AS detection rates and detection accuracy, with a smaller model size. 2. Self-supervised pretraining of “study-level” TTE artifacts provides improvements in AS detection rates and detection accuracy over traditional “image-level” pretraining, or no pretraining at all.

Planned Ablations

The paper has two ablations targeting the attention strategy and the pretraining strategy.

1. Attention: The attention mechanisms within the pooling layer σ to be tested are the

baseline ABMIL model, ABMIL with gated attention, and the SAMIL model without pretraining. The paper compares the performance of these three approaches and identifies that SAMIL’s supervised attention model outperforms ABMIL (the baseline model that SAMIL builds upon) by +1200 bps. The Github repo scripts includes parameters to control the attention mechanism for ABMIL (gated_attention vs. attention), and SAMIL with and without pretraining. 2. Pre Training: The paper introduces a novel approach of built-in study-level (i.e., bag-level) pretraining. This ablation compares different pretraining strategies including: image-level contrastive learning and no pretraining to the study-level pretraining approach. The paper finds no improvements with image-level pretraining, but the study-level pretraining shows improvements of +480 bps. The Github repo scripts include parameters to control pretraining options of: study level, image level, and none.

Methodology

To reproduce this paper, the following pre-requisites must be acquired:

1. Access to the TMED-2 dataset here (https://tmed.cs.tufts.edu/tmed_v2.html)
2. Download the pretrained view classifiers, MOCO pretrained checkpoints, and training curves of SAMIL from the paper's Github repo here (https://tufts.box.com/s/c5w8123j7h3dpls75jye1363uh8qv8us). Once downloaded, upload the entire unzipped folder to your Google Drive (see path below).

The methodology for reproduction is as follows:

1. Create and train the ABMIL model
2. Create and train the SAMIL model with no Pretraining
3. Train the SAMIL model with Image Level Pretraining
4. Train the SAMIL model with Study Level Pretraining

The model definitions and helper methods are pulled from the paper's Github repo.

In [ ]:
```
import os
import zipfile
from google.colab import drive
import warnings
warnings.filterwarnings('ignore')

drive.mount('/content/drive', force_remount=True)

# Modify the paths below if they differ from your upload locations.
# Assumes the SAMIL Github repo has been cloned and uploaded to drive in the `SAMIL
MODEL_CHECKPOINTS = '/content/drive/MyDrive/SAMIL/model_checkpoints'
ROOT_DIR = '/content/drive/MyDrive/SAMIL'
DATA_INFO_DIR = '/content/drive/MyDrive/SAMIL/data_info'
DATA_DIR = '/content/drive/MyDrive/DL4H-TMED2/'

with zipfile.ZipFile(DATA_DIR + 'labeled.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/data')

with zipfile.ZipFile(DATA_DIR + 'unlabeled.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/data')

LOCAL_DATA_DIR = '/content/data/'
```

```
Mounted at /content/drive
```

Environment Setup

In [ ]:
```
import math
import glob
import random

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision
from torchvision import transforms
from tqdm import tqdm

trainingSeed = 0
batchSize = 1
numWorkers = 8
random.seed(trainingSeed)
np.random.seed(trainingSeed)
torch.manual_seed(trainingSeed)

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

Data

The dataset used by the paper is the TMED-2 dataset (https://tmed.cs.tufts.edu/tmed_v2.html), containing transthoracic echocardiogram (TTE) imagery from routine care of patients at Tufts Medical Center.

The paper uses the (`view_and_diagnosis_labeled_set`) from TMED-2, consisting of 599 studies from 577 patients. The patients are labeled by board certified medical staff with the following values: none, early AS, or significant AS. The dataset has been partioned into different splits, each containing 360 training studies, 119 validation studies, and 120 test studies.

Code blocks below should only be executed after you have acquired the TMED-2 dataset and uploaded the `view_and_diagnosis_labeled_set` folder to Drive.

In [ ]:
```
labeled_dir = '/content/drive/MyDrive/DL4H-TMED2/labeled'
unlabeled_dir = '/content/drive/MyDrive/DL4H-TMED2/unlabeled'

# Assumes the `view_and_diagnosis_labeled_set` is uploaded to a folder named `DL4H-

TMED2SummaryTable = pd.read_csv(os.path.join(DATA_INFO_DIR, 'TMED2SummaryTable.csv'
SEED_DIR = DATA_INFO_DIR + '/DataPartition/seed0/DEV479/FullyLabeledSet_studies'

train_PatientStudy_list = pd.read_csv(os.path.join(SEED_DIR, "train_studies.csv"))
val_PatientStudy_list = pd.read_csv(os.path.join(SEED_DIR, "val_studies.csv"))
test_PatientStudy_list = pd.read_csv(os.path.join(SEED_DIR, "test_studies.csv"))

train_PatientStudy_ids = train_PatientStudy_list["study"].values
val_PatientStudy_ids = val_PatientStudy_list["study"].values
test_PatientStudy_ids = test_PatientStudy_list["study"].values

#Debug
#print(train_PatientStudy_ids)
# print(val_PatientStudy_ids)
# print(test_PatientStudy_ids)
```

EchoDataset

The class below is directly from the paper repo and handles transforming and loading the TMED-2 image data.

In [ ]:
```
from PIL import Image
from torch.utils.data import Dataset

DiagnosisStr_to_Int_Mapping={
    'no_AS':0,
    'mild_AS':1,
    'mildtomod_AS':1,
    'moderate_AS':2,
    'severe_AS':2

}

class EchoDataset(Dataset):
    def __init__(self, PatientStudy_list, TMED2SummaryTable, ML_DATA_dir, sampling_

        self.PatientStudy_list = PatientStudy_list
        self.TMED2SummaryTable = TMED2SummaryTable #note: using the patient_id colu

        self.ML_DATA_dir = ML_DATA_dir

        self.sampling_strategy = sampling_strategy

        self.training_seed=training_seed

        self.transform_fn = transform_fn

        self.bag_of_PiatentStudy_images, self.bag_of_PatientStudy_DiagnosisLabels =



    def _create_bags(self):

        bag_of_PatientStudy_images = []
        bag_of_PatientStudy_DiagnosisLabels = []

        for PatientStudy in self.PatientStudy_list:
            this_PatientStudyRecords_from_TMED2SummaryTable = self.TMED2SummaryTabl
            assert this_PatientStudyRecords_from_TMED2SummaryTable.shape[0]!=0, 'ev

            this_PatientStudyRecords_from_TMED2SummaryTable_DiagnosisLabel = list(s
            assert len(this_PatientStudyRecords_from_TMED2SummaryTable_DiagnosisLab

            this_PatientStudy_DiagnosisLabel = this_PatientStudyRecords_from_TMED2S
            this_PatientStudy_DiagnosisLabel = DiagnosisStr_to_Int_Mapping[this_Pat

            this_PatientStudy_Id_ImagesPattern = PatientStudy + "_*.png"
            this_PatientStudy_Id_LabeledImages: list[str] = glob.glob(pathname=this_
            this_PatientStudy_Id_UnlabeledImages: list[str] = glob.glob(pathname=th

            # From paper repo, sort to ensure order of images are consistent each r
            this_PatientStudy_Id_LabeledImages.sort()
            this_PatientStudy_Id_UnlabeledImages.sort()

            this_PatientStudyImages = []

            for ImagePath in this_PatientStudy_Id_LabeledImages:
                this_PatientStudyImages.append(
                    np.array(Image.open(self.ML_DATA_dir + '/labeled/' + ImagePath)
                )
```

```
            for ImagePath in this_PatientStudy_Id_UnlabeledImages:
                this_PatientStudyImages.append(
                    np.array(Image.open(self.ML_DATA_dir + '/unlabeled/' + ImagePat
                )

            bag_of_PatientStudy_images.append(np.array(this_PatientStudyImages))
            bag_of_PatientStudy_DiagnosisLabels.append(this_PatientStudy_DiagnosisL

        return bag_of_PatientStudy_images, bag_of_PatientStudy_DiagnosisLabels


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


    def __getitem__(self, index):

        bag_image = self.bag_of_PiatentStudy_images[index]

        if self.transform_fn is not None:
            bag_image = torch.stack([self.transform_fn(Image.fromarray(image)) for


        DiagnosisLabel = self.bag_of_PatientStudy_DiagnosisLabels[index]

        return bag_image, DiagnosisLabel
```

Transformations

In [ ]:
```
import PIL
import PIL.ImageOps
import PIL.ImageEnhance
import PIL.ImageDraw
from PIL import Image

PARAMETER_MAX = 10


def AutoContrast(img, **kwarg):
    return PIL.ImageOps.autocontrast(img)


def Brightness(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    return PIL.ImageEnhance.Brightness(img).enhance(v)


def Color(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    return PIL.ImageEnhance.Color(img).enhance(v)


def Contrast(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    return PIL.ImageEnhance.Contrast(img).enhance(v)


def Cutout(img, v, max_v, bias=0):
    if v == 0:
        return img
    v = _float_parameter(v, max_v) + bias
    v = int(v * min(img.size))
    return CutoutAbs(img, v)


def CutoutAbs(img, v, **kwarg):
    w, h = img.size
    x0 = np.random.uniform(0, w)
    y0 = np.random.uniform(0, h)
    x0 = int(max(0, x0 - v / 2.))
    y0 = int(max(0, y0 - v / 2.))
    x1 = int(min(w, x0 + v))
    y1 = int(min(h, y0 + v))
    xy = (x0, y0, x1, y1)
    # gray
    color = (127, 127, 127)
    img = img.copy()
    PIL.ImageDraw.Draw(img).rectangle(xy, color)
    return img


def Equalize(img, **kwarg):
    return PIL.ImageOps.equalize(img)
```

```python
def Identity(img, **kwarg):
    return img


def Invert(img, **kwarg):
    return PIL.ImageOps.invert(img)


def Posterize(img, v, max_v, bias=0):
    v = _int_parameter(v, max_v) + bias
    return PIL.ImageOps.posterize(img, v)


def Rotate(img, v, max_v, bias=0):
    v = _int_parameter(v, max_v) + bias
    if random.random() < 0.5:
        v = -v
    return img.rotate(v)


def Sharpness(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    return PIL.ImageEnhance.Sharpness(img).enhance(v)


def ShearX(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    if random.random() < 0.5:
        v = -v
    return img.transform(img.size, PIL.Image.AFFINE, (1, v, 0, 0, 1, 0))


def ShearY(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    if random.random() < 0.5:
        v = -v
    return img.transform(img.size, PIL.Image.AFFINE, (1, 0, 0, v, 1, 0))


def Solarize(img, v, max_v, bias=0):
    v = _int_parameter(v, max_v) + bias
    return PIL.ImageOps.solarize(img, 256 - v)


def SolarizeAdd(img, v, max_v, bias=0, threshold=128):
    v = _int_parameter(v, max_v) + bias
    if random.random() < 0.5:
        v = -v
    img_np = np.array(img).astype(np.int)
    img_np = img_np + v
    img_np = np.clip(img_np, 0, 255)
    img_np = img_np.astype(np.uint8)
    img = Image.fromarray(img_np)
    return PIL.ImageOps.solarize(img, threshold)


def TranslateX(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    if random.random() < 0.5:
        v = -v
```

```python
    v = int(v * img.size[0])
    return img.transform(img.size, PIL.Image.AFFINE, (1, 0, v, 0, 1, 0))


def TranslateY(img, v, max_v, bias=0):
    v = _float_parameter(v, max_v) + bias
    if random.random() < 0.5:
        v = -v
    v = int(v * img.size[1])
    return img.transform(img.size, PIL.Image.AFFINE, (1, 0, 0, 0, 1, v))


def _float_parameter(v, max_v):
    return float(v) * max_v / PARAMETER_MAX


def _int_parameter(v, max_v):
    return int(v * max_v / PARAMETER_MAX)


def fixmatch_augment_pool():
    # FixMatch paper
    augs = [(AutoContrast, None, None),
            (Brightness, 0.9, 0.05),
            (Color, 0.9, 0.05),
            (Contrast, 0.9, 0.05),
            (Equalize, None, None),
            (Identity, None, None),
            (Posterize, 4, 4),
            (Rotate, 30, 0),
            (Sharpness, 0.9, 0.05),
            (ShearX, 0.3, 0),
            (ShearY, 0.3, 0),
            (Solarize, 256, 0),
            (TranslateX, 0.3, 0),
            (TranslateY, 0.3, 0)]
    return augs

class RandAugmentMC(object):
    def __init__(self, n, m):
        assert n >= 1
        assert 1 <= m <= 10
        self.n = n
        self.m = m
        self.augment_pool = fixmatch_augment_pool()

    def __call__(self, img):
        ops = random.choices(self.augment_pool, k=self.n)
        for op, max_v, bias in ops:
            v = np.random.randint(1, self.m)
            if random.random() < 0.5:
                img = op(img, v=v, max_v=max_v, bias=bias)
        img = CutoutAbs(img, int(32*0.5))
        return img
```

In [ ]:
```
transform_eval = transforms.Compose([
    transforms.ToTensor(),
])

transform_labeledtrain = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(size=112,
                          padding=int(112*0.125),
                          padding_mode='reflect'),
    RandAugmentMC(n=2, m=10),
    transforms.ToTensor(),
])
```

Create Dataset

The following code blocks preprocess the data and load the data using a modified implementation from the original paper to handle the TMED-2 dataset structure.

In [ ]:
```
train_dataset = EchoDataset(train_PatientStudy_ids, TMED2SummaryTable, LOCAL_DATA_DI

trainmemory_dataset = EchoDataset(train_PatientStudy_ids, TMED2SummaryTable, LOCAL_

val_dataset = EchoDataset(val_PatientStudy_ids, TMED2SummaryTable, LOCAL_DATA_DIR,

test_dataset = EchoDataset(test_PatientStudy_ids, TMED2SummaryTable, LOCAL_DATA_DIR
```

In [ ]:
```
print("train: {}, trainmemory: {}, val: {}, test: {}".format(len(train_dataset), le
```

```
train: 360, trainmemory: 360, val: 119, test: 120
```

In [ ]:

In [ ]:
```
train_loader = DataLoader(train_dataset, batch_size=batchSize, shuffle=True, num_wo
trainmemory_loader = DataLoader(trainmemory_dataset, batch_size=batchSize, shuffle=
val_loader = DataLoader(val_dataset, batch_size=batchSize, shuffle=False, num_worke
test_loader = DataLoader(test_dataset, batch_size=batchSize, shuffle=False, num_wor
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:558: UserWar
ning: This DataLoader will create 8 worker processes in total. Our suggested max nu
mber of worker in current system is 2, which is smaller than what this DataLoader i
s going to create. Please be aware that excessive worker creation might get DataLoa
der running slow or even freeze, lower the worker number to avoid potential slownes
s/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

Dataset Analysis

The TMED-2 `view_and_diagnosis_labeled_set` has 3 labels: `no_AS` which maps to `0`, `mod_AS` which maps to `1`, and `sev_AS` which maps to 2. In simple terms, these labels stand for No AS diagnosis, Moderate AS diagnosis, and Severe AS diagnosis.

From the dataset statistics method, we find the dataset to be imbalanced towards positive diagnoses of AS, namely Severe AS. There is risk in the trained models biasing towards a diagnosis vs. no AS diagnosis.

In [ ]:
```
def dataset_statistics(loader, name):
    num_batches = len(loader)
    num_samples = len(loader.dataset)
    label_mapping = {0: "no_AS", 1: "mod_AS", 2: "sev_AS"}

    print(f"{name} DataLoader:")
    print(f"  Total number of batches: {num_batches}")
    print(f"  Total number of samples: {num_samples}")

    labels = []
    for _, batch_labels in loader:
        labels.extend(batch_labels.tolist())
    num_classes = len(set(labels))
    print(f"  Number of classes: {num_classes}")

    class_distribution = {label_mapping[label]: labels.count(label) for label in se
    print(f"  Class distribution: {class_distribution}")

dataset_statistics(train_loader, "Train")
dataset_statistics(trainmemory_loader, "Train Memory")
dataset_statistics(val_loader, "Validation")
dataset_statistics(test_loader, "Test")
```

```
Train DataLoader:
  Total number of batches: 360
  Total number of samples: 360
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was
called. os.fork() is incompatible with multithreaded code, and JAX is multithreade
d, so this will likely lead to a deadlock.
  self.pid = os.fork()
  Number of classes: 3
  Class distribution: {'no_AS': 76, 'mod_AS': 103, 'sev_AS': 181}
Train Memory DataLoader:
  Total number of batches: 360
  Total number of samples: 360
  Number of classes: 3
  Class distribution: {'no_AS': 76, 'mod_AS': 103, 'sev_AS': 181}
Validation DataLoader:
  Total number of batches: 119
  Total number of samples: 119
  Number of classes: 3
  Class distribution: {'no_AS': 25, 'mod_AS': 34, 'sev_AS': 60}
Test DataLoader:
  Total number of batches: 120
  Total number of samples: 120
  Number of classes: 3
  Class distribution: {'no_AS': 26, 'mod_AS': 34, 'sev_AS': 60}
```

Model

The paper evaluates multiple models in addition to the SAMIL model they have contributed. These models include ABMIL and DSMIL. As mentioned previously, only ABMIL and SAMIL will be trained in this reproduction.

SAMIL Model

View Classifier

One of the novel contributions of the SAMIL paper is the introduction of supervised attention MIL (SAMIL). This is by introducing a "view-type relevance clasifier" to only pay attention to specific views of echo imagery. This so called "View Classifier" is a separately trained model based on the WideResNet architecture.

The following class for the View Classifier is directly ported from the paper's Github repo.

In [ ]:
```python
import logging

import torch
import torch.nn as nn
import torch.nn.functional as F
import sys

logging.basicConfig(format='%(asctime)s | %(levelname)s : %(message)s',
                    level=logging.INFO, stream=sys.stdout)
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)



def mish(x):
    """Mish: A Self Regularized Non-Monotonic Neural Activation Function (https://a
    return x * torch.tanh(F.softplus(x))


class PSBatchNorm2d(nn.BatchNorm2d):
    """How Does BN Increase Collapsed Neural Network Filters? (https://arxiv.org/ab

    def __init__(self, num_features, alpha=0.1, eps=1e-05, momentum=0.001, affine=T
        super().__init__(num_features, eps, momentum, affine, track_running_stats)
        self.alpha = alpha

    def forward(self, x):
        return super().forward(x) + self.alpha


class BasicBlock(nn.Module):
    def __init__(self, in_planes, out_planes, stride, drop_rate=0.0, activate_befor
        super(BasicBlock, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_planes, momentum=0.001)
        self.relu1 = nn.LeakyReLU(negative_slope=0.1, inplace=True)
        self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                               padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_planes, momentum=0.001)
        self.relu2 = nn.LeakyReLU(negative_slope=0.1, inplace=True)
        self.conv2 = nn.Conv2d(out_planes, out_planes, kernel_size=3, stride=1,
                               padding=1, bias=False)
        self.drop_rate = drop_rate
        self.equalInOut = (in_planes == out_planes)
        self.convShortcut = (not self.equalInOut) and nn.Conv2d(in_planes, out_plan
                                                                padding=0, bias=Fal
        self.activate_before_residual = activate_before_residual

    def forward(self, x):
        if not self.equalInOut and self.activate_before_residual == True:
            x = self.relu1(self.bn1(x))
        else:
            out = self.relu1(self.bn1(x))
        out = self.relu2(self.bn2(self.conv1(out if self.equalInOut else x)))
        if self.drop_rate > 0:
            out = F.dropout(out, p=self.drop_rate, training=self.training)
        out = self.conv2(out)
        return torch.add(x if self.equalInOut else self.convShortcut(x), out)
```

```
class NetworkBlock(nn.Module):
    def __init__(self, nb_layers, in_planes, out_planes, block, stride, drop_rate=0
        super(NetworkBlock, self).__init__()
        self.layer = self._make_layer(
            block, in_planes, out_planes, nb_layers, stride, drop_rate, activate_be

    def _make_layer(self, block, in_planes, out_planes, nb_layers, stride, drop_rat
        layers = []
        for i in range(int(nb_layers)):
            layers.append(block(i == 0 and in_planes or out_planes, out_planes,
                                i == 0 and stride or 1, drop_rate, activate_before_
        return nn.Sequential(*layers)

    def forward(self, x):
        return self.layer(x)

# args.model_depth = 28
# args.model_width = 2

class WideResNet(nn.Module):
    def __init__(self, num_classes, depth=28, widen_factor=2, drop_rate=0.0):
        super(WideResNet, self).__init__()
        channels = [16, 16*widen_factor, 32*widen_factor, 64*widen_factor, 128*wide
        assert((depth - 4) % 6 == 0)
        n = (depth - 4) / 6 #equivalent to 'repeat' in tf repo
        block = BasicBlock
        # 1st conv before any network block
        self.conv1 = nn.Conv2d(3, channels[0], kernel_size=3, stride=1,
                               padding=1, bias=False)
        # 1st block
        self.block1 = NetworkBlock(
            n, channels[0], channels[1], block, 1, drop_rate, activate_before_resid
        # 2nd block
        self.block2 = NetworkBlock(
            n, channels[1], channels[2], block, 2, drop_rate)
        # 3rd block
        self.block3 = NetworkBlock(
            n, channels[2], channels[3], block, 2, drop_rate)

        # 4th block (hz added)
        self.block4 = NetworkBlock(
            n, channels[3], channels[4], block, 2, drop_rate)

        # global average pooling and classifier
        self.bn1 = nn.BatchNorm2d(channels[4], momentum=0.001)
        self.relu = nn.LeakyReLU(negative_slope=0.1, inplace=True)
        self.fc = nn.Linear(channels[4], num_classes)
        self.channels = channels[4]

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight,
                                        mode='fan_out',
                                        nonlinearity='leaky_relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1.0)
                nn.init.constant_(m.bias, 0.0)
            elif isinstance(m, nn.Linear):
                nn.init.xavier_normal_(m.weight)
                nn.init.constant_(m.bias, 0.0)
```

```python
    def forward(self, x):
        out = self.conv1(x)
        out = self.block1(out)
        out = self.block2(out)
        out = self.block3(out)
        out = self.block4(out)

        out = self.relu(self.bn1(out))
        out = F.adaptive_avg_pool2d(out, 1)
        out = out.view(-1, self.channels)
        return self.fc(out)


def build_wideresnet(depth, widen_factor, dropout, num_classes):
    logger.info(f"Model: WideResNet {depth}x{widen_factor}")
    return WideResNet(depth=depth,
                      widen_factor=widen_factor,
                      drop_rate=dropout,
                      num_classes=num_classes)
```

SAMIL Model

The following class represents the SAMIL model. This code is unchanged from the paper's Github repo.

The SAMIL model consist of a 3 layer, sequential Feature Extractor to generate an aggregated bag-level embedding from an embedding of each instance. Then two sequential attention layers steer focus towards relevant echo views, and lastly a classifier layer.

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F

class SAMIL(nn.Module):
    def __init__(self, num_classes=3):
        super(SAMIL, self).__init__()
        self.L = 500
        self.B = 250
        self.D = 128
        self.K = 1
        self.num_classes = num_classes

        self.feature_extractor_part1 = nn.Sequential(
#             nn.Conv2d(1, 20, kernel_size=5),
            nn.Conv2d(3, 20, kernel_size=5),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
            nn.Conv2d(20, 50, kernel_size=5),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
             #hz added
            nn.Conv2d(50, 100, kernel_size=5),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
            nn.Conv2d(100, 200, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
        )

        self.feature_extractor_part2 = nn.Sequential(
#             nn.Linear(50 * 4 * 4, self.L),
            nn.Linear(200 * 4 * 4, self.L),
            nn.ReLU(),
        )

        self.feature_extractor_part3 = nn.Sequential(

            nn.Linear(self.L, self.B),
            nn.ReLU(),
            nn.Linear(self.B, self.L),
            nn.ReLU(),
        )

        self.attention_V = nn.Sequential(
            nn.Linear(self.L, self.D),
            nn.Tanh(),
            nn.Linear(self.D, self.K)
        )

        self.attention_U = nn.Sequential(
            nn.Linear(self.L, self.D),
            nn.Tanh(),
            nn.Linear(self.D, self.K)
        )

#         self.attention_weights = nn.Linear(self.D, self.K)

        self.classifier = nn.Sequential(
```

```
#             nn.Linear(self.L*self.K, 1),
            nn.Linear(self.L*self.K, self.num_classes),
#             nn.Sigmoid()
        )

    def forward(self, x):

#         print('Inside forward: input x shape: {}'.format(x.shape))
        x = x.squeeze(0)
#         print('Inside forward: after squeeze x shape: {}'.format(x.shape))

        H = self.feature_extractor_part1(x)
#         print('Inside forward: after feature_extractor_part1 H shape: {}'.format(


#         H = H.view(-1, 50 * 4 * 4)
        H = H.view(-1, 200 * 4 * 4)
#         print('Inside forward: after view H shape: {}'.format(H.shape))

        H = self.feature_extractor_part2(H)  # NxL
#         print('Inside forward: after feature_extractor_part2 H shape: {}'.format(

        A_V = self.attention_V(H)  # NxK
#         print('Inside forward: A_V is {}, shape: {}'.format(A_V, A_V.shape))

        A_V = torch.transpose(A_V, 1, 0)  # KxN
#         print('Inside forward: A_V is {}, shape: {}'.format(A_V, A_V.shape))

        A_V = F.softmax(A_V, dim=1)  # softmax over N
#         print('Inside forward: A_V (View) is {}, shape: {}'.format(A_V, A_V.shape


        H = self.feature_extractor_part3(H)

        A_U = self.attention_U(H)  # NxK
#         print('Inside forward: A_U is {}, shape: {}'.format(A_U, A_U.shape))

        A_U = torch.transpose(A_U, 1, 0)  # KxN
#         print('Inside forward: A_U is {}, shape: {}'.format(A_U, A_U.shape))

        A_U = F.softmax(A_U, dim=1)  # softmax over N
#         print('Inside forward: A_U (Diagnosis) is {}, shape: {}'.format(A_U, A_U.

#         A = A_V * A_U
#         print('Inside forward: final A is {}, shape: {}'.format(A, A.shape))
        A = torch.exp(torch.log(A_V) + torch.log(A_U)) #numerically more stable?

        A = A/torch.sum(A)
#         A = F.softmax(A, dim=1)
#         print('Inside forward: final A is {}, shape: {}'.format(A, A.shape))
#         A = self.attention_weights(A_V * A_U) # element wise multiplication # NxK
#         print('Inside forward: A is {}, shape: {}'.format(A, A.shape))

#         A = torch.transpose(A, 1, 0)  # KxN
# #         print('Inside forward: A is {}, shape: {}'.format(A, A.shape))

#         A = F.softmax(A, dim=1)  # softmax over N
# #         print('Inside forward: A is {}, shape: {}'.format(A, A.shape))

        M = torch.mm(A, H)  # KxL #M can be regarded as final representation of thi
```

```
#          print('Inside forward: M is {}, shape: {}'.format(M, M.shape))
        out = self.classifier(M)


        return out, A_V #only view regularize one branch of the attention weights
```

SAMIL Helpers

The following helper methods are from the paper's Github repo. Specifically the `src/SAMIL/main.py` file.

In [ ]:
```
import pandas as pd
import numpy as np
import torch
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import LambdaLR
from torch.utils.data import DataLoader
from torchvision import transforms

from torch.utils.tensorboard import SummaryWriter

logger = logging.getLogger(__name__)

def str2bool(s):
    if s == 'True':
        return True
    elif s == 'False':
        return False
    else:
        raise NameError('Bad string')

def save_checkpoint(state, checkpoint_dir, filename='last_checkpoint.pth.tar'):
    '''last_checkpoint.pth.tar or xxx_model_best.pth.tar'''

    filepath = os.path.join(checkpoint_dir, filename)
    torch.save(state, filepath)

def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)

def get_cosine_schedule_with_warmup(optimizer,
                                    lr_warmup_epochs,
                                    lr_cycle_epochs, #total train epochs
                                    num_cycles=7./16.,
                                    last_epoch=-1):
    def _lr_lambda(current_epoch):
        if current_epoch < lr_warmup_epochs:
            return float(current_epoch) / float(max(1, lr_warmup_epochs))
#         no_progress = float(current_epoch - lr_warmup_epochs) / \
#             float(max(1, float(lr_cycle_epochs) - lr_warmup_epochs))

        #see if using restart
        ###############################################################
        if current_epoch%lr_cycle_epochs==0:
            current_cycle_epoch=lr_cycle_epochs
        else:
            current_cycle_epoch = current_epoch%lr_cycle_epochs

        no_progress = float(current_cycle_epoch - lr_warmup_epochs) / \
            float(max(1, float(lr_cycle_epochs) - lr_warmup_epochs))
        #################################################################

        return max(0., math.cos(math.pi * num_cycles * no_progress))

    return LambdaLR(optimizer, _lr_lambda, last_epoch)

def get_fixed_lr(optimizer,
```

```
                lr_warmup_epochs,
                lr_cycle_epochs, #total train iterations
                num_cycles=7./16.,
                last_epoch=-1):
    def _lr_lambda(current_epoch):

        return 1.0

    return LambdaLR(optimizer, _lr_lambda, last_epoch)

def create_view_model(args):

    view_model = build_wideresnet(depth=28,
                                    widen_factor=2,
                                    dropout=0.0,
                                    num_classes=3)

    logger.info("Total params for View Model: {:.2f}M".format(
        sum(p.numel() for p in view_model.parameters())/1e6))


    #load the saved checkpoint
    if args['data_seed']==0:
        args['view_checkpoint_path'] = os.path.join(args['checkpoint_dir'], 'view_c
    elif args['data_seed']==1:
        args['view_checkpoint_path'] = os.path.join(args['checkpoint_dir'], 'view_c
    elif args['data_seed']==2:
        args['view_checkpoint_path'] = os.path.join(args['checkpoint_dir'], 'view_c
    else:
        raise NameError('?')


    view_checkpoint = torch.load(args['view_checkpoint_path'], map_location=device)

    view_model.load_state_dict(view_checkpoint['ema_state_dict'])

    view_model.eval()

    return view_model

def create_model(args):
    model = SAMIL()

    if args['MIL_checkpoint_path'] !='':
        print('!!!!!!!!!!!!!!!!!!!!!initializing from pretrained checkpoint!!!!!!!
        pretrained_dict = torch.load(args['MIL_checkpoint_path'], map_location=devi

        #https://discuss.pytorch.org/t/dataparallel-changes-parameter-names-issue-w
        #rename tensor in the pretrained dict
        from collections import OrderedDict
        new_state_dict = OrderedDict()
        for k, v in pretrained_dict.items():
#                         print(k)
            if 'encoder_q' in k:
#                             print('!extract: {}'.format(k))
                name = '.'.join(k.split('.')[1:])
#                             print('new_name: {}'.format(name))
                new_state_dict[name] = v

        model_dict = model.state_dict()
```

```
        new_state_dict = {k: v for k, v in new_state_dict.items() if k in model_dic
        model_dict.update(new_state_dict)

        # 3. load the new state dict
        model.load_state_dict(model_dict)



    logger.info("Total params: {:.2f}M".format(
        sum(p.numel() for p in model.parameters() if p.requires_grad)/1e6))

    return model
```

SAMIL Training

The method below sets up various arguments around pretraining. The paper explores three methods of training: No Pretraining, pre training the Feature Extrator (to learn instance-level representations), and pre training the study-level representations of all *K* images in a routine echocardiogram.

In [ ]:
```
logging.basicConfig(
    format="%(asctime)s - %(levelname)s - %(name)s -   %(message)s",
    datefmt="%m/%d/%Y %H:%M:%S",
    level=logging.INFO
)

def setup_samil_train(args):

    if args['training_seed'] is not None:
        print('setting training seed{}'.format(args['training_seed']), flush=True)
        set_seed(args['training_seed'])


    if args['Pretrained'] == 'Whole':

        if args['data_seed']==0:
            args['MIL_checkpoint_path'] = os.path.join(args['checkpoint_dir'],'MOCO_
        elif args['data_seed']==1:
            args['MIL_checkpoint_path'] = os.path.join(args['checkpoint_dir'],'MOCO_
        elif args['data_seed']==2:
            args['MIL_checkpoint_path'] = os.path.join(args['checkpoint_dir'],'MOCO_
        else:
            raise NameError('NOT VALID PRETRAINED MODEL')

    elif args['Pretrained'] == 'FeatureExtractor1':

        if args['data_seed']==0:
            args['MIL_checkpoint_path']=os.path.join(args['checkpoint_dir'], 'MOCO_
        elif args['data_seed']==1:
            args['MIL_checkpoint_path']=os.path.join(args['checkpoint_dir'], 'MOCO_
        elif args['data_seed']==2:
            args['MIL_checkpoint_path']=os.path.join(args['checkpoint_dir'], 'MOCO_
        else:
            raise NameError('NOT VALID PRETRAINED MODEL')


    elif args['Pretrained'] == 'NoPretrain':
        args['MIL_checkpoint_path']=''

    else:
        raise NameError('invalid pretrain option')

    if args['use_class_weights'] == 'True':
        print('!!!!!!!!Using pre-calculated class weights!!!!!!!!')

        #indeed, every split should have the same class weight for diagnosis by our
        if args['data_seed'] == 0 and args['development_size'] == 'DEV479':
            args['class_weights'] = '0.463,0.342,0.195'
        elif args['data_seed'] == 1 and args['development_size'] == 'DEV479':
            args['class_weights'] = '0.463,0.342,0.195'
        elif args['data_seed'] == 2 and args['development_size'] == 'DEV479':
            args['class_weights'] = '0.463,0.342,0.195'
        else:
            raise NameError('not valid class weights setting')

    else:
        args['class_weights'] = '1.0,1.0,1.0'
        print('?????????Not using pre-calculated class weights?????????')
```

```
    experiment_name = "{}".format(args['Pretrained'])

    args['experiment_dir'] = os.path.join(args['train_dir'], experiment_name)

    if args['resume'] != 'None':
        args['resume_checkpoint_fullpath'] = os.path.join(args['experiment_dir'], a
        print('args.resume_checkpoint_fullpath: {}'.format(args['resume_checkpoint_
    else:
        args['resume_checkpoint_fullpath'] = None

    os.makedirs(args['experiment_dir'], exist_ok=True)
    args['writer'] = SummaryWriter(args['experiment_dir'])

    brief_summary = {}
    brief_summary['val_progression_view'] = {}

    brief_summary['dataset_name'] = args['dataset_name']
    brief_summary['algorithm'] = 'Echo_MIL'
    brief_summary['hyperparameters'] = {
        'train_epoch': args['train_epoch'],
        'optimizer': args['optimizer_type'],
        'lr': args['lr'],
        'wd': args['wd'],
        'T':args['T'],
        'lambda_ViewRegularization':args['lambda_ViewRegularization']
    }

    return args, brief_summary
```

Train One Epoch and Early Stop Logic

The code block below contains the methods to train a single epoch and the early stop logic.

In [ ]:
```
from copy import deepcopy
from sklearn.metrics import balanced_accuracy_score

import torch

class ModelEMA(object):
    def __init__(self, args, model, decay):
        self.ema = deepcopy(model)
        self.ema.to(args['device'])
        self.ema.eval()
        self.decay = decay
        self.ema_has_module = hasattr(self.ema, 'module')
        # Fix EMA. https://github.com/valencebond/FixMatch_pytorch thank you!
        self.param_keys = [k for k, _ in self.ema.named_parameters()]
        self.buffer_keys = [k for k, _ in self.ema.named_buffers()]

        print('self.param_keys: {}'.format(self.param_keys))
        print('self.buffer_keys: {}'.format(self.buffer_keys))

        for p in self.ema.parameters():
#             print('Inside ModelEMA, p dtype is {}'.format(p.dtype))
            p.requires_grad_(False)


    def update(self, model):
        needs_module = hasattr(model, 'module') and not self.ema_has_module
        with torch.no_grad():
            msd = model.state_dict()
            esd = self.ema.state_dict()
            for k in self.param_keys:
                if needs_module:
                    j = 'module.' + k
                else:
                    j = k
                model_v = msd[j].detach()
                ema_v = esd[k]
                esd[k].copy_(ema_v * self.decay + (1. - self.decay) * model_v)

            for k in self.buffer_keys:
                if needs_module:
                    j = 'module.' + k
                else:
                    j = k
                esd[k].copy_(msd[j])

import time
from tqdm import tqdm
import torch.nn.functional as F

import logging
from sklearn.metrics import confusion_matrix as sklearn_cm
import numpy as np
import os
import pickle

import torch
import torch.nn as nn

import numpy as np
```

```
from sklearn.metrics import confusion_matrix as sklearn_cm


class EarlyStopping:
    """Early stops the training if validation acc doesn't improve after a given pat

    def __init__(self, patience=300, initial_count=0, delta=0):

        """
        Args:
            patience (int): How long to wait after last time validation loss improv
                            Default: 20
            delta (float): Minimum change in the monitored quantity to qualify as a
                            Default: 0

        """

        self.patience = patience
        self.counter = initial_count
        self.best_score = None
        self.early_stop = False
        self.delta = delta


    def __call__(self, val_acc):

        score = val_acc

        if self.best_score is None:
            self.best_score = score

        elif score <= self.best_score + self.delta:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True

        else:
            self.best_score = score
            self.counter = 0

        print('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!counter: {}, score: {}, best_s

        return self.counter


def train_one_epoch(args, weights, train_loader, model, ema_model, view_model, opti

    args['writer'].add_scalar('train/lr', scheduler.get_last_lr()[0], epoch)

    model.train()

    TotalLoss_this_epoch, LabeledCELoss_this_epoch, ViewRegularizationLoss_this_epo

    train_iter = iter(train_loader)
    n_steps_per_epoch = 360 #360 train studies, batch size 1
    p_bar = tqdm(range(n_steps_per_epoch), disable=False)

#     for batch_idx, (data, bag_label, view_relevance) in enumerate(tqdm(train_load
    for batch_idx in range(n_steps_per_epoch):
```

```
        try:
            data, bag_label = next(train_iter)
        except:
            train_iter = iter(train_loader)
            data, bag_label = next(train_iter)

#         print('batch_idx: {}'.format(batch_idx))

#         print('type(data): {}, data.size: {}, require grad: {}'.format(type(data)
#         print('type(bag_label): {}, bag_label: {}'.format(type(bag_label), bag_la
#         print('type(view_relevance): {}, view_relevance: {}'.format(type(view_rel
        data, bag_label = data.to(args['device']), bag_label.to(args['device'])


        outputs, attentions = model(data)

        log_attentions = torch.log(attentions)


        with torch.no_grad():
            view_predictions = view_model(data.squeeze(0))
            softmax_view_predictions = F.softmax(view_predictions, dim=1)
            predicted_relevance = softmax_view_predictions[:, :2]
            predicted_relevance = torch.sum(predicted_relevance, dim=1)
            predicted_relative_relevance = F.softmax(predicted_relevance/args['T'])
            predicted_relative_relevance = predicted_relative_relevance.unsqueeze(0


        #element shape in F.cross_entropy: prediction torch.size([batch_size, num_c
        if args['use_class_weights'] == 'True':
            LabeledCELoss = F.cross_entropy(outputs, bag_label, weights, reduction=
        else:
            LabeledCELoss = F.cross_entropy(outputs, bag_label, reduction='mean')


#         parser.add_argument('--ViewRegularization_warmup_pos', default=0.4, type=

# parser.add_argument('--ViewRegularization_warmup_schedule_type', default='NoWarmu

        #ViewRegularization warmup schedule choice
        if args['ViewRegularization_warmup_schedule_type'] == 'NoWarmup':
            current_warmup = 1
        elif args['ViewRegularization_warmup_schedule_type'] == 'Linear':
            current_warmup = np.clip(epoch/(float(args['ViewRegularization_warmup_p
        elif args['ViewRegularization_warmup_schedule_type'] == 'Sigmoid':
            current_warmup = math.exp(-5 * (1 - min(epoch/(float(args['ViewRegulari
        else:
            raise NameError('Not supported ViewRegularization warmup schedule')



        ViewRegularizationLoss = F.kl_div(input=log_attentions, target=predicted_re

        # backward pass
        total_loss = LabeledCELoss + args['lambda_ViewRegularization'] * ViewRegula

        total_loss.backward()
```

```python
        TotalLoss_this_epoch.append(total_loss.item())
        LabeledCELoss_this_epoch.append(LabeledCELoss.item())
        ViewRegularizationLoss_this_epoch.append(ViewRegularizationLoss.item())
        scaled_ViewRegularizationLoss_this_epoch.append(args['lambda_ViewRegulariza

        # step
        optimizer.step()

        #update ema model
        ema_model.update(model)

        model.zero_grad()

    scheduler.step()

    return TotalLoss_this_epoch, LabeledCELoss_this_epoch, ViewRegularizationLoss_t



#regular eval_model
def eval_model(args, data_loader, raw_model, ema_model, epoch):

    raw_model.eval()
    ema_model.eval()

    data_loader = tqdm(data_loader, disable=False)

    with torch.no_grad():
        total_targets = []
        total_raw_outputs = []
        total_ema_outputs = []


        for batch_idx, (data, bag_label) in enumerate(data_loader):

#             print('EVAL type(data): {}, data.size: {}, require grad: {}'.format(t
#             print('EVAL type(bag_label): {}, bag_label: {}'.format(type(bag_label

            data, bag_label = data.to(args['device']), bag_label.to(args['device'])

            raw_outputs, raw_attention_weights = raw_model(data)
            ema_outputs, ema_attention_weights = ema_model(data)
#             print('target is {}, raw_outputs is: {}, ema_outputs is {}'.format(ba

            total_targets.append(bag_label.detach().cpu())
            total_raw_outputs.append(raw_outputs.detach().cpu())
            total_ema_outputs.append(ema_outputs.detach().cpu())


        total_targets = np.concatenate(total_targets, axis=0)
        total_raw_outputs = np.concatenate(total_raw_outputs, axis=0)
        total_ema_outputs = np.concatenate(total_ema_outputs, axis=0)
#         print('RegularEval total_targets: {}'.format(total_targets))
#         print('RegularEval total_raw_outputs: {}'.format(total_raw_outputs))
#         print('RegularEval total_ema_outputs: {}'.format(total_ema_outputs))

        raw_Bacc = calculate_balanced_accuracy(total_raw_outputs, total_targets)
        ema_Bacc = calculate_balanced_accuracy(total_ema_outputs, total_targets)
```

```
#         print('raw Bacc this evaluation step: {}'.format(raw_Bacc), flush=True)
#         print('ema Bacc this evaluation step: {}'.format(ema_Bacc), flush=True)


    return raw_Bacc, ema_Bacc, total_targets, total_raw_outputs, total_ema_outputs

def eval_model_test(args, data_loader, raw_model):
    raw_model.eval()

    with torch.no_grad():
        ground_truth_labels = []
        pred_labels = []

        for data, bag_label in data_loader:
            data, bag_label = data.to(device), bag_label.to(device)

            pred_logit, _ = raw_model(data)

            pred_label = torch.softmax(pred_logit, dim=-1)
            pred_label = torch.argmax(pred_label).item()

            pred_labels.append(pred_label)
            ground_truth_labels.append(bag_label.item())

        bal_acc = balanced_accuracy_score(ground_truth_labels, pred_labels)

    return bal_acc

def calculate_balanced_accuracy(prediction, true_target, return_type = 'only balanc

    confusion_matrix = sklearn_cm(true_target, prediction.argmax(1))
    n_class = confusion_matrix.shape[0]
    print('Inside calculate_balanced_accuracy, {} classes passed in'.format(n_class

    assert n_class==3

    recalls = []
    for i in range(n_class):
        recall = confusion_matrix[i,i]/np.sum(confusion_matrix[i])
        recalls.append(recall)
        print('class{} recall: {}'.format(i, recall), flush=True)

    balanced_accuracy = np.mean(np.array(recalls))


    if return_type == 'all':
#         return balanced_accuracy * 100, class0_recall * 100, class1_recall * 100,
        return balanced_accuracy * 100, recalls

    elif return_type == 'only balanced_accuracy':
        return balanced_accuracy * 100
    else:
        raise NameError('Unsupported return_type in this calculate_balanced_accurac


 #shared helper fct across different algos
def save_pickle(save_dir, save_file_name, data):
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)
```

```
    data_save_fullpath = os.path.join(save_dir, save_file_name)
    with open(data_save_fullpath, 'wb') as handle:
        pickle.dump(data, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Training Runner

The code block below contains the logic to train the model, one epoch at a time, with early stop, and the logic to write the results out. This method is called in subsequent training blocks after the training arguments are defined.

In [ ]:

```
import json

def train_samil(args):
    best_val_ema_Bacc = 0
    best_test_ema_Bacc_at_val = 0
    best_train_ema_Bacc_at_val = 0

    best_val_raw_Bacc = 0
    best_test_raw_Bacc_at_val = 0
    best_train_raw_Bacc_at_val = 0


    current_count=0

    if os.path.isfile(args.get('resume_checkpoint_fullpath')):

        print('Resuming from checkpoint: {}'.format(args.get('resume_checkpoint_ful

        checkpoint = torch.load(args['resume_checkpoint_fullpath'])
        args['start_epoch'] = checkpoint['epoch']
        model.load_state_dict(checkpoint['state_dict'])
        ema_model.ema.load_state_dict(checkpoint['ema_state_dict'])
        current_count = checkpoint['current_count']
        optimizer.load_state_dict(checkpoint['optimizer'])
        scheduler.load_state_dict(checkpoint['scheduler'])

        best_val_ema_Bacc = checkpoint['val_progression_view']['best_val_ema_Bacc']
        best_test_ema_Bacc_at_val = checkpoint['val_progression_view']['best_test_e
        best_train_ema_Bacc_at_val = checkpoint['val_progression_view']['best_train_

        best_val_raw_Bacc = checkpoint['val_progression_view']['best_val_raw_Bacc']
        best_test_raw_Bacc_at_val = checkpoint['val_progression_view']['best_test_r
        best_train_raw_Bacc_at_val = checkpoint['val_progression_view']['best_train_


    else:
        print('!!!!Does not have checkpoint yet!!!!')


    logger.info("***** Running training *****")
    logger.info(f"  Task = {args['dataset_name']}")
    logger.info(f"  Num Epochs = {args['train_epoch']}")
    logger.info(f"  Total optimization steps = {args['train_epoch'] * len(train_dat

    train_loss_dict = dict()
    train_loss_dict['Totalloss'] = []
    train_loss_dict['LabeledCEloss'] = []
    train_loss_dict['ViewRegularizationLoss'] = []

    early_stopping = EarlyStopping(patience=args['patience'], initial_count=current_
    early_stopping_warmup = args['early_stopping_warmup']

    for epoch in tqdm(range(args['start_epoch'], args['train_epoch'])):
        val_predictions_save_dict = dict()
        test_predictions_save_dict = dict()
        train_predictions_save_dict = dict()

        TotalLoss_list, LabeledCEloss_list, ViewRegularizationLoss_list, scaled_Vie
        train_loss_dict['Totalloss'].extend(TotalLoss_list)
```

```
        train_loss_dict['LabeledCEloss'].extend(LabeledCEloss_list)
        train_loss_dict['ViewRegularizationLoss'].extend(ViewRegularizationLoss_lis

        if epoch % args['eval_every_Xepoch'] == 0:
            val_raw_Bacc, val_ema_Bacc, val_true_labels, val_raw_predictions, val_e
            val_predictions_save_dict['raw_Bacc'] = val_raw_Bacc
            val_predictions_save_dict['ema_Bacc'] = val_ema_Bacc
            val_predictions_save_dict['true_labels'] = val_true_labels
            val_predictions_save_dict['raw_predictions'] = val_raw_predictions
            val_predictions_save_dict['ema_predictions'] = val_ema_predictions

            test_raw_Bacc, test_ema_Bacc, test_true_labels, test_raw_predictions, t

            test_predictions_save_dict['raw_Bacc'] = test_raw_Bacc
            test_predictions_save_dict['ema_Bacc'] = test_ema_Bacc
            test_predictions_save_dict['true_labels'] = test_true_labels
            test_predictions_save_dict['raw_predictions'] = test_raw_predictions
            test_predictions_save_dict['ema_predictions'] = test_ema_predictions

            train_raw_Bacc, train_ema_Bacc, train_true_labels, train_raw_prediction

            train_predictions_save_dict['raw_Bacc'] = train_raw_Bacc
            train_predictions_save_dict['ema_Bacc'] = train_ema_Bacc
            train_predictions_save_dict['true_labels'] = train_true_labels
            train_predictions_save_dict['raw_predictions'] = train_raw_predictions
            train_predictions_save_dict['ema_predictions'] = train_ema_predictions

            if val_raw_Bacc > best_val_raw_Bacc:

                best_val_raw_Bacc = val_raw_Bacc
                best_test_raw_Bacc_at_val = test_raw_Bacc
                best_train_raw_Bacc_at_val = train_raw_Bacc

                save_pickle(os.path.join(args['experiment_dir'], 'val_progression_v

                save_pickle(os.path.join(args['experiment_dir'], 'val_progression_v


                save_pickle(os.path.join(args['experiment_dir'], 'val_progression_v

                save_checkpoint(
                {
                'epoch': epoch+1,
                'state_dict': model.state_dict(),
                'ema_state_dict': ema_model.ema.state_dict(),
                'current_count':current_count,
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict(),

                'val_progression_view':
                    {'epoch': epoch+1,
                    'best_val_ema_Bacc': best_val_ema_Bacc,
                    'best_val_raw_Bacc': best_val_raw_Bacc,
                    'best_test_ema_Bacc_at_val': best_test_ema_Bacc_at_val,
                    'best_test_raw_Bacc_at_val': best_test_raw_Bacc_at_val,
                    'best_train_ema_Bacc_at_val': best_train_ema_Bacc_at_val,
                    'best_train_raw_Bacc_at_val': best_train_raw_Bacc_at_val,
                      },

                }, args['experiment_dir'], filename='val_progression_view/best_pred
```

```
if val_ema_Bacc > best_val_ema_Bacc:

    best_val_ema_Bacc = val_ema_Bacc
    best_test_ema_Bacc_at_val = test_ema_Bacc
    best_train_ema_Bacc_at_val = train_ema_Bacc

    save_pickle(os.path.join(args['experiment_dir'], 'val_progression_v

    save_pickle(os.path.join(args['experiment_dir'], 'val_progression_v

    save_pickle(os.path.join(args['experiment_dir'], 'val_progression_v

    save_checkpoint(
    {
    'epoch': epoch+1,
    'state_dict': model.state_dict(),
    'ema_state_dict': ema_model.ema.state_dict(),
    'current_count':current_count,
    'optimizer': optimizer.state_dict(),
    'scheduler': scheduler.state_dict(),

    'val_progression_view':
        {'epoch': epoch+1,
        #regular val
        'best_val_ema_Bacc': best_val_ema_Bacc,
        'best_val_raw_Bacc': best_val_raw_Bacc,
        'best_test_ema_Bacc_at_val': best_test_ema_Bacc_at_val,
        'best_test_raw_Bacc_at_val': best_test_raw_Bacc_at_val,
        'best_train_ema_Bacc_at_val': best_train_ema_Bacc_at_val,
        'best_train_raw_Bacc_at_val': best_train_raw_Bacc_at_val,
          },

    }, args['experiment_dir'], filename='val_progression_view/best_pred



logger.info('val progression view:')
logger.info('At RAW Best val, validation/test/train %.2f %.2f %.2f' % (
logger.info('At EMA Best val, validation/test/train %.2f %.2f %.2f' % (

args['writer'].add_scalar('train/1.train_raw_Bacc', train_raw_Bacc, epo
args['writer'].add_scalar('train/1.train_ema_Bacc', train_ema_Bacc, epo
args['writer'].add_scalar('train/1.LabeledCEloss', np.mean(LabeledCElos

args['writer'].add_scalar('val/1.val_raw_Bacc', val_raw_Bacc, epoch)
args['writer'].add_scalar('val/2.val_ema_Bacc', val_ema_Bacc, epoch)


args['writer'].add_scalar('test/1.test_raw_Bacc', test_raw_Bacc, epoch)
args['writer'].add_scalar('test/2.test_ema_Bacc', test_ema_Bacc, epoch)

brief_summary['val_progression_view']['best_val_ema_Bacc'] = best_val_e
brief_summary['val_progression_view']['best_val_raw_Bacc'] = best_val_r
brief_summary['val_progression_view']['best_test_ema_Bacc_at_val'] = be
brief_summary['val_progression_view']['best_test_raw_Bacc_at_val'] = be
brief_summary['val_progression_view']['best_train_ema_Bacc_at_val'] = b
brief_summary['val_progression_view']['best_train_raw_Bacc_at_val'] = b
```

```
            with open(os.path.join(args['experiment_dir'], "brief_summary.json"), "
                json.dump(brief_summary, f)

            if epoch > early_stopping_warmup:
                current_count = early_stopping(val_ema_Bacc)

            save_checkpoint(
                {
                'epoch': epoch+1,
                'state_dict': model.state_dict(),
                'ema_state_dict': ema_model.ema.state_dict(),
                'current_count':current_count,
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict(),

                'val_progression_view':
                    {'epoch': epoch+1,
                    #regular val
                    'best_val_ema_Bacc': best_val_ema_Bacc,
                    'best_val_raw_Bacc': best_val_raw_Bacc,
                    'best_test_ema_Bacc_at_val': best_test_ema_Bacc_at_val,
                    'best_test_raw_Bacc_at_val': best_test_raw_Bacc_at_val,
                    'best_train_ema_Bacc_at_val': best_train_ema_Bacc_at_val,
                    'best_train_raw_Bacc_at_val': best_train_raw_Bacc_at_val,
                      },


                }, args['experiment_dir'], filename='last_checkpoint.pth.tar')


            if early_stopping.early_stop:
                break

    brief_summary['val_progression_view']['best_val_ema_Bacc'] = best_val_ema_Bacc
    brief_summary['val_progression_view']['best_val_raw_Bacc'] = best_val_raw_Bacc
    brief_summary['val_progression_view']['best_test_ema_Bacc_at_val'] = best_test_
    brief_summary['val_progression_view']['best_test_raw_Bacc_at_val'] = best_test_
    brief_summary['val_progression_view']['best_train_ema_Bacc_at_val'] = best_trai
    brief_summary['val_progression_view']['best_train_raw_Bacc_at_val'] = best_trai



    args['writer'].close()

    with open(os.path.join(args['experiment_dir'], "brief_summary.json"), "w") as f
        json.dump(brief_summary, f)
```

Configure SAMIL Arguments for Training

The (3) code blocks below configure the arguments for training the SAMIL model with no pretraining, with image-level pretraining, and study-level pretraining per the specified hyperparameters in the paper for Split 1 and the Github repo here: Hyperparameters (https://github.com/tufts-ml/SAMIL/blob/main/Hyperparameters/Hyperparameters.txt)

SAMIL (with study-level SSL)	split1	split2	split3
Learning rate	0.0008	0.0005	0.0005

SAMIL (with study-level SSL)	split1	split2	split3
Weight decay	0.0001	0.0001	0.001
Temperature T	0.1	0.05	0.1
λ_{sA}	15.0	20.0	20.0
Learning rate schedule	cosine	cosine	cosine

Table C.1: Hyperparameter settings for SAMIL across different data splits.

Computational Requirements

It is highly recommended to train the model variants on A100 GPUs as suggested in the paper. Training in a stable environment (not in Google Colab) is preferable due to the long training times necessary for each variant.

For the SAMIL models on an A100 GPU in Colab, each epoch was taking an average of 22 seconds. This represents an upper bound of roughly 13 hours without Early Stopping.

SAMIL with No Pretraining

The arguments below setup the runner to train SAMIL with no pretraining. This is achieved by specifying `NoPretrain` in the `Pretrained` argument.

The number of epochs specified in the paper is 2,000, however the arguments below are set to 1 to enable the testing of this training run.

In [ ]:
```
RUNS_DIR = '/content/runs/'

args = {
    'training_seed': 0,
    'Pretrained': 'NoPretrain',
    'data_seed': 0,
    'checkpoint_dir': MODEL_CHECKPOINTS,
    'MIL_checkpoint_path': '',
    'use_class_weights': 'True',
    'ViewRegularization_warmup_schedule_type': 'Linear',
    'optimizer_type': 'SGD',
    'lr_schedule_type': 'CosineLR',
    'lr_cycle_epochs': 1,
    'lr': 0.0008, # learning rate
    'wd': 0.0001, # weight decay
    'T': 0.1, # tempertature
    'lambda_ViewRegularization': 15.0, # λsA
    'train_dir': RUNS_DIR + 'SAMIL',
    'resume': 'last_checkpoint.pth.tar',
    'dataset_name': 'echo',
    'train_epoch': 2, # number of epochs, 2000 defined in the paper. CHANGE ME!
    'development_size': 'DEV479',
    'lr_warmup_epochs': 0,
    'ema_decay': 0.999,
    'device': device,
    'start_epoch': 0,
    'patience': 200,
    'early_stopping_warmup': 200,
    'ViewRegularization_warmup_pos': 0.4,
    'eval_every_Xepoch': 1
}

args, brief_summary = setup_samil_train(args)

weights = args['class_weights']
weights = [float(i) for i in weights.split(',')]
weights = torch.Tensor(weights)
weights = weights.to(device)

#load the view model, the output is unnormalized logits, need to use softmax on the
view_model = create_view_model(args)
view_model.to(device)

model = create_model(args)
model.to(device)

no_decay = ['bias', 'bn']
grouped_parameters = [
    {'params': [p for n, p in model.named_parameters() if not any(
        nd in n for nd in no_decay)], 'weight_decay': args['wd']},
    {'params': [p for n, p in model.named_parameters() if any(
        nd in n for nd in no_decay)], 'weight_decay': 0.0}
]

if args['optimizer_type'] == 'SGD':
    optimizer = optim.SGD(grouped_parameters, lr=args['lr'],
                          momentum=0.9, nesterov=True)

elif args['optimizer_type'] == 'Adam':
```

```
    optimizer = optim.Adam(grouped_parameters, lr=args['lr'])

elif args['optimizer_type'] == 'AdamW':
    optimizer = optim.AdamW(grouped_parameters, lr=args['lr'])

else:
    raise NameError('Not supported optimizer setting')

#lr_schedule_type choice
if args['lr_schedule_type'] == 'CosineLR':
    scheduler = get_cosine_schedule_with_warmup(optimizer, args['lr_warmup_epochs']

elif args['lr_schedule_type'] == 'FixedLR':
    scheduler = get_fixed_lr(optimizer, args['lr_warmup_epochs'], args['lr_cycle_ep

else:
    raise NameError('Not supported lr scheduler setting')


#instantiate the ema_model object
ema_model = ModelEMA(args, model, args['ema_decay'])


# !!! Start training
train_samil(args)
```

```
setting training seed0
INFO:__main__:Model: WideResNet 28x2
!!!!!!!!Using pre-calculated class weights!!!!!!!!
args.resume_checkpoint_fullpath: /content/runs/SAMIL/NoPretrain/last_checkpoint.pt
h.tar
INFO:__main__:Total params for View Model: 5.93M
INFO:__main__:Total params: 2.31M
INFO:__main__:***** Running training *****
INFO:__main__:  Task = echo
INFO:__main__:  Num Epochs = 2
INFO:__main__:  Total optimization steps = 720
self.param_keys: ['feature_extractor_part1.0.weight', 'feature_extractor_part1.0.bi
as', 'feature_extractor_part1.3.weight', 'feature_extractor_part1.3.bias', 'feature
_extractor_part1.6.weight', 'feature_extractor_part1.6.bias', 'feature_extractor_pa
rt1.9.weight', 'feature_extractor_part1.9.bias', 'feature_extractor_part2.0.weigh
t', 'feature_extractor_part2.0.bias', 'feature_extractor_part3.0.weight', 'feature_
extractor_part3.0.bias', 'feature_extractor_part3.2.weight', 'feature_extractor_par
t3.2.bias', 'attention_V.0.weight', 'attention_V.0.bias', 'attention_V.2.weight',
'attention_V.2.bias', 'attention_U.0.weight', 'attention_U.0.bias', 'attention_U.2.
weight', 'attention_U.2.bias', 'classifier.0.weight', 'classifier.0.bias']
self.buffer_keys: []
Resuming from checkpoint: /content/runs/SAMIL/NoPretrain/last_checkpoint.pth.tar
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/360 [00:40<?, ?it/s]

  0%|          | 0/119 [00:00<?, ?it/s]
  1%|          | 1/119 [00:00<01:25,  1.38it/s]
  2%|▏         | 2/119 [00:00<00:51,  2.25it/s]
  4%|▍         | 5/119 [00:01<00:18,  6.33it/s]
  7%|▋         | 8/119 [00:01<00:11,  9.57it/s]
  9%|▉         | 11/119 [00:01<00:08, 12.85it/s]
 13%|█▎        | 15/119 [00:01<00:05, 17.74it/s]
 15%|█▌        | 18/119 [00:01<00:05, 18.77it/s]
 18%|█▊        | 22/119 [00:01<00:04, 22.46it/s]
 21%|██        | 25/119 [00:01<00:04, 23.36it/s]
 24%|██▍       | 29/119 [00:01<00:03, 27.31it/s]
 28%|██▊       | 33/119 [00:02<00:03, 27.76it/s]
 30%|███       | 36/119 [00:02<00:03, 27.33it/s]
 33%|███▎      | 39/119 [00:02<00:02, 27.24it/s]
 35%|███▌      | 42/119 [00:02<00:02, 27.37it/s]
 39%|███▊      | 46/119 [00:02<00:02, 29.59it/s]
 42%|████▏     | 50/119 [00:02<00:02, 28.65it/s]
 45%|████▍     | 53/119 [00:02<00:02, 27.54it/s]
 48%|████▊     | 57/119 [00:02<00:02, 28.82it/s]
 52%|█████▏    | 62/119 [00:03<00:01, 31.31it/s]
 55%|█████▌    | 66/119 [00:03<00:01, 32.75it/s]
 59%|█████▉    | 70/119 [00:03<00:01, 31.20it/s]
 62%|██████▏   | 74/119 [00:03<00:01, 31.49it/s]
 66%|██████▌   | 78/119 [00:03<00:01, 29.81it/s]
 69%|██████▉   | 82/119 [00:03<00:01, 30.08it/s]
 72%|███████▏  | 86/119 [00:03<00:01, 29.13it/s]
 76%|███████▌  | 90/119 [00:04<00:00, 31.27it/s]
 79%|███████▉  | 94/119 [00:04<00:00, 30.62it/s]
 82%|████████▏ | 98/119 [00:04<00:00, 29.34it/s]
 85%|████████▍ | 101/119 [00:04<00:00, 28.77it/s]
 87%|████████▋ | 104/119 [00:04<00:00, 28.74it/s]
 93%|█████████▎| 111/119 [00:04<00:00, 38.07it/s]
100%|██████████| 119/119 [00:05<00:00, 23.18it/s]
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
```

```
  0%|          | 0/120 [00:00<?, ?it/s]
  1%|          | 1/120 [00:00<01:45,  1.13it/s]
  2%|▎         | 3/120 [00:01<00:33,  3.51it/s]
  5%|▌         | 6/120 [00:01<00:15,  7.33it/s]
  8%|▊         | 10/120 [00:01<00:08, 12.71it/s]
 12%|█▏        | 14/120 [00:01<00:06, 17.02it/s]
 15%|█▌        | 18/120 [00:01<00:04, 21.34it/s]
 18%|█▊        | 22/120 [00:01<00:03, 24.73it/s]
 22%|██▏       | 26/120 [00:01<00:03, 24.73it/s]
 24%|██▍       | 29/120 [00:01<00:03, 24.33it/s]
 28%|██▊       | 33/120 [00:02<00:03, 26.39it/s]
 30%|███       | 36/120 [00:02<00:03, 26.63it/s]
 33%|███▎      | 40/120 [00:02<00:02, 28.87it/s]
 38%|███▊      | 45/120 [00:02<00:02, 33.08it/s]
 41%|████      | 49/120 [00:02<00:02, 29.03it/s]
 44%|████▍     | 53/120 [00:02<00:02, 29.47it/s]
 48%|████▊     | 57/120 [00:02<00:02, 27.28it/s]
 50%|█████     | 60/120 [00:03<00:02, 24.92it/s]
 52%|█████▎    | 63/120 [00:03<00:02, 25.07it/s]
 55%|█████▌    | 66/120 [00:03<00:02, 22.09it/s]
 57%|█████▊    | 69/120 [00:03<00:02, 20.47it/s]
 60%|██████    | 72/120 [00:03<00:02, 20.70it/s]
 62%|██████▎   | 75/120 [00:03<00:02, 18.79it/s]
 64%|██████▍   | 77/120 [00:03<00:02, 17.73it/s]
 66%|██████▌   | 79/120 [00:04<00:02, 17.01it/s]
 68%|██████▊   | 82/120 [00:04<00:02, 18.28it/s]
 70%|███████   | 84/120 [00:04<00:02, 16.24it/s]
 72%|███████▏  | 86/120 [00:04<00:01, 17.01it/s]
 73%|███████▎  | 88/120 [00:04<00:01, 17.41it/s]
 75%|███████▌  | 90/120 [00:04<00:01, 17.06it/s]
 78%|███████▊  | 93/120 [00:04<00:01, 20.06it/s]
 80%|████████  | 96/120 [00:05<00:01, 18.23it/s]
 82%|████████▏ | 98/120 [00:05<00:01, 17.51it/s]
 84%|████████▍ | 101/120 [00:05<00:01, 18.20it/s]
 87%|████████▋ | 104/120 [00:05<00:00, 19.24it/s]
 91%|█████████ | 109/120 [00:05<00:00, 25.66it/s]
100%|██████████| 120/120 [00:06<00:00, 18.89it/s]
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
```

```
  0%|          | 0/360 [00:00<?, ?it/s]
  0%|          | 1/360 [00:00<04:59,  1.20it/s]
  1%|          | 2/360 [00:01<03:04,  1.94it/s]
  1%|          | 3/360 [00:01<02:00,  2.96it/s]
  2%|▏         | 6/360 [00:01<00:48,  7.31it/s]
  3%|▎         | 10/360 [00:01<00:27, 12.88it/s]
  4%|▎         | 13/360 [00:01<00:23, 14.67it/s]
  5%|▍         | 17/360 [00:01<00:17, 19.28it/s]
  6%|▌         | 20/360 [00:01<00:16, 21.23it/s]
  7%|▋         | 24/360 [00:01<00:13, 24.84it/s]
  8%|▊         | 30/360 [00:02<00:10, 30.71it/s]
  9%|▉         | 34/360 [00:02<00:12, 27.01it/s]
 10%|█         | 37/360 [00:02<00:11, 27.42it/s]
 11%|█▏        | 41/360 [00:02<00:10, 29.13it/s]
 12%|█▎        | 45/360 [00:02<00:10, 30.50it/s]
 14%|█▎        | 49/360 [00:02<00:09, 32.57it/s]
 15%|█▍        | 53/360 [00:02<00:09, 33.08it/s]
 16%|█▌        | 57/360 [00:03<00:09, 31.18it/s]
 17%|█▋        | 61/360 [00:03<00:09, 29.94it/s]
 18%|█▊        | 65/360 [00:03<00:09, 30.54it/s]
 19%|█▉        | 69/360 [00:03<00:09, 31.45it/s]
 20%|██        | 73/360 [00:03<00:10, 27.03it/s]
 21%|██        | 76/360 [00:03<00:12, 23.29it/s]
 22%|██▏       | 79/360 [00:03<00:12, 22.10it/s]
 23%|██▎       | 82/360 [00:04<00:12, 22.77it/s]
 24%|██▎       | 85/360 [00:04<00:11, 24.38it/s]
 24%|██▍       | 88/360 [00:04<00:11, 23.39it/s]
 25%|██▌       | 91/360 [00:04<00:11, 24.33it/s]
 26%|██▌       | 94/360 [00:04<00:11, 23.07it/s]
 27%|██▋       | 97/360 [00:04<00:10, 24.12it/s]
 28%|██▊       | 100/360 [00:04<00:11, 22.23it/s]
 29%|██▊       | 103/360 [00:04<00:10, 23.96it/s]
 30%|██▉       | 107/360 [00:05<00:09, 26.40it/s]
 31%|███       | 110/360 [00:05<00:09, 26.54it/s]
 31%|███▏      | 113/360 [00:05<00:09, 27.25it/s]
 33%|███▎      | 118/360 [00:05<00:07, 31.32it/s]
 34%|███▍      | 123/360 [00:05<00:06, 34.39it/s]
 35%|███▌      | 127/360 [00:05<00:06, 33.34it/s]
 36%|███▋      | 131/360 [00:05<00:07, 31.09it/s]
 38%|███▊      | 135/360 [00:05<00:07, 30.36it/s]
 39%|███▊      | 139/360 [00:06<00:06, 32.40it/s]
 40%|███▉      | 143/360 [00:06<00:07, 30.33it/s]
 41%|████      | 147/360 [00:06<00:07, 30.20it/s]
 42%|████▏     | 151/360 [00:06<00:06, 30.70it/s]
 43%|████▎     | 155/360 [00:06<00:06, 30.03it/s]
 44%|████▍     | 159/360 [00:06<00:06, 30.93it/s]
 45%|████▌     | 163/360 [00:06<00:06, 29.27it/s]
 46%|████▌     | 166/360 [00:07<00:07, 26.78it/s]
 47%|████▋     | 170/360 [00:07<00:06, 29.62it/s]
 48%|████▊     | 174/360 [00:07<00:06, 28.70it/s]
 49%|████▉     | 177/360 [00:07<00:06, 28.55it/s]
 50%|█████     | 181/360 [00:07<00:06, 29.43it/s]
 52%|█████▏    | 186/360 [00:07<00:05, 32.51it/s]
 53%|█████▎    | 190/360 [00:07<00:04, 34.02it/s]
 54%|█████▍    | 195/360 [00:07<00:04, 35.95it/s]
 55%|█████▌    | 199/360 [00:07<00:04, 33.15it/s]
 56%|█████▋    | 203/360 [00:08<00:04, 32.42it/s]
 58%|█████▊    | 208/360 [00:08<00:04, 34.58it/s]
```

```
 59%|█████▉    | 213/360 [00:08<00:04, 36.41it/s]
 60%|██████    | 217/360 [00:08<00:04, 29.94it/s]
 61%|██████▏   | 221/360 [00:08<00:04, 31.04it/s]
 62%|██████▎   | 225/360 [00:08<00:04, 30.56it/s]
 64%|██████▎   | 229/360 [00:08<00:04, 27.85it/s]
 64%|██████▍   | 232/360 [00:09<00:04, 28.21it/s]
 66%|██████▌   | 236/360 [00:09<00:04, 29.73it/s]
 67%|██████▋   | 240/360 [00:09<00:04, 29.91it/s]
 68%|██████▊   | 244/360 [00:09<00:03, 30.78it/s]
 69%|██████▉   | 248/360 [00:09<00:03, 29.77it/s]
 70%|███████   | 252/360 [00:09<00:03, 29.99it/s]
 71%|███████   | 256/360 [00:09<00:03, 30.01it/s]
 72%|███████▏  | 260/360 [00:10<00:03, 29.86it/s]
 74%|███████▎  | 265/360 [00:10<00:03, 30.93it/s]
 75%|███████▍  | 269/360 [00:10<00:03, 27.74it/s]
 76%|███████▌  | 273/360 [00:10<00:03, 28.77it/s]
 77%|███████▋  | 277/360 [00:10<00:02, 29.77it/s]
 78%|███████▊  | 281/360 [00:10<00:02, 29.44it/s]
 79%|███████▉  | 284/360 [00:10<00:03, 25.08it/s]
 80%|███████▉  | 287/360 [00:11<00:03, 23.10it/s]
 81%|████████  | 290/360 [00:11<00:03, 21.63it/s]
 82%|████████▏ | 294/360 [00:11<00:02, 24.36it/s]
 82%|████████▎ | 297/360 [00:11<00:03, 20.75it/s]
 83%|████████▎ | 300/360 [00:11<00:02, 20.24it/s]
 84%|████████▍ | 303/360 [00:11<00:03, 16.42it/s]
 85%|████████▍ | 305/360 [00:12<00:03, 16.10it/s]
 85%|████████▌ | 307/360 [00:12<00:03, 16.57it/s]
 86%|████████▌ | 309/360 [00:12<00:03, 16.44it/s]
 86%|████████▋ | 311/360 [00:12<00:03, 16.14it/s]
 87%|████████▋ | 313/360 [00:12<00:03, 15.45it/s]
 88%|████████▊ | 315/360 [00:12<00:02, 16.10it/s]
 88%|████████▊ | 317/360 [00:12<00:02, 16.48it/s]
 89%|████████▉ | 320/360 [00:12<00:02, 17.96it/s]
 89%|████████▉ | 322/360 [00:13<00:02, 16.96it/s]
 90%|█████████ | 324/360 [00:13<00:02, 17.42it/s]
 91%|█████████ | 326/360 [00:13<00:01, 17.55it/s]
 91%|█████████ | 328/360 [00:13<00:01, 17.53it/s]
 92%|█████████▏| 330/360 [00:13<00:01, 17.22it/s]
 92%|█████████▎| 333/360 [00:13<00:01, 18.68it/s]
 93%|█████████▎| 336/360 [00:13<00:01, 20.09it/s]
 94%|█████████▍| 338/360 [00:13<00:01, 19.60it/s]
 94%|█████████▍| 340/360 [00:14<00:01, 19.39it/s]
 95%|█████████▌| 343/360 [00:14<00:00, 19.97it/s]
 96%|█████████▋| 347/360 [00:14<00:00, 23.88it/s]
 98%|█████████▊| 353/360 [00:14<00:00, 32.05it/s]
100%|██████████| 360/360 [00:15<00:00, 23.75it/s]
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0

INFO:__main__:val progression view:
INFO:__main__:At RAW Best val, validation/test/train 33.33 33.33 33.33
INFO:__main__:At EMA Best val, validation/test/train 33.33 33.33 33.33
100%|██████████| 1/1 [01:07<00:00, 67.77s/it]
```

Type 2 - Image-Level Pre Training

The arguments below train the SAMIL model with "Image-Level" Pre Training. This is achieved by passing in `FeatureExtractor1` as the value for the `Pretrained` argument.

In [ ]:

```python
RUNS_DIR = '/content/runs/'

args = {
    'training_seed': 0,
    'Pretrained': 'FeatureExtractor1', # Options are Whole for Study Level, Feature
    'data_seed': 0,
    'checkpoint_dir': MODEL_CHECKPOINTS,
    'MIL_checkpoint_path': '',
    'use_class_weights': 'True',
    'ViewRegularization_warmup_schedule_type': 'Linear',
    'optimizer_type': 'SGD',
    'lr_schedule_type': 'CosineLR',
    'lr_cycle_epochs': 1,
    'lr': 0.0008, # learning rate
    'wd': 0.0001, # weight decay
    'T': 0.1, # tempertature
    'lambda_ViewRegularization': 15.0, # λsA
    'train_dir': RUNS_DIR + 'SAMIL',
    'resume': 'last_checkpoint.pth.tar',
    'dataset_name': 'echo',
    'train_epoch': 1, # number of epochs, 2000 defined in the paper. CHANGE ME!
    'development_size': 'DEV479',
    'lr_warmup_epochs': 0,
    'ema_decay': 0.999,
    'device': device,
    'start_epoch': 0,
    'patience': 200,
    'early_stopping_warmup': 200,
    'ViewRegularization_warmup_pos': 0.4,
    'eval_every_Xepoch': 1
}

args, brief_summary = setup_samil_train(args)

weights = args['class_weights']
weights = [float(i) for i in weights.split(',')]
weights = torch.Tensor(weights)
weights = weights.to(device)

#load the view model, the output is unnormalized logits, need to use softmax on the
view_model = create_view_model(args)
view_model.to(device)

model = create_model(args)
model.to(device)

no_decay = ['bias', 'bn']
grouped_parameters = [
    {'params': [p for n, p in model.named_parameters() if not any(
        nd in n for nd in no_decay)], 'weight_decay': args['wd']},
    {'params': [p for n, p in model.named_parameters() if any(
        nd in n for nd in no_decay)], 'weight_decay': 0.0}
]

if args['optimizer_type'] == 'SGD':
    optimizer = optim.SGD(grouped_parameters, lr=args['lr'],
                          momentum=0.9, nesterov=True)

elif args['optimizer_type'] == 'Adam':
```

```python
    optimizer = optim.Adam(grouped_parameters, lr=args['lr'])

elif args['optimizer_type'] == 'AdamW':
    optimizer = optim.AdamW(grouped_parameters, lr=args['lr'])

else:
    raise NameError('Not supported optimizer setting')

#lr_schedule_type choice
if args['lr_schedule_type'] == 'CosineLR':
    scheduler = get_cosine_schedule_with_warmup(optimizer, args['lr_warmup_epochs']

elif args['lr_schedule_type'] == 'FixedLR':
    scheduler = get_fixed_lr(optimizer, args['lr_warmup_epochs'], args['lr_cycle_ep

else:
    raise NameError('Not supported lr scheduler setting')


#instantiate the ema_model object
ema_model = ModelEMA(args, model, args['ema_decay'])


# !!! Start training
train_samil(args)
```

```
setting training seed0
INFO:__main__:Model: WideResNet 28x2
INFO:__main__:Total params for View Model: 5.93M
!!!!!!!!Using pre-calculated class weights!!!!!!!!
args.resume_checkpoint_fullpath: /content/runs/SAMIL/FeatureExtractor1/last_checkpo
int.pth.tar
!!!!!!!!!!!!!!!!!!!!!initializing from pretrained checkpoint!!!!!!!!!!!!!!!!!!!!!
INFO:__main__:Total params: 2.31M
INFO:__main__:***** Running training *****
INFO:__main__:  Task = echo
INFO:__main__:  Num Epochs = 1
INFO:__main__:  Total optimization steps = 360
self.param_keys: ['feature_extractor_part1.0.weight', 'feature_extractor_part1.0.bi
as', 'feature_extractor_part1.3.weight', 'feature_extractor_part1.3.bias', 'feature
_extractor_part1.6.weight', 'feature_extractor_part1.6.bias', 'feature_extractor_pa
rt1.9.weight', 'feature_extractor_part1.9.bias', 'feature_extractor_part2.0.weigh
t', 'feature_extractor_part2.0.bias', 'feature_extractor_part3.0.weight', 'feature_
extractor_part3.0.bias', 'feature_extractor_part3.2.weight', 'feature_extractor_par
t3.2.bias', 'attention_V.0.weight', 'attention_V.0.bias', 'attention_V.2.weight',
'attention_V.2.bias', 'attention_U.0.weight', 'attention_U.0.bias', 'attention_U.2.
weight', 'attention_U.2.bias', 'classifier.0.weight', 'classifier.0.bias']
self.buffer_keys: []
!!!!Does not have checkpoint yet!!!!
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/360 [00:42<?, ?it/s]

  0%|          | 0/119 [00:00<?, ?it/s]
  1%|          | 1/119 [00:00<01:23,  1.41it/s]
  2%|▏         | 2/119 [00:00<00:43,  2.67it/s]
  3%|▎         | 4/119 [00:01<00:24,  4.71it/s]
  6%|▌         | 7/119 [00:01<00:12,  8.93it/s]
  8%|▊         | 9/119 [00:01<00:11,  9.45it/s]
 10%|█         | 12/119 [00:01<00:08, 12.61it/s]
 13%|█▎        | 15/119 [00:01<00:06, 14.88it/s]
 14%|█▍        | 17/119 [00:01<00:06, 15.82it/s]
 17%|█▋        | 20/119 [00:01<00:05, 17.33it/s]
 19%|█▉        | 23/119 [00:02<00:05, 18.65it/s]
 22%|██▏       | 26/119 [00:02<00:04, 21.13it/s]
 24%|██▍       | 29/119 [00:02<00:04, 21.03it/s]
 28%|██▊       | 33/119 [00:02<00:03, 23.58it/s]
 30%|███       | 36/119 [00:02<00:03, 24.60it/s]
 33%|███▎      | 39/119 [00:02<00:03, 21.62it/s]
 35%|███▌      | 42/119 [00:03<00:04, 15.54it/s]
 37%|███▋      | 44/119 [00:03<00:04, 15.32it/s]
 39%|███▉      | 47/119 [00:03<00:04, 17.21it/s]
 41%|████      | 49/119 [00:03<00:04, 15.83it/s]
 43%|████▎     | 51/119 [00:03<00:04, 13.82it/s]
 45%|████▌     | 54/119 [00:03<00:04, 15.64it/s]
 48%|████▊     | 57/119 [00:03<00:03, 17.96it/s]
 50%|█████     | 60/119 [00:04<00:03, 19.39it/s]
 53%|█████▎    | 63/119 [00:04<00:03, 16.75it/s]
 55%|█████▍    | 65/119 [00:04<00:03, 17.00it/s]
 56%|█████▋    | 67/119 [00:04<00:03, 15.36it/s]
 59%|█████▉    | 70/119 [00:04<00:02, 17.26it/s]
 61%|██████    | 72/119 [00:04<00:02, 17.29it/s]
 62%|██████▏   | 74/119 [00:04<00:02, 17.39it/s]
 65%|██████▍   | 77/119 [00:05<00:02, 17.79it/s]
 66%|██████▋   | 79/119 [00:05<00:02, 16.70it/s]
 68%|██████▊   | 81/119 [00:05<00:02, 17.10it/s]
 70%|██████▉   | 83/119 [00:05<00:02, 17.45it/s]
 71%|███████▏  | 85/119 [00:05<00:02, 16.92it/s]
 73%|███████▎  | 87/119 [00:05<00:01, 16.85it/s]
 75%|███████▍  | 89/119 [00:05<00:01, 17.02it/s]
 76%|███████▋  | 91/119 [00:06<00:01, 14.33it/s]
 79%|███████▉  | 94/119 [00:06<00:01, 15.28it/s]
 81%|████████  | 96/119 [00:06<00:01, 15.55it/s]
 82%|████████▏ | 98/119 [00:06<00:01, 14.65it/s]
 84%|████████▍ | 100/119 [00:06<00:01, 15.51it/s]
 87%|████████▋ | 103/119 [00:06<00:00, 17.31it/s]
 89%|████████▉ | 106/119 [00:06<00:00, 19.09it/s]
 92%|█████████▏| 110/119 [00:06<00:00, 23.32it/s]
100%|██████████| 119/119 [00:07<00:00, 15.75it/s]
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
```

```
  0%|          | 0/120 [00:00<?, ?it/s]
  1%|          | 1/120 [00:00<01:42,  1.16it/s]
  2%|▏         | 2/120 [00:01<00:50,  2.32it/s]
  2%|▎         | 3/120 [00:01<00:38,  3.07it/s]
  4%|▍         | 5/120 [00:01<00:20,  5.62it/s]
  6%|▌         | 7/120 [00:01<00:13,  8.08it/s]
  8%|▊         | 10/120 [00:01<00:08, 12.25it/s]
 11%|█         | 13/120 [00:01<00:06, 15.79it/s]
 13%|█▎        | 16/120 [00:01<00:06, 17.12it/s]
 17%|█▋        | 20/120 [00:01<00:04, 21.31it/s]
 20%|██        | 24/120 [00:02<00:03, 25.32it/s]
 22%|██▎       | 27/120 [00:02<00:04, 22.17it/s]
 25%|██▌       | 30/120 [00:02<00:04, 22.14it/s]
 28%|██▊       | 33/120 [00:02<00:03, 23.35it/s]
 30%|███       | 36/120 [00:02<00:03, 24.88it/s]
 32%|███▎      | 39/120 [00:02<00:03, 24.92it/s]
 36%|███▌      | 43/120 [00:02<00:02, 26.43it/s]
 38%|███▊      | 46/120 [00:02<00:02, 27.27it/s]
 41%|████      | 49/120 [00:03<00:02, 27.87it/s]
 43%|████▎     | 52/120 [00:03<00:02, 28.40it/s]
 46%|████▌     | 55/120 [00:03<00:02, 24.82it/s]
 48%|████▊     | 58/120 [00:03<00:02, 24.42it/s]
 51%|█████     | 61/120 [00:03<00:02, 24.93it/s]
 53%|█████▎    | 64/120 [00:03<00:02, 24.95it/s]
 57%|█████▊    | 69/120 [00:03<00:01, 29.41it/s]
 60%|██████    | 72/120 [00:03<00:01, 24.94it/s]
 62%|██████▎   | 75/120 [00:04<00:01, 24.08it/s]
 65%|██████▌   | 78/120 [00:04<00:01, 23.91it/s]
 68%|██████▊   | 81/120 [00:04<00:01, 23.21it/s]
 70%|███████   | 84/120 [00:04<00:01, 22.40it/s]
 72%|███████▎  | 87/120 [00:04<00:01, 20.65it/s]
 76%|███████▌  | 91/120 [00:04<00:01, 23.99it/s]
 78%|███████▊  | 94/120 [00:04<00:01, 23.63it/s]
 81%|████████  | 97/120 [00:05<00:00, 23.74it/s]
 83%|████████▎ | 100/120 [00:05<00:00, 21.59it/s]
 87%|████████▋ | 104/120 [00:05<00:00, 24.64it/s]
 92%|█████████▎| 111/120 [00:05<00:00, 34.78it/s]
100%|██████████| 120/120 [00:06<00:00, 19.90it/s]
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
```

```
  0%|          | 0/360 [00:00<?, ?it/s]
  0%|          | 1/360 [00:00<04:51,  1.23it/s]
  1%|          | 2/360 [00:01<02:50,  2.10it/s]
  1%|          | 3/360 [00:01<01:56,  3.07it/s]
  2%|▏         | 6/360 [00:01<00:47,  7.38it/s]
  2%|▏         | 8/360 [00:01<00:37,  9.51it/s]
  3%|▎         | 10/360 [00:01<00:31, 11.02it/s]
  4%|▎         | 13/360 [00:01<00:24, 14.44it/s]
  4%|▍         | 16/360 [00:01<00:19, 17.98it/s]
  5%|▌         | 19/360 [00:01<00:17, 19.21it/s]
  6%|▋         | 23/360 [00:02<00:14, 23.87it/s]
  7%|▋         | 26/360 [00:02<00:13, 25.19it/s]
  8%|▊         | 29/360 [00:02<00:12, 26.42it/s]
  9%|▉         | 32/360 [00:02<00:12, 26.32it/s]
 10%|▉         | 35/360 [00:02<00:12, 25.51it/s]
 11%|█         | 38/360 [00:02<00:13, 23.90it/s]
 12%|█▏        | 42/360 [00:02<00:11, 26.93it/s]
 13%|█▎        | 46/360 [00:02<00:10, 28.78it/s]
 14%|█▎        | 49/360 [00:02<00:11, 28.23it/s]
 15%|█▍        | 53/360 [00:03<00:10, 29.35it/s]
 16%|█▌        | 56/360 [00:03<00:10, 28.78it/s]
 16%|█▋        | 59/360 [00:03<00:11, 26.21it/s]
 18%|█▊        | 64/360 [00:03<00:09, 29.81it/s]
 19%|█▊        | 67/360 [00:03<00:12, 23.97it/s]
 19%|█▉        | 70/360 [00:03<00:12, 23.23it/s]
 20%|██        | 73/360 [00:04<00:16, 17.38it/s]
 21%|██        | 76/360 [00:04<00:16, 17.65it/s]
 22%|██▏       | 78/360 [00:04<00:16, 16.73it/s]
 22%|██▏       | 80/360 [00:04<00:17, 15.79it/s]
 23%|██▎       | 82/360 [00:04<00:17, 15.63it/s]
 23%|██▎       | 84/360 [00:04<00:19, 14.10it/s]
 24%|██▍       | 86/360 [00:05<00:22, 12.41it/s]
 24%|██▍       | 88/360 [00:05<00:21, 12.64it/s]
 25%|██▌       | 91/360 [00:05<00:17, 15.34it/s]
 26%|██▌       | 93/360 [00:05<00:17, 14.95it/s]
 26%|██▋       | 95/360 [00:05<00:18, 14.53it/s]
 27%|██▋       | 97/360 [00:05<00:17, 14.95it/s]
 28%|██▊       | 99/360 [00:05<00:20, 12.96it/s]
 28%|██▊       | 102/360 [00:06<00:17, 15.01it/s]
 29%|██▉       | 104/360 [00:06<00:18, 14.04it/s]
 29%|██▉       | 106/360 [00:06<00:17, 14.58it/s]
 30%|███       | 108/360 [00:06<00:17, 14.19it/s]
 31%|███       | 110/360 [00:06<00:17, 14.04it/s]
 31%|███       | 112/360 [00:06<00:18, 13.54it/s]
 32%|███▏      | 114/360 [00:06<00:16, 14.85it/s]
 32%|███▎      | 117/360 [00:07<00:13, 17.87it/s]
 33%|███▎      | 120/360 [00:07<00:12, 18.93it/s]
 34%|███▍      | 122/360 [00:07<00:13, 17.92it/s]
 34%|███▍      | 124/360 [00:07<00:12, 18.31it/s]
 35%|███▌      | 126/360 [00:07<00:13, 16.86it/s]
 36%|███▌      | 128/360 [00:07<00:13, 17.29it/s]
 36%|███▌      | 130/360 [00:07<00:14, 16.12it/s]
 37%|███▋      | 132/360 [00:08<00:14, 15.90it/s]
 37%|███▋      | 134/360 [00:08<00:13, 16.28it/s]
 38%|███▊      | 136/360 [00:08<00:13, 16.66it/s]
 38%|███▊      | 138/360 [00:08<00:13, 16.74it/s]
 39%|███▉      | 140/360 [00:08<00:13, 16.82it/s]
 40%|███▉      | 143/360 [00:08<00:11, 19.45it/s]
```

```
 41%|████      | 146/360 [00:08<00:10, 21.17it/s]
 41%|████      | 149/360 [00:08<00:09, 22.74it/s]
 42%|████▏     | 152/360 [00:08<00:09, 21.05it/s]
 43%|████▎     | 155/360 [00:09<00:09, 21.80it/s]
 44%|████▍     | 158/360 [00:09<00:09, 21.31it/s]
 45%|████▍     | 161/360 [00:09<00:09, 20.06it/s]
 46%|████▌     | 164/360 [00:09<00:09, 20.34it/s]
 46%|████▋     | 167/360 [00:09<00:11, 16.64it/s]
 47%|████▋     | 170/360 [00:09<00:10, 18.92it/s]
 48%|████▊     | 173/360 [00:10<00:09, 19.62it/s]
 49%|████▉     | 176/360 [00:10<00:08, 21.27it/s]
 50%|█████     | 180/360 [00:10<00:07, 24.01it/s]
 51%|█████     | 183/360 [00:10<00:07, 24.12it/s]
 52%|█████▏    | 187/360 [00:10<00:06, 27.94it/s]
 53%|█████▎    | 190/360 [00:10<00:06, 26.34it/s]
 54%|█████▎    | 193/360 [00:10<00:06, 27.07it/s]
 54%|█████▍    | 196/360 [00:10<00:06, 23.49it/s]
 56%|█████▌    | 200/360 [00:11<00:06, 26.48it/s]
 56%|█████▋    | 203/360 [00:11<00:06, 23.27it/s]
 57%|█████▊    | 207/360 [00:11<00:05, 26.30it/s]
 58%|█████▊    | 210/360 [00:11<00:06, 23.67it/s]
 59%|█████▉    | 213/360 [00:11<00:06, 23.61it/s]
 60%|██████    | 216/360 [00:11<00:05, 24.10it/s]
 61%|██████    | 219/360 [00:11<00:05, 23.96it/s]
 62%|██████▏   | 222/360 [00:11<00:05, 23.69it/s]
 62%|██████▎   | 225/360 [00:12<00:06, 22.04it/s]
 63%|██████▎   | 228/360 [00:12<00:06, 20.02it/s]
 64%|██████▍   | 231/360 [00:12<00:06, 21.28it/s]
 65%|██████▌   | 234/360 [00:12<00:06, 20.67it/s]
 66%|██████▌   | 237/360 [00:12<00:05, 22.18it/s]
 67%|██████▋   | 241/360 [00:12<00:05, 23.52it/s]
 68%|██████▊   | 244/360 [00:13<00:05, 22.58it/s]
 69%|██████▊   | 247/360 [00:13<00:05, 22.49it/s]
 69%|██████▉   | 250/360 [00:13<00:04, 23.18it/s]
 70%|███████   | 253/360 [00:13<00:04, 22.71it/s]
 71%|███████▏  | 257/360 [00:13<00:04, 25.00it/s]
 72%|███████▏  | 260/360 [00:13<00:04, 24.70it/s]
 73%|███████▎  | 264/360 [00:13<00:03, 27.92it/s]
 74%|███████▍  | 267/360 [00:13<00:03, 25.12it/s]
 75%|███████▌  | 270/360 [00:14<00:04, 21.59it/s]
 76%|███████▌  | 273/360 [00:14<00:04, 20.84it/s]
 77%|███████▋  | 276/360 [00:14<00:03, 21.48it/s]
 78%|███████▊  | 279/360 [00:14<00:03, 21.11it/s]
 78%|███████▊  | 282/360 [00:14<00:03, 21.51it/s]
 79%|███████▉  | 285/360 [00:14<00:03, 21.99it/s]
 80%|████████  | 289/360 [00:14<00:02, 24.48it/s]
 81%|████████  | 292/360 [00:15<00:02, 23.11it/s]
 82%|████████▏ | 295/360 [00:15<00:02, 23.27it/s]
 83%|████████▎ | 298/360 [00:15<00:02, 21.65it/s]
 84%|████████▎ | 301/360 [00:15<00:02, 19.93it/s]
 84%|████████▍ | 304/360 [00:15<00:02, 19.89it/s]
 86%|████████▌ | 308/360 [00:15<00:02, 22.55it/s]
 86%|████████▋ | 311/360 [00:16<00:02, 20.41it/s]
 87%|████████▋ | 314/360 [00:16<00:02, 17.65it/s]
 88%|████████▊ | 316/360 [00:16<00:02, 17.95it/s]
 88%|████████▊ | 318/360 [00:16<00:02, 17.93it/s]
 89%|████████▉ | 321/360 [00:16<00:02, 19.39it/s]
 90%|█████████ | 324/360 [00:16<00:01, 20.72it/s]
 91%|█████████ | 327/360 [00:16<00:01, 22.34it/s]
 92%|█████████▏| 330/360 [00:17<00:01, 20.15it/s]
```

```
 92%|█████████▎| 333/360 [00:17<00:01, 21.09it/s]
 93%|█████████▎| 336/360 [00:17<00:01, 22.83it/s]
 94%|█████████▍| 339/360 [00:17<00:00, 23.32it/s]
 95%|█████████▌| 342/360 [00:17<00:00, 23.47it/s]
 96%|█████████▌| 346/360 [00:17<00:00, 26.68it/s]
 98%|█████████▊| 351/360 [00:17<00:00, 32.10it/s]
 99%|█████████▊| 355/360 [00:17<00:00, 34.03it/s]
100%|██████████| 360/360 [00:18<00:00, 19.49it/s]
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
Inside calculate_balanced_accuracy, 3 classes passed in
class0 recall: 0.0
class1 recall: 0.0
class2 recall: 1.0
INFO:__main__:val progression view:
INFO:__main__:At RAW Best val, validation/test/train 33.33 33.33 33.33
INFO:__main__:At EMA Best val, validation/test/train 33.33 33.33 33.33
100%|██████████| 1/1 [01:16<00:00, 76.14s/it]
```

Training Type 3 - Study-Level (i.e., Bag-Level) Pre Training

This training run trains the model with the two novel contributions of the paper: 1) the supervised-attention mechanism, and 2) the self-supervised (SSL) pre-training of the entire **study-level representations** that builds upon MoCo (V2). MoCo is a "recent method for self-supervised image-level contrastive learning (img-CL). (Huang et., al, 2023).

The arguments below specify the epoch count (`train_epoch`), the Hyperparameters (`lr` , `wd` , `T` , and `lambda_ViewRegularization`). Most importantanly, it species `Pretrained` argument as `Whole` which represents the Study Level pretraining.

In [ ]:
```
RUNS_DIR = '/content/runs/'

args = {
    'training_seed': 0,
    'Pretrained': 'Whole', # Options are Whole for Study Level, FeatureExtractor fo
    'data_seed': 0,
    'checkpoint_dir': MODEL_CHECKPOINTS,
    'MIL_checkpoint_path': '',
    'use_class_weights': 'True',
    'ViewRegularization_warmup_schedule_type': 'Linear',
    'optimizer_type': 'SGD',
    'lr_schedule_type': 'CosineLR',
    'lr_cycle_epochs': 1,
    'lr': 0.0008, # learning rate
    'wd': 0.0001, # weight decay
    'T': 0.1, # tempertature
    'lambda_ViewRegularization': 15.0, # λsA
    'train_dir': RUNS_DIR + 'SAMIL',
    'resume': 'last_checkpoint.pth.tar',
    'dataset_name': 'echo',
    'train_epoch': 1, # number of epochs, 2000 defined in the paper. CHANGE ME!
    'development_size': 'DEV479',
    'lr_warmup_epochs': 0,
    'ema_decay': 0.999,
    'device': device,
    'start_epoch': 0,
    'patience': 200,
    'early_stopping_warmup': 200,
    'ViewRegularization_warmup_pos': 0.4,
    'eval_every_Xepoch': 1
}

args, brief_summary = setup_samil_train(args)

weights = args['class_weights']
weights = [float(i) for i in weights.split(',')]
weights = torch.Tensor(weights)
weights = weights.to(device)

#load the view model, the output is unnormalized logits, need to use softmax on the
view_model = create_view_model(args)
view_model.to(device)

model = create_model(args)
model.to(device)

no_decay = ['bias', 'bn']
grouped_parameters = [
    {'params': [p for n, p in model.named_parameters() if not any(
        nd in n for nd in no_decay)], 'weight_decay': args['wd']},
    {'params': [p for n, p in model.named_parameters() if any(
        nd in n for nd in no_decay)], 'weight_decay': 0.0}
]

if args['optimizer_type'] == 'SGD':
    optimizer = optim.SGD(grouped_parameters, lr=args['lr'],
                          momentum=0.9, nesterov=True)

elif args['optimizer_type'] == 'Adam':
```

```python
    optimizer = optim.Adam(grouped_parameters, lr=args['lr'])

elif args['optimizer_type'] == 'AdamW':
    optimizer = optim.AdamW(grouped_parameters, lr=args['lr'])

else:
    raise NameError('Not supported optimizer setting')

#lr_schedule_type choice
if args['lr_schedule_type'] == 'CosineLR':
    scheduler = get_cosine_schedule_with_warmup(optimizer, args['lr_warmup_epochs']

elif args['lr_schedule_type'] == 'FixedLR':
    scheduler = get_fixed_lr(optimizer, args['lr_warmup_epochs'], args['lr_cycle_ep

else:
    raise NameError('Not supported lr scheduler setting')


#instantiate the ema_model object
ema_model = ModelEMA(args, model, args['ema_decay'])


# !!! Start training
train_samil(args)
```

```
setting training seed0
INFO:__main__:Model: WideResNet 28x2
INFO:__main__:Total params for View Model: 5.93M
!!!!!!!!Using pre-calculated class weights!!!!!!!!
args.resume_checkpoint_fullpath: /content/runs/SAMIL/Whole/last_checkpoint.pth.tar
INFO:__main__:Total params: 2.31M
INFO:__main__:***** Running training *****
INFO:__main__:  Task = echo
INFO:__main__:  Num Epochs = 1
INFO:__main__:  Total optimization steps = 360
!!!!!!!!!!!!!!!!!!!!!initializing from pretrained checkpoint!!!!!!!!!!!!!!!!!!!!
self.param_keys: ['feature_extractor_part1.0.weight', 'feature_extractor_part1.0.bi
as', 'feature_extractor_part1.3.weight', 'feature_extractor_part1.3.bias', 'feature
_extractor_part1.6.weight', 'feature_extractor_part1.6.bias', 'feature_extractor_pa
rt1.9.weight', 'feature_extractor_part1.9.bias', 'feature_extractor_part2.0.weigh
t', 'feature_extractor_part2.0.bias', 'feature_extractor_part3.0.weight', 'feature_
extractor_part3.0.bias', 'feature_extractor_part3.2.weight', 'feature_extractor_par
t3.2.bias', 'attention_V.0.weight', 'attention_V.0.bias', 'attention_V.2.weight',
'attention_V.2.bias', 'attention_U.0.weight', 'attention_U.0.bias', 'attention_U.2.
weight', 'attention_U.2.bias', 'classifier.0.weight', 'classifier.0.bias']
self.buffer_keys: []
Resuming from checkpoint: /content/runs/SAMIL/Whole/last_checkpoint.pth.tar
0it [00:00, ?it/s]
```

Evaluation

The sections below show the balanced accuracy scores after training for each of the models compared to the paper.

At the time of draft submission, only the SAMIL w/ Study Level Pretraining has been trained.

SAMIL with Study Level Pretraining

The reproduction of the SAMIL model with Study Level Pretraining was trained for 446 epochs before Colab terminated the A100 GPU runtime. At that time, the best balanced accuracy achieved was 0.663 vs. the paper's 0.754. We will continue training from the checkpoint once a more stable environment has been chosen.

The code block below compares the balanced accuracy of the SAMIL with Study Level Pre Training from the checkpoint file captured during training by downloading the checkpoint from Google Drive on a publicly available link using `gdown` .

In [ ]:
```
import os

target_balanced_accuracy = 0.754

args = {
    'training_seed': 0,
    'Pretrained': 'Whole', # Options are Whole for Study Level, FeatureExtractor fo
    'data_seed': 0,
    'checkpoint_dir': MODEL_CHECKPOINTS,
    'MIL_checkpoint_path': '',
    'use_class_weights': 'True',
    'ViewRegularization_warmup_schedule_type': 'Linear',
    'optimizer_type': 'SGD',
    'lr_schedule_type': 'CosineLR',
    'lr_cycle_epochs': 1,
    'lr': 0.0008, # learning rate
    'wd': 0.0001, # weight decay
    'T': 0.1, # tempertature
    'lambda_ViewRegularization': 15.0, # λsA
    'train_dir': RUNS_DIR + 'SAMIL',
    'resume': 'last_checkpoint.pth.tar',
    'dataset_name': 'echo',
    'train_epoch': 2000, # number of epochs, 2000 defined in the paper. CHANGE ME!
    'development_size': 'DEV479',
    'lr_warmup_epochs': 0,
    'ema_decay': 0.999,
    'device': device,
    'start_epoch': 0,
    'patience': 200,
    'early_stopping_warmup': 200,
    'ViewRegularization_warmup_pos': 0.4,
    'eval_every_Xepoch': 1
}


# Download model checkpoint
!gdown 'https://drive.google.com/uc?id=1q4ROvzCUlZdfR1qArH1_G27BPiU-Zonz'
model_checkpoint = os.path.join('', 'samil_bag_last_checkpoint.pth.tar')
```

```
Downloading...
From: https://drive.google.com/uc?id=1q4ROvzCUlZdfR1qArH1_G27BPiU-Zonz
To: /content/samil_bag_last_checkpoint.pth.tar
100% 27.8M/27.8M [00:00<00:00, 63.3MB/s]
```

In [ ]:
```
checkpoint = torch.load(model_checkpoint)

model = SAMIL().to(device)
model.load_state_dict(torch.load(model_checkpoint)['state_dict'])

model.eval()

test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False, num_workers=8)
repro_balanced_accuracy_score = eval_model_test(args, test_loader, model)

print(f"Target bal_acc (from paper): {target_balanced_accuracy}")
print(f"Reproduced bal_acc (from repro): {repro_balanced_accuracy_score}")
```

```
Target bal_acc (from paper): 0.754
Reproduced bal_acc (from repro): 0.663298139768728
```

Results

At the time of draft submission, the only result to report upon is the partial training of the SAMIL model with study-level pretraining. Training did not complete, but the reproduced balanced accuracy reached 0.663 after 446 epochs.

Variant	**Reproduced Balanced Accuracy**	**Paper Balanced Accuracy**
ABMIL	TBD	58.5
SAMIL w/ No Pretraining	TBD	72.7
SAMIL w/ Image-Level Pretraining	TBD	71.2
SAMIL w/ Study-Level Pretraining	63.3	75.4

Analyses

No analyses conducted at this time.

Discussion

Reproducibility

At this time, it cannot be concluded whether the results in the paper can be reproduced, however it can be confirmed that the steps necessary to attempt reproducibility are feasible. The code in the paper's Github repo is usable, functional, and requires little to no modification to get started. The dataset, while difficult to obtail initially, is also maintained and accessible when access is acquired from the owners.

Challenges

1. The code for this paper assumes you will run it outside of a Jupyter notebook, so refactoring of the classes was required to get this functional in a notebook.

2. While the TMED-2 dataset has an access request form, it took three attempts over several weeks to gain access. Ultimately, emailing the staff on the TMED-2 website was required beyond their signup form. For individuals looking to reproduce the results of the paper in short notice, this might pose a challenge.

3. The documentation for the paper's code in the repo was poor. There is only one example of running an experiment without making any changes to the parameters. The authors should have provided robust documentation in the Github repo for how to reproduce each model they evaluated in the various configurations.

4. The computational requirements to train each model pose a challenge. Training on a standard GPU provided in colab is estimated to take 3-4 days to run the 2,000 epoch upperbound defined in the paper. A100 GPU units were purchased from Colab and enabled, which improved the speed of training, but on several occasions, the instance would get terminated by Colab/timeout. The reproducibility of this paper should only be considered feasible if individuals are willing to acquire GPUs such as the A100 through Colab/AWS/Azure/etc.

5. Instructions in the Github repo readme were unclear/incorrect on how to load in the dataset. Modification of the data loading method was necessary to get this running based on the structure of the data from TMED-2. Perhaps this was a misunderstanding on my part, but I could not get this to work based on the original instructions.

Despite the challenges above, once the code was refactored into a notebook and the A100 GPU was used in Colab, the code itself was functional without any bug fixing or adjustment to solve for out of date libraries.

Suggestions

To improve reproducibility of this paper, the following suggestions are recommended:

1. Update the Github repo readme to specify exactly how to structure the downloaded data from TMED-2 (where to put it, what the folder structure should be, etc.)

2. TMED-2 dataset needs a better mechanism to acquire access in a faster more transparent way. The Google Form was filled out 3 times over 3 weeks with no response. An email to all the authors of the dataset was required to gain access and a response.

3. Github repo readme should outline the steps to reproduce the exact results in the paper, step by step. The readme only gives one example, and no steps on what to adjust for each experiment run in the paper. In summary, for each experiment in the paper, there should be a 1:1 instruction in the readme.

4. Paper author's should included the runtime of their model on the A100. They mentioned it ran on a single A100 in the paper, but no reference to how long for each run.

Future Plans

The following work remains to complete the reproduction of this paper. Each item will be completed by the final submission.

1. Include the ABMIL model class from the paper, train it, and include in the Results/Analysis section. This is one of the models evaluated by the paper.

2. Fully train the SAMIL model with No Pretraining by moving to an A100 instance on Lambda Labs.

3. Complete the following sections that were not completed in the draft, and the additional sections required in the final report:

    - Results: Include final Balanced Accuracy results for all 4 models.
    - Analysis: Complete an Analysis of the 4 models for the following metrics: Balanced Accuracy, Numer of Epochs before Early Stop. Analysis to include a chart of the values for each model, and graphs for Balanced Accuracy vs. Epoch, Loss vs. Epoch.
    - Include a section for Environment setup (Python version, packages), Data visualizations, detailed information on the Hyperparams used by this notebook vs. in the paper, additional writeups on the models, and include a section for the Ablation Study.
4. Simulate image data to enable an individual to run through the notebook without having access to the TMED-2 dataset.

I kindly ask you to consider these Future Plans during the assessment of this draft. I have recently moved into a new role in my full time job and have been unable to dedicate sufficient time to this draft, in additional to hitting multiple blockers during training in Colab.

References

@misc{huang2021new, title={A New Semi-supervised Learning Benchmark for Classifying View and Diagnosing Aortic Stenosis from Echocardiograms}, author={Zhe Huang and Gary Long and Benjamin Wessler and Michael C. Hughes}, year={2021}, eprint={2108.00080}, archivePrefix={arXiv}, primaryClass={cs.CV} }

@misc{huang2024detecting, title={Detecting Heart Disease from Multi-View Ultrasound Images via Supervised Attention Multiple Instance Learning}, author={Zhe Huang and Benjamin S. Wessler and Michael C. Hughes}, year={2024}, eprint={2306.00003}, archivePrefix={arXiv}, primaryClass={eess.IV} }