In [ ]:
```
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
import torch

from torch.utils.data import DataLoader, Dataset
from torchvision import models, transforms
device = "cuda" if torch.cuda.is_available() else 'cpu'
print(device)
import wandb
import torch.nn as nn

import time
```

```
cuda
```

In [ ]:
```
wandb.login()
```

```
Failed to detect the name of this notebook, you can set it manually with the WANDB_
NOTEBOOK_NAME environment variable to enable code saving.
wandb: Currently logged in as: sup3rm. Use `wandb login --relogin` to force relogin
```

Out[ ]:
```
True
```

In [ ]:
```
from data_utils import load_dataset, LESION_TYPE
```

CLIP Zero-Shot Classification

In [ ]:
```
import clip
```

In [ ]:
```
clip_model, clip_preprocess = clip.load("ViT-B/32", device=device)
```

In [ ]:
```
BATCH_SIZE = 128
```

In [ ]:
```
def clip_zero_shot(data_set, classes):
    # https://colab.research.google.com/drive/1IqJfogZdC61dgE4BDQILCJS-zUiphD4y?aut
    data_loader = DataLoader(data_set, batch_size=BATCH_SIZE, shuffle=True, num_wor
    # Encode text features here
    text_inputs = torch.cat([clip.tokenize(f"a photo of a {c}, a type of skin lesio
    with torch.no_grad():
        text_features = clip_model.encode_text(text_inputs)
    text_features /= text_features.norm(dim=-1, keepdim=True)
    # Encode image features here
    correct = 0
    total = 0
    for image, label in tqdm(data_loader):
        image, label = image.to(device), label.to(device)
        with torch.no_grad():
            image_features = clip_model.encode_image(image)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        similarity = (100.0 * image_features @ text_features.T).softmax(dim=-1)
        _, pred = similarity.max(dim=-1)
        correct += (pred == label).sum().item()
        total += len(label)

    return correct / total
```

Testing NIH dataset with CLIP zero-shot classification w/ NIH labels

In [ ]:
```
from data_utils import NIH_CLASS_TYPES


nih_train, nih_test = load_dataset("NIH", transform=clip_preprocess)
print(f"Train size: {len(nih_train)}")
print(f"Test size: {len(nih_test)}")

# NIH_CLASS_TYPES
nih_classes = list(NIH_CLASS_TYPES)  # From the data_utils.py file
```

```
Loading NIH dataset...
Train size: 100908
Test size: 11212
```

In [ ]:
```
BATCH_SIZE = 128
```

In [ ]:
```
t0 = time.time()

accuracy = clip_zero_shot(data_set=nih_train, classes=nih_classes)

t1 = time.time()
total = t1-t0
print(f"\n CLIP Zero Shot Accuracy on NIH Train: {100*accuracy:.3f}%")
print(f"Time taken: {total} seconds")
```

```
  0%|          | 0/789 [00:00<?, ?it/s]
 CLIP Zero Shot Accuracy on NIH Train: 0.5727899869187775
Time taken: 470.55077934265137 seconds
```

CLIP Linear-Probe Classification

Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
def get_features(data_set):
    all_features = []
    all_labels = []

    with torch.no_grad():
        for images, labels in tqdm(DataLoader(data_set, batch_size=BATCH_SIZE)):
            features = clip_model.encode_image(images.to(device))
            all_features.append(features)
            all_labels.append(labels)

    return torch.cat(all_features).cpu().numpy(), torch.cat(all_labels).cpu().numpy
```

NIH dataset with CLIP Logistic Regression w/ NIH labels

In [ ]:
```
# calculate the image features
t0 = time.time()

nih_train_features, nih_train_labels = get_features(nih_train)
nih_test_features, nih_test_labels = get_features(nih_test)

t1 = time.time()
total = t1-t0
print(f"Time taken to get features: {total} seconds")
```

```
  0%|          | 0/1577 [00:00<?, ?it/s]
  0%|          | 0/176 [00:00<?, ?it/s]
Time taken to get features: 1065.6440699100494 seconds
```

In [ ]:
```
t0 = time.time()

# Perform logistic regression for the NIH dataset
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1,
classifier.fit(nih_train_features, nih_train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(nih_test_features)

accuracy = np.mean((nih_test_labels == predictions).astype(float))
print(f"\n CLIP logistic regression accuracy = {100*accuracy:.3f}%")

t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate logistic regression: {total} seconds")
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed: 10.8min finished
 CLIP logistic regression accuracy = 56.975%
Time taken to train and evaluate logistic regression: 648.9348454475403 seconds
```

SVM

In [ ]:
```
from sklearn import svm
```

NIH dataset with CLIP SVM classification w/ NIH labels

In [ ]:
```
t0 = time.time()

# Perform logistic regression
classifier = svm.SVC(random_state=0, C=0.316, max_iter=5000, verbose=1)
classifier.fit(nih_train_features, nih_train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(nih_test_features)

accuracy = np.mean((nih_test_labels == predictions).astype(float))
print(f"\nAccuracy = {100*accuracy:.3f}%")

t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate SVM: {total} seconds")
```

```
[LibSVM]
c:\Users\mario\anaconda3\Lib\site-packages\sklearn\svm\_base.py:297: ConvergenceWar
ning: Solver terminated early (max_iter=5000).  Consider pre-processing your data w
ith StandardScaler or MinMaxScaler.
  warnings.warn(
Accuracy = 54.245%
Time taken to train and evaluate SVM: 2172.5718846321106 seconds
```

K-Means Clusteriungfrom scipy import stats

NIH dataset with CLIP K-Means clustering w/ NIH labels

In [ ]:
```
from scipy import stats
```

In [ ]:
```
def knn(x_train, y_train, x_test, y_test, K=15):
    # Needs code here
    test_pred = []
    for i in tqdm(range(len(x_test))):
        distance = np.linalg.norm(x_train - x_test[i], axis=-1)
        indices = np.argsort(distance)[:K]
        neighbors_labels = y_train[indices]
        test_pred.append(stats.mode(neighbors_labels).mode[0])

    correct = (test_pred == y_test).sum()
    total = len(y_test)

    return correct / total
```

In [ ]:
```
t0 = time.time()

accuracy = knn(nih_train_features, nih_train_labels, nih_test_features, nih_test_la
print(f"\nNIH CLIP K-NN Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate K-NN: {total} seconds")
```

```
  0%|          | 0/11212 [00:00<?, ?it/s]
C:\Users\mario\AppData\Local\Temp\ipykernel_13924\1507260031.py:8: FutureWarning: U
nlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of
`mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior w
ill change: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no longer be a
ccepted. Set `keepdims` to True or False to avoid this warning.
  test_pred.append(stats.mode(neighbors_labels).mode[0])
NIH CLIP K-NN Accuracy = 40.885%
Time taken to train and evaluate K-NN: 7065.8533890247345 seconds
```

In [ ]:
```
from sklearn.cluster import KMeans
```

In [ ]:
```
t0 = time.time()

# Perform logistic regression
classifier = KMeans(n_clusters=15)
classifier.fit(nih_train_features, nih_train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(nih_test_features)
accuracy = np.mean((nih_test_labels == predictions).astype(float))
print(f"\nNIH CLIP KMeans Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate KMeans: {total} seconds")
```

```
c:\Users\mario\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureW
arning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the
value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
NIH CLIP KMeans Accuracy = 7.742%
Time taken to train and evaluate KMeans: 35.73114538192749 seconds
```

Random Forest

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

NIH dataset with CLIP Random Forest classification w/ NIH labels

In [ ]:
```
t0 = time.time()
# Perform logistic regression
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(nih_train_features, nih_train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(nih_test_features)
accuracy = np.mean((nih_test_labels == predictions).astype(float))
print(f"\n NIH CLIP Random Forest Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate Random Forest: {total} seconds")
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:   13.2s
 NIH CLIP Random Forest Accuracy = 55.342%
Time taken to train and evaluate Random Forest: 46.70379066467285 seconds
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   46.4s finished
[Parallel(n_jobs=16)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  18 tasks      | elapsed:    0.0s
[Parallel(n_jobs=16)]: Done 100 out of 100 | elapsed:    0.0s finished
```

ResNet 50

In [ ]:
```
resnet_preprocess = models.ResNet50_Weights.IMAGENET1K_V2.transforms()
weights = models.ResNet50_Weights.IMAGENET1K_V2
resnet50 = models.resnet50(weights=weights)

# Change last layer
num_features = resnet50.fc.in_features
resnet50.fc = nn.Linear(num_features, len(NIH_CLASS_TYPES))

resnet50.to(device)
```

Out[ ]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fa
lse)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=Fa
lse)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
```

```
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
```

```
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
```

```
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
```

```
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=15, bias=True)
)
```

In [ ]:
```
from torch.optim import Adam
```

In [ ]:
```
NIH_train_data, NIH_test_data = load_dataset("NIH", transform=resnet_preprocess)
```

```
Loading NIH dataset...
```

In [ ]:
```
def evaluate(model, dataloader):
    model.eval()
    with torch.no_grad():
        num_correct = 0
        total = 0
        for images, labels in tqdm(dataloader, desc="Evaluating", position=2, leave
            num_correct += torch.sum(labels.to(device) == torch.argmax(model(images
            total += labels.size(0)
        return num_correct / total
```

In [ ]:

```
def train(model, optim, loss_fn, train_data, test_data, config):
    """
    Train a PyTorch model using the provided parameters.

    :param model: PyTorch model to train
    :param optim: Optimizer to use for training
    :param loss_fn: Loss function to use for training
    :param train_data: Training dataset
    :param test_data: Test dataset
    :param num_epochs: Number of epochs to train for (default is 100)
    :param batch_size: Batch size to use for data loading (default is 32)
    """
    model.train()
    run = wandb.init(
    # Set the project where this run will be logged
    project="vision-project-resnet",
    # Track hyperparameters and run metadata
    config=config)

    num_epochs = config['epochs']
    batch_size = config['batch_size']
    # Create data loaders
    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_
    test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_w

    for epoch in tqdm(range(num_epochs), desc="Epochs", position=0, leave=True):
        train_loss = 0.0
        correct_train = 0
        total_train = 0

        for inputs, targets in tqdm(train_loader, desc="Training", position=1, leav
            # Forward pass
            targets = targets.long().to(device)
            outputs = model(inputs.to(device))
            loss = loss_fn(outputs, targets)

            # Backward pass and optimization
            optim.zero_grad()
            loss.backward()
            optim.step()

            # Calculate train loss
            train_loss += loss.item()
            predicted = torch.argmax(outputs, 1)
            total_train += targets.size(0)
            correct_train += (predicted == targets).sum().item()

        if (epoch+1) % 2 == 0 or epoch == num_epochs - 1:
            train_loss /= len(train_loader)
            train_accuracy = correct_train / total_train

            test_accuracy = evaluate(model, test_loader)
            model.train()

            # , Test Loss: {test_loss:.4f}
            # print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, T

            # Log metrics to wandb
```

```
            wandb.log({
                "epoch": epoch+1,
                "train_loss": train_loss,
                "train_accuracy": train_accuracy,
                "test_accuracy": test_accuracy
            })
```

In [ ]:
```
config = {
    "learning_rate":1e-5,
    "batch_size":64,
    "epochs":50,
    "weight_decay":1e-5,
}
```

Zero-Shot Resnet

NIH Chest X-Ray Dataset

In [ ]:
```
NIH_test_loader = DataLoader(NIH_test_data, batch_size=64, shuffle=False, num_worke
```

In [ ]:
```
t0 = time.time()

print(evaluate(resnet50, NIH_test_loader))

t1 = time.time()
total = t1-t0
print(f"Time taken to evaluate ResNet50: {total} seconds")
```

```
Evaluating:   0%|          | 0/176 [00:00<?, ?it/s]
0.04504102747056725
Time taken to evaluate ResNet50: 51.259777784347534 seconds
```

Fine-Tuned Resnet

In [ ]:
```
optim = Adam(resnet50.parameters(), lr=config['learning_rate'], weight_decay=config
loss = nn.CrossEntropyLoss()
```

In [ ]:
```
NIH_train_data, NIH_test_data = load_dataset("NIH", transform=resnet_preprocess)
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
import torch

t0 = time.time()
train(resnet50, optim, loss, NIH_train_data, NIH_test_data, config)

t1 = time.time()
total = t1-t0
print(f"Time taken to train ResNet50: {total} seconds")
```

```
Loading NIH dataset...
```

wandb version 0.16.1 is available! To upgrade, please run: $ pip install wandb --upgrade

Tracking run with wandb version 0.16.0

Run data is saved locally in c:\GitHub\Evaluating-CLIP-Features-for-Medical-Image-Classification\wandb\run-20231211_230036-1dlcv37f

Syncing run **stellar-darkness-14** to Weights & Biases (docs)

View project at https://wandb.ai/sup3rm/vision-project-resnet

View run at https://wandb.ai/sup3rm/vision-project-resnet/runs/1dlcv37f

```
Epochs:   0%|          | 0/50 [00:00<?, ?it/s]
Training:   0%|          | 0/1577 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[33], line 7
      4 import torch
      6 t0 = time.time()
----> 7 train(resnet50, optim, loss, NIH_train_data, NIH_test_data, config)
      9 t1 = time.time()
     10 total = t1-t0

Cell In[28], line 39, in train(model, optim, loss_fn, train_data, test_data, config)
     37 # Backward pass and optimization
     38 optim.zero_grad()
---> 39 loss.backward()
     40 optim.step()
     42 # Calculate train loss

File c:\Users\mario\anaconda3\Lib\site-packages\torch\_tensor.py:492, in Tensor.backward(self, gradient, retain_graph, create_graph, inputs)
    482 if has_torch_function_unary(self):
    483     return handle_torch_function(
    484         Tensor.backward,
    485         (self,),
   (...)
    490         inputs=inputs,
    491     )
--> 492 torch.autograd.backward(
    493     self, gradient, retain_graph, create_graph, inputs=inputs
    494 )

File c:\Users\mario\anaconda3\Lib\site-packages\torch\autograd\__init__.py:251, in backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables, inputs)
    246     retain_graph = create_graph
    248 # The reason we repeat the same comment below is that
    249 # some Python versions print out the first line of a multi-line function
    250 # calls in the traceback and some print out the last line
--> 251 Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
    252     tensors,
    253     grad_tensors_,
    254     retain_graph,
    255     create_graph,
    256     inputs,
    257     allow_unreachable=True,
    258     accumulate_grad=True,
    259 )

RuntimeError: CUDA error: device-side assert triggered
Compile with `TORCH_USE_CUDA_DSA` to enable device-side assertions.
```

In [ ]:
```
t0 = time.time()

print(evaluate(resnet50, NIH_test_loader))

t1 = time.time()
total = t1-t0
print(f"Time taken to evaluate ResNet50: {total} seconds")
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

def plot_confusion_matrix(model, data_loader, classes):
    model.eval()
    y_pred = []
    y_true = []
    with torch.no_grad():
        for images, labels in tqdm(data_loader, desc="Evaluating", position=2, leav
            y_pred.extend(torch.argmax(model(images.to(device)), 1).cpu().numpy())
            y_true.extend(labels.cpu().numpy())

    cm = confusion_matrix(y_true, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.figure(figsize=(20,20))
    sns.heatmap(cm, annot=True, fmt='.2f', cmap="Blues", xticklabels=classes, ytick
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

plot_confusion_matrix(resnet50, NIH_test_loader, list(NIH_CLASS_TYPES))
```

Implement a zero-shot function for medclip

Load NIH Chest X-ray dataset

In [ ]:
```
import os
# os.chdir('../')
print(os.getcwd())
```

```
c:\GitHub\Evaluating-CLIP-Features-for-Medical-Image-Classification
```

In [ ]:

```
import torch
import torchvision
import torch.nn.functional as F
from tqdm import tqdm
from transformers import AutoTokenizer
from torch.utils.data import DataLoader

# Device configuration
from data_utils import load_nih_dataset_split, NIH_CLASS_TYPES, load_dataset
from medclip import MedCLIPModel, MedCLIPVisionModelViT, MedCLIPVisionModel
from build.lib.medclip.constants import BERT_TYPE, IMG_MEAN, IMG_STD, IMG_SIZE

# debuggin
from PIL import Image

BATCH_SIZE = 128

transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize((IMG_SIZE, IMG_SIZE)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean=[IMG_MEAN], std=[IMG_STD])
])


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# NIH_CLASS_TYPES
classes = list(NIH_CLASS_TYPES)  # From the data_utils.py file
print(classes)

# nih_train, nih_test = load_nih_dataset_split(transform=transform)
nih_train, nih_test = load_dataset("NIH", transform=transform, data_dir='data/nih/'
```

```
['Atelectasis', 'Consolidation', 'Infiltration', 'Pneumothorax', 'Edema', 'Emphysem
a', 'Fibrosis', 'Effusion', 'Pneumonia', 'Pleural_thickening', 'Cardiomegaly', 'Nod
ule', 'Mass', 'Hernia', 'No Finding']
Loading NIH dataset...
```

In [ ]:
```
def medclip_zero_shot(model, test_dataset, classes, batch_size=BATCH_SIZE):
    # Data loader for the dataset
    data_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True, num

    # Prepare text prompts
    text_prompts = [f"a photo of a {c}, a type of Chest x ray." for c in classes]
    # Initialize the tokenizer
    tokenizer = AutoTokenizer.from_pretrained(BERT_TYPE)

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

    # Tokenize text prompts and convert to tensors
    text_tokens = [tokenizer(text, return_tensors='pt', padding=True, truncation=Fa

    # Encode text prompts using MedClip's text model
    # Inside the medclip_zero_shot function
    text_features = [
        model.encode_text(
            input_ids=tokens['input_ids'].to(device),
            attention_mask=tokens['attention_mask'].to(device)
        )
        for tokens in text_tokens
    ]

    # Initialize variables for accuracy calculation
    correct = 0
    total = 0
    # print('text_features', text_features)
    for images, labels in tqdm(data_loader):
        images, labels = images.to(device), labels.to(device)
        # Encode images using MedClip's vision model
        # with torch.no_grad():
        image_features = model.encode_image(images)
        # Flatten text_features into a single 2D tensor
        text_features_tensor = torch.cat(text_features, dim=0)

        # Calculate similarity and make predictions
        similarity = torch.matmul(image_features, text_features_tensor.t())
        _, predictions = similarity.max(dim=-1)

        # Update correct and total counts
        correct += (predictions == labels).sum().item()
        total += len(labels)

    return correct / total
```

Load MedCLIP-ResNet50

In [ ]:
```
MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:208: UserWa
rning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in th
e future, please use 'weights' instead.
  warnings.warn(
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223: UserWa
rning: Arguments other than a weight enum or `None` for 'weights' are deprecated si
nce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet50_
Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not us
ed when initializing BertModel: ['cls.seq_relationship.weight', 'cls.predictions.tr
ansform.dense.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.prediction
s.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.wei
ght', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
```

In [ ]:
```
t0 = time.time()

accuracy = medclip_zero_shot(MedCLIP_ResNet50_model, nih_train, classes)
print(f"\n MedCLIP ResNet50 Zero Shot Accuracy = {100*accuracy:.3f}%")

t1 = time.time()
total = t1-t0
print(f"Time taken to evaluate MedCLIP ResNet50 Zero Shot: {total} seconds")
```

```
Device: cuda
100%|██████████| 789/789 [10:34<00:00,  1.24it/s]
 MedCLIP ResNet50 Zero Shot Accuracy = 53.138%
Time taken to evaluate MedCLIP ResNet50 Zero Shot: 635.1536309719086 seconds
```

Load MedCLIP-ViT

In [ ]:
```
MedCLIP_ViT_model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torch\functional.py:504: UserWarning: to
rch.meshgrid: in an upcoming release, it will be required to pass the indexing argu
ment. (Triggered internally at C:\cb\pytorch_1000000000000\work\aten\src\ATen\nativ
e\TensorShape.cpp:3527.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Some weights of the model checkpoint at microsoft/swin-tiny-patch4-window7-224 were
not used when initializing SwinModel: ['classifier.weight', 'classifier.bias']
- This IS expected if you are initializing SwinModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing SwinModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not us
ed when initializing BertModel: ['cls.seq_relationship.weight', 'cls.predictions.tr
ansform.dense.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.prediction
s.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.wei
ght', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
```

In [ ]:

```
t0 = time.time()

accuracy = medclip_zero_shot(MedCLIP_ViT_model, nih_train, classes)
print(f"\n MedCLIP ViT Zero Shot Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to evaluate MedCLIP ViT Zero Shot: {total} seconds")
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torch\functional.py:504: UserWarning: to
rch.meshgrid: in an upcoming release, it will be required to pass the indexing argu
ment. (Triggered internally at C:\cb\pytorch_1000000000000\work\aten\src\ATen\nativ
e\TensorShape.cpp:3527.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Some weights of the model checkpoint at microsoft/swin-tiny-patch4-window7-224 were
not used when initializing SwinModel: ['classifier.weight', 'classifier.bias']
- This IS expected if you are initializing SwinModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing SwinModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not us
ed when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.s
eq_relationship.bias', 'cls.predictions.transform.dense.bias', 'cls.predictions.bia
s', 'cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.weight', 'c
ls.predictions.transform.dense.weight', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
Device: cuda
100%|██████████| 789/789 [3:36:49<00:00, 16.49s/it]
Accuracy = 16.531%
Time taken to evaluate MedCLIP ViT Zero Shot: 13009.078595161438 seconds
```

In [ ]:
```python
import numpy as np
def get_features(data_set, model):
    all_features = []
    all_labels = []

    with torch.no_grad():
        for images, labels in tqdm(DataLoader(data_set, batch_size=BATCH_SIZE)):
            features = model.encode_image(images.to(device))
            all_features.append(features)
            all_labels.append(labels)

    return torch.cat(all_features).cpu().numpy(), torch.cat(all_labels).cpu().numpy
```

In [ ]:
```python
# Calculate the ResNet50 features for the NIH dataset
t0 = time.time()

MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)
medclip_resnet50_nih_train_features, medclip_resnet50_nih_train_labels = get_featur
medclip_resnet50_nih_test_features, medclip_resnet50_nih_test_labels = get_features

t1 = time.time()
total = t1-t0
print(f"Time taken to get MedCLIP ResNet50 features: {total} seconds")
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:208: UserWa
rning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in th
e future, please use 'weights' instead.
  warnings.warn(
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223: UserWa
rning: Arguments other than a weight enum or `None` for 'weights' are deprecated si
nce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet50_
Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not us
ed when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.s
eq_relationship.bias', 'cls.predictions.transform.dense.bias', 'cls.predictions.bia
s', 'cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.weight', 'c
ls.predictions.transform.dense.weight', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
100%|██████████| 789/789 [20:23<00:00,  1.55s/it]
100%|██████████| 88/88 [02:14<00:00,  1.52s/it]
```

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
t0 = time.time()

# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1,
classifier.fit(medclip_resnet50_nih_train_features, medclip_resnet50_nih_train_labe

# Evaluate using the logistic regression classifier
predictions = classifier.predict(medclip_resnet50_nih_test_features)
accuracy = np.mean((medclip_resnet50_nih_test_labels == predictions).astype(float))
print(f"\n MedClip ResNet50 logistic regression Accuracy = {100*accuracy:.3f}%")

t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ResNet50 logistic regression: {tot
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:  1.5min finished
 MedClip ResNet50 NIH Image Features Accuracy = 54.995%
Time taken to train and evaluate MedClip ResNet50 logistic regression: 92.214000701
9043 seconds
```

In [ ]:
```python
t0 = time.time()

# Calculate the ViT features for the NIH dataset
MedCLIP_ViT_model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)
medclip_vit_nih_train_features, medclip_vit_nih_train_labels = get_features(nih_tra
medclip_vit_nih_test_features, medclip_vit_nih_test_labels = get_features(nih_test,

t1 = time.time()
total = t1-t0
print(f"Time taken to get MedCLIP ViT features: {total} seconds")
```

```
Some weights of the model checkpoint at microsoft/swin-tiny-patch4-window7-224 were
not used when initializing SwinModel: ['classifier.weight', 'classifier.bias']
- This IS expected if you are initializing SwinModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing SwinModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not us
ed when initializing BertModel: ['cls.predictions.bias', 'cls.predictions.transfor
m.LayerNorm.weight', 'cls.predictions.transform.dense.weight', 'cls.predictions.tra
nsform.dense.bias', 'cls.seq_relationship.bias', 'cls.seq_relationship.weight', 'cl
s.predictions.decoder.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
100%|██████████| 789/789 [20:41<00:00,  1.57s/it]
100%|██████████| 88/88 [02:18<00:00,  1.58s/it]
Time taken to get MedCLIP ViT features: 1381.7960591316223 seconds
```

In [ ]:

```python
from sklearn.linear_model import LogisticRegression

t0 = time.time()
# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1,
classifier.fit(medclip_vit_nih_train_features, medclip_vit_nih_train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(medclip_vit_nih_test_features)
accuracy = np.mean((medclip_vit_nih_test_labels == predictions).astype(float))
print(f"\n MedClip ViT NIH Image Features Accuracy = {100*accuracy:.3f}%")

t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ViT logistic regression: {total} s
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:  1.4min finished
 MedClip ViT NIH Image Features Accuracy = 56.894%
Time taken to train and evaluate MedClip ViT logistic regression: 85.84021830558777
seconds
```

SVM testing for MedCLIP-ResNet50 and MedCLIP-ViT

In [ ]:
```
import torch
from torch.utils.data import DataLoader
from tqdm import tqdm

# Device configuration
from medclip import MedCLIPModel, MedCLIPVisionModelViT
from medclip.modeling_medclip import MedCLIPVisionModel

import numpy as np
def get_features(data_set, model):
    all_features = []
    all_labels = []

    with torch.no_grad():
        for images, labels in tqdm(DataLoader(data_set, batch_size=BATCH_SIZE)):
            features = model.encode_image(images.to(device))
            all_features.append(features)
            all_labels.append(labels)

    return torch.cat(all_features).cpu().numpy(), torch.cat(all_labels).cpu().numpy
```

ResNet50

In [ ]:
```
t0 = time.time()

# ResNet50
MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)

# NIH
MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NIH_train_labels
MedCLIP_ResNet50_model_NIH_test_features, MedCLIP_ResNet50_model_NIH_test_labels =

t1 = time.time()
total = t1-t0
print(f"Time taken to get MedCLIP ResNet50 features: {total} seconds")
```

```
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not us
ed when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.p
redictions.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transf
orm.LayerNorm.weight', 'cls.predictions.decoder.weight', 'cls.seq_relationship.bias
', 'cls.seq_relationship.weight', 'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
100%|██████████| 141/141 [01:04<00:00,  2.19it/s]
100%|██████████| 16/16 [00:07<00:00,  2.28it/s]
100%|██████████| 1577/1577 [22:22<00:00,  1.17it/s]
100%|██████████| 176/176 [02:30<00:00,  1.17it/s]
```

ViT

In [ ]:
```
t0 = time.time()

# ViT
MedCLIP_ViT_model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)

# NIH
MedCLIP_ViT_model_NIH_train_features, MedCLIP_ViT_model_NIH_train_labels = get_feat
MedCLIP_ViT_model_NIH_test_features, MedCLIP_ViT_model_NIH_test_labels = get_featur

t1 = time.time()
total = t1-t0
print(f"Time taken to get MedCLIP ViT features: {total} seconds")
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torch\functional.py:504: UserWarning: to
rch.meshgrid: in an upcoming release, it will be required to pass the indexing argu
ment. (Triggered internally at C:\cb\pytorch_1000000000000\work\aten\src\ATen\nativ
e\TensorShape.cpp:3527.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Some weights of the model checkpoint at microsoft/swin-tiny-patch4-window7-224 were
not used when initializing SwinModel: ['classifier.weight', 'classifier.bias']
- This IS expected if you are initializing SwinModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing SwinModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not us
ed when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.p
redictions.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transf
orm.LayerNorm.weight', 'cls.predictions.decoder.weight', 'cls.seq_relationship.bias
', 'cls.seq_relationship.weight', 'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassif
ication model from a BertForSequenceClassification model).
100%|██████████| 141/141 [01:13<00:00,  1.93it/s]
100%|██████████| 16/16 [00:08<00:00,  1.97it/s]
100%|██████████| 1577/1577 [24:34<00:00,  1.07it/s]
100%|██████████| 176/176 [02:36<00:00,  1.13it/s]
```

In [ ]:
```
from sklearn import svm
from sklearn.preprocessing import StandardScaler
import numpy as np

# Perform SVM regression
classifier = svm.SVC(random_state=0, C=0.316, max_iter=1000, verbose=1)

# Data preprocessing with StandardScaler
scaler = StandardScaler()
```

In [ ]:
```
# ResNet50 and ViT Models

# NIH
scaler.fit(MedCLIP_ResNet50_model_NIH_train_features)
MedCLIP_ResNet50_model_NIH_train_features = scaler.transform(MedCLIP_ResNet50_model
MedCLIP_ResNet50_model_NIH_test_features = scaler.transform(MedCLIP_ResNet50_model_

scaler.fit(MedCLIP_ViT_model_NIH_train_features)
MedCLIP_ViT_model_NIH_train_features = scaler.transform(MedCLIP_ViT_model_NIH_train
MedCLIP_ViT_model_NIH_test_features = scaler.transform(MedCLIP_ViT_model_NIH_test_f
```

NIH Chest X-ray dataset

In [ ]:
```
t0 = time.time()

# NIH ResNet50
classifier.fit(MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NI
predictions = classifier.predict(MedCLIP_ResNet50_model_NIH_test_features)
accuracy = np.mean((MedCLIP_ResNet50_model_NIH_test_labels == predictions).astype(f
print(f"\n MedClip ResNet50 NIH SVM Image Features Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ResNet50 SVM: {total} seconds")
```

```
[LibSVM]
c:\Users\mario\anaconda3\Lib\site-packages\sklearn\svm\_base.py:297: ConvergenceWar
ning: Solver terminated early (max_iter=1000).  Consider pre-processing your data w
ith StandardScaler or MinMaxScaler.
  warnings.warn(
 MedClip ResNet50 NIH SVM Image Features Accuracy = 33.518%
```

In [ ]:
```
t0 = time.time()

# NIH ViT
classifier.fit(MedCLIP_ViT_model_NIH_train_features, MedCLIP_ViT_model_NIH_train_la
predictions = classifier.predict(MedCLIP_ViT_model_NIH_test_features)
accuracy = np.mean((MedCLIP_ViT_model_NIH_test_labels == predictions).astype(float)
print(f"\n MedClip ViT NIH SVM Image Features Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ViT SVM: {total} seconds")
```

```
[LibSVM]
c:\Users\mario\anaconda3\Lib\site-packages\sklearn\svm\_base.py:297: ConvergenceWar
ning: Solver terminated early (max_iter=1000).  Consider pre-processing your data w
ith StandardScaler or MinMaxScaler.
  warnings.warn(
 MedClip ViT NIH SVM Image Features Accuracy = 34.748%
```

K-Means testing for MedCLIP-ResNet50 and MedCLIP-ViT

In [ ]:
```
# Perform KNN regression
from scipy import stats
def knn(x_train, y_train, x_test, y_test, K=15):
    # Needs code here
    test_pred = []
    for i in tqdm(range(len(x_test))):
        distance = np.linalg.norm(x_train - x_test[i], axis=-1)
        indices = np.argsort(distance)[:K]
        neighbors_labels = y_train[indices]
        test_pred.append(stats.mode(neighbors_labels).mode[0])

    correct = (test_pred == y_test).sum()
    total = len(y_test)

    return correct / total
```

NIH Chest X-ray dataset

In [ ]:
```
t0 = time.time()

# Perform KNN regression for ResNet50 NIH
accuracy = knn(MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NI
print(f"\n MedClip ResNet50 NIH Image Features Accuracy = {100*accuracy:.3f}%")

t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ResNet50 KNN: {total} seconds")
```

```
  0%|          | 0/11212 [00:00<?, ?it/s]C:\Users\mario\AppData\Local\Temp\ipykerne
l_20616\3553024242.py:10: FutureWarning: Unlike other reduction functions (e.g. `sk
ew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it ac
ts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be eliminat
ed, and the value None will no longer be accepted. Set `keepdims` to True or False
to avoid this warning.
  test_pred.append(stats.mode(neighbors_labels).mode[0])
100%|██████████| 11212/11212 [18:25<00:00, 10.14it/s]
 MedClip ResNet50 NIH Image Features Accuracy = 41.322%
```

In [ ]:
```
t0 = time.time()

# Perform KNN regression for ViT NIH
accuracy = knn(MedCLIP_ViT_model_NIH_train_features, MedCLIP_ViT_model_NIH_train_la
print(f"\n MedClip ViT NIH Image Features Accuracy = {100*accuracy:.3f}%")

t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ViT KNN: {total} seconds")
```

```
  0%|          | 0/11212 [00:00<?, ?it/s]C:\Users\mario\AppData\Local\Temp\ipykernel_20616\3553024242.py:10: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  test_pred.append(stats.mode(neighbors_labels).mode[0])
100%|██████████| 11212/11212 [18:33<00:00, 10.07it/s]
 MedClip ViT NIH Image Features Accuracy = 42.133%
```

Random Forest testing for MedCLIP-ResNet50 and MedCLIP-ViT

NIH Chest X-ray dataset

In [ ]:
```
# Perform Random Forest regression for ResNet50 NIH
from sklearn.ensemble import RandomForestClassifier

t0 = time.time()
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NI

# Evaluate using the logistic regression classifier for ResNet50 NIH
predictions = classifier.predict(MedCLIP_ResNet50_model_NIH_test_features)
accuracy = np.mean((MedCLIP_ResNet50_model_NIH_test_labels == predictions).astype(f
print(f"\n MedClip ResNet50 NIH Image Features Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ResNet50 Random Forest: {total} se
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:   18.1s
 MedClip ResNet50 NIH Image Features Accuracy = 53.951%
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.0min finished
[Parallel(n_jobs=16)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  18 tasks      | elapsed:    0.0s
[Parallel(n_jobs=16)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [ ]:
```
# Perform Random Forest regression for ViT NIH
from sklearn.ensemble import RandomForestClassifier

t0 = time.time()
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(MedCLIP_ViT_model_NIH_train_features, MedCLIP_ViT_model_NIH_train_la

# Evaluate using the logistic regression classifier for ViT NIH
predictions = classifier.predict(MedCLIP_ViT_model_NIH_test_features)
accuracy = np.mean((MedCLIP_ViT_model_NIH_test_labels == predictions).astype(float)
print(f"\n MedClip ViT NIH Image Features Accuracy = {100*accuracy:.3f}%")


t1 = time.time()
total = t1-t0
print(f"Time taken to train and evaluate MedClip ViT Random Forest: {total} seconds
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:   18.2s
 MedClip ViT NIH Image Features Accuracy = 54.772%
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.1min finished
[Parallel(n_jobs=16)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  18 tasks      | elapsed:    0.0s
[Parallel(n_jobs=16)]: Done 100 out of 100 | elapsed:    0.0s finished
```