```
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
```

          cuda

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

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 [ ]:   ham_train, ham_test = load_dataset("HAM10000", transform=clip_preprocess)

          print(f"Train size: {len(ham_train)}")
          print(f"Test size: {len(ham_test)}")
          print(ham_train)
          print(ham_test)
```

          Loading HAM10000 dataset...
          Train size: 9013
          Test size: 1002
          <torch.utils.data.dataset.Subset object at 0x000002144EAB63D0>
          <torch.utils.data.dataset.Subset object at 0x000002144EAB6390>

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

```
In [ ]:   def clip_zero_shot(data_set, classes):
              # https://colab.research.google.com/drive/1IqJfogZdC61dgE4BDQILCJS-zUiphD4y?authuser=2#scrol
              data_loader = DataLoader(data_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
              # Encode text features here
              text_inputs = torch.cat([clip.tokenize(f"a photo of a {c}, a type of skin lesion.") for c in
              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 HAM10000 dataset with CLIP zero-shot classification

In [ ]:
```
lesion_classes = LESION_TYPE.values() # This was probably only because the class labels were num
```

In [ ]:
```
accuracy = clip_zero_shot(data_set=ham_train, classes=lesion_classes)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```
```
  0%|          | 0/71 [00:00<?, ?it/s]
Accuracy = 21.147%
```

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 = 64
accuracy = clip_zero_shot(data_set=nih_train, classes=nih_classes)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```
```
  0%|          | 0/1577 [00:00<?, ?it/s]
Accuracy = 0.572%
```

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()
```

HAM10000 dataset with CLIP Logistic Regression

In [ ]:
```
# Calculate the image features
train_features, train_labels = get_features(ham_train)
test_features, test_labels = get_features(ham_test)
```

```
 0%|           | 0/71 [00:00<?, ?it/s]
 0%|           | 0/8 [00:00<?, ?it/s]
```

In [ ]:
```
# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1, n_jobs=-1)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done    1 out of    1 | elapsed:   20.8s finished
Accuracy = 81.737%
```

NIH dataset with CLIP Logistic Regression w/ NIH labels

In [ ]:
```
# calculate the image features
train_features, train_labels = get_features(nih_train)
test_features, test_labels = get_features(nih_test)
```

```
100%|██████████| 1577/1577 [22:23<00:00,  1.17it/s]
100%|██████████| 176/176 [02:29<00:00,  1.18it/s]
```

In [ ]:
```
# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1, n_jobs=-1)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done    1 out of    1 | elapsed: 8.5min finished
Accuracy = 56.894%
```

SVM

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

HAM10000 dataset with CLIP SVM classification

In [ ]:
```
# Perform logistic regression
classifier = svm.SVC(random_state=0, C=0.316, max_iter=5000, verbose=1)
classifier.fit(train_features, train_labels)

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

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

[LibSVM]
Accuracy = 71.457%

NIH dataset with CLIP SVM classification w/ NIH labels

In [ ]:
```
# Perform logistic regression
classifier = svm.SVC(random_state=0, C=0.316, max_iter=5000, verbose=1)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

[LibSVM]
c:\Users\mario\anaconda3\Lib\site-packages\sklearn\svm_base.py:297: ConvergenceWarning: Solver
terminated early (max_iter=5000). Consider pre-processing your data with StandardScaler or MinM
axScaler.
  warnings.warn(
Accuracy = 54.210%

K-Means Clusteriungfrom scipy import stats

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

In [ ]:
```
def knn(x_train, y_train, x_test, y_test, K=5):
    # 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 [ ]:
```
accuracy = knn(train_features, train_labels, test_features, test_labels, K=1)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

  0%| | 0/1002 [00:00<?, ?it/s]
C:\Users\mario\AppData\Local\Temp\ipykernel_12396\3497779333.py:8: FutureWarning: Unlike other r
eduction 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 `keepdi
ms` 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])
Accuracy = 76.347%

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

In [ ]:
```
# Perform logistic regression
classifier = KMeans(n_clusters=7)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
```

```
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
c:\Users\mario\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explici
tly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Accuracy = 19.561%
```

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

In [ ]:
```python
from scipy import stats
def knn(x_train, y_train, x_test, y_test, K=5):
    # 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 [ ]:
```python
accuracy = knn(train_features, train_labels, test_features, test_labels, K=1)
print(f"\nNIH CLIP scipy Accuracy = {100*accuracy:.3f}%")
```

```
  0%|          | 0/11212 [00:00<?, ?it/s]C:\Users\mario\AppData\Local\Temp\ipykernel_9832\588109
221.py:9: 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 wi
ll change: the default value of `keepdims` will become False, the `axis` over which the statisti
c 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 [1:57:28<00:00,  1.59it/s]
NIH CLIP scipy Accuracy = 40.885%
```

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

In [ ]:
```python
# Perform logistic regression
classifier = KMeans(n_clusters=7)
classifier.fit(train_features, train_labels)

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

```
c:\Users\mario\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explici
tly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
NIH CLIP sklearn.KMeans Accuracy = 1.677%
```

Random Forest

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

```python
# Perform logistic regression
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    0.8s
Accuracy = 71.457%
```

```
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    3.3s 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
```

NIH dataset with CLIP Random Forest classification w/ NIH labels

```python
from sklearn.ensemble import RandomForestClassifier
# Perform logistic regression
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\n NIH CLIP sklearn.RandomForestClassifier Accuracy = {100*accuracy:.3f}%")
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:   14.3s
 NIH CLIP sklearn.RandomForestClassifier Accuracy = 55.137%
```

```
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   48.3s 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

```python
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(LESION_TYPE))

resnet50.to(device);
```

```python
from torch.optim import Adam
```

```python
HAM_train_data, HAM_test_data = load_dataset("HAM10000", transform=resnet_preprocess)
NIH_train_data, NIH_test_data = load_dataset("NIH", transform=resnet_preprocess)
```

```
Loading HAM10000 dataset...
Loading NIH dataset...
```

```python
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=False):
            num_correct += torch.sum(labels.to(device) == torch.argmax(model(images.to(device)),
            total += labels.size(0)
        return num_correct / total
```

```python
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_workers=2)
    test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=2)

    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, leave=False):
            # Forward pass
            targets = targets.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}, Train Accuracy
```

```
                            # 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

HAM10000 Dataset

In [ ]:
```
HAM_test_loader = DataLoader(HAM_test_data, batch_size=64, shuffle=False, num_workers=2)
```

In [ ]:
```
print(evaluate(resnet50, HAM_test_loader))
```

```
Evaluating:   0%|              | 0/16 [00:00<?, ?it/s]
0.312375249500998
```

NIH Chest X-Ray Dataset

In [ ]:
```
NIH_test_loader = DataLoader(NIH_test_data, batch_size=64, shuffle=False, num_workers=2)
```

In [ ]:
```
print(evaluate(resnet50, NIH_test_loader))
```

```
Evaluating:   0%|              | 0/176 [00:00<?, ?it/s]
0.02167320727791652
```

Fine-Tuned Resnet

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

HAM10000 Dataset

In [ ]:
```
train(resnet50, optim, loss, HAM_train_data, HAM_test_data, config)
```

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-20231210_000108-mmoyd7kv

Syncing run **grateful-grass-4** 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/mmoyd7kv

```
Epochs:    0%|          | 0/50 [00:00<?, ?it/s]
Training:   0%|          | 0/141 [00:00<?, ?it/s]
Training:   0%|          | 0/141 [00:00<?, ?it/s]
Evaluating:   0%|          | 0/16 [00:00<?, ?it/s]
Training:   0%|          | 0/141 [00:00<?, ?it/s]
```

```
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Evaluating:    0%|             | 0/16 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
Training:    0%|             | 0/141 [00:00<?, ?it/s]
```

```
Evaluating:   0%|                | 0/16 [00:00<?, ?it/s]
Training:     0%|                | 0/141 [00:00<?, ?it/s]
Training:     0%|                | 0/141 [00:00<?, ?it/s]
Evaluating:   0%|                | 0/16 [00:00<?, ?it/s]
Training:     0%|                | 0/141 [00:00<?, ?it/s]
Training:     0%|                | 0/141 [00:00<?, ?it/s]
Evaluating:   0%|                | 0/16 [00:00<?, ?it/s]
```

In [ ]:
```
print(evaluate(resnet50, HAM_test_loader))
```

```
Evaluating:   0%|                | 0/16 [00:00<?, ?it/s]
0.8562874251497006
```

NIH Chest X-Ray Dataset

In [ ]:
```
train(resnet50, optim, loss, NIH_train_data, NIH_test_data, config)
print(evaluate(resnet50, NIH_test_loader))
```

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_004851-w9toyvdz`

Syncing run **worldly-deluge-7** 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/w9toyvdz

```
Epochs:   0%|                | 0/50 [00:07<?, ?it/s]
```

```
-----------------------------------------------------------------------------
RuntimeError                               Traceback (most recent call last)
Cell In[53], line 1
----> 1 train(resnet50, optim, loss, NIH_train_data, NIH_test_data, config)
      2 print(evaluate(resnet50, NIH_test_loader))

Cell In[48], line 35, in train(model, optim, loss_fn, train_data, test_data, config)
     33 targets = targets.to(device)
     34 outputs = model(inputs.to(device))
---> 35 loss = loss_fn(outputs, targets)
     37 # Backward pass and optimization
     38 optim.zero_grad()

File c:\Users\mario\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1518, in Module._wrap
ped_call_impl(self, *args, **kwargs)
   1516     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1517 else:
-> 1518     return self._call_impl(*args, **kwargs)

File c:\Users\mario\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1527, in Module._call
_impl(self, *args, **kwargs)
   1522 # If we don't have any hooks, we want to skip the rest of the logic in
   1523 # this function, and just call forward.
   1524 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self.
_forward_pre_hooks
   1525         or _global_backward_pre_hooks or _global_backward_hooks
   1526         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527     return forward_call(*args, **kwargs)
   1529 try:
   1530     result = None

File c:\Users\mario\anaconda3\Lib\site-packages\torch\nn\modules\loss.py:1179, in CrossEntropyLo
ss.forward(self, input, target)
   1178 def forward(self, input: Tensor, target: Tensor) -> Tensor:
-> 1179     return F.cross_entropy(input, target, weight=self.weight,
   1180                            ignore_index=self.ignore_index, reduction=self.reduction,
   1181                            label_smoothing=self.label_smoothing)

File c:\Users\mario\anaconda3\Lib\site-packages\torch\nn\functional.py:3053, in cross_entropy(in
put, target, weight, size_average, ignore_index, reduce, reduction, label_smoothing)
   3051 if size_average is not None or reduce is not None:
   3052     reduction = _Reduction.legacy_get_string(size_average, reduce)
-> 3053 return torch._C._nn.cross_entropy_loss(input, target, weight, _Reduction.get_enum(reduct
ion), ignore_index, label_smoothing)

RuntimeError: "nll_loss_forward_reduce_cuda_kernel_2d_index" not implemented for 'Char'
```

```python
import os
print(os.getcwd())
```

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

Implement a zero-shot function for medclip

```python
# implement a zero-shot function for medclip

import torch
import torchvision
from transformers import AutoTokenizer
from torch.utils.data import DataLoader
from tqdm import tqdm
```

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

# debuggin
from PIL import Image

# prepare for the demo image and texts
from build.lib.medclip.constants import BERT_TYPE, IMG_MEAN, IMG_STD, IMG_SIZE
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
from data_utils import load_dataset, LESION_TYPE, load_ham10000_dataset

BATCH_SIZE = 64
```

```python
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_workers=4)

    # 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=False, add_spec

    # 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

    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 HAM10000 dataset and test MedClip's zero-shot capabilities

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

ham_train, ham_test = load_ham10000_dataset(data_dir="data/ham10000/", transform=transform)
classes = list(LESION_TYPE.values())  # From the data_utils.py file
```

Loading HAM10000 dataset...

MedCLIP_ResNet50_model

In [ ]:
```python
# load MedCLIP-ResNet50
MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)
accuracy = medclip_zero_shot(MedCLIP_ResNet50_model, ham_train, classes)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not used when initi
alizing BertModel: ['cls.predictions.transform.dense.bias', 'cls.seq_relationship.bias', 'cls.pr
edictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predicti
ons.transform.dense.weight', 'cls.seq_relationship.weight', 'cls.predictions.bias', 'cls.predict
ions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on a
nother task or with another architecture (e.g. initializing a BertForSequenceClassification mode
l from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification model from a BertFo
rSequenceClassification model).
Device: cuda
100%|██████████| 141/141 [00:23<00:00, 6.11it/s]
Accuracy = 22.346%

MedCLIP_ViT_model

In [ ]:
```python
# load MedCLIP-ViT
MedCLIP_ViT_model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)
accuracy = medclip_zero_shot(MedCLIP_ViT_model, ham_train, classes)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
Some weights of the model checkpoint at microsoft/swin-tiny-patch4-window7-224 were not used whe
n initializing SwinModel: ['classifier.bias', 'classifier.weight']
- This IS expected if you are initializing SwinModel from the checkpoint of a model trained on a
nother task or with another architecture (e.g. initializing a BertForSequenceClassification mode
l from a BertForPreTraining model).
- This IS NOT expected if you are initializing SwinModel from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification model from a BertFo
rSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not used when initi
alizing BertModel: ['cls.predictions.transform.dense.bias', 'cls.seq_relationship.bias', 'cls.pr
edictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predicti
ons.transform.dense.weight', 'cls.seq_relationship.weight', 'cls.predictions.bias', 'cls.predict
ions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on a
nother task or with another architecture (e.g. initializing a BertForSequenceClassification mode
l from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification model from a BertFo
rSequenceClassification model).
Device: cuda
100%|██████████| 141/141 [00:25<00:00,  5.63it/s]
Accuracy = 27.593%
```

Load NIH Chest X-ray dataset

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

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

In [ ]:
```python
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
classes

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

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

```python
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_workers=2)

    # 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=False, add_spec
    # print('text_tokens',  text_prompts)
    # 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

```python
MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)
MedCLIP_ResNet50_model
accuracy = medclip_zero_shot(MedCLIP_ResNet50_model, nih_train, classes)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The pa
rameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weig
hts' instead.
  warnings.warn(
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Argume
nts other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be remove
d in the future. The current behavior is equivalent to passing `weights=ResNet50_Weights.IMAGENE
T1K_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 used when initi
alizing BertModel: ['cls.predictions.transform.dense.bias', 'cls.predictions.transform.LayerNor
m.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.decoder.weight', 'cls.pr
edictions.transform.dense.weight', 'cls.seq_relationship.bias', 'cls.seq_relationship.weight',
'cls.predictions.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on a
nother task or with another architecture (e.g. initializing a BertForSequenceClassification mode
l from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification model from a BertFo
rSequenceClassification model).
Device: cuda
100%|██████████| 789/789 [04:57<00:00,  2.66it/s]
Accuracy = 53.138%
```

Load MedCLIP-ViT

In [ ]:
```python
MedCLIP_ViT_model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)
accuracy = medclip_zero_shot(MedCLIP_ViT_model, nih_train, classes)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torch\functional.py:504: UserWarning: torch.meshgrid:
in an upcoming release, it will be required to pass the indexing argument. (Triggered internally
at C:\cb\pytorch_1000000000000\work\aten\src\ATen\native\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 whe
n initializing SwinModel: ['classifier.weight', 'classifier.bias']
- This IS expected if you are initializing SwinModel from the checkpoint of a model trained on a
nother task or with another architecture (e.g. initializing a BertForSequenceClassification mode
l from a BertForPreTraining model).
- This IS NOT expected if you are initializing SwinModel from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification model from a BertFo
rSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not used when initi
alizing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.seq_relationship.bias', 'cl
s.predictions.transform.dense.bias', 'cls.predictions.bias', 'cls.predictions.transform.LayerNor
m.weight', 'cls.seq_relationship.weight', 'cls.predictions.transform.dense.weight', 'cls.predict
ions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on a
nother task or with another architecture (e.g. initializing a BertForSequenceClassification mode
l from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification model from a BertFo
rSequenceClassification model).
Device: cuda
100%|██████████| 789/789 [3:36:49<00:00, 16.49s/it]
Accuracy = 16.531%
```

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

```python
    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
MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)

# Calculate the image features
train_features, train_labels = get_features(nih_train, MedCLIP_ResNet50_model)
test_features, test_labels = get_features(nih_test, MedCLIP_ResNet50_model)
```

```
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The pa
rameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weig
hts' instead.
  warnings.warn(
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Argume
nts other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be remove
d in the future. The current behavior is equivalent to passing `weights=ResNet50_Weights.IMAGENE
T1K_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 used when initi
alizing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.seq_relationship.bias', 'cl
s.predictions.transform.dense.bias', 'cls.predictions.bias', 'cls.predictions.transform.LayerNor
m.weight', 'cls.seq_relationship.weight', 'cls.predictions.transform.dense.weight', 'cls.predict
ions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on a
nother task or with another architecture (e.g. initializing a BertForSequenceClassification mode
l from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification model from a BertFo
rSequenceClassification 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
# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1, n_jobs=-1)
classifier.fit(train_features, train_labels)

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

```
[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%
```

In [ ]:
```python
# same thing for ViT
MedCLIP_ViT_model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)

# Calculate the image features
train_features, train_labels = get_features(nih_train, MedCLIP_ViT_model)
test_features, test_labels = get_features(nih_test, MedCLIP_ViT_model)
```

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1, n_jobs=-1)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\n MedClip ResNet50 HAM1000 Image Features Accuracy = {100*accuracy:.3f}%")
```

In [ ]:
```python
# same thing for ViT
MedCLIP_ViT_model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)

# Calculate the image features
train_features, train_labels = get_features(ham_train, MedCLIP_ViT_model)
test_features, test_labels = get_features(ham_test, MedCLIP_ViT_model)

from sklearn.linear_model import LogisticRegression
# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, verbose=1, n_jobs=-1)
classifier.fit(train_features, train_labels)

# Evaluate using the logistic regression classifier
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\n MedClip ViT HAM1000 Image Features Accuracy = {100*accuracy:.3f}%")
```

SVM testing for MedCLIP-ResNet50 and MedCLIP-ViT

In [ ]:
```python
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 [ ]:
```python
# ResNet50
MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)

# HAM10000
MedCLIP_ResNet50_model_HAM_train_features, MedCLIP_ResNet50_model_HAM_train_labels = get_feature
MedCLIP_ResNet50_model_HAM_test_features, MedCLIP_ResNet50_model_HAM_test_labels = get_features(

# NIH
```

```
MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NIH_train_labels = get_feature
MedCLIP_ResNet50_model_NIH_test_features, MedCLIP_ResNet50_model_NIH_test_labels = get_features(
```

ViT

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

# HAM10000
MedCLIP_ViT_model_HAM_train_features, MedCLIP_ViT_model_HAM_train_labels = get_features(ham_trai
MedCLIP_ViT_model_HAM_test_features, MedCLIP_ViT_model_HAM_test_labels = get_features(ham_test,

# NIH
MedCLIP_ViT_model_NIH_train_features, MedCLIP_ViT_model_NIH_train_labels = get_features(nih_trai
MedCLIP_ViT_model_NIH_test_features, MedCLIP_ViT_model_NIH_test_labels = get_features(nih_test,
```

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

```python
# 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
         # HAM10000
         scaler.fit(MedCLIP_ResNet50_model_HAM_train_features)
         MedCLIP_ResNet50_model_HAM_train_features = scaler.transform(MedCLIP_ResNet50_model_HAM_train_fe
         MedCLIP_ResNet50_model_HAM_test_features = scaler.transform(MedCLIP_ResNet50_model_HAM_test_feat

         scaler.fit(MedCLIP_ViT_model_HAM_train_features)
         MedCLIP_ViT_model_HAM_train_features = scaler.transform(MedCLIP_ViT_model_HAM_train_features)
         MedCLIP_ViT_model_HAM_test_features = scaler.transform(MedCLIP_ViT_model_HAM_test_features)

         # NIH
         scaler.fit(MedCLIP_ResNet50_model_NIH_train_features)
         MedCLIP_ResNet50_model_NIH_train_features = scaler.transform(MedCLIP_ResNet50_model_NIH_train_fe
         MedCLIP_ResNet50_model_NIH_test_features = scaler.transform(MedCLIP_ResNet50_model_NIH_test_feat

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

HAM10000

```
In [ ]:  # HAM10000 ResNet50
         classifier.fit(MedCLIP_ResNet50_model_HAM_train_features, MedCLIP_ResNet50_model_HAM_train_label
         predictions = classifier.predict(MedCLIP_ResNet50_model_HAM_test_features)
         accuracy = np.mean((MedCLIP_ResNet50_model_HAM_test_labels == predictions).astype(float))
         print(f"\n MedClip ResNet50 HAM1000 SVM Image Features Accuracy = {100*accuracy:.3f}%")
```

[LibSVM]

c:\Users\mario\anaconda3\Lib\site-packages\sklearn\svm_base.py:297: ConvergenceWarning: Solver
terminated early (max_iter=1000). Consider pre-processing your data with StandardScaler or MinM
axScaler.
  warnings.warn(
 MedClip ResNet50 HAM1000 SVM Image Features Accuracy = 75.948%

```
In [ ]:  # HAM10000 ViT
         classifier.fit(MedCLIP_ViT_model_HAM_train_features, MedCLIP_ViT_model_HAM_train_labels)
         predictions = classifier.predict(MedCLIP_ViT_model_HAM_test_features)
         accuracy = np.mean((MedCLIP_ViT_model_HAM_test_labels == predictions).astype(float))
         print(f"\n MedClip ViT HAM1000 SVM Image Features Accuracy = {100*accuracy:.3f}%")
```

[LibSVM]

c:\Users\mario\anaconda3\Lib\site-packages\sklearn\svm_base.py:297: ConvergenceWarning: Solver
terminated early (max_iter=1000). Consider pre-processing your data with StandardScaler or MinM
axScaler.
  warnings.warn(
 MedClip ViT HAM1000 SVM Image Features Accuracy = 76.946%

NIH Chest X-ray dataset

```
In [ ]:  # NIH ResNet50
         classifier.fit(MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NIH_train_label
         predictions = classifier.predict(MedCLIP_ResNet50_model_NIH_test_features)
         accuracy = np.mean((MedCLIP_ResNet50_model_NIH_test_labels == predictions).astype(float))
         print(f"\n MedClip ResNet50 NIH SVM Image Features Accuracy = {100*accuracy:.3f}%")
```

[LibSVM]

In [ ]:
```python
# NIH ViT
classifier.fit(MedCLIP_ViT_model_NIH_train_features, MedCLIP_ViT_model_NIH_train_labels)
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}%")
```

```
[LibSVM]
```

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

In [ ]:
```python
# Perform KNN regression
from scipy import stats
def knn(x_train, y_train, x_test, y_test, K=5):
    # 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
```

HAM10000 Dataset

In [ ]:
```python
# Perform KNN regression for ResNet50 HAM10000
accuracy = knn(MedCLIP_ResNet50_model_HAM_train_features, MedCLIP_ResNet50_model_HAM_train_label
print(f"\n MedClip ResNet50 HAM1000 Image Features Accuracy = {100*accuracy:.3f}%")
```

In [ ]:
```python
# Perform KNN regression for ViT HAM10000
accuracy = knn(MedCLIP_ViT_model_HAM_train_features, MedCLIP_ViT_model_HAM_train_labels, MedCLIP_
print(f"\n MedClip ViT HAM1000 Image Features Accuracy = {100*accuracy:.3f}%")
```

NIH Chest X-ray dataset

```python
# Perform KNN regression for ResNet50 NIH
accuracy = knn(MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NIH_train_label
print(f"\n MedClip ResNet50 NIH Image Features Accuracy = {100*accuracy:.3f}%")
```

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

Random Forest testing for MedCLIP-ResNet50 and MedCLIP-ViT

HAM10000 dataset

```python
# Perform Random Forest regression for ResNet50 HAM10000
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(MedCLIP_ResNet50_model_HAM_train_features, MedCLIP_ResNet50_model_HAM_train_label

# Evaluate using the logistic regression classifier for ResNet50 HAM10000
predictions = classifier.predict(MedCLIP_ResNet50_model_HAM_test_features)
accuracy = np.mean((MedCLIP_ResNet50_model_HAM_test_labels == predictions).astype(float))
print(f"\n MedClip ResNet50 HAM1000 Image Features Accuracy = {100*accuracy:.3f}%")
```

```
    MedClip ResNet50 HAM1000 Image Features Accuracy = 68.463%
```

```
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    3.7s 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 [ ]:
```python
# Perform Random Forest regression for ViT HAM10000
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(MedCLIP_ViT_model_HAM_train_features, MedCLIP_ViT_model_HAM_train_labels)

# Evaluate using the logistic regression classifier for ViT HAM10000
predictions = classifier.predict(MedCLIP_ViT_model_HAM_test_features)
accuracy = np.mean((MedCLIP_ViT_model_HAM_test_labels == predictions).astype(float))
print(f"\n MedClip ViT HAM1000 Image Features Accuracy = {100*accuracy:.3f}%")
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    1.2s
 MedClip ViT HAM1000 Image Features Accuracy = 70.060%
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    4.4s 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
```

NIH Chest X-ray dataset

In [ ]:
```python
# Perform Random Forest regression for ResNet50 NIH
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(MedCLIP_ResNet50_model_NIH_train_features, MedCLIP_ResNet50_model_NIH_train_label

# 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(float))
print(f"\n MedClip ResNet50 NIH Image Features Accuracy = {100*accuracy:.3f}%")
```

```
[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 [ ]:
```python
# Perform Random Forest regression for ViT NIH
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=0, verbose=1, n_jobs=-1)
classifier.fit(MedCLIP_ViT_model_NIH_train_features, MedCLIP_ViT_model_NIH_train_labels)

# 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}%")
```

```
[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
```