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 [ ]:
```
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?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 HAM10000 dataset with CLIP zero-shot classification

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

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

accuracy = clip_zero_shot(data_set=ham_train, classes=lesion_classes)
print(f"\n CLIP Zero Shot Accuracy on HAM Train: {accuracy:.3f}")

t1 = time.time()
total = t1-t0
print(f"\n Time taken: {total} seconds")
```

```
  0%|          | 0/71 [00:00<?, ?it/s]
Accuracy = 21.147%
```

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

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

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

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

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

# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, 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}%")

t1 = time.time()
total = t1-t0
print(f"\n 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:   20.8s finished
Accuracy = 81.737%
```

SVM

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

HAM10000 dataset with CLIP SVM classification

In [ ]:
```
t0 = time.time()
# 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}%")

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

```
[LibSVM]
Accuracy = 71.457%
```

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

accuracy = knn(train_features, train_labels, test_features, test_labels, K=1)
print(f"\nAccuracy = {100*accuracy:.3f}%")

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

```
  0%|          | 0/1002 [00:00<?, ?it/s]
C:\Users\mario\AppData\Local\Temp\ipykernel_12396\3497779333.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])
Accuracy = 76.347%
```

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

In [ ]:
```
t0 = time.time()
# 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}%")

t1 = time.time()
total = t1-t0
print(f"\n 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)
Accuracy = 19.561%
```

Random Forest

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

In [ ]:
```
t0 = time.time()
# 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}%")

t1 = time.time()
total = t1-t0
print(f"\n 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:    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
```

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(LESION_TYPE))

resnet50.to(device)
```

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

In [ ]:
```
HAM_train_data, HAM_test_data = load_dataset("HAM10000", transform=resnet_preproces
```

```
Loading HAM10000 dataset...
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.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

HAM10000 Dataset

In [ ]:
```
HAM_test_loader = DataLoader(HAM_test_data, batch_size=64, shuffle=False, num_worke
```

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

print(evaluate(resnet50, HAM_test_loader))

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

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

Fine-Tuned Resnet

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

HAM10000 Dataset

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

train(resnet50, optim, loss, HAM_train_data, HAM_test_data, config)

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

```
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]
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 [ ]:

```
t0 = time.time()

print(evaluate(resnet50, HAM_test_loader))

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

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

In [ ]:

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

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

Implement a zero-shot function for medclip

In [ ]:
```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

# 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
```

In [ ]:
```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

    # 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

    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 [ ]:
```
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=
ham_train, ham_test = load_dataset("HAM10000", transform=transform)
classes = list(LESION_TYPE.values())  # From the data_utils.py file
```

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

MedCLIP_ResNet50_model

In [ ]:
```
# 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 us
ed when initializing BertModel: ['cls.predictions.transform.dense.bias', 'cls.seq_r
elationship.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.t
ransform.LayerNorm.bias', 'cls.predictions.transform.dense.weight', 'cls.seq_relati
onship.weight', 'cls.predictions.bias', '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%|██████████| 141/141 [00:23<00:00,  6.11it/s]
Accuracy = 22.346%
```

MedCLIP_ViT_model

In [ ]:
```
# 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 when initializing SwinModel: ['classifier.bias', 'classifier.weight']
- 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.dense.bias', 'cls.seq_r
elationship.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.t
ransform.LayerNorm.bias', 'cls.predictions.transform.dense.weight', 'cls.seq_relati
onship.weight', 'cls.predictions.bias', '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%|██████████| 141/141 [00:25<00:00,  5.63it/s]
Accuracy = 27.593%
```

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
# now for HAM10000
MedCLIP_ResNet50_model = MedCLIPModel(vision_cls=MedCLIPVisionModel).to(device)

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

```
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%|██████████| 141/141 [01:31<00:00,  1.54it/s]
100%|██████████| 16/16 [00:06<00:00,  2.33it/s]
```

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
# Perform logistic regression
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=10000, 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"\n MedClip ResNet50 HAM1000 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:    2.4s finished
 MedClip ResNet50 HAM1000 Image Features Accuracy = 73.054%
```

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,
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}%")
```

```
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).
100%|████████| 141/141 [04:20<00:00,  1.85s/it]
100%|████████| 16/16 [00:19<00:00,  1.24s/it]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    1.7s finished
 MedClip ViT HAM1000 Image Features Accuracy = 74.152%
```

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

# HAM10000
MedCLIP_ResNet50_model_HAM_train_features, MedCLIP_ResNet50_model_HAM_train_labels
MedCLIP_ResNet50_model_HAM_test_features, MedCLIP_ResNet50_model_HAM_test_labels =
```

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

# HAM10000
MedCLIP_ViT_model_HAM_train_features, MedCLIP_ViT_model_HAM_train_labels = get_feat
MedCLIP_ViT_model_HAM_test_features, MedCLIP_ViT_model_HAM_test_labels = get_featur
```

```
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 [ ]:
```python
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 [ ]:
```python
# ResNet50 and ViT Models
# HAM10000
scaler.fit(MedCLIP_ResNet50_model_HAM_train_features)
MedCLIP_ResNet50_model_HAM_train_features = scaler.transform(MedCLIP_ResNet50_model
MedCLIP_ResNet50_model_HAM_test_features = scaler.transform(MedCLIP_ResNet50_model_

scaler.fit(MedCLIP_ViT_model_HAM_train_features)
MedCLIP_ViT_model_HAM_train_features = scaler.transform(MedCLIP_ViT_model_HAM_train
MedCLIP_ViT_model_HAM_test_features = scaler.transform(MedCLIP_ViT_model_HAM_test_f
```

HAM10000

In [ ]:
```python
# HAM10000 ResNet50
classifier.fit(MedCLIP_ResNet50_model_HAM_train_features, MedCLIP_ResNet50_model_HA
predictions = classifier.predict(MedCLIP_ResNet50_model_HAM_test_features)
accuracy = np.mean((MedCLIP_ResNet50_model_HAM_test_labels == predictions).astype(f
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: ConvergenceWar
ning: Solver terminated early (max_iter=1000).  Consider pre-processing your data w
ith StandardScaler or MinMaxScaler.
  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_la
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: ConvergenceWar
ning: Solver terminated early (max_iter=1000).  Consider pre-processing your data w
ith StandardScaler or MinMaxScaler.
  warnings.warn(
 MedClip ViT HAM1000 SVM Image Features Accuracy = 76.946%
```

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=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 [ ]:
```
# Perform KNN regression for ResNet50 HAM10000
accuracy = knn(MedCLIP_ResNet50_model_HAM_train_features, MedCLIP_ResNet50_model_HA
print(f"\n MedClip ResNet50 HAM1000 Image Features Accuracy = {100*accuracy:.3f}%")
```

```
  0%|          | 0/1002 [00:00<?, ?it/s]C:\Users\mario\AppData\Local\Temp\ipykernel
_20616\3553024242.py:10: FutureWarning: Unlike other reduction functions (e.g. `ske
w`, `kurtosis`), the default behavior of `mode` typically preserves the axis it act
s 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 eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False t
o avoid this warning.
  test_pred.append(stats.mode(neighbors_labels).mode[0])
100%|██████████| 1002/1002 [00:06<00:00, 149.38it/s]
```

```
 MedClip ResNet50 HAM1000 Image Features Accuracy = 76.347%
```

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

```
  0%|          | 0/1002 [00:00<?, ?it/s]C:\Users\mario\AppData\Local\Temp\ipykernel
_20616\3553024242.py:10: FutureWarning: Unlike other reduction functions (e.g. `ske
w`, `kurtosis`), the default behavior of `mode` typically preserves the axis it act
s 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 eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False t
o avoid this warning.
  test_pred.append(stats.mode(neighbors_labels).mode[0])
100%|██████████| 1002/1002 [00:06<00:00, 153.57it/s]
 MedClip ViT HAM1000 Image Features Accuracy = 76.447%
```

Random Forest testing for MedCLIP-ResNet50 and MedCLIP-ViT

HAM10000 dataset

In [ ]:
```
# 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_HA

# 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(f
print(f"\n MedClip ResNet50 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:    1.0s
 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 [ ]:
```
# 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_la

# 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
```