ham10000

December 8, 2023

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

[ ]:
```
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
```

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

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

1 CLIP Zero-Shot Classification

[ ]:
```
import clip
```

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

[ ]:
```
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)
```

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

[ ]:

```
def clip_zero_shot(data_set, classes):
    # https://colab.research.google.com/drive/1IqJfogZdC61dgE4BDQILCJS-zUiphD4y?
 ↪authuser=2#scrollTo=EuZFg3ZlHOVD
    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 classes]).to(device)
    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
```

[ ]:

```
lesion_classes = LESION_TYPE.values() # This was probably only because the␣
 ↪class labels were numbers, not strs
```

[ ]:

```
accuracy = clip_zero_shot(data_set=ham_train, classes=lesion_classes)
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

2 CLIP Linear-Probe Classification

2.1 Logistic Regression

[ ]:

```
from sklearn.linear_model import LogisticRegression
```

[ ]:

```
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()
```

[ ]:

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

[ ]:

```
# 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}%")
```

2.2 SVM

[ ]:

```
from sklearn import svm
```

[ ]:

```
# 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}%")
```

3 K-Means Clusteriungfrom scipy import stats

[ ]:

```
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
```

[ ]:

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

[ ]:

```
from sklearn.cluster import KMeans
```

[ ]:

```
# 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}%")
```

[ ]:

```

```

4 Random Forest

[ ]:

```
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"\nAccuracy = {100*accuracy:.3f}%")
```

5 ResNet 50

[ ]:

```
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
train_data, test_data = load_dataset("HAM10000", transform=resnet_preprocess)
```

[ ]:

```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)), 1)).item()
            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: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.
 ↪4f}")

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

[ ]:

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

5.1 Zero-Shot Resnet

```
[ ]: test_loader = DataLoader(test_data, batch_size=config['batch_size'],␣
      ↪shuffle=False, num_workers=2)
```

```
[ ]: print(evaluate(resnet50, test_loader))
```

5.2 Fine-Tuned Resnet

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

```
[ ]: train(resnet50, optim, loss, train_data, test_data, config)
```

```
[ ]: print(evaluate(resnet50, test_loader))
```

```
[ ]: import os
     print(os.getcwd())
```

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

6 Implement a zero-shot function for medclip

```
[ ]: # 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
```

```
# def medclip_zero_shot_inline(test_dataset, classes, batch_size=BATCH_SIZE):
#     # Device configuration
#     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#     # Data loader for the dataset
#     data_loader = DataLoader(test_dataset, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=4)
#     print(f"Device: {device}")

#     # Initialize MedClip Models
#     model = MedCLIPModel(vision_cls=MedCLIPVisionModelViT).to(device)

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

#     # Tokenize text prompts and convert to tensors
#     text_tokens = [tokenizer(text, return_tensors='pt', padding=True,␣
 ↪truncation=True, add_special_tokens=True) for text in 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

#     for images, labels in tqdm(data_loader):
#         images, labels = images.to(device), labels.to(device)

#         # TODO: Encode images using MedClip's vision model
#         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 += labels.size(0)

#     return correct / total

# # Load HAM10000 dataset
# transform = torchvision.transforms.Compose([
#     torchvision.transforms.Resize((IMG_SIZE, IMG_SIZE)),
#     torchvision.transforms.ToTensor(),
#     torchvision.transforms.Normalize(mean=[IMG_MEAN], std=[IMG_STD])
# ])

# # train_dataset, test_dataset = load_ham10000_dataset(transform=transform,␣
 ↪data_dir='data/ham10000')
# train_dataset, test_dataset = load_dataset("HAM10000", transform=transform,␣
 ↪data_dir='data/ham10000/')
# classes = list(LESION_TYPE.values())  # From the data_utils.py file

# # Run zero-shot classification
# acc = medclip_zero_shot_inline(test_dataset, classes)
# print(f"Accuracy: {acc:.2f}")
```

[ ]:

```
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 skin lesion." 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_special_tokens=True) for text in 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

    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
```

6.1 Load HAM10000 dataset and test MedClip's zero-shot capabilities

[ ]:

```
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

[ ]:

```
# 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 initializing BertModel: ['cls.predictions.transform.dense.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.weight', 'cls.seq_relationship.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
BertForSequenceClassification model 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 BertForSequenceClassification model).
```

```
Device: cuda
```

```
100%|          | 141/141 [00:23<00:00,  6.11it/s]
```

```

Accuracy = 22.346%
```

MedCLIP_ViT_model

[ ]:
```
# 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
BertForSequenceClassification model 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 BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not
used when initializing BertModel: ['cls.predictions.transform.dense.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.weight', 'cls.seq_relationship.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
BertForSequenceClassification model 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 BertForSequenceClassification model).
```

```
Device: cuda
```

```
100%|     | 141/141 [00:25<00:00,  5.63it/s]
```

```
Accuracy = 27.593%
```

6.2 Load NIH Chest X-ray dataset

[ ]:

```
import os
# os.chdir('../')
print(os.getcwd())
```

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

[ ]:

```
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/')
```

```
NIH Dataset:  Compose(
    Resize(size=(224, 224), interpolation=bilinear, max_size=None,
antialias=warn)
    ToTensor()
    Normalize(mean=[0.5862785803043838], std=[0.27950088968644304])
)
```

[ ]:

```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_special_tokens=True) for text in text_prompts]
    # 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

[ ]:

```
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 parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `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
used when initializing BertModel: ['cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.decoder.weight',
'cls.predictions.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 another task or with another architecture (e.g. initializing a
BertForSequenceClassification model 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 BertForSequenceClassification model).
```

```
Device: cuda
```

```
100%|     | 789/789 [04:57<00:00,  2.66it/s]
```

```

Accuracy = 53.138%
```

Load MedCLIP-ViT

[ ]:

```
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 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
BertForSequenceClassification model 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 BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not
used when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.bias',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.weight', 'cls.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
BertForSequenceClassification model 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 BertForSequenceClassification model).
```

```
Device: cuda
```

```
100%|          | 789/789 [3:36:49<00:00, 16.49s/it]


Accuracy = 16.531%
```

[ ]:

```
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()
```

```
[ ]: 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 parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `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
used when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.bias',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.weight', 'cls.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
BertForSequenceClassification model 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 BertForSequenceClassification model).
100%|          | 789/789 [20:23<00:00,  1.55s/it]
100%|          | 88/88 [02:14<00:00,  1.52s/it]
```

```
[ ]: 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%
```

```
[ ]: # 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)
```

```
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
BertForSequenceClassification model 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 BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not
used when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.bias',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.weight', 'cls.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
BertForSequenceClassification model 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 BertForSequenceClassification model).
100%|      | 789/789 [26:12<00:00,  1.99s/it]
100%|      | 88/88 [02:53<00:00,  1.97s/it]
```

```
[ ]: 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 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:  1.6min finished
```

```
 MedClip ViT NIH Image Features Accuracy = 55.342%
```

[ ]:

```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:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
c:\Users\mario\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `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
used when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.bias',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.weight', 'cls.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
BertForSequenceClassification model 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 BertForSequenceClassification model).
100%|     | 141/141 [01:31<00:00,  1.54it/s]
100%|     | 16/16 [00:06<00:00,  2.33it/s]
```

[ ]:

```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}%")
```

```
[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%
```

[ ]:

```
# 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}%")
```

```
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
BertForSequenceClassification model 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 BertForSequenceClassification model).
Some weights of the model checkpoint at emilyalsentzer/Bio_ClinicalBERT were not
used when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.bias',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.weight', 'cls.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
BertForSequenceClassification model 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 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%
```