```
from transformers import CLIPProcessor, CLIPModel

# CLIP 모델 다운로드
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

print("CLIP 모델 로드 완료!")
```

```
CLIP 모델 로드 완료!
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os

# Google Drive의 데이터 경로
data_dir = "/content/drive/My Drive/cifake-2000"
test_fake_dir_without_cat = os.path.join(data_dir, "test-fake-without-cat")  # FAKE 이미지 경로
test_real_dir_without_cat = os.path.join(data_dir, "test-real-without-cat")  # REAL 이미지 경로

# 파일 리스트 확인
test_fake_images_without_cat = [os.path.join(test_fake_dir_without_cat, img) for img in os.listdir(test_fake_dir_without_cat)]
test_real_images_without_cat = [os.path.join(test_real_dir_without_cat, img) for img in os.listdir(test_real_dir_without_cat)]

print(f"FAKE 이미지 개수: {len(test_fake_images_without_cat)}")
print(f"REAL 이미지 개수: {len(test_real_images_without_cat)}")
```

```
FAKE 이미지 개수: 100
REAL 이미지 개수: 100
```

```
prompts = [
    "This is a synthetic image generated by AI.",
    "This is a real image."
]
```

```
from PIL import Image

# 단일 이미지 예측
def predict_image(image_path, model, processor, prompts):
    image = Image.open(image_path)
    inputs = processor(text=prompts, images=image, return_tensors="pt", padding=True)
    outputs = model(**inputs)
    probs = outputs.logits_per_image.softmax(dim=1)
    return probs[0, 0].item(), probs[0, 1].item()  # FAKE 확률, REAL 확률

# 테스트
#fake_prob, real_prob = predict_image(fake_images[0], model, processor, prompts)
#print(f"FAKE 확률: {fake_prob:.2f}, REAL 확률: {real_prob:.2f}")
```

```
def evaluate_dataset(image_paths, model, processor, prompts, label):
    predictions = []
    for image_path in image_paths:
        fake_prob, real_prob = predict_image(image_path, model, processor, prompts)
        predictions.append((label, fake_prob, real_prob))
    return predictions


# 데이터셋 평가
test_fake_results_without_cat = evaluate_dataset(test_fake_images_without_cat, model, processor, prompts, "FAKE")
test_real_results_without_cat = evaluate_dataset(test_real_images_without_cat, model, processor, prompts, "REAL")

# 결과 결합
test_all_results_without_cat = test_fake_results_without_cat + test_real_results_without_cat
```

```
test_all_results_without_cat
```

```
 ('REAL', 0.9404003620147705, 0.05959967523813248),
 ('REAL', 0.9205093383789062, 0.07949068397283554),
 ('REAL', 0.9730965495109558, 0.026903413236141205),
 ('REAL', 0.9300916194915771, 0.06990831345319748),
 ('REAL', 0.9750741720199585, 0.024925880134105682),
 ('REAL', 0.9016796350479126, 0.0983203575015068),
 ('REAL', 0.8750462532043457, 0.12495376169681549),
 ('REAL', 0.7536402344703674, 0.24635978043079376),
 ('REAL', 0.9279583692550659, 0.07204166799783707),
 ('REAL', 0.9177968502044678, 0.08220313489437103),
 ('REAL', 0.9725841283798218, 0.02741587720811367),
 ('REAL', 0.9693737030029297, 0.030626270920038223),
 ('REAL', 0.9672332406044006, 0.03276669979095459),
 ('REAL', 0.9236007928848267, 0.07639920711517334),
 ('REAL', 0.9161434769630432, 0.08385653048753738),
 ('REAL', 0.9414606094360352, 0.05853937566280365),
 ('REAL', 0.9573239684104919, 0.042675960808992386),
 ('REAL', 0.9639925360679626, 0.036007530987262726),
 ('REAL', 0.9304677844047546, 0.06953224539756775),
 ('REAL', 0.9021555781364441, 0.09784436970949173),
 ('REAL', 0.9619117379188538, 0.03808830678462982),
 ('REAL', 0.9266484379768372, 0.07335158437490463),
 ('REAL', 0.9533271789550781, 0.046672821044921875),
 ('REAL', 0.9263799786567688, 0.07362000644207001),
 ('REAL', 0.8870115280151367, 0.11298853158950806),
 ('REAL', 0.9485714435577393, 0.05142851173877716),
 ('REAL', 0.9685027599334717, 0.031497228890657425),
 ('REAL', 0.9192119836807251, 0.08078800141811371),
 ('REAL', 0.9460725784301758, 0.05392736196517944),
 ('REAL', 0.8282051086425781, 0.1717948317527771),
 ('REAL', 0.9493219256401062, 0.050678104162216187),
 ('REAL', 0.9487306475639343, 0.05126935616135597),
 ('REAL', 0.9808908700942993, 0.019109157845377922),
 ('REAL', 0.9859156608581543, 0.014084350317716599),
 ('REAL', 0.9766210317611694, 0.023378973826766014),
 ('REAL', 0.7495397329330444, 0.2504602670669557),
 ('REAL', 0.9836630821228027, 0.01633690856397152),
 ('REAL', 0.913494884967804, 0.08650507777929306),
 ('REAL', 0.7567141652107239, 0.2432858645915985),
 ('REAL', 0.9025894999504089, 0.09741052240133286),
 ('REAL', 0.9876621961593628, 0.012337868101894855),
 ('REAL', 0.9529577493667603, 0.04704227298498154),
 ('REAL', 0.9332798719406128, 0.06672009825706482),
 ('REAL', 0.940459668636322, 0.05954026058316231),
 ('REAL', 0.9524455666542053, 0.04755442962050438),
 ('REAL', 0.8840427994728088, 0.11595720052719116),
 ('REAL', 0.8795379400253296, 0.12046205252408981),
 ('REAL', 0.9354336857795715, 0.06456634402275085),
 ('REAL', 0.9207526445388794, 0.07924733310937881),
 ('REAL', 0.9466441869735718, 0.05335581302642822),
 ('REAL', 0.8936890959739685, 0.1063108891248703),
 ('REAL', 0.9453384280204773, 0.05466158315553936)]
```

```
from sklearn.metrics import classification_report

# 실제 레이블과 예측 레이블 생성
test_true_labels_without_cat = [0] * len(test_fake_results_without_cat) + [1] * len(test_real_results_without_cat)  # 0: FAKE, 1: REAL
test_predicted_labels_without_cat = [
    0 if fake_prob > real_prob else 1
    for _, fake_prob, real_prob in test_all_results_without_cat
]

# 성능 평가
print(classification_report(test_true_labels_without_cat, test_predicted_labels_without_cat, target_names=["FAKE", "REAL"]))
```

```
              precision    recall  f1-score   support

        FAKE       0.50      1.00      0.67       100
        REAL       0.00      0.00      0.00       100

    accuracy                           0.50       200
   macro avg       0.25      0.50      0.33       200
weighted avg       0.25      0.50      0.33       200

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# Google Drive의 데이터 경로
train_data_dir = "/content/drive/My Drive/cifake-2000"
train_fake_dir = os.path.join(train_data_dir, "train-fake-without-cat")  # FAKE 이미지 경로
train_real_dir = os.path.join(train_data_dir, "train-real-without-cat")  # REAL 이미지 경로
```

```
# 파일 리스트 확인
train_fake_images = [os.path.join(train_fake_dir, img) for img in os.listdir(train_fake_dir)]
train_real_images = [os.path.join(train_real_dir, img) for img in os.listdir(train_real_dir)]

print(f"FAKE 이미지 개수: {len(train_fake_images)}")
print(f"REAL 이미지 개수: {len(train_real_images)}")
```

```
FAKE 이미지 개수: 450
REAL 이미지 개수: 450
```

```
from PIL import Image
from torchvision import transforms
import torch

# 데이터 전처리
preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.48145466, 0.4578275, 0.40821073), std=(0.26862954, 0.26130258, 0.27577711)),
])


# FAKE와 REAL 데이터를 전처리
def preprocess_images(image_paths, label):
    images, labels = [], []
    for img_path in image_paths:
        img = Image.open(img_path).convert("RGB")
        img_tensor = preprocess(img)
        images.append(img_tensor)
        labels.append(label)
    return images, labels

# FAKE와 REAL 데이터 전처리
processed_fake_images, fake_labels = preprocess_images(train_fake_images, label=0)  # FAKE = 0
processed_real_images, real_labels = preprocess_images(train_real_images, label=1)  # REAL = 1


# 데이터 병합
train_images = torch.stack(processed_fake_images + processed_real_images)
train_labels = torch.tensor(fake_labels + real_labels)

print(f"전체 학습 이미지 개수: {train_images.shape[0]}")
```

```
전체 학습 이미지 개수: 900
```

```
from torch.utils.data import DataLoader, TensorDataset

# TensorDataset 및 DataLoader 생성
train_dataset = TensorDataset(train_images, train_labels)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

print(f"배치당 데이터 개수: {len(next(iter(train_loader))[0])}")
```

```
배치당 데이터 개수: 32
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
model = model.to(device)


# Vision Encoder와 Text Encoder 학습 가능 상태로 설정
for param in model.vision_model.parameters():
    param.requires_grad = True

for param in model.text_model.parameters():
    param.requires_grad = True
```

```
import torch
import torch.nn.functional as F

# Contrastive Loss 정의
def contrastive_loss(image_features, text_features, temperature=0.07):
    """
    CLIP-style contrastive loss for image and text embeddings.
    """
    # 코사인 유사도 계산
    logits_per_image = image_features @ text_features.T / temperature
    logits_per_text = text_features @ image_features.T / temperature
    # 정답 레이블 생성
    labels = torch.arange(logits_per_image.size(0)).to(image_features.device)

    # CrossEntropyLoss를 사용해 Positive Pair 최적화
    loss_i2t = F.cross_entropy(logits_per_image, labels)  # 이미지 → 텍스트 매칭 손실
    loss_t2i = F.cross_entropy(logits_per_text, labels)  # 텍스트 → 이미지 매칭 손실

    # 평균 손실 반환
    return (loss_i2t + loss_t2i) / 2
```

```
from torch.optim import AdamW

# 옵티마이저 정의
optimizer = AdamW(model.parameters(), lr=1e-5)


# 학습 루프
epochs = 5  # 학습 반복 횟수
temperature = 0.07  # Contrastive Learning의 온도 파라미터

model.train()  # 학습 모드로 설정

for epoch in range(epochs):
    total_loss = 0
    for images, labels in train_loader:
        images = images.to(device)

        # 텍스트 프롬프트 처리 (labels에 따라 프롬프트 선택)
        text_inputs = [prompts[label] for label in labels.cpu().numpy()]

        # Processor 없이 입력 준비
        inputs = {
            "pixel_values": images,  # 정규화된 Tensor 이미지
            "input_ids": processor.tokenizer(text_inputs, padding=True, return_tensors="pt")["input_ids"].to(device)
        }

        # 모델 출력
        outputs = model(**inputs)
        image_features = outputs.image_embeds  # 이미지 임베딩
        text_features = outputs.text_embeds    # 텍스트 임베딩

        # Contrastive Loss 계산
        loss = contrastive_loss(image_features, text_features, temperature)

        # 모델 업데이트
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {total_loss/len(train_loader):.4f}")
```

```
Epoch 1/5, Loss: 3.1266
Epoch 2/5, Loss: 2.8736
Epoch 3/5, Loss: 2.8615
Epoch 4/5, Loss: 2.7774
Epoch 5/5, Loss: 2.7731
```

```
model.eval()  # 평가 모드로 설정
# 100개씩 데이터셋 평가

test_fake_results_without_cat_after_tuning = evaluate_dataset(test_fake_images_without_cat, model, processor, prompts, "FAKE")
test_real_results_without_cat_after_tuning = evaluate_dataset(test_real_images_without_cat, model, processor, prompts, "REAL")

# 결과 결합
test_results_without_cat_after_tuning = test_fake_results_without_cat_after_tuning + test_real_results_without_cat_after_tuning
```

```
test_results_without_cat_after_tuning
```

```
       ('REAL', 2.8222090441485939 4e-18, 1.0),
       ('REAL', 1.0467183537806933e-19, 1.0),
       ('REAL', 0.9999536275863647, 4.636010635294951 5e-05),
       ('REAL', 1.3030596869063159e-19, 1.0),
       ('REAL', 1.7759594750688698e-15, 1.0),
       ('REAL', 9.17844160490931e-16, 1.0),
       ('REAL', 5.556489637390497e-21, 1.0),
       ('REAL', 2.9098949914241956e-19, 1.0),
       ('REAL', 3.3190380128361064e-18, 1.0),
       ('REAL', 7.790428368055042e-20, 1.0),
       ('REAL', 5.867955675941588e-19, 1.0),
       ('REAL', 6.26424706919905e-14, 1.0),
       ('REAL', 5.293241765685723e-16, 1.0),
       ('REAL', 1.0910800758813063e-18, 1.0),
       ('REAL', 2.4998423815312062e-18, 1.0),
       ('REAL', 4.449923441510142e-20, 1.0),
       ('REAL', 8.566076662421782e-19, 1.0),
       ('REAL', 3.459265702781582e-19, 1.0),
       ('REAL', 1.6184958111190983e-18, 1.0),
       ('REAL', 2.115551639731933e-18, 1.0),
       ('REAL', 4.255024482511063e-16, 1.0),
       ('REAL', 1.5689265217733694e-19, 1.0),
       ('REAL', 3.528491104077649e-18, 1.0),
       ('REAL', 2.4892131106598998e-18, 1.0),
       ('REAL', 1.183814818725774e-17, 1.0),
       ('REAL', 1.951600047926919e-19, 1.0),
       ('REAL', 9.880470584876453e-18, 1.0),
       ('REAL', 4.865623005615902e-16, 1.0),
       ('REAL', 6.133025186246323e-19, 1.0),
       ('REAL', 8.60123985176747e-20, 1.0),
       ('REAL', 2.610328168030975e-16, 1.0),
       ('REAL', 3.078639515852526e-17, 1.0),
       ('REAL', 8.842440548509104e-20, 1.0),
       ('REAL', 1.7212309194151484e-18, 1.0),
       ('REAL', 2.24111153558095e-16, 1.0),
       ('REAL', 1.7686070169148753e-19, 1.0),
       ('REAL', 1.3708134046456808e-20, 1.0),
       ('REAL', 4.803193385224422e-18, 1.0),
       ('REAL', 1.5836004897391251e-16, 1.0),
       ('REAL', 9.970920923702748e-19, 1.0),
       ('REAL', 3.530443663913581e-18, 1.0),
       ('REAL', 7.484982879581887e-20, 1.0),
       ('REAL', 9.90143432720357e-21, 1.0),
       ('REAL', 0.9999995231628418, 5.342518534234841e-07),
       ('REAL', 1.0096399591930094e-06, 0.9999990463256836),
       ('REAL', 2.8044674619330737e-19, 1.0),
       ('REAL', 7.43070561423163e-19, 1.0),
       ('REAL', 1.2457971329292097e-16, 1.0),
```

```
# 실제 레이블과 예측 레이블 생성
test_true_labels_without_cat_after_tuning = [0] * len(test_fake_results_without_cat_after_tuning) + [1] * len(test_real_results_without_cat_after_tun
test_predicted_labels_without_cat_after_tuning = [
    0 if fake_prob > real_prob else 1
    for _, fake_prob, real_prob in test_results_without_cat_after_tuning
]

# 성능 평가
print(classification_report(test_true_labels_without_cat_after_tuning, test_predicted_labels_without_cat_after_tuning, target_names=["FAKE", "REAL"])
```

```
              precision    recall  f1-score   support

        FAKE       0.97      0.83      0.89       100
        REAL       0.85      0.97      0.91       100

    accuracy                           0.90       200
   macro avg       0.91      0.90      0.90       200
weighted avg       0.91      0.90      0.90       200
```

```
# Google Drive의 데이터 경로
data_dir = "/content/drive/My Drive/cifake-2000"
test_fake_dir_with_cat = os.path.join(data_dir, "test-fake-with-cat")  # FAKE 이미지 경로
test_real_dir_with_cat = os.path.join(data_dir, "test-real-with-cat")  # REAL 이미지 경로

# 파일 리스트 확인
test_fake_images_with_cat = [os.path.join(test_fake_dir_with_cat, img) for img in os.listdir(test_fake_dir_with_cat)]
test_real_images_with_cat = [os.path.join(test_real_dir_with_cat, img) for img in os.listdir(test_real_dir_with_cat)]

print(f"FAKE 이미지 개수: {len(test_fake_images_with_cat)}")
print(f"REAL 이미지 개수: {len(test_real_images_with_cat)}")
```

```
FAKE 이미지 개수: 100
REAL 이미지 개수: 100
```

```
model.eval()  # 평가 모드로 설정
# 100개씩 데이터셋 평가
```

```
test_fake_results_with_cat_after_tuning = evaluate_dataset(test_fake_images_with_cat, model, processor, prompts, "FAKE")
test_real_results_with_cat_after_tuning = evaluate_dataset(test_real_images_with_cat, model, processor, prompts, "REAL")

# 결과 결합
test_results_with_cat_after_tuning = test_fake_results_with_cat_after_tuning + test_real_results_with_cat_after_tuning
```

```
test_results_with_cat_after_tuning
```

```
[('FAKE', 0.9998124241828918, 0.00018764582637231797),
 ('FAKE', 1.0, 3.6905884570614944e-08),
 ('FAKE', 1.0, 1.2909829533606182e-18),
 ('FAKE', 1.0, 4.002322739893316e-08),
 ('FAKE', 4.2165358347834195e-16, 1.0),
 ('FAKE', 1.0, 5.743875469723628e-16),
 ('FAKE', 0.00697876513004303, 0.9930211901664734),
 ('FAKE', 0.0005342710646800697, 0.9994657635688782),
 ('FAKE', 4.129812472357906e-15, 1.0),
 ('FAKE', 5.706515988912475e-13, 1.0),
 ('FAKE', 5.067492111265892e-06, 0.9999948740005493),
 ('FAKE', 0.999862551689148, 0.00013745526666752994),
 ('FAKE', 1.7913743405317817e-11, 1.0),
 ('FAKE', 1.0, 3.599196434887151e-18),
 ('FAKE', 3.397805582901203e-14, 1.0),
 ('FAKE', 1.6928927877579714e-13, 1.0),
 ('FAKE', 1.646833502899625e-10, 1.0),
 ('FAKE', 1.0, 3.0918785001974065e-18),
 ('FAKE', 4.709127310320582e-09, 1.0),
 ('FAKE', 0.025593169033527374, 0.9744068384170532),
 ('FAKE', 1.0, 4.116292376399322e-12),
 ('FAKE', 5.849878914609974e-10, 1.0),
 ('FAKE', 7.120573372106538e-18, 1.0),
 ('FAKE', 6.883937864188283e-09, 1.0),
 ('FAKE', 1.0, 3.5852935204028213e-11),
 ('FAKE', 1.2885054134770257e-11, 1.0),
 ('FAKE', 0.9999991655349731, 7.87387534728623e-07),
 ('FAKE', 1.1732042420886657e-11, 1.0),
 ('FAKE', 0.0023200525902211666, 0.9976800084114075),
 ('FAKE', 1.1754897968785372e-05, 0.9999881982803345),
 ('FAKE', 1.0, 2.4455101572257165e-13),
 ('FAKE', 1.0, 3.6043798298468163e-16),
 ('FAKE', 1.8617258776276735e-11, 1.0),
 ('FAKE', 8.101653977679243e-14, 1.0),
 ('FAKE', 1.0, 1.4057250795931594e-15),
 ('FAKE', 2.7139108424749023e-14, 1.0),
 ('FAKE', 1.0, 1.1767152433335171e-16),
 ('FAKE', 1.6735336173438094e-17, 1.0),
 ('FAKE', 1.0, 3.7054569190786424e-08),
 ('FAKE', 6.0310111939243516e-12, 1.0),
 ('FAKE', 7.800194062366843e-17, 1.0),
 ('FAKE', 2.6992432822225965e-07, 0.9999997615814209),
 ('FAKE', 1.0, 1.699545138886152e-16),
 ('FAKE', 1.0, 1.0343950470001755e-16),
 ('FAKE', 1.0595115930434051e-11, 1.0),
 ('FAKE', 1.0, 2.8863263927246636e-18),
 ('FAKE', 3.9605402435083045e-16, 1.0),
 ('FAKE', 1.3449170111349692e-17, 1.0),
 ('FAKE', 2.0356789864894864e-10, 1.0),
 ('FAKE', 1.0, 7.069232623325289e-14),
 ('FAKE', 2.4132388531938886e-08, 1.0),
 ('FAKE', 2.1567121422094715e-08, 1.0),
 ('FAKE', 5.212704712320715e-10, 1.0),
 ('FAKE', 2.819791973251995e-07, 0.9999997615814209),
 ('FAKE', 0.0004522087110672146, 0.9995477795600891),
 ('FAKE', 1.0, 4.3440330443900166e-08),
 ('FAKE', 4.2691368455057075e-16, 1.0),
 ('FAKE', 6.19098941001539e-08, 0.9999998807907104),
```

```
# 실제 레이블과 예측 레이블 생성
test_true_labels_with_cat_after_tuning = [0] * len(test_fake_results_with_cat_after_tuning) + [1] * len(test_real_results_with_cat_after_tuning)  # 0:
test_predicted_labels_with_cat_after_tuning = [
    0 if fake_prob > real_prob else 1
    for _, fake_prob, real_prob in test_results_with_cat_after_tuning
]

# 성능 평가
print(classification_report(test_true_labels_with_cat_after_tuning, test_predicted_labels_with_cat_after_tuning, target_names=["FAKE", "REAL"]))
```

```
              precision    recall  f1-score   support

        FAKE       1.00      0.40      0.57       100
        REAL       0.62      1.00      0.77       100

    accuracy                           0.70       200
   macro avg       0.81      0.70      0.67       200
weighted avg       0.81      0.70      0.67       200
```