Appendix S1. Overview of the code for fine-tuning and evaluation of the deep learning algorithm. Other codes are available at https://github.com/facebookresearch/dino

```
import csv, os, sys, argparse
from pathlib import Path

import torch
from torch import nn
import torch.backends.cudnn as cudnn
from torchvision import datasets, transforms as T

import vision_transformer as vits          # local file
import utils                               # local file
# ---------------------------------------------------------------------------
# ---------------------------- USER PARAMETERS ------------------------------
BACKBONE_CKPT = "./linear_eval/checkpoint0500.pth" # "" if not needed
LINEAR_CKPT = "./output_dermo_addition/checkpoint.pth.tar"
TEST_DIR = "./test_dermo_final" #
CSV_PATH = "./predictions_dermo_final.csv"

ARCH           = "vit_small"          # vit_small / vit_base / ...
PATCH          = 16
NUM_CLASSES    = 2
BATCH_SIZE     = 128
NUM_WORKERS    = 8
DEVICE         = "cuda" if torch.cuda.is_available() else "cpu"
# ---------------------------------------------------------------------------
# 1. Backbone (rebuild exactly as during training)
if ARCH in vits.__dict__:
    backbone = vits.__dict__[ARCH](patch_size=PATCH, num_classes=0)
    embed_dim = backbone.embed_dim * 4          # n_last_blocks = 4 during training
else:
    raise ValueError(f"Unknown architecture: {ARCH}")
backbone.to(DEVICE).eval()

if BACKBONE_CKPT:          # optional: load SSL / DINO weights
    utils.load_pretrained_weights(
        backbone, BACKBONE_CKPT, checkpoint_key="teacher",
        model_name=ARCH, patch_size=PATCH
    )

# 2. Linear head
class LinearClassifier(nn.Module):
    def __init__(self, dim, num_labels):
        super().__init__()
        self.linear = nn.Linear(dim, num_labels)
```

```
    def forward(self, x):
        x = x.view(x.size(0), -1)
        return self.linear(x)

head = LinearClassifier(embed_dim, NUM_CLASSES).to(DEVICE)

ckpt  = torch.load(LINEAR_CKPT, map_location=DEVICE)
state = ckpt.get("state_dict", ckpt)
state = {k.replace("module.", ""): v for k, v in state.items()}
head.load_state_dict(state, strict=False)
head.eval()

# 3. Dataset that also returns the file path
class ImageFolderWithPaths(datasets.ImageFolder):
    def __getitem__(self, index):
        path, target = self.samples[index]
        img = self.loader(path)
        if self.transform is not None:
            img = self.transform(img)
        return img, target, path      # <-- path is new

val_tf = T.Compose([
    T.Resize(256, interpolation=3),
    T.CenterCrop(224),
    T.ToTensor(),
    T.Normalize((0.485,0.456,0.406), (0.229,0.224,0.225)),
])

dataset = ImageFolderWithPaths(TEST_DIR, transform=val_tf)
loader  = torch.utils.data.DataLoader(
    dataset, batch_size=BATCH_SIZE, shuffle=False,
    num_workers=NUM_WORKERS, pin_memory=True
)

# 4. Inference & CSV dump
softmax = nn.Softmax(dim=1)
Path(CSV_PATH).parent.mkdir(parents=True, exist_ok=True)

header = ["filename",
          "gt",
          "pred1", "prob1",
          "pred2", "prob2",
```

```
        ]

with open(CSV_PATH, "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(header)

    with torch.no_grad():
        for images, targets, paths in loader:
            images  = images.to(DEVICE, non_blocking=True)
            targets = targets.to(DEVICE, non_blocking=True)

            # backbone CLS-token features (4 last blocks concatenated)
            feats = backbone.get_intermediate_layers(images, n=4)
            feats = torch.cat([x[:, 0] for x in feats], dim=-1)

            logits = head(feats)                # (B, NUM_CLASSES)
            probs  = softmax(logits)            # (B, NUM_CLASSES)

            top2_probs, top2_idxs = probs.topk(2, dim=1)

            # Move to CPU once per batch
            targets    = targets.cpu()
            top2_probs = top2_probs.cpu()
            top2_idxs  = top2_idxs.cpu()

            for i, (gt, pth) in enumerate(zip(targets, paths)):
                gt_name              = dataset.classes[gt.item()]
                pred1_idx, pred2_idx = top2_idxs[i].tolist()
                prob1, prob2         = top2_probs[i].tolist()

                row = [
                    Path(pth).name,             # filename only
                    gt_name,
                    dataset.classes[pred1_idx], f"{prob1:.6f}",
                    dataset.classes[pred2_idx], f"{prob2:.6f}",
                        ]
                writer.writerow(row)

print(f"Saved predictions to {CSV_PATH}")
```