BME646 and ECE60146: Homework 7

Spring 2026
Due Date: Tuesday, March 24, 2026, 11:59 pm ET
TA: Somosmita Mitra (mitra26@purdue.edu)

Turn in typed solutions via Gradescope. Post questions to Piazza. Additional instructions can be found at the end. **Late submissions will be accepted with penalty: -10 points per late day, up to 5 days.**

1 Introduction

In the previous homeworks, you trained networks to make explicit predictions: a class label, a bounding box, or a per-pixel segmentation mask. All of these relied on matching network outputs directly to ground-truth labels.

In this homework, you will explore **metric learning**, a different training paradigm in which the network learns to produce *embedding vectors* such that similar inputs end up close together in the embedding space, and dissimilar inputs are pushed far apart. This idea powers many modern applications: face verification, image retrieval, and large-scale multi-modal systems like CLIP.

The homework is organized into three parts:

1. **Supervised Metric Learning:** You will run DLStudio's `MetricLearning` module with both the *Pairwise Contrastive Loss* and the *Triplet Loss* on CIFAR-10, study the results, and implement the two core loss functions from scratch.

2. **Noise Contrastive Estimation (NCE):** You will run the NCE distribution-estimation script from the lecture and answer questions about how NCE connects to contrastive metric learning.

3. **CLIP, Multi-Modal Metric Learning:** You will experiment with a pretrained CLIP model on your own LVIS dataset, performing zero-shot classification and image retrieval.

As discussed in the lecture on metric learning [1], these topics introduce several new concepts:

- **Embedding spaces:** Representing inputs as vectors such that geometric proximity implies semantic similarity.

- **Pairwise Contrastive Loss:** Pulling together embeddings for same-class pairs and pushing apart different-class pairs using a margin.
- **Triplet Loss:** Ensuring that the anchor is closer to a positive (same class) than to a negative (different class) by at least a margin δ.
- **Batch mining:** Efficiently identifying positive/negative pairs and triplets within a batch using tensor operations rather than for-loops.
- **Noise Contrastive Estimation (NCE):** Training a discriminator to distinguish real data from injected noise as a proxy for learning the data distribution.
- **CLIP:** Jointly embedding images and text captions into a shared metric space via a symmetric InfoNCE loss, enabling zero-shot transfer.

1.1 Quick Reference: Lecture Slide Numbers

HW7 Topic	**Lecture**	**Slides**
Embedding vectors and L2 metric	MetricLearning	15–18
Pairwise Contrastive Loss	MetricLearning	19–25
Triplet Loss and mining strategies	MetricLearning	26–34
GPU-efficient tensor-based mining	MetricLearning	35–60
DLStudio MetricLearning results	MetricLearning	101–105
NCE for distribution estimation	MetricLearning	111–126
InfoNCE Loss	MetricLearning	127–136
CLIP multi-modal metric learning	MetricLearning	143–149
Experiments with CLIP	MetricLearning	150–175

2 Setting Up Your Environment

2.1 Prerequisites

Ensure you have:

- DLStudio version 2.5.6 installed (with the `ExamplesMetricLearning` directory)
- LVIS annotations and COCO 2017 images
- The CIFAR-10 dataset (downloaded automatically by `torchvision`)

2.2 Installing Required Packages

```
pip install faiss-cpu        # or faiss-gpu if available
pip install scikit-learn     # for t-SNE visualization
pip install openai-clip      # for Task 3 CLIP experiments
```

Verify:

```
import faiss
import clip
from sklearn.manifold import TSNE
print("All imports successful")
```

2.3 Key DLStudio Components

1. `ExamplesMetricLearning/example_for_pairwise_contrastive_loss.py`
2. `ExamplesMetricLearning/example_for_triplet_loss.py`
3. `NCE_for_learning_point_distro.py` (from lecture slides, Slides 121–126)

Read the docstrings for the `MetricLearning` co-class in DLStudio before starting. The two example scripts in `ExamplesMetricLearning` are your primary entry points.

3 Programming Tasks (100 Points Total)

3.1 Task 1: Pairwise Contrastive Loss (25 points)

Objective: Understand the Pairwise Contrastive Loss by running DLStudio's example and implementing the loss function from scratch.

3.1.1 Running the DLStudio Example (10 points)

Run the example script:

`example_for_pairwise_contrastive_loss.py`

This trains an embedding network on CIFAR-10 for 16 epochs using the Pairwise Contrastive Loss. Refer to Slides 19–25 and 101–103 of the metric learning lecture [1].

Deliverables:

- The training loss curve over all iterations

- A t-SNE visualization of the test-set embeddings, colored by class label (this is produced automatically by the script)
- The Precision@1 accuracy printed by the script

3.1.2 Understanding the Loss (5 points)

Report (1–2 paragraphs each):

- The Pairwise Contrastive Loss has two components: one for positive pairs and one for negative pairs. Write out the full loss formula (Eq. 6 in the lecture, Slide 25). Explain the role of the margin m: what happens to a negative pair whose distance already exceeds m?
- Why is it necessary to mine both positive and negative pairs from a batch, rather than simply computing a loss over all pairs? What is the danger of using too many easy negatives?

3.1.3 Implementing Pairwise Contrastive Loss (10 points)

Implement the following two functions from scratch. **You must use tensor operations, no Python for-loops.** Study Slides 40–51 of the lecture carefully before writing any code.

```
import torch

def compute_distance_matrix(embeddings):
    """
    Compute the pairwise squared Euclidean distance
    matrix for a batch of L2-normalized embedding vectors.

    Use the identity:
        ||x - y||^2 = ||x||^2 - 2*(x . y^T) + ||y||^2

    Args:
        embeddings: FloatTensor of shape (B, D)
    Returns:
        dist_matrix: FloatTensor of shape (B, B),
            where dist_matrix[i, j] = ||e_i - e_j||^2
    """
    # Step 1: dot products of all pairs
    dot = torch.mm(embeddings, embeddings.T)   # (B, B)
    # Step 2: squared norms (the diagonal of dot)
    sq_norms = dot.diagonal()                  # (B,)
    # Step 3: apply the identity above
    dist = (sq_norms.unsqueeze(1)
```

```
            - 2.0 * dot
            + sq_norms.unsqueeze(0))
    return dist.clamp(min=0)   # numerical safety


def pairwise_contrastive_loss(embeddings,
                              labels,
                              margin=1.0):
    """
    Compute Pairwise Contrastive Loss over all
    valid pairs in the batch.

    Positive pair (same label, i != j):
        loss_ij = dist[i, j]
    Negative pair (different labels):
        loss_ij = max(0, margin - sqrt(dist[i,j]))^2

    Args:
        embeddings: FloatTensor (B, D), L2-normalized
        labels:     LongTensor (B,)
        margin:     float
    Returns:
        loss: scalar tensor (mean over all pairs)
    """
    B = embeddings.shape[0]
    dist_sq = compute_distance_matrix(embeddings)

    # Build positive / negative masks
    # using broadcasting -- no for-loops
    labels_equal = (labels.unsqueeze(0)
                    == labels.unsqueeze(1))    # (B, B)
    not_diagonal = ~torch.eye(
        B, dtype=torch.bool,
        device=embeddings.device)

    pos_mask = labels_equal & not_diagonal     # (B, B)
    neg_mask = (~labels_equal) & not_diagonal  # (B, B)

    # Positive loss: squared distance
    pos_loss = dist_sq * pos_mask.float()

    # Negative loss: hinge on Euclidean distance
    dist_euc = dist_sq.clamp(min=0).sqrt()
    neg_loss = (torch.clamp(margin - dist_euc, min=0)
                ** 2) * neg_mask.float()

    # Average over all pairs (positive + negative)
    num_pairs = pos_mask.sum() + neg_mask.sum()
    loss = (pos_loss.sum() + neg_loss.sum()) \
```

```
            / (num_pairs.float() + 1e-8)
    return loss
```

Verification, include this output in your report:

```
import torch.nn.functional as F
torch.manual_seed(42)
B, D = 8, 16
emb = F.normalize(torch.randn(B, D), dim=1)
labels = torch.tensor([0, 0, 1, 1, 2, 2, 3, 3])

# Distance matrix diagonal must be ~0
dm = compute_distance_matrix(emb)
assert dm.shape == (B, B)
assert dm.diagonal().abs().max().item() < 1e-5, \
    "Diagonal should be zero"

# Loss must be non-negative
loss = pairwise_contrastive_loss(emb, labels)
assert loss.item() >= 0
print(f"Distance matrix diagonal max: "
      f"{dm.diagonal().abs().max().item():.2e}")
print(f"Contrastive loss (margin=1.0): "
      f"{loss.item():.4f}")
```

Report:

- Your complete implementation with comments
- The printed output of the verification block
- In 2–3 sentences, explain why `labels.unsqueeze(0) == labels.unsqueeze(1)` produces the correct positive-pair mask without any for-loops

Grading:

- DLStudio example: loss curve, t-SNE, Precision@1 (10 points)
- Written explanation of the loss and margin (5 points)
- Correct implementation with verification output (10 points)

3.2 Task 2: Triplet Loss (25 points)

Objective: Understand the Triplet Loss and its mining strategies by running DLStudio's example and implementing the loss.

3.2.1 Running the DLStudio Example (10 points)

Run the example script:

`example_for_triplet_loss.py`

Deliverables:

- The training loss curve
- The t-SNE visualization of test-set embeddings produced by the script
- The Precision@1 accuracy

3.2.2 Understanding Triplet Loss (5 points)

Report (1–2 paragraphs each):

- Write the Triplet Loss formula (Eq. 10 on Slide 34). Define what makes a triplet (a, p, n) *valid*. Explain why the `max(0, ...)` operation means that easy negatives (those already well separated from the anchor) contribute zero to the loss.
- Describe the three negative mining strategies, hard, semi-hard, and easy, using the margin δ. Explain intuitively why using *only* hard negatives often leads to training instability or embedding collapse, while semi-hard mining is generally preferred.

3.2.3 Implementing Triplet Loss (10 points)

Implement the all-triplets version of the Triplet Loss using tensor masking. **No for-loops.** Use `compute_distance_matrix` from Task 1.

```
def get_triplet_mask(labels):
    """
    Return a Boolean mask of shape (B, B, B) where
    mask[i, j, k] is True iff ALL of the following hold:
      1. i, j, k are all distinct
      2. labels[i] == labels[j]  (valid anchor-positive)
      3. labels[i] != labels[k]  (valid anchor-negative)

    No for-loops. Use tensor broadcasting.
    See Slides 52-60 of the lecture.

    Args:
        labels: LongTensor (B,)
    Returns:
```

```
        mask: BoolTensor (B, B, B)
    """
    B = labels.shape[0]
    device = labels.device

    # All-distinct indices mask
    not_eq = ~torch.eye(B, dtype=torch.bool,
                        device=device)
    i_ne_j = not_eq.unsqueeze(2)   # (B, B, 1)
    i_ne_k = not_eq.unsqueeze(1)   # (B, 1, B)
    j_ne_k = not_eq.unsqueeze(0)   # (1, B, B)
    distinct = i_ne_j & i_ne_k & j_ne_k  # (B, B, B)

    # Label equality mask
    eq = (labels.unsqueeze(0)
          == labels.unsqueeze(1))   # (B, B)
    # i==j (anchor-positive) shaped for axis 2
    ij_eq = eq.unsqueeze(2)         # (B, B, 1)
    # i==k shaped for axis 1
    ik_eq = eq.unsqueeze(1)         # (B, 1, B)

    valid_labels = ij_eq & ~ik_eq   # (B, B, B)
    return distinct & valid_labels


def triplet_loss(embeddings, labels, margin=0.2):
    """
    Compute the Triplet Loss over all valid triplets
    in the batch (batch-all strategy).

    loss = mean over valid triplets of:
        max(0, dist(a,p) - dist(a,n) + margin)

    Args:
        embeddings: FloatTensor (B, D), L2-normalized
        labels:     LongTensor (B,)
        margin:     float (delta in the lecture)
    Returns:
        loss:         scalar tensor
        num_triplets: int, number of non-zero triplets
    """
    dist = compute_distance_matrix(embeddings) # (B, B)
    mask = get_triplet_mask(labels)            # (B, B, B)

    # dist_ap[i,j,k] = dist[i,j] (anchor-positive)
    dist_ap = dist.unsqueeze(2).expand(
        dist.shape[0], dist.shape[1], dist.shape[0])
    # dist_an[i,j,k] = dist[i,k] (anchor-negative)
    dist_an = dist.unsqueeze(1).expand(
```

```
        dist.shape[0], dist.shape[0], dist.shape[0])

    triplet_loss_vals = torch.clamp(
        dist_ap - dist_an + margin, min=0.0)

    # Apply mask and average over non-zero losses
    triplet_loss_vals = triplet_loss_vals * mask.float()
    num_pos = (triplet_loss_vals > 1e-16).sum().item()
    loss = triplet_loss_vals.sum() \
           / (num_pos + 1e-8)
    return loss, int(num_pos)
```

Verification, include this output in your report:

```
torch.manual_seed(7)
B, D = 12, 32
emb = F.normalize(torch.randn(B, D), dim=1)
labels = torch.tensor([0,0,0,1,1,1,2,2,2,3,3,3])

mask = get_triplet_mask(labels)
assert mask.shape == (B, B, B)
# Self-pairs must never appear in valid triplets:
# mask[i,i,:] and mask[:,i,i] should always be False
assert mask.diagonal(dim1=0, dim2=1).any() == False

loss, n = triplet_loss(emb, labels)
assert loss.item() >= 0
print(f"Triplet loss: {loss.item():.4f}, "
      f"non-zero triplets: {n}")
```

Report:

- Your complete implementation with comments
- The printed output of the verification block
- Compare the Precision@1 you obtained from running the DLStudio triplet example to the Pairwise Contrastive result from Task 1. Which loss performs better on CIFAR-10, and is this consistent with the results shown on Slides 103–104 of the lecture?

Grading:

- DLStudio example: loss curve, t-SNE, Precision@1 (10 points)
- Written explanations of triplet loss and mining strategies (5 points)
- Correct implementation with verification output (10 points)

3.3 Task 3: Noise Contrastive Estimation (20 points)

Objective: Understand how NCE trains a discriminator to learn an implicit probability density model by distinguishing real data from injected noise.

3.3.1 Running the NCE Script (10 points)

The script `NCE_for_learning_point_distro.py` (reproduced in full on Slides 121–126 of the lecture [1]) trains a small discriminator network to separate samples from a 2D multi-Gaussian distribution from wide-variance background noise. Run the script for three values of the number of negative (noise) samples per positive (real) sample:

```
python3 NCE_for_learning_point_distro.py 1
python3 NCE_for_learning_point_distro.py 5
python3 NCE_for_learning_point_distro.py 20
```

Each run produces two plots: (a) the learned density contours over the real data, and (b) the same contours shown alongside all data points used including the NCE-mandated noise.

Deliverables:

- For each of the three values of N, include both output plots (6 plots total)
- The discriminator accuracy printed at the end of each run, collected into a simple table:

Negatives per positive (N)	**Discriminator accuracy**
1	?
5	?
20	?

3.3.2 Conceptual Questions (10 points)

Answer each of the following clearly in 3–5 sentences:

1. **Effect of N:** As N increases (more noise samples per real sample), how does the quality of the estimated density improve, as visible in the contour plots? Why does the NCE estimate become more accurate with larger N, and what is the cost of increasing N?

2. **NCE as a binary classifier:** The discriminator is trained with `nn.BCELoss` to output 1 for real samples and 0 for noise. Explain in intuitive terms why a well-trained discriminator's output is proportional to the true data density $p_{\text{data}}(x)$ divided by the noise density $p_{\text{noise}}(x)$.

3. **Connection to metric learning:** In Pairwise Contrastive and Triplet Loss, we distinguish between positive (same-class) and negative (different-class) samples. In NCE, we distinguish between real data and noise. Identify the conceptual parallel: what plays the role of the "positive" and "negative" in each framework? In what sense are both approaches doing "contrastive" learning?

4. **From NCE to InfoNCE:** The InfoNCE loss (Slide 138, Eq. 40) replaces the binary BCE loss of NCE with a multi-class softmax over one positive and N negatives. Write the InfoNCE formula and explain in one sentence what the temperature parameter τ controls.

Grading:

- Correct execution with all plots and accuracy table (10 points)
- Conceptual questions (10 points, 2.5 points each)

3.4 Task 4: CLIP Experiments on Your LVIS Dataset (30 points)

Objective: Understand CLIP's multi-modal metric learning framework and evaluate the pretrained CLIP model on your own LVIS data from HW6, both for zero-shot classification and image retrieval.

3.4.1 Understanding CLIP (10 points)

Study Slides 143–149 of the lecture [1] and **report**:

1. **Architecture (2 points):** CLIP uses two separate encoders. Describe the role of each and how their outputs are combined into a single similarity score. Why must both embeddings be L2-normalized before computing cosine similarity?

2. **Symmetric loss (3 points):** The CLIP loss applies the InfoNCE loss in *both* directions. Using the $N \times N$ similarity matrix shown on Slide

146, explain what is meant by the "image-to-text" and "text-to-image" directions. Write the combined loss:

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2}(\mathcal{L}_{\text{image}\to\text{text}} + \mathcal{L}_{\text{text}\to\text{image}}) \tag{1}$$

and explain why symmetry is important.

3. **Zero-shot transfer (2 points):** Explain how a trained CLIP model can classify an image into a category it has never seen during fine-tuning. What text prompts are typically used?

4. **Batch size (3 points):** In CLIP, the off-diagonal entries of the $B \times B$ similarity matrix serve as negatives. Why does a larger batch size improve the quality of CLIP training? What is the conceptual connection to the NCE result in Task 3, where using more noise samples (N) improved the density estimate?

3.4.2 Zero-Shot Classification with Pretrained CLIP (10 points)

Using the OpenAI CLIP model (`ViT-B/32`), classify images in your LVIS **validation set** from HW6 (the same 3 categories you used there) in a zero-shot fashion. Use the text prompt template `"a photo of a {class_name}"` for each of your 3 categories, where `class_name` is a simplified version of the LVIS category name (e.g., `'car'` instead of `'car_(automobile)'`).

```
import clip
import torch
import torch.nn.functional as F
from PIL import Image

device = "cuda" if torch.cuda.is_available() \
         else "cpu"
model, preprocess = clip.load(
    "ViT-B/32", device=device)
model.eval()

# Example for class_names = ['car','chair','bottle']
prompts = [f"a photo of a {c}"
           for c in class_names]
text_tokens = clip.tokenize(prompts).to(device)

with torch.no_grad():
    text_emb = model.encode_text(text_tokens)
    text_emb = F.normalize(text_emb, dim=1)

def classify_image(img_path):
```

```
    img = preprocess(
        Image.open(img_path).convert('RGB')
    ).unsqueeze(0).to(device)
    with torch.no_grad():
        img_emb = model.encode_image(img)
        img_emb = F.normalize(img_emb, dim=1)
    # Cosine similarity to each text prompt
    sims = (img_emb @ text_emb.T).squeeze(0)
    return sims.argmax().item()
```

Deliverables:

- Your complete zero-shot classification code
- A table reporting per-class accuracy and overall accuracy:

Category	**Zero-Shot Accuracy**
Category 1 (your name)	?
Category 2 (your name)	?
Category 3 (your name)	?
Overall	?

- A 3×3 confusion matrix
- Discuss in 2–3 sentences: How well does pretrained CLIP generalize to your LVIS categories without any fine-tuning? Which category is hardest and why?

3.4.3 Image-to-Image Retrieval with CLIP (10 points)

Using the CLIP image encoder, extract embeddings for all images in your LVIS **training set**. Build a FAISS index over these embeddings. Then, for each image in the validation set, retrieve its 5 nearest neighbors from the training set.

```
import faiss
import numpy as np

# Extract embeddings for the training set
# train_embeddings: np.ndarray, shape (N_train, 512)
# (ViT-B/32 produces 512-dimensional embeddings)

# Build FAISS index using inner product
# (equivalent to cosine similarity for L2-normalized
# vectors)
embed_dim = 512
```

```
index = faiss.IndexFlatIP(embed_dim)
index.add(train_embeddings.astype('float32'))

# Query with validation embeddings
# val_embeddings: np.ndarray, shape (N_val, 512)
D, I = index.search(
    val_embeddings.astype('float32'), k=5)
# D[i]: similarity scores for query i
# I[i]: indices of top-5 training-set neighbors
```

Deliverables:

- Your complete retrieval code (embedding extraction + FAISS index + query)
- A table of per-category Precision@1 and Precision@5:

Category	**Precision@1**	**Precision@5**
Category 1	?	?
Category 2	?	?
Category 3	?	?
Overall	?	?

- A visualization grid of at least **9 example retrievals** (3 per category): for each query image, show the query and its top-3 retrieved images. Draw a **green border** around each retrieved image if it belongs to the same category as the query, and a **red border** if it does not.
- Discuss in 2–3 sentences: How does the CLIP-based Precision@1 compare to what you would expect from a random baseline ($1/3 \approx 33\%$ for 3 classes)? Which category retrieves most accurately?

Grading:

- CLIP conceptual questions (10 points)
- Zero-shot classification with table and confusion matrix (10 points)
- Image retrieval with precision table and visualization grid (10 points)

4 Bonus Task (20 Points)

4.1 Margin Ablation Study for Triplet Loss

Re-train the DLStudio `EmbeddingGenerator1` backbone using your `triplet_loss` implementation from Task 2 for three values of the margin: $\delta \in \{0.1, 0.2, 0.5\}$. Use 8 training epochs for each.

Deliverables:

1. The three training loss curves on a single figure with a legend
2. Per-class and overall Precision@1 for each margin value (use FAISS for nearest-neighbor search, following the DLStudio example)
3. A brief discussion (1 paragraph): How does the margin δ control the geometry of the learned embedding space? Does a larger margin always lead to better Precision@1 on CIFAR-10?

5 Submission Instructions

5.1 What to Submit in the Report

Read this section carefully. Every item listed below must be present in your submission.

1. **Task 1: Pairwise Contrastive Loss (25 points)**
   (a) **Code:** Script used to run the DLStudio example
   (b) **Figure:** Training loss curve from the DLStudio example
   (c) **Figure:** t-SNE plot of test-set embeddings (from the script)
   (d) **Text:** Precision@1 accuracy
   (e) **Text:** Written explanation of the loss formula and the role of margin m; discussion of easy negatives
   (f) **Code:** `compute_distance_matrix` and `pairwise_contrastive_loss` implementations
   (g) **Output:** Printed output of the verification block
   (h) **Text:** Brief explanation of the broadcasting trick
2. **Task 2: Triplet Loss (25 points)**
   (a) **Code:** Script used to run the DLStudio triplet example
   (b) **Figure:** Training loss curve from the DLStudio example
   (c) **Figure:** t-SNE plot of test-set embeddings (from the script)
   (d) **Text:** Precision@1 accuracy
   (e) **Text:** Written explanation of the Triplet Loss formula, valid triplet definition, and the three mining strategies

(f) **Code:** `get_triplet_mask` and `triplet_loss` implementations

(g) **Output:** Printed output of the verification block

(h) **Text:** Comparison of Precision@1 between Task 1 and Task 2

3. **Task 3: NCE (20 points)**

   (a) **Figures:** Both output plots for each of $N \in \{1, 5, 20\}$ (6 figures total)

   (b) **Table:** Discriminator accuracy for each N

   (c) **Text:** Answers to all four conceptual questions

4. **Task 4: CLIP on LVIS (30 points)**

   (a) **Text:** Answers to the four CLIP conceptual questions

   (b) **Code:** Zero-shot classification code

   (c) **Table:** Per-class and overall zero-shot accuracy

   (d) **Figure:** 3×3 confusion matrix

   (e) **Text:** Discussion of zero-shot performance

   (f) **Code:** Retrieval code (embedding extraction + FAISS + query)

   (g) **Table:** Per-category Precision@1 and Precision@5

   (h) **Figure:** Retrieval visualization grid (≥ 9 examples)

   (i) **Text:** Discussion of retrieval results

5. **Bonus: Margin Ablation (20 points, optional)**

   (a) **Figure:** Three triplet loss curves for $\delta \in \{0.1, 0.2, 0.5\}$

   (b) **Table:** Precision@1 per margin

   (c) **Text:** Discussion of margin effect

5.2 Master Summary Table

Include the following summary table in your report:

Network	**Dataset**	**Loss**	**Key Metric**
EmbeddingGenerator1	CIFAR-10	Pairwise Contrastive	P@1: ?
EmbeddingGenerator1	CIFAR-10	Triplet	P@1: ?
NCE Discriminator	2D Gaussians	BCE	Accuracy: ?
CLIP ViT-B/32 (pretrained)	LVIS (3 cat)	,	Zero-shot acc: ?
CLIP ViT-B/32 (pretrained)	LVIS (3 cat)	,	P@1 retrieval: ?

Write a short paragraph (5–8 sentences) summarizing your key takeaways:

- How did Pairwise Contrastive Loss and Triplet Loss compare on CIFAR-10?
- What did the NCE experiments reveal about the role of negative samples?
- How well did pretrained CLIP transfer to your LVIS categories without any fine-tuning?

5.3 Code Files to Submit

1. `losses.py`: `compute_distance_matrix`, `pairwise_contrastive_loss`, `get_triplet_mask`, `triplet_loss`
2. `clip_experiments.py`: zero-shot classification + FAISS retrieval
3. `requirements.txt`: package dependencies
4. `README.md`: brief instructions to run your code

5.4 Autograder Interface Requirements

The autograder will import `losses.py` and call the functions below. Default argument values must match exactly.

```
from losses import (compute_distance_matrix,
                    pairwise_contrastive_loss,
                    get_triplet_mask,
                    triplet_loss)
import torch, torch.nn.functional as F

#, distance matrix,
emb = F.normalize(torch.randn(6, 32), dim=1)
dm = compute_distance_matrix(emb)
assert dm.shape == (6, 6)
assert dm.diagonal().abs().max() < 1e-4

#, pairwise contrastive loss,
labels = torch.tensor([0, 0, 1, 1, 2, 2])
loss_c = pairwise_contrastive_loss(emb, labels)
assert loss_c.item() >= 0

#, triplet mask,
mask = get_triplet_mask(labels)
```

```
assert mask.shape == (6, 6, 6)
# self-pairs must never appear
for i in range(6):
    assert not mask[i, i, :].any()
    assert not mask[:, i, i].any()

#, triplet loss,
loss_t, n = triplet_loss(emb, labels)
assert loss_t.item() >= 0
assert isinstance(n, int)

#, edge cases,
iou_zero = pairwise_contrastive_loss(
    F.normalize(torch.eye(6), dim=1),
    torch.tensor([0,0,1,1,2,2]),
    margin=1.0)
assert iou_zero.item() >= 0
```

File	Required Functions	Autograder Tests
losses.py	compute_distance_matrix	Shape (B, B), zero diagonal
	pairwise_contrastive_loss	Non-negative, correct signature
	get_triplet_mask	Shape (B, B, B), no self-pairs
	triplet_loss	Non-negative loss, int triplet count

5.5 Additional Guidelines

- **Include code snippets in the report for every task.** For each task, show the code alongside the output it produced.
- Convert Jupyter notebooks to `.py` files. **Do NOT submit `.ipynb` files.**
- Guard all executable code with `if __name__ == '__main__':` blocks.
- **Do NOT submit dataset files.**
- Your code and report must be your own work.
- Cite DLStudio wherever you borrow or adapt code.

6 Frequently Asked Questions (FAQ)

Q: Can I use `pytorch-metric-learning` for the loss functions?

A: No. You must implement `compute_distance_matrix`, `pairwise_contrastive_loss`, `get_triplet_mask`, and `triplet_loss` from scratch. You may use it only as a reference to verify your numerical results.

Q: My triplet loss is 0 after just a few iterations.

A: This is embedding collapse, all vectors have converged to the same point. Make sure your embeddings are L2-normalized before passing them to `triplet_loss`, and verify that `get_triplet_mask` returns a non-zero number of valid triplets.

Q: The NCE script raises an import error.

A: Copy the script exactly from the lecture slides and save it as `NCE_for_learning_point_distro.py` The only non-standard dependency is `matplotlib`; all other imports (`torch`, `numpy`, `random`) are standard.

Q: CLIP's `ViT-B/32` produces 512-dimensional embeddings. Should my FAISS index use 512?

A: Yes. Use `faiss.IndexFlatIP(512)` for inner-product search. Since CLIP embeddings are L2-normalized, inner product equals cosine similarity.

Q: My CLIP zero-shot accuracy is lower than I expected.

A: This is normal. Zero-shot CLIP works best when the text prompts closely match the distribution of captions used during pretraining. LVIS categories with ambiguous or compound names (e.g., `car_(automobile)`) may need simplified prompts. Try `"a photo of a car"` rather than `"a photo of a car_(automobile)"`.

Q: I don't have a GPU. Can I still do this homework?

A: Yes. Tasks 1–3 run comfortably on CPU. For Task 4, extracting CLIP embeddings for a few thousand LVIS images on CPU will take 10–20 minutes but is feasible. Use Google Colab's free GPU if needed.

Q: Can I work in a group?

A: No. This is individual work.

References

[1] Avinash Kak, *Metric Learning with Deep Neural Networks*. Available at: https://engineering.purdue.edu/kak/pdf-kak/MetricLearning.pdf

[2] Avinash Kak, *DLStudio Platform*. Available at: https://engineering.purdue.edu/kak/distDLS/

[3] Raia Hadsell, Sumit Chopra, and Yann LeCun, *Dimensionality Reduction by Learning an Invariant Mapping*, IEEE Conference on Com-

puter Vision and Pattern Recognition (CVPR), 2006. Available at: https://www.academia.edu/download/56222713/cvpr06.pdf

[4] Florian Schroff, Dmitry Kalenichenko, and James Philbin, *FaceNet: A Unified Embedding for Face Recognition and Clustering*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015. Available at: https://arxiv.org/pdf/1503.03832.pdf

[5] Aaron van den Oord, Yazhe Li, and Oriol Vinyals, *Representation Learning with Contrastive Predictive Coding*, arXiv preprint, 2018. Available at: https://arxiv.org/abs/1807.03748

[6] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, et al., *Learning Transferable Visual Models From Natural Language Supervision*, International Conference on Machine Learning (ICML), 2021. Available at: https://arxiv.org/abs/2103.00020

[7] Michael Gutmann and Aapo Hyvärinen, *Noise-Contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models*, International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou, *Billion-Scale Similarity Search with GPUs*, IEEE Transactions on Big Data, 2019. Available at: https://arxiv.org/pdf/1702.08734.pdf