ECE60146 Homework 4

> Tianyuan Qiu
> PUID: 37772148
> Email: qiu270@purdue.edu

Environment Setup

- Python 3.11.14 on Ubuntu 24.04.1 LTS and x86_64 CPU
    - `requirements.txt` included
- Python codes are formatted with `black` (PEP 8)

3.1 Task 1 - DLStudio Exploration

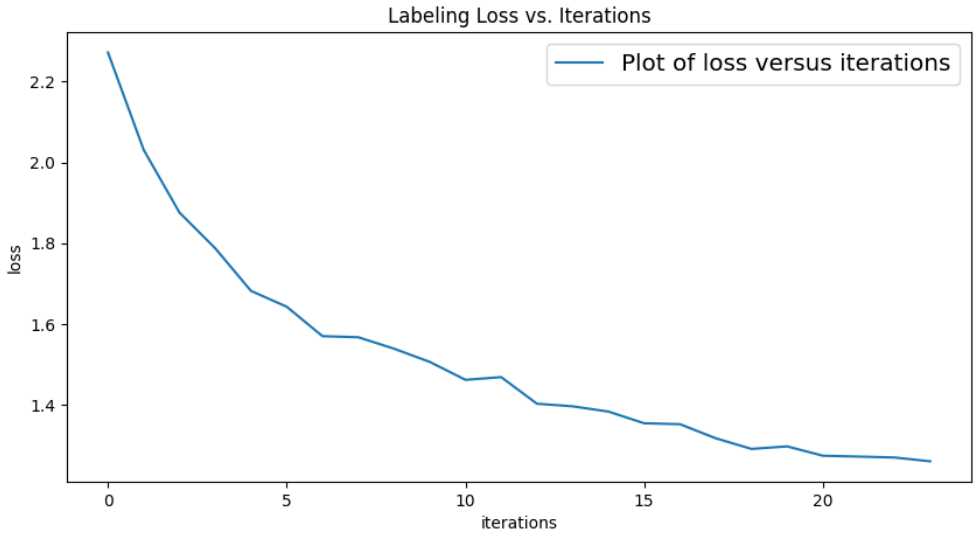
3.1.1 Running the CIFAR-10 Example

Run the official example script exactly as required: `python playing_with_cifar10.py`.

Run command:

```
$ python -c "import DLStudio; print(DLStudio.__version__)"
2.5.5
$ python ../DLStudio-2.5.5/Examples/playing_with_cifar10.py >
playing_with_cifar10.log
```

Output




```
# playing_with_cifar10.log
The number of learnable parameters in the model: 62006
...
Prediction accuracy for plane : 56 %
Prediction accuracy for   car : 73 %
Prediction accuracy for  bird : 32 %
Prediction accuracy for   cat : 25 %
Prediction accuracy for  deer : 60 %
Prediction accuracy for   dog : 47 %
Prediction accuracy for  frog : 62 %
Prediction accuracy for horse : 64 %
Prediction accuracy for  ship : 69 %
Prediction accuracy for truck : 55 %

Overall accuracy of the network on the 10000 test images: 54 %
```

3.1.2 Understanding the Network Architecture

`ExperimentsWithCIFAR.Net` structure:

Layer (`torch.nn`)	Key setting	Output shape	Trainable params
`nn.Conv2d`	`in_channels=3, out_channels=6, kernel_size=5, stride=1, padding=0`	`[N, 6, 28, 28]`	`(6 * 3 * 5 * 5) + 6 = 456`
`nn.MaxPool2d`	`kernel_size=2, stride=2`	`[N, 6, 14, 14]`	`0`
`nn.Conv2d`	`in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=0`	`[N, 16, 10, 10]`	`(16 * 6 * 5 * 5) + 16 = 2416`
`nn.MaxPool2d`	`kernel_size=2, stride=2`	`[N, 16, 5, 5]`	`0`
`nn.Linear`	`in_features=400, out_features=120`	`[N, 120]`	`(400 * 120) + 120 = 48120`
`nn.Linear`	`in_features=120, out_features=84`	`[N, 84]`	`(120 * 84) + 84 = 10164`
`nn.Linear`	`in_features=84, out_features=10`	`[N, 10]`	`(84 * 10) + 10 = 850`

Total trainable parameters:

`456 + 2416 + 48120 + 10164 + 850 = 62006`.

Questions

1. `nn.Conv2d` applies learnable filters over local image regions to produce feature maps that capture patterns such as edges, textures, and object parts. Its key parameters are `in_channels`, `out_channels`, `kernel_size`, `stride`, and `padding`, which control feature extraction capacity and spatial size changes. The layer has trainable `weight` and `bias` parameters that are updated by backpropagation.
2. `nn.MaxPool2d` downsamples feature maps by taking local maxima, which reduces spatial resolution and computation. It keeps the strongest activations while discarding less informative details, helping the network focus on salient features. This also makes features more robust to small translations in the input.
3. Convolution and pooling produce 4D tensors shaped like `[N, C, H, W]`, while fully connected layers require 2D input `[N, F]`. `view()` or `flatten()` reshapes each sample into a single feature vector without changing the batch dimension. Without this

step, `nn.Linear` cannot perform the required matrix multiplication.

4. `nn.CrossEntropyLoss` combines `log_softmax` with negative log-likelihood to measure how well logits match integer class labels. It penalizes low probability assigned to the correct class and encourages separation between classes. This is the standard loss for single-label multi-class classification tasks like CIFAR-10.

3.2 Task 2 - LVIS Dataset Creation

3.2.1 Selecting the 5 Categories

I listed all categories with `instance_count > 10000` and `frequency = frequent`. Then selected categories that are more likely to co-occur in everyday scenes to increase Dataset 3 (multi-instance-different) coverage.

Final selected categories used in all three datasets:

Category	LVIS frequency group	Instance count in LVIS train	Unique image count in LVIS train
`person`	frequent (`f`)	13439	1928
`chair`	frequent (`f`)	11549	1927
`book`	frequent (`f`)	33353	1903
`bottle`	frequent (`f`)	7969	1901
`cup`	frequent (`f`)	4637	1521

These classes are visually distinct and semantically related, which improves cross-category co-occurrence for Dataset 3.

3.2.2 Dataset 1: Single-Instance Images

Applied filtering criteria:

- exactly one instance of the target category in the image
- target area ratio threshold (`area / image_area`) to keep reasonably visible objects
- image can contain other non-target categories
- one image is used at most once globally across selected classes

```
# dataset_creation.py
def create_single_instance_dataset(
    lvis, category_name, category_info, min_area_ratio=0.05, max_images=500
):
    ...
    target_anns = [ann for ann in anns if ann["category_id"] == cat_id]
    if len(target_anns) != 1:
        continue
    if float(target_anns[0]["area"]) / img_area < min_area_ratio:
        continue
```

Output counts after train/val split (before augmentation):

```
person 400/100, chair 363/91, book 197/50, bottle 243/61, cup 266/67
```

3.2.3 Dataset 2: Multi-Instance Same Object

Applied filtering criteria:

- at least two instances of the target category in one image
- combined target-instance area must satisfy a minimum ratio
- one image is used at most once globally across selected classes

```python
# dataset_creation.py
def create_multi_same_dataset(
    lvis,
    category_name,
    category_info,
    min_instances=2,
    min_total_area_ratio=0.10,
):
    ...
    if len(target_anns) < min_instances:
        continue
    total_target_area = sum(float(ann["area"]) for ann in target_anns)
    if total_target_area / img_area < min_total_area_ratio:
        continue
```

Output counts after train/val split (before augmentation):

```
person 400/100, chair 400/100, book 400/100, bottle 400/100, cup 400/100
```

3.2.4 Dataset 3: Multi-Instance Different Objects

Applied filtering criteria:

- at least two different categories from the selected five must appear
- each image is assigned to exactly one class using the largest total category area
- assigned class must satisfy minimum area ratio

```
# dataset_creation.py
def create_multi_different_dataset(
    lvis, selected_categories, category_info, min_area_ratio=0.01
):
    ...
    if len(cats_present) < 2:
        continue
    assigned_category = max(cats_present, key=lambda name:
area_by_category[name])
    if area_by_category[assigned_category] / img_area < min_area_ratio:
        continue
```

Output counts after train/val split (before augmentation):

```
person 76/20, chair 146/37, book 68/18, bottle 45/12, cup 32/8
```

3.3 Task 3 - Image Processing and Storage

Image Processing Configuration

All extracted images are converted to RGB and resized to `(64, 64)` using bicubic interpolation.

```
# task3_prepare_data.py
with Image.open(src_path).convert("RGB") as img:
    resized = img.resize((64, 64), resample=Image.BICUBIC)
    resized.save(dst_path, quality=95)
```

To satisfy the target dataset size (`400` train / `100` val per class), I added split-safe augmentation when raw filtered samples are insufficient. The split is created first, then augmentation is applied independently inside each split directory (no cross-split reuse), preventing train/val leakage.

```
# task3_prepare_data.py
augmented_train = augment_split_to_target(train_dir, train_target)
augmented_val = augment_split_to_target(val_dir, val_target)

# augmentation modes (cycled):
# hflip, vflip, ±10 deg rotate, brightness up/down,
ImageOps.mirror(img)
ImageOps.flip(img)
img.rotate(10, resample=Image.BICUBIC)
img.rotate(-10, resample=Image.BICUBIC)
ImageEnhance.Brightness(img).enhance(1.2)
ImageEnhance.Brightness(img).enhance(0.8)
ImageEnhance.Contrast(img).enhance(1.2)
ImageEnhance.Contrast(img).enhance(0.8)
ImageEnhance.Sharpness(img).enhance(1.5)
# contrast up/down, sharpness up
```

Final dataset sizes are all 400/100.

Output Location and Directory Structure

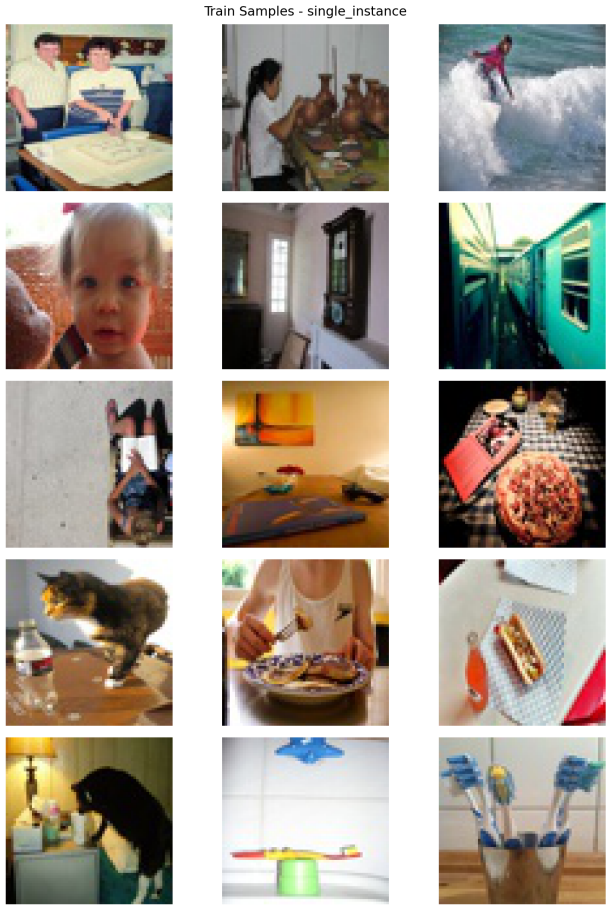
- dataset root: `data/lvis_datasets/`
- summary file: `data/dataset_summary.json`

```
data/lvis_datasets/
|- single_instance/train|val/{person,chair,book,bottle,cup}
|- multi_instance_same/train|val/{person,chair,book,bottle,cup}
|- multi_instance_different/train|val/{person,chair,book,bottle,cup}
```

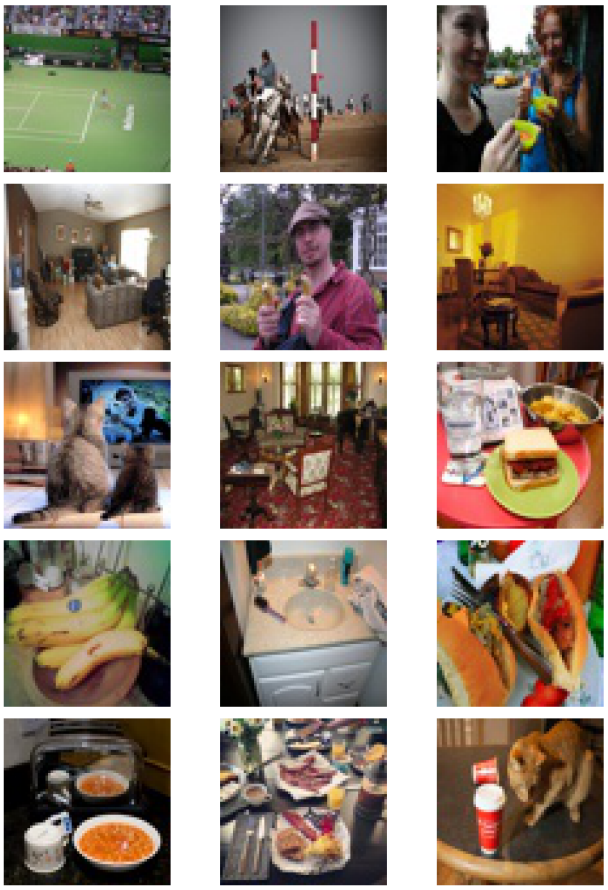
Result Artifacts

I used `task3_generate_samples.py` to create the sample grids shown below from the generated training splits.

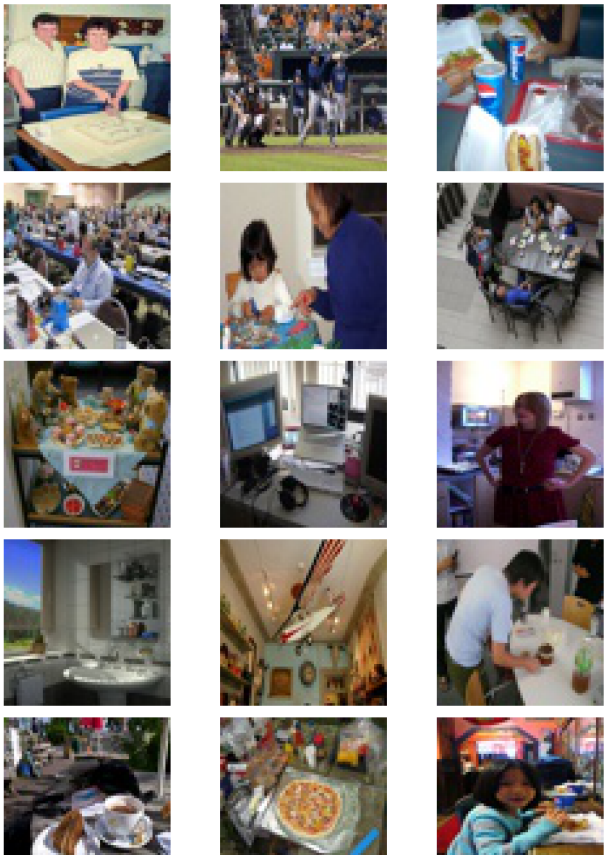
Sample grids (3 images per class per dataset type):



Train Samples - multi_instance_same



Train Samples - multi_instance_different





3.4 Task 4 - CNN Architecture Implementation

Implemented in `models.py` as required (`LVISNet(nn.Module)`):

```
# models.py
class LVISNet(nn.Module):
    def __init__(self, num_classes=5):
        super(LVISNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(2, 2)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool3 = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(128 * 8 * 8, 256)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(256, 128)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(128, num_classes)
```

Architecture Analysis (required order from the assignment):

1. Total number of trainable parameters

```
Total trainable parameters = 2,224,645
```

2. Output dimensions after each layer

`B` denotes batch size.

- after `conv1 + pool1`: `(B, 32, 32, 32)`
- after `conv2 + pool2`: `(B, 64, 16, 16)`
- after `conv3 + pool3`: `(B, 128, 8, 8)`
- after flatten: `(B, 8192)`
- after `fc1`: `(B, 256)`
- after `fc2`: `(B, 128)`

- after `fc3`: `(B, 5)`

3. Number of parameters in each required layer (`conv1`, `conv2`, `conv3`, `fc1`, `fc2`, `fc3`)

Layer	Formula	Parameters
`conv1`	`(32 * 3 * 3 * 3) + 32`	896
`conv2`	`(64 * 32 * 3 * 3) + 64`	18,496
`conv3`	`(128 * 64 * 3 * 3) + 128`	73,856
`fc1`	`(8192 * 256) + 256`	2,097,408
`fc2`	`(256 * 128) + 128`	32,896
`fc3`	`(128 * 5) + 5`	645

Required layer subtotal (`conv1/2/3 + fc1/2/3`):

```
896 + 18,496 + 73,856 + 2,097,408 + 32,896 + 645 = 2,224,197
```

`BatchNorm2d` layers also have trainable scale and bias parameters:

```
bn1: 32 + 32 = 64
bn2: 64 + 64 = 128
bn3: 128 + 128 = 256
BatchNorm total = 448
```

Final total trainable parameters:

```
2,224,197 (conv/fc subtotal) + 448 (batch norm) = 2,224,645
```

3.5 Task 5: Custom Dataset and DataLoader

Key code

```
# task5_data_utils.py
class LVISImageFolderDataset(Dataset):
    def __init__(self, root, class_names, transform=None):
        self.root = Path(root)
        self.class_names = list(class_names)
        self.transform = transform
        self.class_to_idx = {name: idx for idx, name in
enumerate(self.class_names)}
        self.samples = []

        for class_name in self.class_names:
            class_dir = self.root / class_name
            image_paths = sorted(
                p for p in class_dir.iterdir() if p.suffix.lower() in (".jpg",
".jpeg", ".png", ".bmp")
            )
            for image_path in image_paths:
                self.samples.append((image_path,
self.class_to_idx[class_name]))

    def __getitem__(self, index):
        image_path, label = self.samples[index]
        with Image.open(image_path).convert("RGB") as image:
            if self.transform is not None:
                image = self.transform(image)
        return image, label


def default_transforms(train=True):
    if train:
        return transforms.Compose(
            [
                transforms.Resize((72, 72),
interpolation=transforms.InterpolationMode.BICUBIC),
                transforms.RandomResizedCrop(64, scale=(0.8, 1.0)),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
            ]
        )
```

```
        )
    return transforms.Compose(

        [
            transforms.Resize((64, 64),
interpolation=transforms.InterpolationMode.BICUBIC),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
        ]
    )


train_loader, val_loader = create_dataloaders(
    dataset_root=dataset_root,
    class_names=SELECTED_CATEGORIES,
    batch_size=32,
    num_workers=2,
)
```

Output

I verified Task 5 using `task5_dataloader_check.py`.

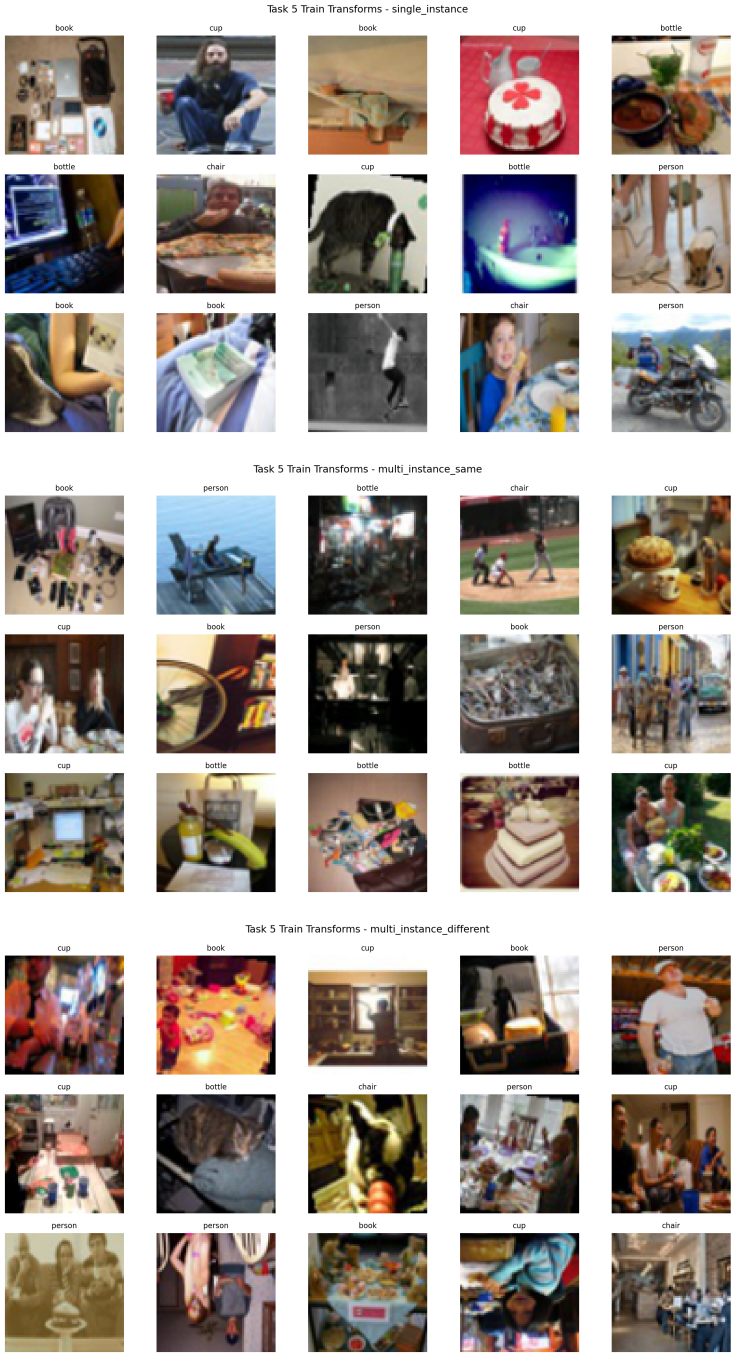
```
Task 5 Dataset/DataLoader verification

[single_instance]
train counts: person:400, chair:400, book:400, bottle:400, cup:400
val counts:   person:100, chair:100, book:100, bottle:100, cup:100
train batch shape: images=(32, 3, 64, 64), labels=(32,)

[multi_instance_same]
train counts: person:400, chair:400, book:400, bottle:400, cup:400
val counts:   person:100, chair:100, book:100, bottle:100, cup:100
train batch shape: images=(32, 3, 64, 64), labels=(32,)

[multi_instance_different]
train counts: person:400, chair:400, book:400, bottle:400, cup:400
val counts:   person:100, chair:100, book:100, bottle:100, cup:100
train batch shape: images=(32, 3, 64, 64), labels=(32,)
```

Class-count summary:

Dataset	Train per class	Val per class
single_instance	person/chair/book/bottle/cup = 400 each	person/chair/book/bottle/cup = 100 each
multi_instance_same	person/chair/book/bottle/cup = 400 each	person/chair/book/bottle/cup = 100 each
multi_instance_different	person/chair/book/bottle/cup = 400 each	person/chair/book/bottle/cup = 100 each

Data augmentation note: these balanced counts are achieved by split-safe augmentation in `task3_prepare_data.py` when raw LVIS filtered samples are below `400/100`.

Sample transformed training images generated by `task5_dataloader_check.py`:



3.6 Task 6: Training Implementation

Train with fixed config: `epochs=7`, `batch_size=32`, `lr=1e-3`, `Adam(betas=(0.9,0.99))`, `num_workers=32`, seed 42.

Key code

```
# task6_run_training.py
SEED = 42
EPOCHS = 7
BATCH_SIZE = 32
LEARNING_RATE = 1e-3
NUM_WORKERS = 32


def train_one_dataset(dataset_name, dataset_root, device):
    train_loader, val_loader = create_dataloaders(
        dataset_root=dataset_root,
        class_names=SELECTED_CATEGORIES,
        batch_size=BATCH_SIZE,
        num_workers=NUM_WORKERS,
    )

    model = LVISNet(num_classes=len(SELECTED_CATEGORIES)).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, betas=(0.9,
0.99))

    history = {
        "train_losses": [],
        "train_accuracies": [],
        "val_losses": [],
        "val_accuracies": [],
        "epoch_times": [],
    }

    for epoch in range(EPOCHS):
        ...  # Exactly same code from handouts

    cm = compute_confusion_matrix(model, val_loader, len(SELECTED_CATEGORIES),
device)
    class_acc = per_class_accuracy(cm)
    oa = overall_accuracy(cm)
    return {"history": history, "cm": cm, "class_acc": class_acc,
```

```
"overall_acc": oa, "model": model}
```

Output

Generated by `task6_run_training.py`.

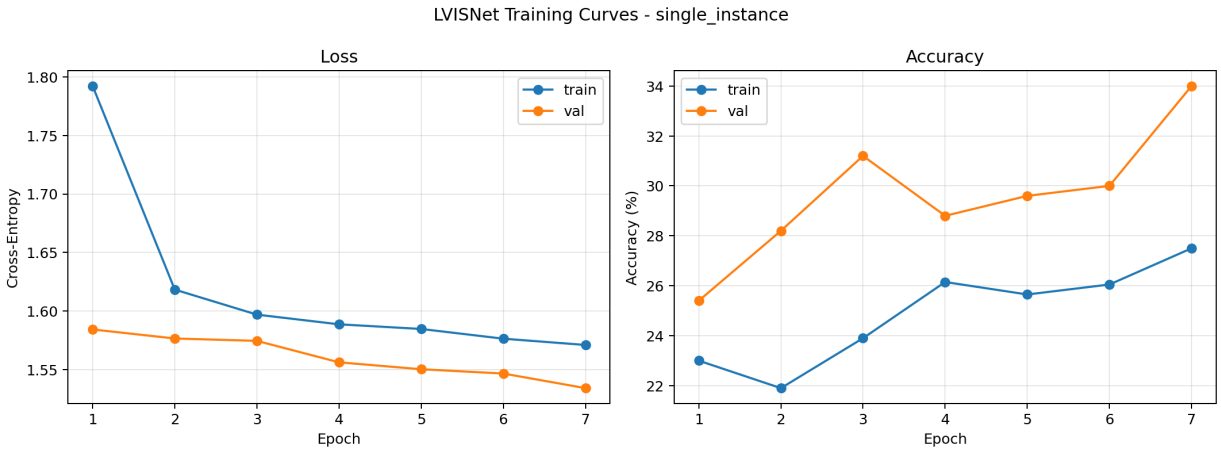
```
Using device: cuda
Done single_instance: val_acc=34.00%, overall=34.00%
Done multi_instance_same: val_acc=27.60%, overall=27.60%
Done multi_instance_different: val_acc=18.20%, overall=18.20%
Saved metrics to: results/metrics.json
```

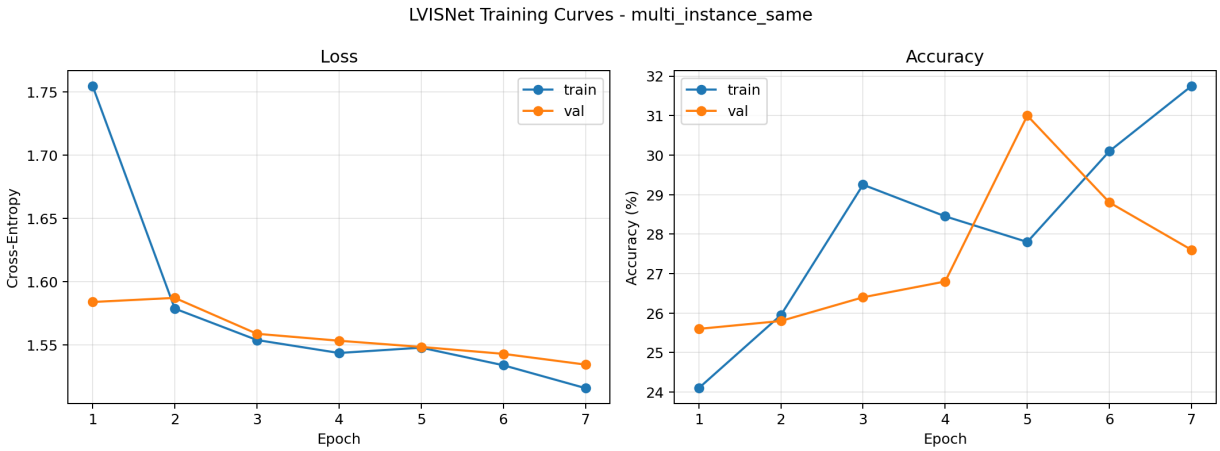
`overall accuracy` is computed from the confusion matrix and is equivalent to final validation accuracy when both are evaluated on the same validation split.

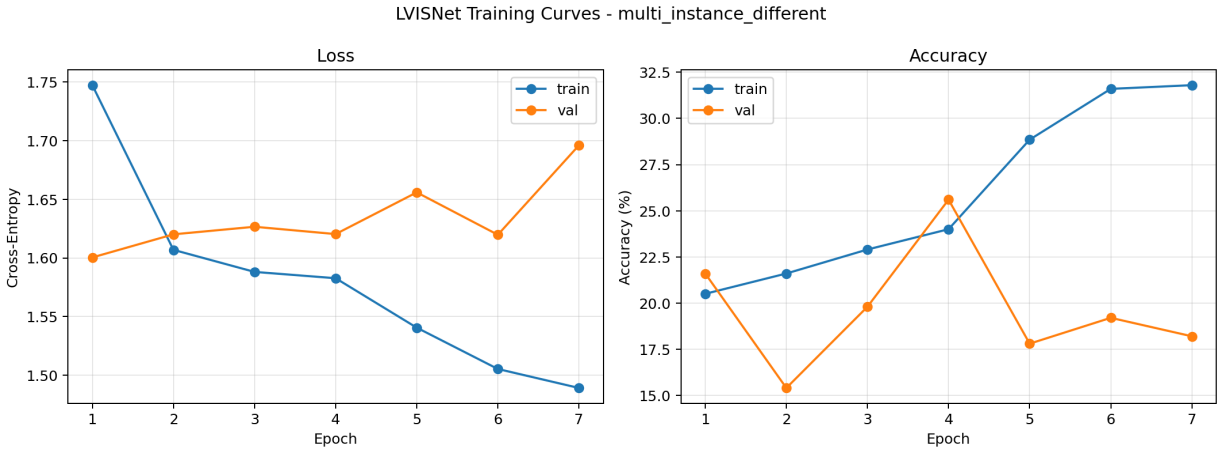
Summary table:

Dataset	Final Val Acc (%)	Overall Acc (%)	Train Time (s)
single_instance	34.00	34.00	23.57
multi_instance_same	27.60	27.60	18.78
multi_instance_different	18.20	18.20	18.51

Training/validation curves saved by `task6_run_training.py`:




LVISNet Training Curves - multi_instance_same
Loss
Cross-Entropy
Epoch
train
val
Accuracy
Accuracy (%)
Epoch
train
val


LVISNet Training Curves - multi_instance_different
Loss
Cross-Entropy
Epoch
train
val
Accuracy
Accuracy (%)
Epoch
train
val

3.7 Task 7: Confusion Matrix

Key code

```python
# evaluate.py
def compute_confusion_matrix(model, dataloader, num_classes, device):
    cm = np.zeros((num_classes, num_classes), dtype=np.int64)
    model.eval()
    with torch.no_grad():
        for images, labels in dataloader:
            outputs = model(images.to(device))
            preds = outputs.argmax(dim=1).cpu().numpy()
            truths = labels.cpu().numpy()
            for t, p in zip(truths, preds):
                cm[int(t), int(p)] += 1
    return cm
```

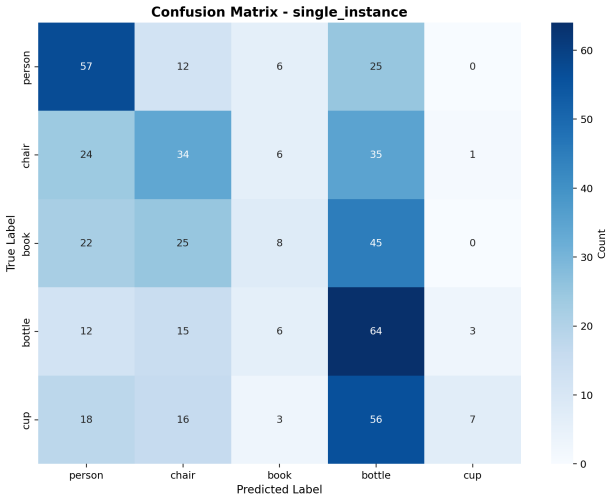
Output

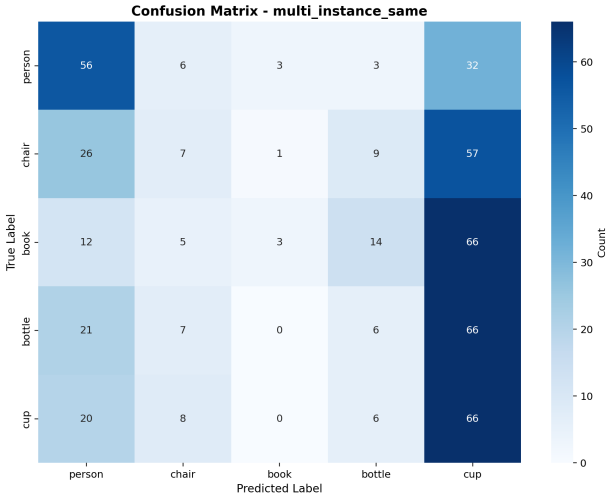
```
single_instance overall_acc = 0.3400
multi_instance_same overall_acc = 0.2760
multi_instance_different overall_acc = 0.1820
```

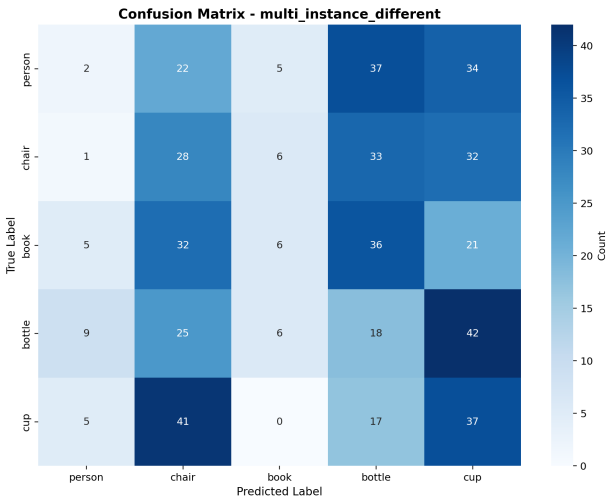
Per-class accuracy (generated from `results/metrics.json`):

Dataset	person (%)	chair (%)	book (%)	bottle (%)	cup (%)	overall (%)
single_instance	57.0	34.0	8.0	64.0	7.0	34.0
multi_instance_same	56.0	7.0	3.0	6.0	66.0	27.6
multi_instance_different	2.0	28.0	6.0	18.0	37.0	18.2

Confusion matrices:







Short observation:

- `single_instance`: strongest confusion is `cup -> bottle` (56), while `bottle` has the best class accuracy (64%).

- `multi_instance_same`: strongest confusion is `book -> cup` (66), showing frequent merge into `cup` predictions.

- `multi_instance_different`: strongest confusion is `bottle -> cup` (42); this split has the heaviest cross-class confusion and lowest overall accuracy.

---

4 Bonus Tasks

Key code

```
# bonus_run.py
experiments = [(LVISNet, "baseline"), (LVISNetDeep5, "deep5"), (LVISNetDeep10,
"deep10")]
for model_cls, name in experiments:
    result = train_model(model_cls, name, train_loader, val_loader, device)
```

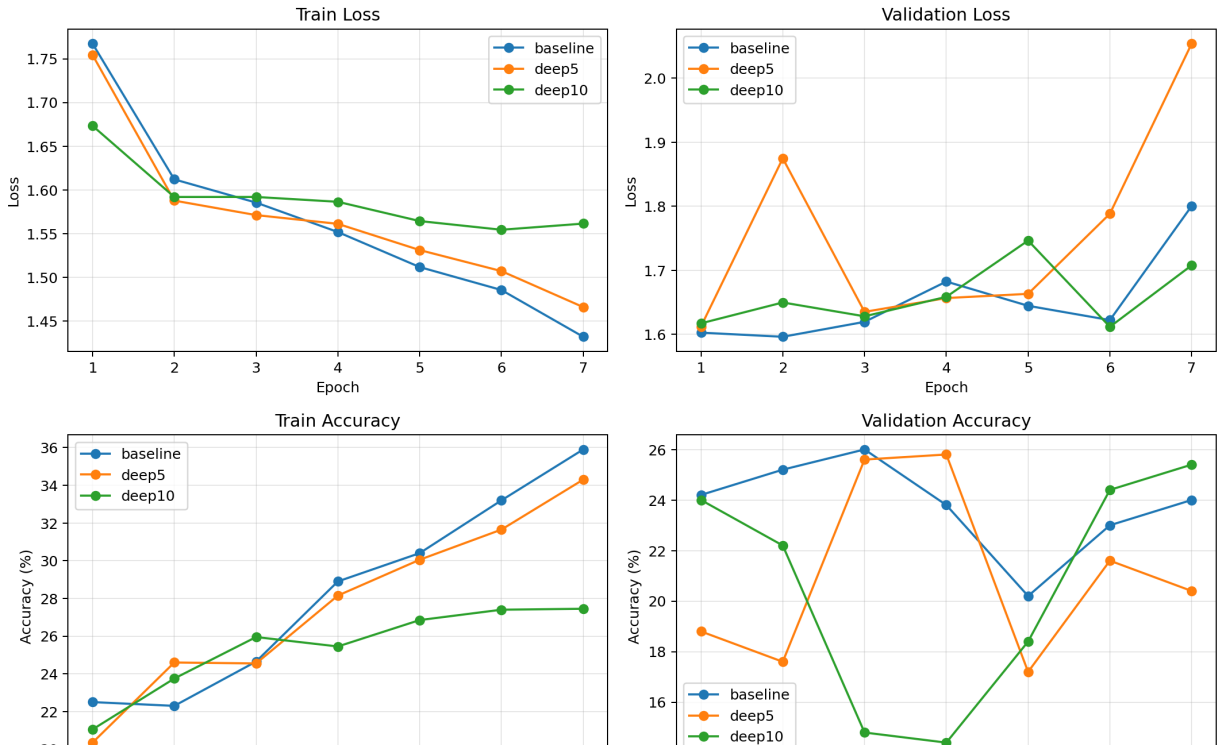
Output

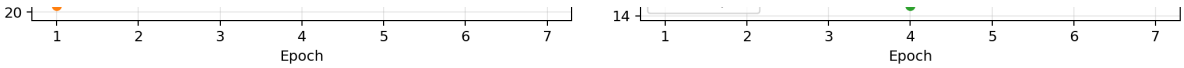
Generated by `bonus_run.py` on `multi_instance_different` only.

```
baseline overall: 0.2400, params: 2224645
deep5    overall: 0.2040, params: 2520325
deep10   overall: 0.2540, params: 3010213
```

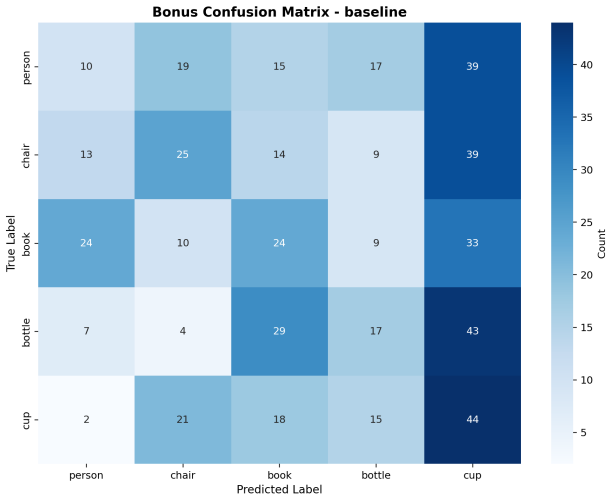
`overall` means overall validation accuracy on the bonus dataset.

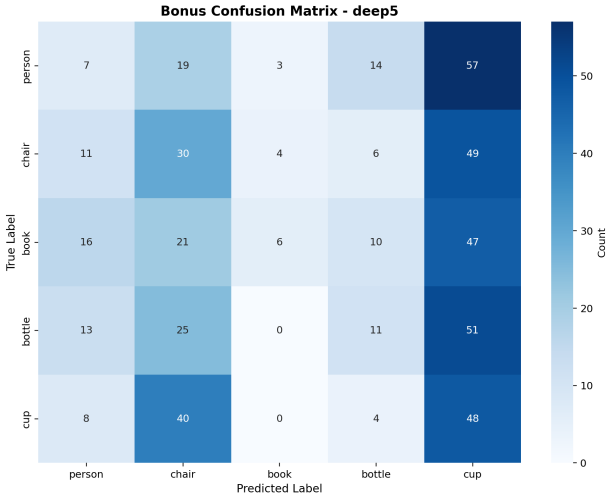
`bonus_curves.png` now uses four subplots: Train Loss, Validation Loss, Train Accuracy, and Validation Accuracy (baseline vs deep5 vs deep10).

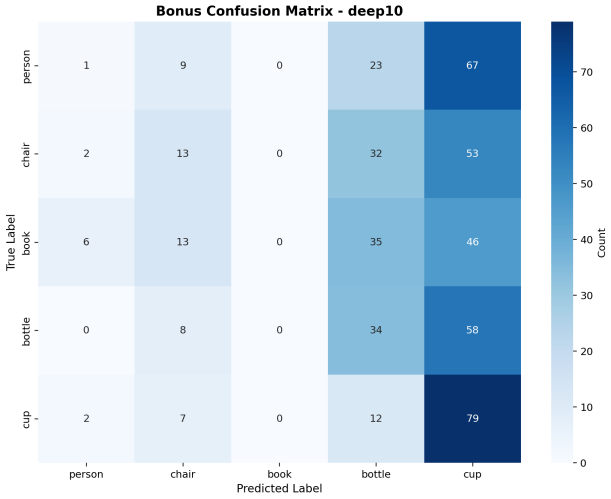




Bonus confusion matrices (generated by `bonus_run.py`):







Bonus comparison table:

Model	Params	Overall Acc (%)	Avg Epoch Time (s)
baseline	2,224,645	24.00	3.34
deep5	2,520,325	20.40	2.73
deep10	3,010,213	25.40	2.98

Does depth help?

- On this run, `deep10` is best and `deep5` is worse than baseline: `deep10 (25.4%) > baseline (24.0%) > deep5 (20.4%)`.
- So increased depth can help, but improvement is not monotonic and depends on optimization stability.

What challenges arise with deeper networks?

- With the same 7-epoch budget, deeper models are harder to optimize fully and remain undertrained.
- The dataset is still difficult (`multi_instance_different`), so class confusion remains high even with added depth.
- Compute/memory cost increases with depth, while accuracy gain is limited in this homework setting.