Lab02
Crowd Counting

PyTorch tutorial

- Official tutorial
  - https://pytorch.org/tutorials/
- 莫凡
  - https://mofanpy.com/tutorials/machine-learning/torch/
- AssemblyAI - PyTorch Crash Course
  - https://www.youtube.com/watch?v=OIenNRt2bjg

You can only use PyTorch in this Lab!!

Crowd Counting

- **Crowd counting** is a computer vision technique that aims to estimate the number of people in crowded images using deep learning models.
- It is essential for analyzing large gatherings where manual counting is impractical.
- Applications: Public safety monitoring, event management, smart city planning, transportation hubs (metro, airports), retail analytics, and disaster response.

Dataset

- UCSD Pedestrian Dataset
- Image size: 238*158 grayscale
- Ground Truth: 3 values for counts of walking away, toward, and total
- Training: 2500 Validation: 700 Testing: 800 (200 public + 600 private)

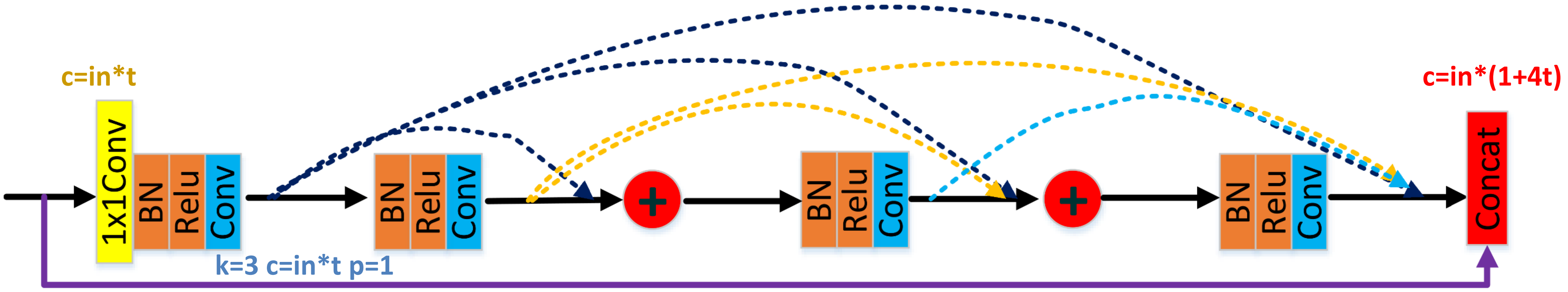
Task 1 of This Lab

- In “Lab02_CDenseNet.ipynb”
  - Build CDenseNet by yourself
  - Achieve MAE of **2.4 or lower** on public testing data

  (Put the screenshot in your report)

CDenseNet

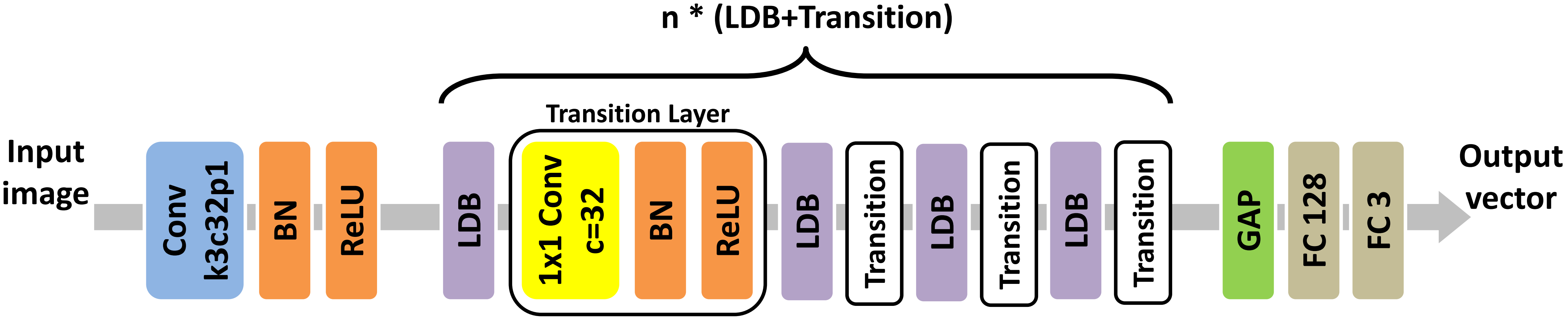
- DenseNet concatenates all prior features → channels explode → heavy compute/weights.
- **Lightweight Dense Block (LDB)** fuse by element-wise sum; use concat only at block input/output → fixed channel width with reuse.



(d) Lightweight dense block

CDenseNet

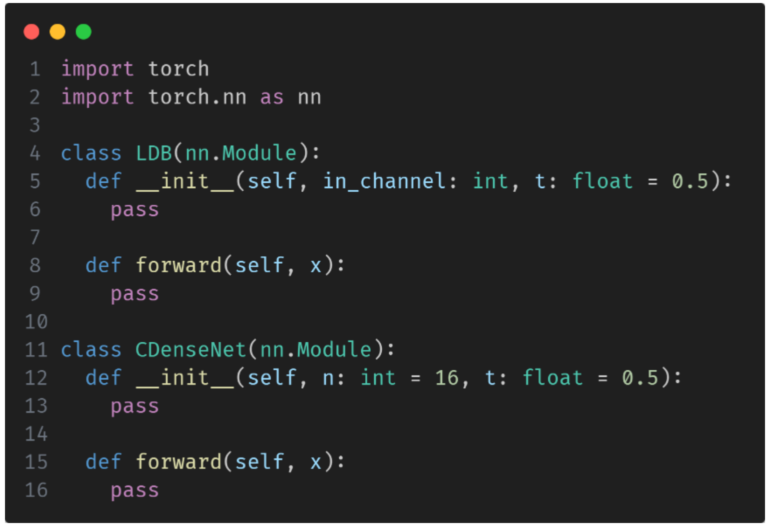
- Simplified Compress DenseNet (CDenseNet)
- Choose t = 0.5 & n = 16



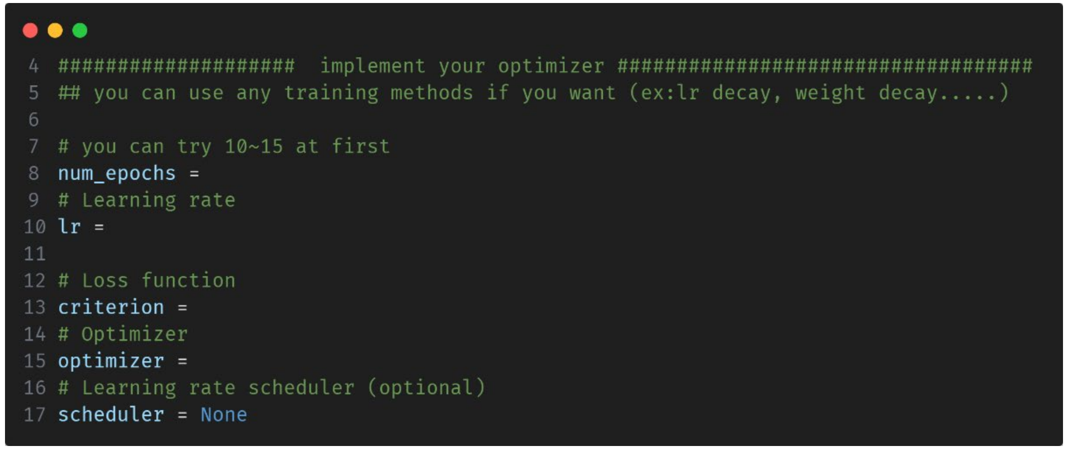
Please follow this model architecture!
We will check your implementation

Task 1 of This Lab

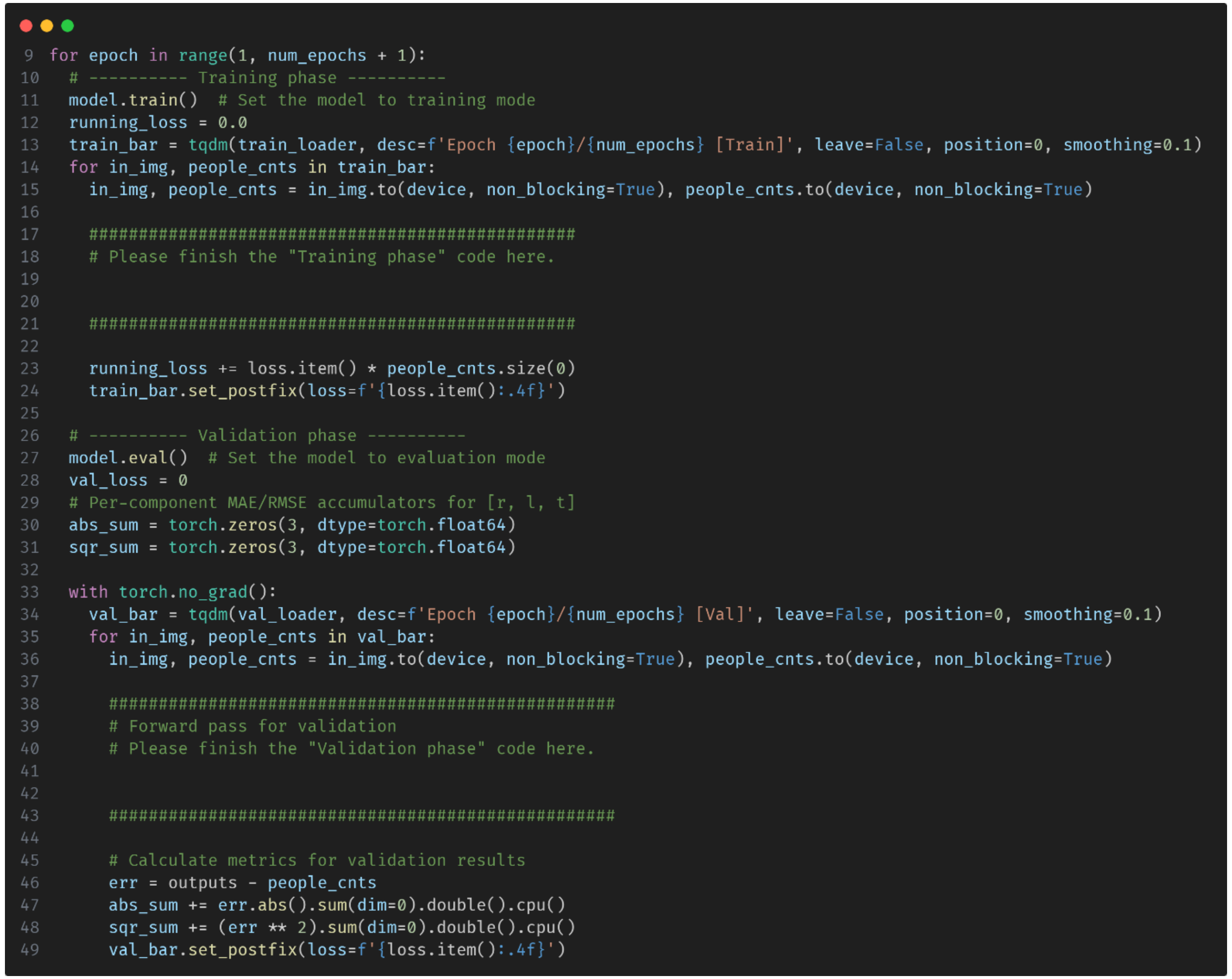
- Finish these parts (`CDenseNet.py` & training flow).



```python
import torch
import torch.nn as nn

class LDB(nn.Module):
  def __init__(self, in_channel: int, t: float = 0.5):
    pass

  def forward(self, x):
    pass

class CDenseNet(nn.Module):
  def __init__(self, n: int = 16, t: float = 0.5):
    pass

  def forward(self, x):
    pass
```



```python
####################  implement your optimizer ###################################
## you can use any training methods if you want (ex:lr decay, weight decay.....)

# you can try 10~15 at first
num_epochs =
# Learning rate
lr =

# Loss function
criterion =
# Optimizer
optimizer =
# Learning rate scheduler (optional)
scheduler = None
```



```python
for epoch in range(1, num_epochs + 1):
  # ---------- Training phase ----------
  model.train()  # Set the model to training mode
  running_loss = 0.0
  train_bar = tqdm(train_loader, desc=f'Epoch {epoch}/{num_epochs} [Train]', leave=False, position=0, smoothing=0.1)
  for in_img, people_cnts in train_bar:
    in_img, people_cnts = in_img.to(device, non_blocking=True), people_cnts.to(device, non_blocking=True)

    ##################################################
    # Please finish the "Training phase" code here.


    ##################################################

    running_loss += loss.item() * people_cnts.size(0)
    train_bar.set_postfix(loss=f'{loss.item():.4f}')

  # ---------- Validation phase ----------
  model.eval()  # Set the model to evaluation mode
  val_loss = 0
  # Per-component MAE/RMSE accumulators for [r, l, t]
  abs_sum = torch.zeros(3, dtype=torch.float64)
  sqr_sum = torch.zeros(3, dtype=torch.float64)

  with torch.no_grad():
    val_bar = tqdm(val_loader, desc=f'Epoch {epoch}/{num_epochs} [Val]', leave=False, position=0, smoothing=0.1)
    for in_img, people_cnts in val_bar:
      in_img, people_cnts = in_img.to(device, non_blocking=True), people_cnts.to(device, non_blocking=True)

      ##################################################
      # Forward pass for validation
      # Please finish the "Validation phase" code here.


      ##################################################

      # Calculate metrics for validation results
      err = outputs - people_cnts
      abs_sum += err.abs().sum(dim=0).double().cpu()
      sqr_sum += (err ** 2).sum(dim=0).double().cpu()
      val_bar.set_postfix(loss=f'{loss.item():.4f}')
```

Task 2 of This Lab

- In Task2
  - Do your best to improve the prediction accuracy
  - Calling different models with pretrained weight is allowed
  - Basically, any methods you learn are allowed
  - Achieve MAE of **2.0 or lower** on public testing data

  (put the screenshot in your report)

Report

- Your report should include/answer
  - Required
    - Screenshot of Task 1 (MAE on public testing data **<= 2.4**)
    - Screenshot of Task 2 (MAE on public testing data **<= 2.0**)
    - In Task 2
      - What model did you choose?
      - Why did you choose this model? What advantages does it offer?
    - Compare the characteristics of **MAE (Mean Absolute Error)** and **RMSE (Root Mean Square Error)**. In what types of scenarios might one be preferred over the other?
    - Another popular method in crowd counting is “**density map estimation**.” Briefly explain what density map estimation means in the context of crowd counting. How does it differ from the regression-based approach used in our implementation? Give at least one metrics used to evaluate it.
  - Can include but not limited to
    - Anything you do to improve the accuracy.
    - Discuss any challenges you faced.

Score

- MAE on public testing data in Task 1 <= 2.4 (30%)
  - If the model architecture in Task 1 is incorrect, points will be deducted accordingly
- MAE on public testing data in Task 2 <= 2.0 (30%)
- Report (30%)
- Performance ranking for Task 1 (10%)
  - Ranked based on MAE on the full testing data in Task 1
  - 0 points will be given if your model is found trained in an abnormal way

- Please do not plagiarize, or you will receive 0 points if caught

Reminder

- Submit Deadline : 2 week (2025-10-06 23:59)
- Upload these files to E3
  - `Lab02_CDenseNet_StudentID.ipynb`
  - `CDenseNet_StudentID.py`
  - `model_StudentID.pth` (of Task 1)
  - `summary_StudentID.txt` (of Task 1)
  - `Lab02_report_StudentID.pdf`

Supplements

- **paper**
  - https://arxiv.org/abs/1912.07016
  - https://ieeexplore.ieee.org/document/6054049

HAVE FUN !!!