Assignment 2:

Recognition/Classification

Computer Vision

NTU, Spring 2023

Announced: 2023/03/24 (Fri.)

Due: 2023/04/13 (Thu.) 23:59

Outline

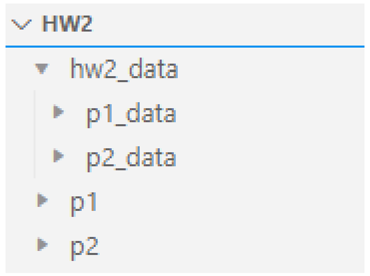
Part 1: Bag-of-Words Scene Recognition

Part 2: CNN Image Classification

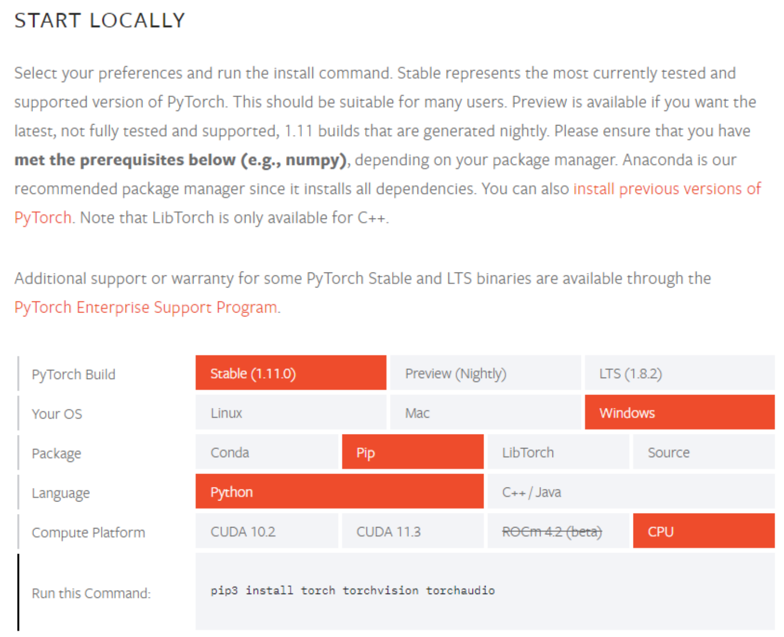
Before you start…

- Download the dataset
  - hw2_data.zip
  - https://drive.google.com/file/d/14jRYKi9mgrn2oJ5TYrPB5svh_GxN9pne/view?usp=sharing

- Install pytorch package in official website (strongly recommended)
  - Start Locally | PyTorch



```
HW2
  hw2_data
    p1_data
    p2_data
  p1
  p2
```



START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.11 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

Additional support or warranty for some PyTorch Stable and LTS binaries are available through the PyTorch Enterprise Support Program.

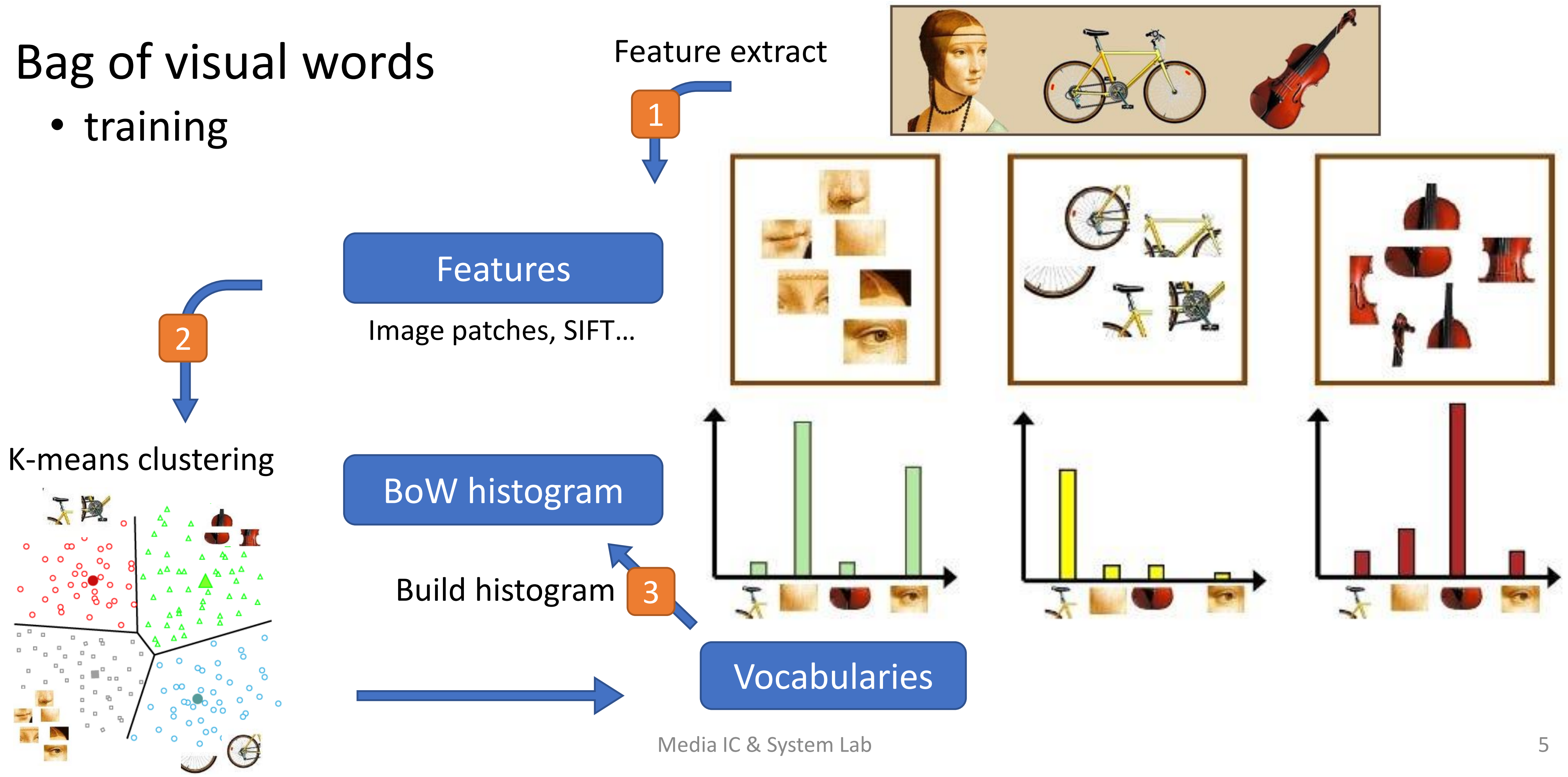
PyTorch Build	[✓ Stable (1.11.0)]	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	[✓ Windows]	
Package	Conda	[✓ Pip]	LibTorch	Source
Language	[✓ Python]	C++ / Java		
Compute Platform	CUDA 10.2	CUDA 11.3	~~ROCm 4.2 (beta)~~	[✓ CPU]
Run this Command:	`pip3 install torch torchvision torchaudio`			

Part 1 :

Bag-of-Words Scene Recognition

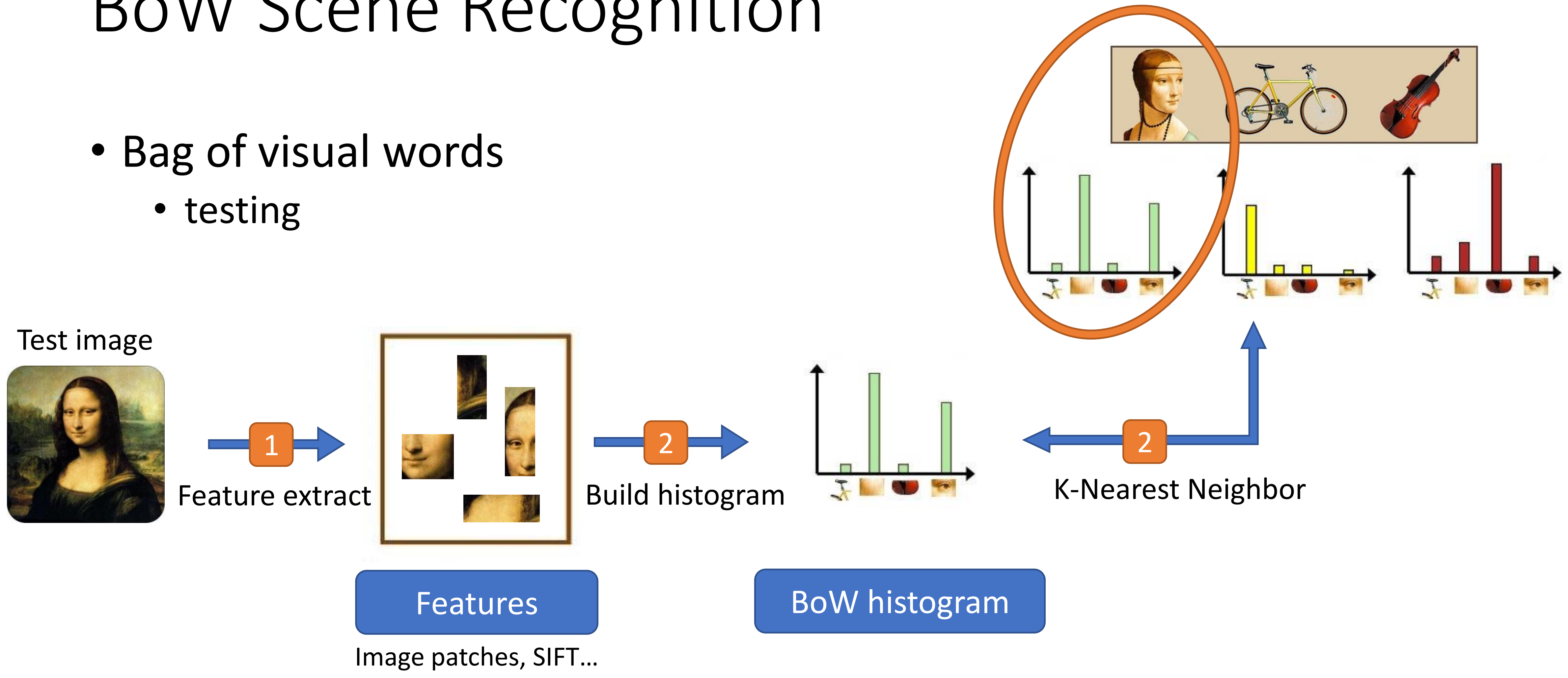
BoW Scene Recognition

- Bag of visual words
  - training



BoW Scene Recognition

- Bag of visual words
  - testing



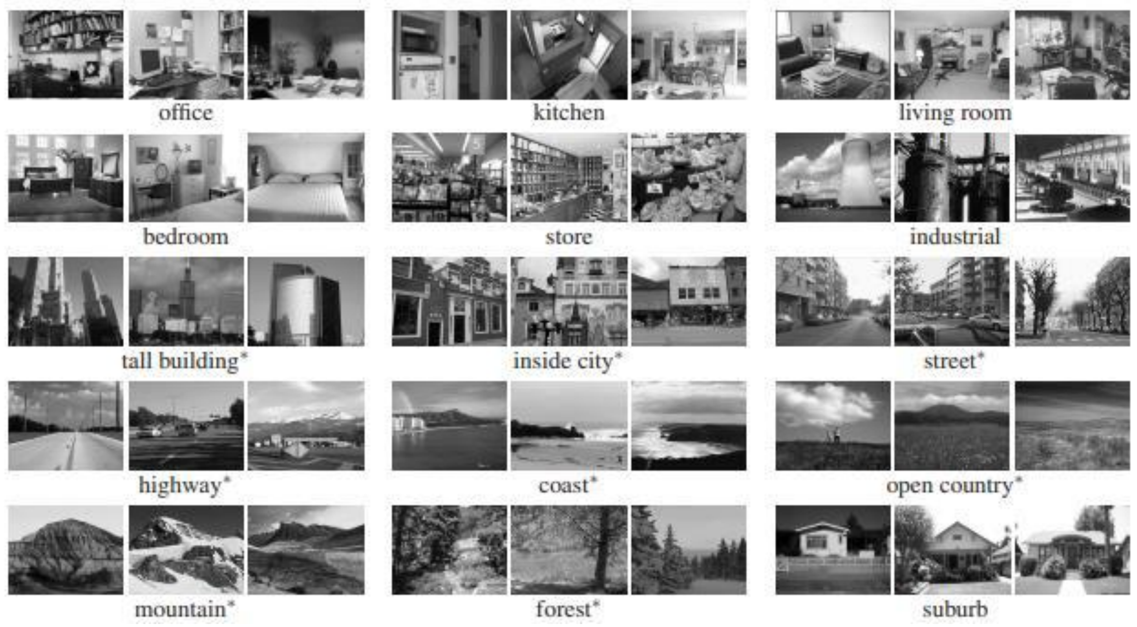
BoW Scene Recognition

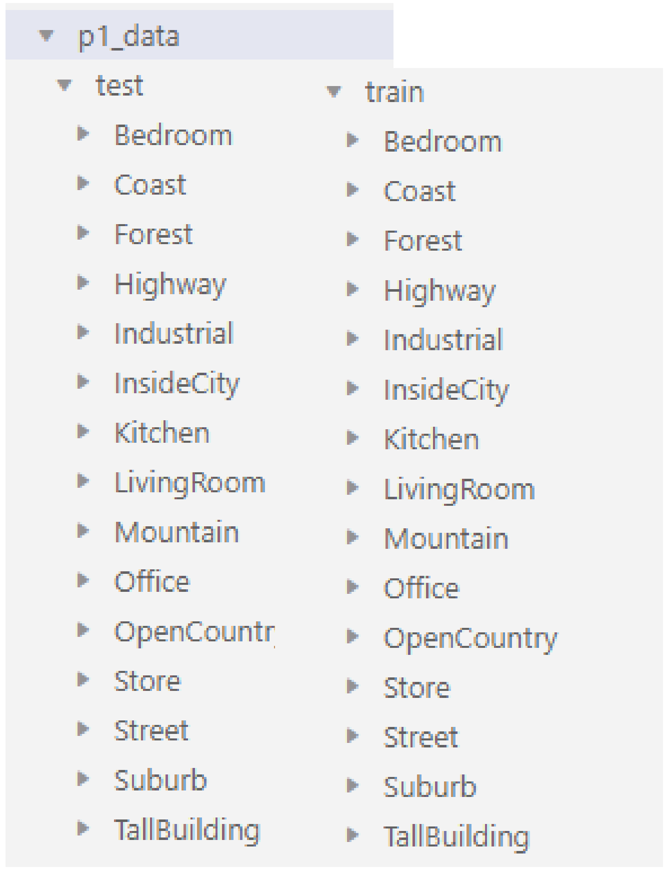
- You will have to implement two kinds of feature representations
  1. Tiny image
     - get_tiny_images()
  2. Bag of SIFT (cyvlfeat.sift is allowed)
     - build_vocabulary()
     - get_bags_of_sifts()

- Then apply hand-craft classifier
  1. K-Nearest Neighbor (sklearn.neighbors.KNeighborsClassifier is NOT allowed)
     - nearest_neighbor_classify()

Dataset

- hw2_data/p1_data/train
  - 100 grayscale images per category
- hw2_data/p1_data/test
  - 100+ grayscale images per category





Assignment Description

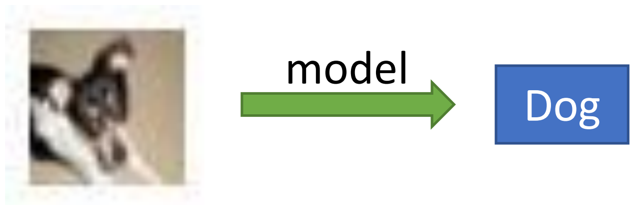
- p1/p1.py
  - Read data, feature extract, classification, compute accuracy, plot confusion matrix.
  - TA will run this code to evaluate your result.
- p1/utils.py
  - get_tiny_images() ## TODO ##
    - Build tiny image features.
  - build_vocabulary() ## TODO ##
    - Sample SIFT descriptors from training images, cluster them with kmeans and return centroid.
  - get_bags_of_sifts() ## TODO ##
    - Construct SIFT and build a histogram indicating how many times each centroid was used.
  - nearest_neighbor_classify() ## TODO ##
    - Predict the category for each test image.
    - CAN NOT USE sklearn.neighbors.KNeighborsClassifier
- p1/p1_run.sh
  - example for code execution.

Part 2 :

CNN Image Classification

CNN Image Classification

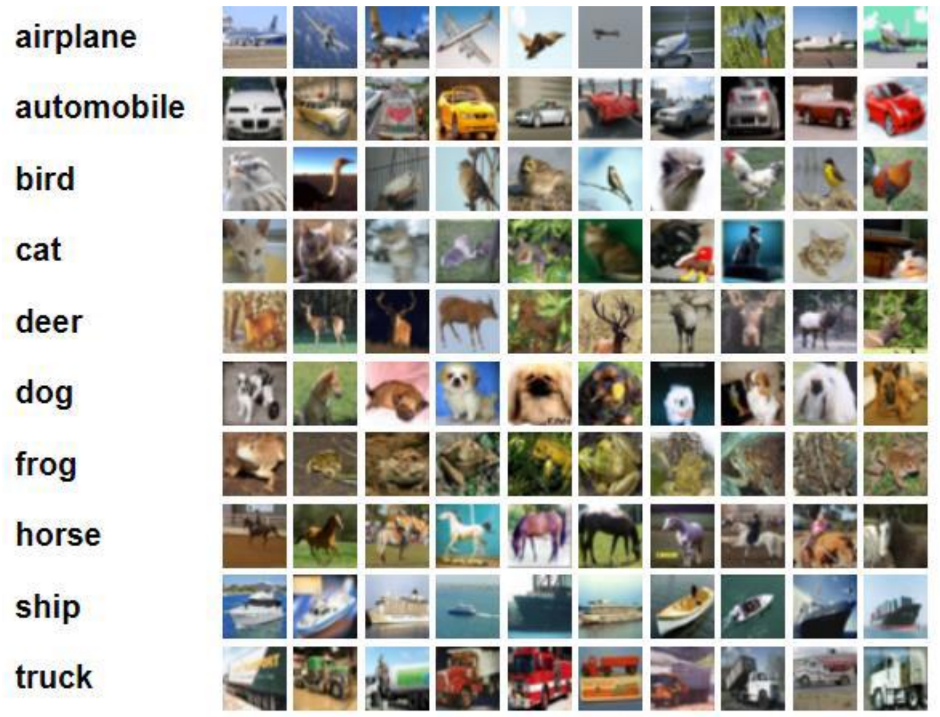
- Image classification – predict a label for each image
  - Input : RGB image
  - Output : classification label

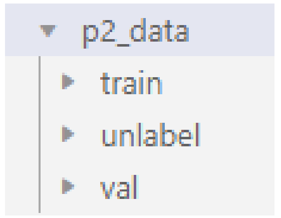


- You need to perform image classification with the following methods:
  - A. Build and train a CNN model from scratch
    - torchvision.models, pretrained weights is NOT allowed.
  - B. Try ResNet18 model
    - You can use torchvision.models.resnet18(pretrained=True).
    - You may have to modify the structure to pass the strong baseline.

Dataset

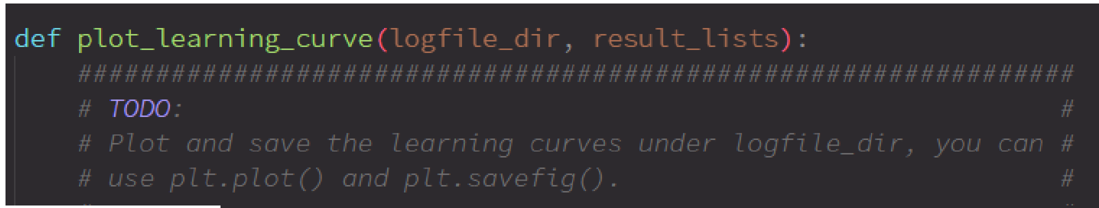
- Original CIFAR-10 dataset (32x32 RGB images)
  - 50000 training images, 5000 each class
  - 10000 test images
- Provided dataset (You can only use this)
  - p2_data/train
    - 20000 images, 1 annotation file (w/ labels)
  - p2_data/val
    - 5000 images, 1 annotation file (w/ labels)
    - You cannot use the validation labels to train your model in a fully supervised manner.
  - p2_data/unlabel
    - 30000 images, 1 annotation file (w/o labels)
    - For semi-supervised (optional)



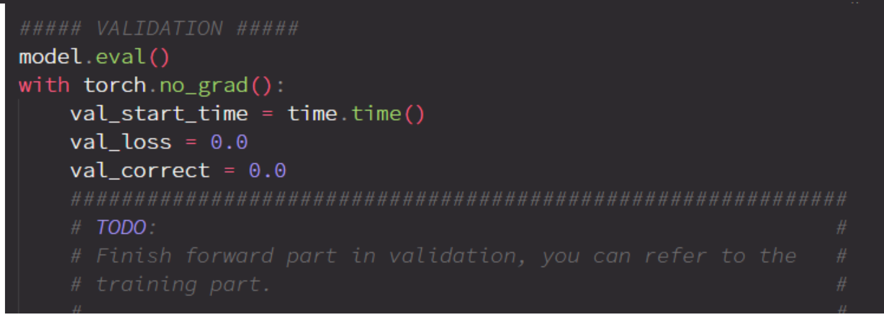


Assignment Description

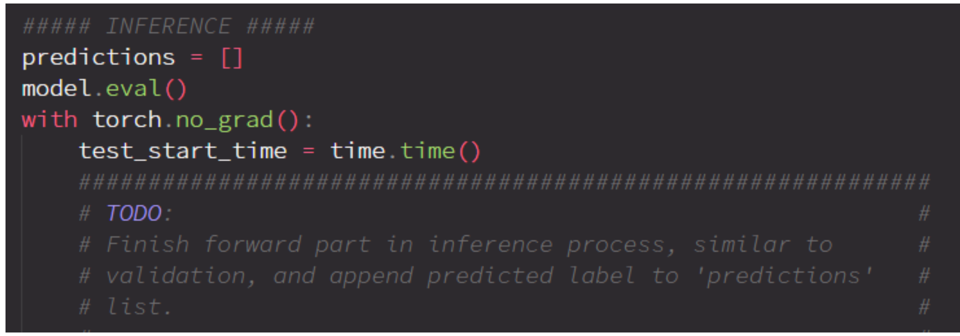
- p2/p2_train.py ## TODO ##
  - Start training, write log files, plot learning curves, etc.
- p2/p2_inference.py ## TODO ##
  - Inferencing, generate predicted output to csv file.
  - TA will run this code to generate your result.
- p2/p2_eval.py ## DO NOT MODIFY ##
  - Evaluate accuracy between predicted and ground truth labels.
  - TA will run this code to evaluate your result.



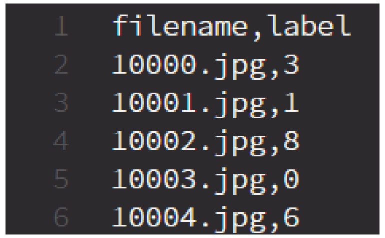
```
def plot_learning_curve(logfile_dir, result_lists):
    ###############################################################
    # TODO:                                                       #
    # Plot and save the learning curves under logfile_dir, you can #
    # use plt.plot() and plt.savefig().                           #
```



```
##### VALIDATION #####
model.eval()
with torch.no_grad():
    val_start_time = time.time()
    val_loss = 0.0
    val_correct = 0.0
    ###########################################################
    # TODO:                                                   #
    # Finish forward part in validation, you can refer to the #
    # training part.                                          #
```



```
##### INFERENCE #####
predictions = []
model.eval()
with torch.no_grad():
    test_start_time = time.time()
    ###########################################################
    # TODO:                                                   #
    # Finish forward part in inference process, similar to    #
    # validation, and append predicted label to 'predictions' #
    # list.                                                   #
```

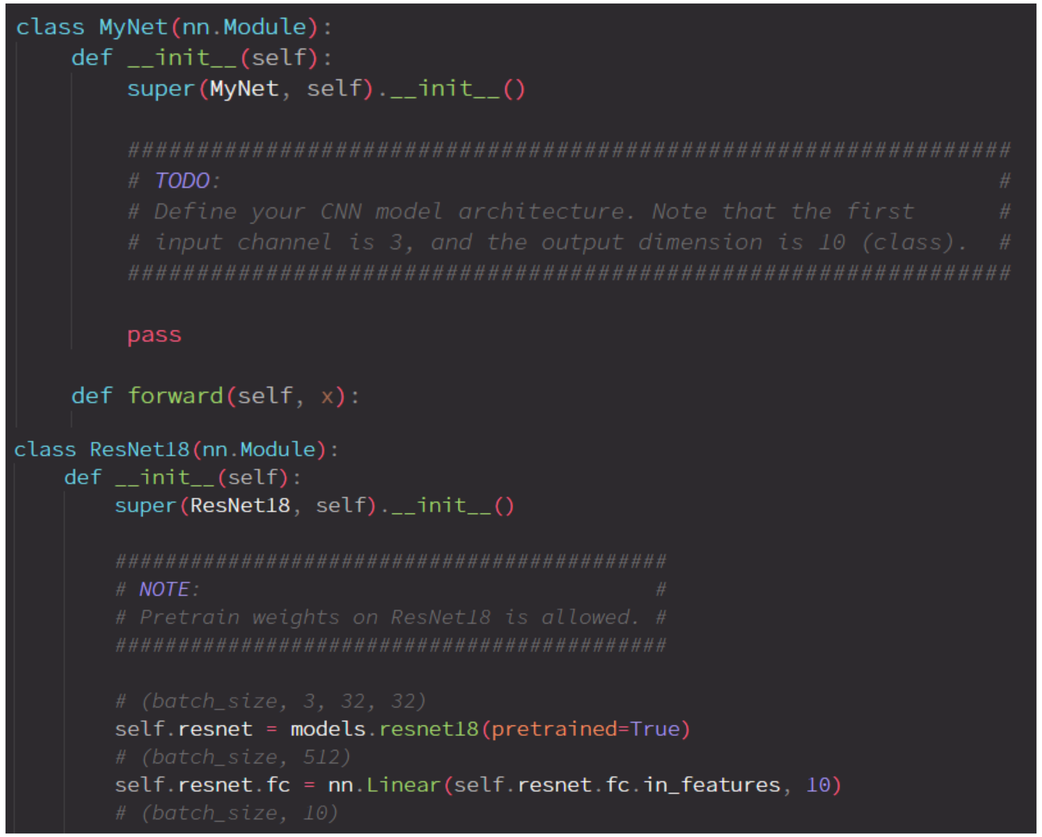


```
filename,label
10000.jpg,3
10001.jpg,1
10002.jpg,8
10003.jpg,0
10004.jpg,6
```

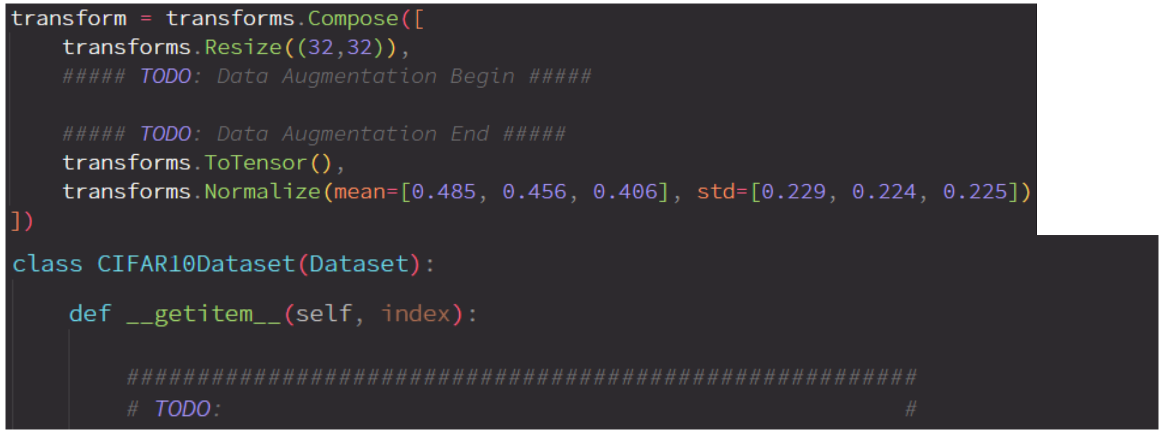
output cvs file format

Assignment Description

- p2/model.py ## TODO ##
  - Define your own models.
- p2/dataset.py ## TODO ##
  - Define your customized dataset.
- p2/config.py
  - Hyperparameters setting for training.
- p2/utils.py ## DO NOT MODIFY ##
  - Some helper functions.
- p2/p2_run_train.sh, p2_run_test.sh
  - example for code execution.



```
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()

        ################################################################
        # TODO:                                                        #
        # Define your CNN model architecture. Note that the first      #
        # input channel is 3, and the output dimension is 10 (class).  #
        ################################################################

        pass

    def forward(self, x):

class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()

        ############################################
        # NOTE:                                    #
        # Pretrain weights on ResNet18 is allowed. #
        ############################################

        # (batch_size, 3, 32, 32)
        self.resnet = models.resnet18(pretrained=True)
        # (batch_size, 512)
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 10)
        # (batch_size, 10)
```

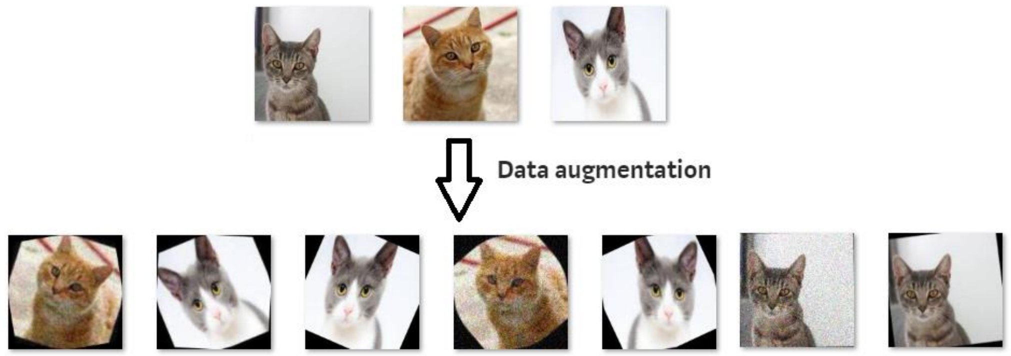


```
transform = transforms.Compose([
    transforms.Resize((32,32)),
    ##### TODO: Data Augmentation Begin #####

    ##### TODO: Data Augmentation End #####
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
class CIFAR10Dataset(Dataset):

    def __getitem__(self, index):

        ############################################################
        # TODO:                                                    #
```

Data Augmentation

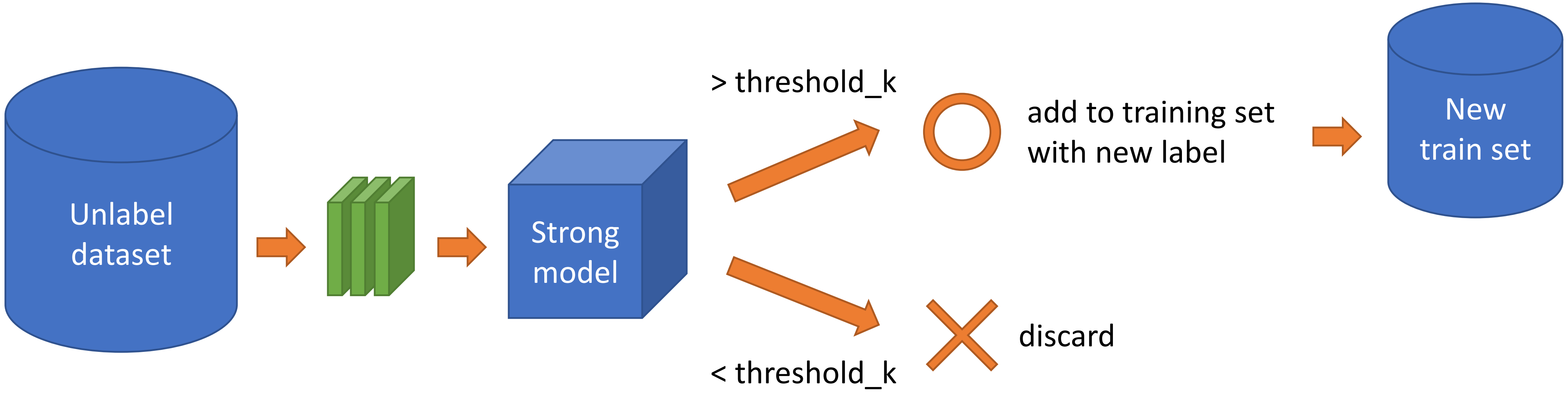
- CNNs are not rotationally invariant! We can apply a serial of data augmentation on training images for a robust model.

- You can simply apply a built-in function in pytorch
  - https://pytorch.org/vision/stable/transforms.html



Semi-supervised Learning (optional)

- Apply a well-trained model to verify if the images need to be preserved.



Execution

- TAs will execute your code in the following manner, each of them must be finished in 10 mins
- Part 1.
  - python3 p1.py $1 $2 $3
    - $1: type of feature representation (tiny_image/bag_of_sift)
    - $2: type of classifier (nearest_neighbor)
    - $3: dataset directory
  - E.g. python3 p1.py --feature tiny_image --classifier nearest_neighbor --dataset_dir ../hw2_data/p1_data/
- Part 2.
  - python3 p2_inference.py $1 $2 $3
    - $1: test dataset directory
    - $2: type of model (mynet, resnet18)
    - $3: path to output csvfile
  - E.g. python3 p2_inference.py --test_datadir ../hw2_data/p2_data/val --model_type resnet18 --output_path ./output/pred.csv

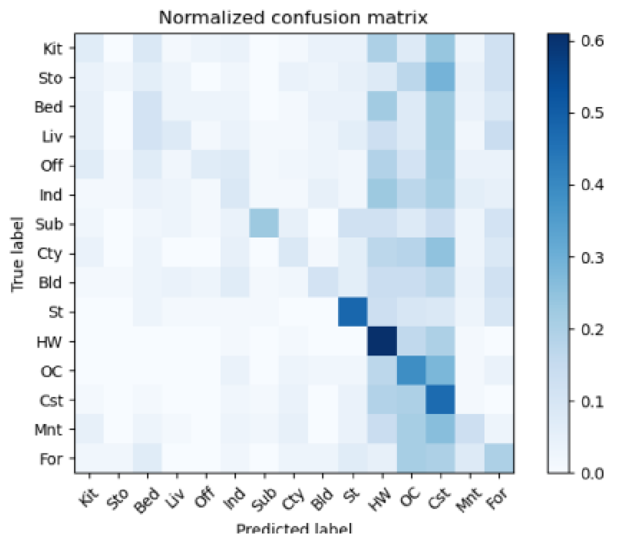
  - python3 p2_eval.py $1 $2   =>   this should reproduce the accuracy in your report.
    - $1: path to output csvfile
    - $2: path to ground truth annotations
  - E.g. python3 p2_eval.py --csv_path ./output/pred.csv --annos_path ../hw2_data/p2_data/val/annotations.json

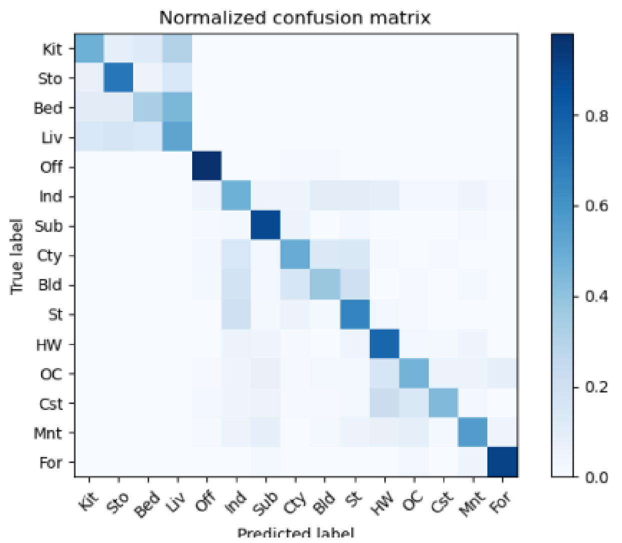
Grading

- Part 1 : 25%
  - (10%) Tiny image + kNN baseline : 0.2
  - (15%) Bag of SIFT + kNN baseline :
    - (9%) Weak baseline : 0.55
    - (6%) Strong baseline : 0.6
  - Accuracy should be above the baseline score to get points.
- Part 2 : 40%
  - (7.5%, 7.5%) Simple baseline : 0.70, for both public and private sets.
  - (7.5%, 7.5%) Medium baseline : 0.78, for both public and private sets.
  - (5%, 5%) Strong baseline : 0.84, for both public and private sets.
  - The best result of your two models (A and B) will be used here.
- Report : 35%

Report (35%)

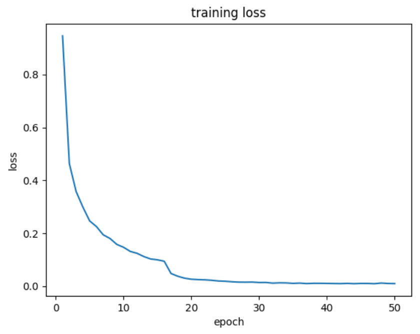
- Follow the template we provide and submit pdf file
- Part 1 : BoW Scene Recognition (10%)
  - (5%) Plot confusion matrix of two settings.
  - (5%) Compare the results / accuracy of both settings and explain the result.

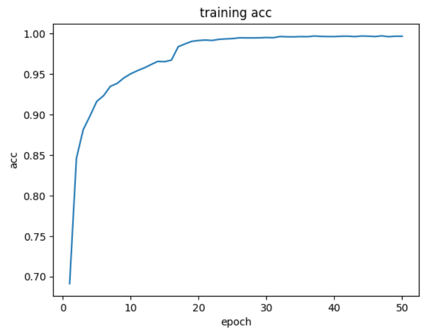


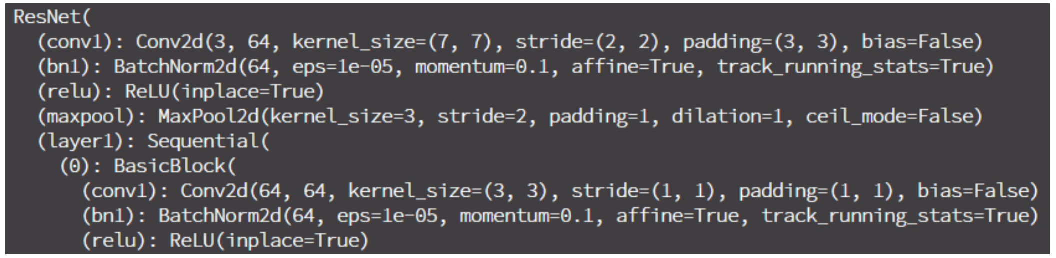


Report (35%)

- Part 2 : CNN Image Classification (25%)
  - (2%) Report accuracy of both models (both A & B) on the validation (public) set.
  - (5%) Print the network architectures & number of parameters of both models. What is the main difference between ResNet and other CNN architectures?
  - (8%) Plot four learning curves (loss & accuracy) of the training process (train/validation) for both models. Total 8 plots.
  - (10%) Briefly describe what method do you apply on your best model? (e.g. data augmentation, model architecture, loss function, etc)



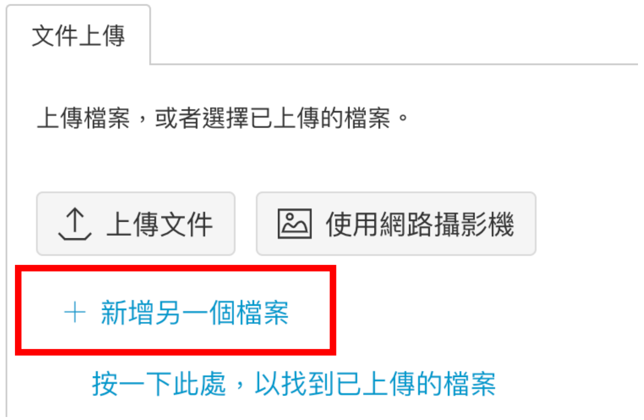




```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
```

Submission

- Deadline: 2023/04/13 (Thu.) 23:59
- R01234567_hw2/
  - p1/
    - p1.py, utils.py
    - vocab.pkl, test_image_feats.pkl, train_image_feats.pkl
  - p2/
    - p2_train.py, p2_inference.py
    - config.py, model.py, dataset.py
    - download.sh (optional)
    - checkpoint/
      - mynet_best.pth, resnet18_best.pth



- Submit **report.pdf** & **StudentID_hw2.zip** (e.g. R01234567_hw2.zip) to NTU COOL
  - IMPORTANT: There must be a directory named StudentID_hw2/ after unzipped

Submission

- DO NOT upload the dataset
- Late policy (refer to hw1)
- Do not delete your trained model before the TAs disclose your homework score and before you make sure that your score is correct.
- Plagiarism is forbidden!

Submission

- For part2,
  - For each model, the size should be less than 80MB.
  - If you cannot match the validation accuracy in your report…
    - TA will send you an e-mail to run another chance only for one time.
  - If your models are too large to submit on NTU COOL, **provide download.sh** under p2/ to download them from cloud drive (Dropbox, Google Drive…) to checkpoint/ (use **-O** argument)
    - TA will run \`bash download.sh\` first if this script exists.
    - You can use **wget, gdown, unzip**, etc. command.
    - Remember to set permission to **public**.

Packages

- python: 3.6.13+
- numpy: 1.19.2+
- cyvlfeat: 0.5.1+
- matplotlib: 3.3.4+
- Pillow: 8.4.0+
- scipy: 1.5.4+
- opencv-python: 4.7.0.72+
- sklearn: 0.24.2+
- torch: 1.10.1+
- torchvision: 0.11.2+
- tdqm: 4.64.1+
- gdown: 4.6.4+
- other standard python packages
- If you want to use any other packages, please email TA first.

If You Still Have Problems…

1. Use NTU COOL (strongly recommended)
   - TA will answer the questions ASAP and record some common problems you may face.
   - https://cool.ntu.edu.tw/courses/26914/discussion_topics/189119

2. E-mail to TA or use office hours
3. Goooooogle yourself

TA Information

- Yung-Wei Fan (范詠為)
  - E-mail: ywfan@media.ee.ntu.edu.tw
  - TA hour: Tue. 15:30-17:30
  - Location: 博理 421

- Yu-Kai Chen (陳昱愷)
  - E-mail: chenyukai@media.ee.ntu.edu.tw
  - Location: 博理 421

Supplementary

Tips for Achieving the Baselines

- In Part 1.
  - Consider different metrics to evaluate the distance between features.
  - Ref: scipy.spatial.distance.cdist -- SciPy Manual
- In Part 2.
  - In addition to data augmentation and semi-supervised techniques, you can consider modifying ResNet18 architecture since the first convolution layer's kernel size of ResNet18 is 7x7 which is relatively large to 32x32 images and may loss some information.

TA’s Experiment for Part 2. (ResNet18)

- Equipment & Time
  - CPU only: Intel(R) Core(TM) i5-12500 @ 3.00 GHz + 16GB RAM => ~ 4 mins / epoch
  - With GPU: NVIDIA GeForce GTX 1080Ti => ~ 2 mins / epoch
- Training experiments
  - pretrained weights: 77.8%
  - pretrained weights + DA: 79.9%
  - pretrained weights + DA + modify architecture: 91.8%
  - pretrained weights + DA + modify architecture + semi-supervised: 93.1%
    - took longer time since having more training data