Data loading

Import

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packages (1.5.10)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.6/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.6/dist-packages (from kaggle) (4.0.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist-packages (from kaggle) (2.8.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from kaggle) (4.41.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-packages (from kaggle) (2020.12.5)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->kaggle) (3.0
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->kaggle) (2.10)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.6/dist-packages (from python-slugify->kagg
```

```
pip install --upgrade kaggle
```

```
Requirement already up-to-date: kaggle in /usr/local/lib/python3.6/dist-packages (1.5.10)
Requirement already satisfied, skipping upgrade: certifi in /usr/local/lib/python3.6/dist-packages (from kaggle) (2020.
Requirement already satisfied, skipping upgrade: six>=1.10 in /usr/local/lib/python3.6/dist-packages (from kaggle) (1.1
Requirement already satisfied, skipping upgrade: tqdm in /usr/local/lib/python3.6/dist-packages (from kaggle) (4.41.1)
Requirement already satisfied, skipping upgrade: python-slugify in /usr/local/lib/python3.6/dist-packages (from kaggle)
Requirement already satisfied, skipping upgrade: requests in /usr/local/lib/python3.6/dist-packages (from kaggle) (2.23
Requirement already satisfied, skipping upgrade: python-dateutil in /usr/local/lib/python3.6/dist-packages (from kaggle
Requirement already satisfied, skipping upgrade: urllib3 in /usr/local/lib/python3.6/dist-packages (from kaggle) (1.24.
Requirement already satisfied, skipping upgrade: text-unidecode>=1.3 in /usr/local/lib/python3.6/dist-packages (from py
Requirement already satisfied, skipping upgrade: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests-
Requirement already satisfied, skipping upgrade: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requ
```

```
!mkdir .kaggle
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/My Drive/Study/Kaggle"
```

```
%cd /content/gdrive/My Drive/Study/DL/Assignments/Project
```

```
/content/gdrive/My Drive/Study/DL/Assignments/Project
```

```
!kaggle competitions download -c final-project-dl-spring-2020
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.5.10 / client 1.5.4)
404 - Not Found
```

```
!ls
```

```
 alexnet_pretrained.pt              preds_4.1.csv
 final-project-dl-spring-2020.zip   preds_4.1.gsheet
 model_5.pt                         preds_4.2.csv
 model_6.pt                         preds_4.3.csv
 model_7.pt                         preds_5.csv
 model_8.pt                         preds_6.csv
```

```
 pred_5.csv                        preds_7.csv
 preds_1.2.csv                     preds_7.gsheet
 preds_1.2.gsheet                  preds_8.csv
 preds_1.3.csv                     preds_8.gsheet
 preds_1.4.csv                     Project.docx
 preds_1.6.csv                     resnet18_pretrained.pt
 preds_1.6.gsheet                  resnet34_pretrained.pt
'preds_1.7 (1).gsheet'             resnet50_pretrained.pt
 preds_1.7.csv                    '~$roject.docx'
 preds_1.7.gsheet                  sample_submission.csv
 preds_1.csv                       submission.csv
 preds_1.gsheet                    test_data
 preds_3.1.csv                     train_data
 preds_3.1.gsheet
```

```
#unzipping the zip files and deleting the zip files
#!unzip \*.zip  && rm *.zip
```

```
import torch
import torch.nn as nn

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from datetime import datetime
from pathlib import Path
import pandas as pd
import torch.nn.functional as F
import torchtext.data as ttd
import torchvision
from torchvision import datasets, transforms,utils


from torch.optim.lr_scheduler import StepLR, CyclicLR
```

```
folder=Path('/content/gdrive/My Drive/Study/DL/Assignments/Project')
```

Data transformation

```
transformer = transforms.Compose([
#    transforms.Resize(256),
#    transforms.RandomCrop(256),
#    transforms.RandomHorizontalFlip(),
#    transforms.RandomVerticalFlip(),
    transforms.ToTensor(),
 #   transforms.Normalize([meanR, meanG, meanB], [stdR, stdG, stdB]),
     ])
```

```
train0_ds = datasets.ImageFolder(folder/'train_data/train',transform=transformer)
test_ds = datasets.ImageFolder(folder/'test_data',transform=transformer)
```

```
train0_ds.classes
```

```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

Train/val split

```
from sklearn.model_selection import StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n_splits=1, train_size=0.7, random_state=0)
```

```
indices=list(range(len(train0_ds)))
y_train0=[y for _,y in train0_ds]
for train_index, val_index in sss.split(indices, y_train0):
    print("train:", train_index, "val:", val_index)
    print(len(val_index),len(train_index))
```

```
 train: [5279 5574 7391 ... 8774 2047 3129] val: [1900 5887 8619 ... 5449 1106 6628]
 2708 6317
```

```
from torch.utils.data import Subset

val_ds=Subset(train0_ds,val_index)
train_ds=Subset(train0_ds,train_index)
```

```
print(f'Number of training examples: {len(train_ds)}')
print(f'Number of validation examples: {len(val_ds)}')
print(f'Number of testing examples: {len(test_ds)}')
```

```
 Number of training examples: 6317
 Number of validation examples: 2708
 Number of testing examples: 3929
```

```
import numpy as np

# RGB mean and std
meanRGB=[np.mean(x.numpy(),axis=(1,2)) for x,_ in train_ds]
stdRGB=[np.std(x.numpy(),axis=(1,2)) for x,_ in train_ds]

meanR=np.mean([m[0] for m in meanRGB])
meanG=np.mean([m[1] for m in meanRGB])
meanB=np.mean([m[2] for m in meanRGB])

stdR=np.mean([s[0] for s in stdRGB])
stdG=np.mean([s[1] for s in stdRGB])
stdB=np.mean([s[2] for s in stdRGB])

print(meanR,meanG,meanB)
print(stdR,stdG,stdB)
```

```
 0.48499483 0.45491496 0.39288142
 0.22034366 0.21472108 0.21670675
```

Preprocessing

```
#additional transformation changes if required

train_transformer_1 = transforms.Compose([
    transforms.Resize(512),
#    transforms.RandomHorizontalFlip(), # randomly flip and rotate
    transforms.RandomCrop(512),
#    transforms.RandomHorizontalFlip(),
#    transforms.RandomVerticalFlip(),
    transforms.ToTensor(),
#    transforms.Normalize([meanR, meanG, meanB], [stdR, stdG, stdB]),
#    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

     ])


test_transformer_1 = transforms.Compose([
    transforms.Resize(512),
    transforms.RandomCrop(512),
    transforms.ToTensor(),
#    transforms.Normalize([meanR, meanG, meanB], [stdR, stdG, stdB]),
```

```
#    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

    ])

train0_ds.transform=train_transformer_1
test_ds.transform=test_transformer_1
```

Data loader

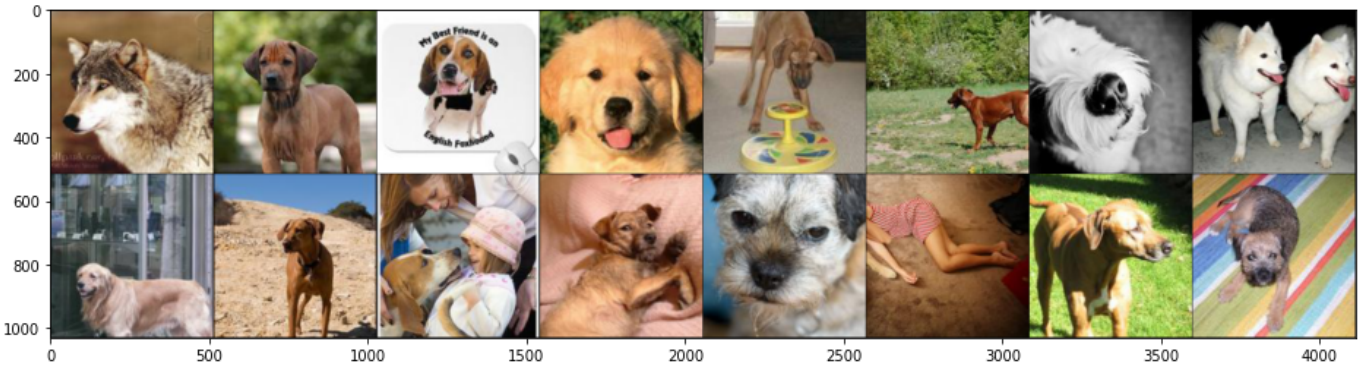
```
from torch.utils.data import DataLoader

train_dl = DataLoader(train_ds, batch_size = 32, shuffle=True, num_workers=4)
val_dl = DataLoader(val_ds, batch_size = 32, shuffle=True, num_workers=4)
test_dl = DataLoader(test_ds, batch_size = 32, shuffle=False, num_workers=4)
```

Sample check

```
samples, labels = iter(train_dl).next()
plt.figure(figsize=(16,24))
grid_imgs = torchvision.utils.make_grid(samples[:24])
np_grid_imgs = grid_imgs.numpy()
# in tensor, image is (batch, width, height), so you have to transpose it to (width, height, batch) in numpy to show it.
plt.imshow(np.transpose(np_grid_imgs, (1,2,0)))
```

```
<matplotlib.image.AxesImage at 0x7fade135e978>
```




```
# extract a batch from training data
for x, y in train_dl:
    print(x.shape)
    print(y.shape)
    break
# extract a batch from val data
for x, y in val_dl:
    print(x.shape)
    print(y.shape)
    break
```

```
torch.Size([32, 3, 512, 512])
torch.Size([32])
torch.Size([32, 3, 512, 512])
torch.Size([32])
```

```
import collections

# get labels
y_train=[y for _,y in train_ds]

# count labels
```

```
counter_train=collections.Counter(y_train)
print(counter_train)
```

```
Counter({9: 664, 7: 664, 6: 660, 8: 659, 0: 659, 5: 658, 3: 652, 1: 650, 2: 645, 4: 406})
```

Predefined functions

```
def train_val(model, params):
    # extract model parameters
    num_epochs=params["num_epochs"]
    loss_func=params["loss_func"]
    opt=params["optimizer"]
    train_dl=params["train_dl"]
    val_dl=params["val_dl"]
    sanity_check=params["sanity_check"]
    lr_scheduler=params["lr_scheduler"]
    path2weights=params["path2weights"]

    # history of loss values in each epoch
    loss_history={
        "train": [],
        "val": [],
    }

    # histroy of metric values in each epoch
    metric_history={
        "train": [],
        "val": [],
    }

    # a deep copy of weights for the best performing model
    best_model_wts = copy.deepcopy(model.state_dict())

    # initialize best loss to a large value
    best_loss=float('inf')

    # main loop
    for epoch in range(num_epochs):

        # get current learning rate
        current_lr=get_lr(opt)
        print('Epoch {}/{}, current lr={}'.format(epoch, num_epochs - 1, current_lr))

        # train model on training dataset
        model.train()
        train_loss, train_metric=loss_epoch(model,loss_func,train_dl,sanity_check,opt)

        # collect loss and metric for training dataset
        loss_history["train"].append(train_loss)
        metric_history["train"].append(train_metric)

        # evaluate model on validation dataset
        model.eval()
        with torch.no_grad():
            val_loss, val_metric=loss_epoch(model,loss_func,val_dl,sanity_check)


        # store best model
        if val_loss < best_loss:
            best_loss = val_loss
            best_model_wts = copy.deepcopy(model.state_dict())

            # store weights into a local file
            torch.save(model.state_dict(), path2weights)
            print("Copied best model weights!")
```

```
        # collect loss and metric for validation dataset
        loss_history["val"].append(val_loss)
        metric_history["val"].append(val_metric)

        # learning rate schedule
        lr_scheduler.step()

        print("train loss: %.6f, dev loss: %.6f, accuracy: %.2f" %(train_loss,val_loss,100*val_metric))
        print("-"*10)

    # load best model weights
    model.load_state_dict(best_model_wts)

    return model, loss_history, metric_history
```

```
# a helper function to compute the loss value and the performance metric for the entire dataset or an epoch.

# define device as a global variable
#device = torch.device("cuda")

def loss_epoch(model,loss_func,dataset_dl,sanity_check=False,opt=None):
    running_loss=0.0
    running_metric=0.0
    len_data=len(dataset_dl.dataset)

    for xb, yb in dataset_dl:
        # move batch to device
        xb=xb.to(device)
        yb=yb.to(device)

        # get model output
        output=model(xb)

        # get loss per batch
        loss_b,metric_b=loss_batch(loss_func, output, yb, opt)

        # update running loss
        running_loss+=loss_b

        # update running metric
        if metric_b is not None:
            running_metric+=metric_b

        # break the loop in case of sanity check
        if sanity_check is True:
            break

    # average loss value
    loss=running_loss/float(len_data)

    # average metric value
    metric=running_metric/float(len_data)

    return loss, metric
```

```
# A helper function to compute the loss value per batch of data:
def loss_batch(loss_func, output, target, opt=None):

    # get loss
    loss = loss_func(output, target)

    # get performance metric
    metric_b = metrics_batch(output,target)

    if opt is not None:
        opt.zero_grad()
        loss.backward()
```

```
        opt.step()

    return loss.item(), metric_b
```

```
# A helper function to count the number of correct predictions per data batch:
def metrics_batch(output, target):
    # get output class
    pred = output.argmax(dim=1, keepdim=True)

    # compare output class with target class
    corrects=pred.eq(target.view_as(pred)).sum().item()
    return corrects
```

Model_7

Densenet 161

Densenet 161

```
from torchvision import models
import torch

# load model with random weights
model_7= models.densenet161(pretrained=True)
```

```
print(model_7)
```

```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 96, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (norm0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (denseblock1): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(96, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(144, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(192, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(240, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer5): _DenseLayer(
```

```
      (norm1): BatchNorm2d(288, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): ReLU(inplace=True)
      (conv1): Conv2d(288, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu2): ReLU(inplace=True)
      (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (denselayer6): _DenseLayer(
      (norm1): BatchNorm2d(336, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): ReLU(inplace=True)
      (conv1): Conv2d(336, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu2): ReLU(inplace=True)
      (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
  )
  (transition1): _Transition(
    (norm): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
```

```
# upload model to GPU
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda:0
```

```
from torch import nn
# change the output layer
num_classes=10

model_7.classifier.out_features = num_classes

#device = torch.device("cuda:0")
model_7.to(device)
```

```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 96, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (norm0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (denseblock1): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(96, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(144, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(192, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(240, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
```

```
    (denselayer5): _DenseLayer(
      (norm1): BatchNorm2d(288, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): ReLU(inplace=True)
      (conv1): Conv2d(288, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu2): ReLU(inplace=True)
      (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (denselayer6): _DenseLayer(
      (norm1): BatchNorm2d(336, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu1): ReLU(inplace=True)
      (conv1): Conv2d(336, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu2): ReLU(inplace=True)
      (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
  )
  (transition1): _Transition(
    (norm): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
```

Training

```
from torch import optim
opt = optim.Adam(model_7.parameters(), lr=1e-4)
```

```
# get learning rate
def get_lr(opt):
    for param_group in opt.param_groups:
        return param_group['lr']

current_lr=get_lr(opt)
print('current lr={}'.format(current_lr))
```

```
current lr=0.0001
```

```
from torch.optim.lr_scheduler import CosineAnnealingLR, StepLR

# define learning rate scheduler
# Several methods exist to adjust the learning rate. For a list of supported methods by PyTorch,
# please visit the following link: https://pytorch.org/docs/stable/optim.html.

#lr_scheduler = CosineAnnealingLR(opt,T_max=2,eta_min=1e-5)
lr_scheduler = StepLR(opt,step_size=1)
```

```
lrs=[]
for i in range(5):
    lr_scheduler.step()
    lr=get_lr(opt)
    print("epoch %s, lr: %.1e" %(i,lr))
    lrs.append(lr)
```

```
epoch 0, lr: 1.0e-05
epoch 1, lr: 1.0e-06
epoch 2, lr: 1.0e-07
epoch 3, lr: 1.0e-08
epoch 4, lr: 1.0e-09
/usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:136: UserWarning: Detected call of `lr_scheduler.ste
  "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarning)
```

```
(counter_train)
```

```
Counter({0: 753,
         1: 742,
         2: 737,
         3: 746,
```

```
             4: 464,
             5: 752,
             6: 754,
             7: 759,
             8: 754,
             9: 759})
```

```
from sortedcontainers import SortedDict

total_sum = sum(counter_train.values())

weights = SortedDict(dict(counter_train))
weights = np.array(list(weights.values()))
weights = np.around(weights/total_sum,decimals=2)
weights = (torch.tensor(weights)).float().to(device)
weights
```

```
tensor([0.1000, 0.1000, 0.1000, 0.1000, 0.0600, 0.1000, 0.1000, 0.1100, 0.1000,
        0.1100], device='cuda:0')
```

```
import copy

loss_func = nn.CrossEntropyLoss(reduction="sum")

opt = optim.Adam(model_7.parameters(), lr=1e-4)
lr_scheduler = CosineAnnealingLR(opt,T_max=5,eta_min=1e-6)
#lr_scheduler = StepLR(opt,step_size=1)

params_train={
 "num_epochs": 5,
 "optimizer": opt,
 "loss_func": loss_func,
 "train_dl": train_dl,
 "val_dl": val_dl,
 "sanity_check": False,
 "lr_scheduler": lr_scheduler,
 "path2weights": folder / "model_7.pt",
}

# train and validate the model
model_7,loss_hist,metric_hist=train_val(model_7,params_train)
```

```
Epoch 0/4, current lr=0.0001
Copied best model weights!
train loss: 0.757316, dev loss: 0.309049, accuracy: 89.92
----------
Epoch 1/4, current lr=9.05463412215599e-05
Copied best model weights!
train loss: 0.217857, dev loss: 0.261615, accuracy: 91.19
----------
Epoch 2/4, current lr=6.57963412215599e-05
Copied best model weights!
train loss: 0.092043, dev loss: 0.246794, accuracy: 93.91
----------
Epoch 3/4, current lr=3.52036587784401e-05
Copied best model weights!
train loss: 0.041339, dev loss: 0.224900, accuracy: 93.68
----------
Epoch 4/4, current lr=1.0453658778440105e-05
Copied best model weights!
train loss: 0.020577, dev loss: 0.196145, accuracy: 94.85
----------
```

Prediction

```
from torch import nn
from torchvision import models
import torch
```

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda:0
```

```
# load model
model_7 = models.densenet161(pretrained=True)

from torch import nn
# change the output layer
num_classes=10

model_7.classifier.out_features = num_classes

#device = torch.device("cuda:0")
model_7.to(device)
```

```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 96, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (norm0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (denseblock1): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(96, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(144, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(192, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(240, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(288, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(288, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(336, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(336, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
```

```
        )
        (transition1): _Transition(
          (norm): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
```

```
import torch

# load state_dict into model
path2weights=folder / "model_7.pt"
model_7.load_state_dict(torch.load(path2weights))
```

```
    <All keys matched successfully>
```

```
test_dl.dataset.samples[0][0][78:]
```

```
    '0.JPEG'
```

```
model_7.eval()
model_7.to(device)
fn_list = []
pred_list = []
for i, (x,fn) in enumerate(test_dl, 0):
    with torch.no_grad():
        x = x.to(device)
        output = model_7(x)
        pred = torch.argmax(output, dim=1)
        pred_list += [p.item() for p in pred]

for m,n in enumerate(test_dl.dataset.samples,0):
        fn_list += [n[0][78:]]

submission = pd.DataFrame({"file_names":fn_list, "target":pred_list})
submission.to_csv('preds_7.csv', index=False)
```

```

```