TypeB_AntsBees_VGG16_PyTorch_Training_for_loop_dataloader_05

January 5, 2022

1 Ants Bees VGG16

2 Fitting

3 TypeBTraining

[ ]:

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

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

[ ]:

```
'''
3.
'''
import numpy as np
from PIL import Image
from torchvision import transforms
import matplotlib.pyplot as plt
import os
import torch
import torch.utils.data as data
from pathlib import Path
```

[ ]:

```
torch.cuda.synchronize()
print(torch.cuda.memory_allocated())
```

```
0
```

[ ]:

```
data_dir = '/content/gdrive/My Drive/Colab Notebooks/AntsBees/data'
root_dir= '/content/gdrive/My Drive/Colab Notebooks/AntsBees/data/
 ↪hymenoptera_data'
```

[ ]:

```
os.chdir('/content/gdrive/My Drive/Colab Notebooks/AntsBees')
```

3.0.1 Device

[ ]:
```
# (CPUGPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

4 (1) DataSet

4.0.1 train_list, val_list

[ ]:
```
import glob
import pprint

def make_datapath_list(phase='train'):
    '''



    Parameters:
      phase(str): 'train''val'

    Returns:
      path_list(list):
    '''
    #
    #
    # rootpath +
    #   train/ants/*.jpg
    #   train/bees/*.jpg
    #   val/ants/*.jpg
    #   val/bees/*.jpg
    tt= phase +'/**/*.jpg'
    target_path = os.path.join(root_dir,tt)
    #
    path_list = []   #

    # glob()
    for path in glob.glob(target_path):
        path_list.append(path)

    return path_list

#
train_list = make_datapath_list(phase='train')
val_list = make_datapath_list(phase='val')
```

```
print(train_list)
```

[ ]:

```
p=Path(train_list[5])
print(p.parts[-2])

p2=Path(val_list[-4])
print(p2.parts[-2])
```

```
ants
bees
```

5 DataSet

[ ]:

```
from dsets.dsets import MakeDataset
from util.ImageTransform import ImageTransform
```

[ ]:

```
'''
6.
'''
import torch

#
batch_size = 10
SIZE = 224
# RGB
MEAN = (0.485, 0.456, 0.406) # ImageNet
# RGB
STD = (0.229, 0.224, 0.225)  # ImageNet


#
size, mean, std = SIZE, MEAN, STD

# MakeDataset
train_dataset = MakeDataset(
    file_list=train_list, #
    transform=ImageTransform(size, mean, std), #
    phase='train')
# MakeDataset
val_dataset = MakeDataset(
    file_list=val_list, #
    transform=ImageTransform(size, mean, std), #
    phase='val')
```

[ ]:

```
print(len(train_list))
print(len(val_list))
```

```
243
153
```

6 (2) DataLoader

[ ]:

```
'''
6.
'''
from torch.utils.data import DataLoader

# :(, 3, 224, 224)
train_dl = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# :(, 3, 224, 224)
val_dl = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```

7 (4)

[ ]:

```
'''
7. VGG16
'''
from torchvision import models
import torch.nn as nn

# ImageNetVGG16
model = models.vgg16(pretrained=True)

# VGG162
model.classifier[6] = nn.Linear(
    in_features=4096, # 4096
    out_features=2)   # 10002


model = model.to(device)
print(model)
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to
/root/.cache/torch/hub/checkpoints/vgg16-397923af.pth

  0%|          | 0.00/528M [00:00<?, ?B/s]


VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

8 For Loop with DataLoader

8.0.1 enumerate(tain_dl)

[ ]:

```
import sys
for batch_ndx, batch_tup in enumerate(train_dl):
  input_t, label_t = batch_tup
  print(batch_ndx,input_t.shape)
  var_name = 'input_t'
  print(sys.getsizeof(eval(var_name)))

print(sys.getsizeof(eval(var_name)))
```

```
0 torch.Size([10, 3, 224, 224])
88
1 torch.Size([10, 3, 224, 224])
88
2 torch.Size([10, 3, 224, 224])
88
3 torch.Size([10, 3, 224, 224])
88
4 torch.Size([10, 3, 224, 224])
88
5 torch.Size([10, 3, 224, 224])
88
6 torch.Size([10, 3, 224, 224])
88
7 torch.Size([10, 3, 224, 224])
88
8 torch.Size([10, 3, 224, 224])
88
9 torch.Size([10, 3, 224, 224])
88
10 torch.Size([10, 3, 224, 224])
88
11 torch.Size([10, 3, 224, 224])
88
12 torch.Size([10, 3, 224, 224])
88
13 torch.Size([10, 3, 224, 224])
88
14 torch.Size([10, 3, 224, 224])
88
15 torch.Size([10, 3, 224, 224])
88
16 torch.Size([10, 3, 224, 224])
88
17 torch.Size([10, 3, 224, 224])
```

```
88
18 torch.Size([10, 3, 224, 224])
88
19 torch.Size([10, 3, 224, 224])
88
20 torch.Size([10, 3, 224, 224])
88
21 torch.Size([10, 3, 224, 224])
88
22 torch.Size([10, 3, 224, 224])
88
23 torch.Size([10, 3, 224, 224])
88
24 torch.Size([3, 3, 224, 224])
88
88
```

[ ]:

```
metrics_g= torch.zeros(4,len(train_dl.dataset),device=device)
var_name = 'metrics_g'
print(sys.getsizeof(eval(var_name)))
```

```
88
```

[ ]:

```
print('Allocated:', round(torch.cuda.memory_allocated(0)/1024**3,1), 'GB')
print('Cached:   ', round(torch.cuda.memory_reserved(0)/1024**3,1), 'GB')
```

```
Allocated: 0.5 GB
Cached:    0.5 GB
```

9 epochmetrics_g

[ ]:

```
metrics_g= torch.zeros(3,len(train_dl.dataset),device=device)

import sys

for batch_ndx, batch_tup in enumerate(train_dl):

  input_t, label_t = batch_tup

  input_g = input_t.to(device)
  label_g = label_t.to(device)
  model=model.to(device)

  outputs  = model(input_g)

  softmax = nn.Softmax(dim=1)
```

```
prob_g= softmax(outputs)
pred_label_g= outputs.argmax(dim=-1)
#print(prob_g[:,1])

loss_func = nn.CrossEntropyLoss(reduction='none')
loss_g = loss_func(outputs,label_g)

print('Allocated:', round(torch.cuda.memory_allocated(0)/1024**3,1), 'GB')
print('Cached:   ', round(torch.cuda.memory_reserved(0)/1024**3,1), 'GB')

start_ndx = batch_ndx * batch_size
end_ndx = start_ndx + label_g.size()[0]
print('start_ndx,end_ndx',start_ndx,end_ndx,label_g.shape[0],prob_g.
↪shape[0],loss_g.shape[0])

metrics_g[0, start_ndx:end_ndx] = label_g
metrics_g[1, start_ndx:end_ndx] = prob_g[:,1].detach()
metrics_g[2, start_ndx:end_ndx] = loss_g.detach()

var_name = 'metrics_g'
print(sys.getsizeof(eval(var_name)))
```

```
Allocated: 1.2 GB
Cached:    1.5 GB
start_ndx,end_ndx 0 10 10 10 10
88
Allocated: 1.2 GB
Cached:    2.2 GB
start_ndx,end_ndx 10 20 10 10 10
88
Allocated: 1.9 GB
Cached:    2.2 GB
start_ndx,end_ndx 20 30 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 30 40 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 40 50 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 50 60 10 10 10
88
Allocated: 1.9 GB
```

```
Cached:    2.9 GB
start_ndx,end_ndx 60 70 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 70 80 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 80 90 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 90 100 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 100 110 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 110 120 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 120 130 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 130 140 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 140 150 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 150 160 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 160 170 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 170 180 10 10 10
88
Allocated: 1.9 GB
```

```
Cached:    2.9 GB
start_ndx,end_ndx 180 190 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 190 200 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 200 210 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 210 220 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 220 230 10 10 10
88
Allocated: 1.9 GB
Cached:    2.9 GB
start_ndx,end_ndx 230 240 10 10 10
88
Allocated: 1.4 GB
Cached:    2.9 GB
start_ndx,end_ndx 240 243 3 3 3
88
```

[ ]:

```
print(metrics_g.shape)
print(metrics_g.dtype)
```

```
torch.Size([3, 243])
```

10 metrics_gepoch

[ ]:

```
print(metrics_t.device)
print(metrics_g.device)
```

```
cuda:0
cuda:0
```

[ ]:

```
metrics_t = metrics_g

negLabel_mask = metrics_t[0] <= 0.5
negPred_mask = metrics_t[1] <= 0.5
```

[ ]:
```python
metrics_cpu=metrics_g.to('cpu')
print(metrics_cpu.device)

negLabel_mask_cpu = metrics_cpu[0] <= 0.5
negPred_mask_cpu = metrics_cpu[1] <= 0.5

print(negLabel_mask_cpu)
```

```
cpu
tensor([ True, False,  True, False, False,  True,  True,  True, False, False,
        False,  True,  True,  True, False, False,  True,  True, False,  True,
        False,  True, False, False,  True,  True,  True, False, False, False,
         True, False, False, False,  True,  True,  True,  True, False, False,
        False,  True, False,  True, False, False,  True,  True,  True,  True,
         True, False,  True, False, False,  True, False, False, False,  True,
        False,  True, False, False, False,  True, False,  True, False, False,
         True,  True, False,  True,  True, False,  True,  True, False,  True,
        False,  True,  True,  True, False, False, False, False,  True,  True,
        False, False,  True,  True,  True,  True, False,  True, False, False,
        False, False, False, False, False,  True, False, False,  True,  True,
         True, False,  True, False,  True, False, False, False,  True,  True,
        False, False,  True,  True, False, False,  True,  True, False, False,
         True, False,  True, False,  True, False, False, False, False,  True,
        False, False,  True, False, False,  True,  True,  True,  True,  True,
         True,  True, False, False, False,  True, False,  True,  True, False,
         True,  True,  True, False,  True,  True,  True,  True,  True,  True,
         True, False,  True, False, False, False, False,  True,  True,  True,
        False,  True, False,  True, False, False, False,  True,  True, False,
         True, False, False,  True, False,  True, False,  True, False,  True,
        False,  True, False, False, False,  True, False,  True, False,  True,
         True,  True,  True,  True, False, False, False,  True, False,  True,
        False, False,  True, False,  True,  True, False, False,  True,  True,
         True, False, False,  True, False,  True,  True, False,  True, False,
         True, False,  True])
```

[ ]:
```python
print(negLabel_mask)
print(negPred_mask)
print(negLabel_mask.shape)
print(negPred_mask.shape)
```

```
tensor([ True, False,  True, False, False,  True,  True,  True, False, False,
        False,  True,  True,  True, False, False,  True,  True, False,  True,
        False,  True, False, False,  True,  True,  True, False, False, False,
         True, False, False, False,  True,  True,  True,  True, False, False,
        False,  True, False,  True, False, False,  True,  True,  True,  True,
         True, False,  True, False, False,  True, False, False, False,  True,
        False,  True, False, False, False,  True, False,  True, False, False,
         True,  True, False,  True,  True, False,  True,  True, False,  True,
        False,  True,  True,  True, False, False, False, False,  True,  True,
        False, False,  True,  True,  True,  True, False,  True, False, False,
        False, False, False, False, False,  True, False, False,  True,  True,
         True, False,  True, False,  True, False, False, False,  True,  True,
        False, False,  True,  True, False, False,  True,  True, False, False,
         True, False,  True, False,  True, False, False, False, False,  True,
        False, False,  True, False, False,  True,  True,  True,  True,  True,
         True,  True, False, False, False,  True, False,  True,  True, False,
         True,  True,  True, False,  True,  True,  True,  True,  True,  True,
         True, False,  True, False, False, False, False,  True,  True,  True,
        False,  True, False,  True, False, False, False,  True,  True, False,
         True, False, False,  True, False,  True, False,  True, False,  True,
        False,  True, False, False, False,  True, False,  True, False,  True,
         True,  True,  True,  True, False, False, False,  True, False,  True,
        False, False,  True, False,  True,  True, False, False,  True,  True,
         True, False, False,  True, False,  True,  True, False,  True, False,
         True, False,  True], device='cuda:0')
tensor([ True, False, False,  True, False, False, False,  True,  True, False,
         True, False, False,  True, False, False, False, False,  True, False,
        False, False,  True,  True,  True, False, False,  True,  True,  True,
         True, False,  True, False, False, False,  True,  True,  True, False,
        False, False, False, False, False, False, False, False,  True, False,
        False,  True, False,  True,  True, False,  True,  True, False, False,
        False,  True, False, False,  True,  True, False,  True,  True, False,
        False, False,  True, False,  True, False, False, False, False, False,
        False,  True,  True, False, False,  True, False,  True, False, False,
        False,  True, False, False, False, False, False, False,  True, False,
        False,  True, False,  True,  True,  True,  True, False, False,  True,
        False, False,  True,  True, False,  True,  True, False, False,  True,
         True, False, False, False, False, False,  True, False, False,  True,
        False, False,  True,  True,  True,  True, False, False,  True,  True,
         True, False, False, False, False, False, False, False, False, False,
         True,  True, False, False,  True,  True, False, False,  True, False,
         True,  True,  True,  True,  True, False,  True,  True,  True, False,
         True, False,  True,  True, False, False, False, False, False,  True,
         True, False, False, False, False, False, False, False, False, False,
         True, False, False,  True, False,  True,  True,  True, False,  True,
        False, False,  True, False, False,  True, False,  True, False, False,
         True,  True, False,  True, False, False, False,  True,  True,  True,
         True,  True,  True, False, False,  True,  True,  True, False, False,
         True, False, False,  True,  True,  True, False, False, False, False,
         True, False,  True], device='cuda:0')
torch.Size([243])
torch.Size([243])
```

[ ]:

```
posLabel_mask = ~negLabel_mask
posPred_mask = ~negPred_mask

print(posLabel_mask)
print(posPred_mask)
```

```
tensor([False,  True, False,  True,  True, False, False, False,  True,  True,
         True, False, False, False,  True,  True, False, False,  True, False,
         True, False,  True,  True, False, False, False,  True,  True,  True,
        False,  True,  True,  True, False, False, False, False,  True,  True,
         True, False,  True, False,  True,  True, False, False, False, False,
        False,  True, False,  True,  True, False,  True,  True,  True, False,
         True, False,  True,  True,  True, False,  True, False,  True,  True,
        False, False,  True, False, False,  True, False, False,  True, False,
         True, False, False, False,  True,  True,  True,  True, False, False,
         True,  True, False, False, False, False,  True, False,  True,  True,
         True,  True,  True,  True,  True, False,  True,  True, False, False,
        False,  True, False,  True, False,  True,  True,  True, False, False,
         True,  True, False, False,  True,  True, False, False,  True,  True,
        False,  True, False,  True, False,  True,  True,  True,  True, False,
         True,  True, False,  True,  True, False, False, False, False, False,
        False, False,  True,  True,  True, False,  True, False, False,  True,
        False, False, False,  True, False, False, False, False, False, False,
        False,  True, False,  True,  True,  True,  True, False, False, False,
         True, False,  True, False,  True,  True,  True, False, False,  True,
        False,  True,  True, False,  True, False,  True, False,  True, False,
         True, False,  True,  True,  True, False,  True, False,  True, False,
        False, False, False, False,  True,  True,  True, False,  True, False,
         True,  True, False,  True, False, False,  True,  True, False, False,
        False,  True,  True, False,  True, False, False,  True, False,  True,
        False,  True, False], device='cuda:0')
tensor([False,  True,  True, False,  True,  True,  True, False, False,  True,
        False,  True,  True, False,  True,  True,  True,  True, False,  True,
         True,  True, False, False, False,  True,  True, False, False, False,
        False,  True, False,  True,  True,  True, False, False, False,  True,
         True,  True,  True,  True,  True,  True,  True,  True, False,  True,
         True, False,  True, False, False,  True, False, False,  True,  True,
         True, False,  True,  True, False, False,  True, False, False,  True,
         True,  True, False,  True, False,  True,  True,  True,  True,  True,
         True, False, False,  True,  True, False,  True, False,  True,  True,
         True, False,  True,  True,  True,  True,  True,  True, False,  True,
         True, False,  True, False, False, False, False,  True,  True, False,
         True,  True, False, False,  True, False, False,  True,  True, False,
        False,  True,  True,  True,  True,  True, False,  True,  True, False,
         True,  True, False, False, False, False,  True,  True, False, False,
        False,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        False, False,  True,  True, False, False,  True,  True, False,  True,
        False, False, False, False, False,  True, False, False, False,  True,
        False,  True, False, False,  True,  True,  True,  True,  True, False,
        False,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        False,  True,  True, False,  True, False, False, False,  True, False,
         True,  True, False,  True,  True, False,  True, False,  True,  True,
        False, False,  True, False,  True,  True,  True, False, False, False,
        False, False, False,  True,  True, False, False, False,  True,  True,
        False,  True,  True, False, False, False,  True,  True,  True,  True,
        False,  True, False], device='cuda:0')
```

[ ]:

```
neg_count = int(negLabel_mask.sum())
pos_count = int(posLabel_mask.sum())

print(neg_count)
print(pos_count)
```

```
122
121
```

[ ]:

```
trueNeg_count = neg_correct = int((negLabel_mask & negPred_mask).sum())
truePos_count = pos_correct = int((posLabel_mask & posPred_mask).sum())

#trueNeg_count = neg_correct
#truePos_count = pos_correct

print(trueNeg_count,neg_correct)
```

```
72 72
```

[ ]:

```
falsePos_count = neg_count - neg_correct
falseNeg_count = pos_count - pos_correct

metrics_dict = {}
metrics_dict['loss/all'] = metrics_t[2].mean()
#Beesloss
metrics_dict['loss/neg'] = metrics_t[2, negLabel_mask].mean()
#Antsloss
metrics_dict['loss/pos'] = metrics_t[2, posLabel_mask].mean()

#accuracy
metrics_dict['correct/all'] = (pos_correct + neg_correct) / metrics_t.shape[1]␣
 ↪* 100
metrics_dict['correct/neg'] = (neg_correct / neg_count) * 100
metrics_dict['correct/pos'] = (pos_correct/ pos_count) * 100

precision = metrics_dict['pr/precision'] = truePos_count / np.
 ↪float32(truePos_count + falsePos_count)
```

```
recall = metrics_dict['pr/recall'] = truePos_count / np.float32(truePos_count +␣
 ↪falseNeg_count)


metrics_dict['pr/f1_score'] = 2 * (precision * recall) / (precision + recall)

print(precision)
print(recall)
print(metrics_dict['correct/pos'])
```

```
0.5575221238938053
0.5206611570247934
52.066115702479344
```

[ ]:

```
def logMetrics(epoch_ndx,metrics_t,classificationThreshold=0.5,):

    negLabel_mask = metrics_t[0] <= classificationThreshold
    negPred_mask = metrics_t[1] <= classificationThreshold

    posLabel_mask = ~negLabel_mask
    posPred_mask = ~negPred_mask

    neg_count = int(negLabel_mask.sum())
    pos_count = int(posLabel_mask.sum())

    trueNeg_count = neg_correct = int((negLabel_mask & negPred_mask).sum())
    truePos_count = pos_correct = int((posLabel_mask & posPred_mask).sum())

    falsePos_count = neg_count - neg_correct
    falseNeg_count = pos_count - pos_correct

    metrics_dict = {}
    metrics_dict['loss/all'] = metrics_t[2].mean()
    metrics_dict['loss/neg'] = metrics_t[2, negLabel_mask].mean()
    metrics_dict['loss/pos'] = metrics_t[2, posLabel_mask].mean()

    metrics_dict['correct/all'] = (pos_correct + neg_correct) / metrics_t.
↪shape[1] * 100
    metrics_dict['correct/neg'] = (neg_correct) / neg_count * 100
    metrics_dict['correct/pos'] = (pos_correct) / pos_count * 100

    precision = metrics_dict['pr/precision'] = truePos_count / np.
↪float32(truePos_count + falsePos_count)
    recall = metrics_dict['pr/recall'] = truePos_count / np.
↪float32(truePos_count + falseNeg_count)
```

```
      metrics_dict['pr/f1_score'] = 2 * (precision * recall) / (precision +␣
 ↪recall)

      return metrics_dict
```

[ ]:

```
metrics_c=metrics_g.to('cpu')
metrics_dict=logMetrics(1,metrics_c)
```

[ ]:

```
print(metrics_dict)
```

```
{'loss/all': tensor(0.6729), 'loss/neg': tensor(0.6491), 'loss/pos':
tensor(0.6969), 'correct/all': 55.55555555555556, 'correct/neg':
59.01639344262295, 'correct/pos': 52.066115702479344, 'pr/precision':
0.5575221238938053, 'pr/recall': 0.5206611570247934, 'pr/f1_score':
0.5384615384615384}
```

[ ]:

```
# dict
history = {'train_loss':[],'train_accuracy':[], 'val_loss':[], 'val_accuracy':
 ↪[]}
```

[ ]:

```
  def logMetrics2(epoch_ndx,metrics_t,classificationThreshold=0.5,):

      negLabel_mask = metrics_t[0] <= classificationThreshold
      negPred_mask = metrics_t[1] <= classificationThreshold

      posLabel_mask = ~negLabel_mask
      posPred_mask = ~negPred_mask

      neg_count = int(negLabel_mask.sum())
      pos_count = int(posLabel_mask.sum())

      trueNeg_count = neg_correct = int((negLabel_mask & negPred_mask).sum())
      truePos_count = pos_correct = int((posLabel_mask & posPred_mask).sum())

      falsePos_count = neg_count - neg_correct
      falseNeg_count = pos_count - pos_correct

      metrics_dict = {'epoch_ndx':[],'loss/all':[],'loss/neg':[], 'loss/pos':
 ↪[],'correct/all':[],'correct/neg':[],'correct/pos':[],'pr/precision':[],'pr/
 ↪recall':[],'pr/f1_score':[]}
      metrics_dict['epoch_ndx'].append(epoch_ndx)
      metrics_dict['loss/all'].append(metrics_t[2].mean().item())
      metrics_dict['loss/neg'].append(metrics_t[2, negLabel_mask].mean().item())
      metrics_dict['loss/pos'].append(metrics_t[2, posLabel_mask].mean().item())

      metrics_dict['correct/all'].append((pos_correct + neg_correct) /␣
 ↪metrics_t.shape[1] * 100)
      metrics_dict['correct/neg'].append((neg_correct) / neg_count * 100)
```

```
    metrics_dict['correct/pos'].append((pos_correct) / pos_count * 100)

    metrics_dict['pr/precision'].append(truePos_count / np.
↪float32(truePos_count + falsePos_count))
    metrics_dict['pr/recall'].append(truePos_count / np.float32(truePos_count␣
↪+ falseNeg_count))

    precision = truePos_count / np.float32(truePos_count + falsePos_count)
    recall = truePos_count / np.float32(truePos_count + falseNeg_count)

    metrics_dict['pr/f1_score'].append(2 * (precision * recall) / (precision␣
↪+ recall))

    return metrics_dict
```

[ ]:
```
metrics_c=metrics_g.to('cpu')
metrics_dict=logMetrics2(1,metrics_c)
```

[ ]:
```
print(metrics_dict)
```

```
{'loss/all': [0.6728792190551758], 'loss/neg': [0.6490968465805054], 'loss/pos':
[0.6968579292297363], 'correct/all': [55.55555555555556], 'correct/neg':
[59.01639344262295], 'correct/pos': [52.066115702479344], 'pr/precision':
[0.5575221238938053], 'pr/recall': [0.5206611570247934], 'pr/f1_score':
[0.5384615384615384]}
```

11 END