


โมเดล Deep learning จำแนกโรคลูคีเมียเซลล์เม็ดเลือดด้วย PyTorch lightning

มะเร็งเม็ดเลือดขาวเฉียบพลันชนิดลิมโฟบลาสต์ [Acute lymphoblastic leukemia](#) (ALL) เป็นมะเร็งชนิดที่พบบ่อยที่สุดในเด็กและคิดเป็นประมาณ 25 เปอร์เซ็นต์ของมะเร็งในวัยเด็ก โดยชุดข้อมูลนี้เป็นชุดรูปภาพที่ถูกคัดเลือกเป็นรูปภาพเซลล์เม็ดเลือดจากกล้องจุลทรรศน์ โดยวันนี้เราจะนำทุกคนมาสร้างโมเดลจำแนกเซลล์เม็ดเลือดเหล่านี้ด้วย PyTorch Lightning ด้วยวิธี Transfer learning


Transfer learning อธิบายแบบง่ายๆคือการที่เรานำโมเดลที่มีคนเทรนมาแล้วนำมาเทรนต่อเพื่อจุดประสงค์ที่เราต้องการ โดยที่เราจะใช้ Pre-trained โมเดลที่เรียกว่า Resnet ในการเทรนโมเดลในครั้งนี้

 LARXEL · UPDATED 3 YEARS AGO

▲ 199

New Notebook

Download (909 MB)



Leukemia Classification

Identify cancer cells in the most prevalent childhood cancer type.

Data Card

Code (45)

Discussion (5)

About Dataset

Abstract

Tackle one of the major childhood cancer types by creating a model to classify normal from abnormal cell images.

About this dataset

Acute lymphoblastic leukemia (ALL) is the most common type of childhood cancer and accounts for approximately 25% of the pediatric cancers.
These cells have been segmented from microscopic images and are representative of images in the real-world because they contain some staining noise and illumination errors, although these errors have largely been fixed in the course of acquisition.
The task of identifying immature leukemic blasts from normal cells under the microscope is challenging due to morphological similarity and thus the ground truth labels were annotated by an expert oncologist.

Usability 8.75

License Other (specified in description)

Expected update frequency Never

Tags

Health

Biology

Cancer

Image

Health Conditions

Public Health

Medicine

<https://www.kaggle.com/datasets/andrewmvd/leukemia-classification/data>



รูปภาพตัวอย่างเซลล์

ข้อมูลประกอบด้วย **15,135** รูป จากผู้ป่วย **118** คน โดยแบ่งได้เป็น label2 class ดังนี้:

- Normal cell ถึงจะถูกจัดเก็บไว้ใน Folder hem
- Leukemia blast ถึงจะถูกจัดเก็บไว้ใน Folder all

Data preparation

จากข้อมูลที่ได้รับมาจะพบได้ว่า ใน Folder testing_data นั้นไม่มี Label กำหนดมาจึงไม่สามารถใช้งานได้เราจึง
ต้องทำการเตรียมข้อมูลใหม่

Current file structure

```
| - C-NMC_Leukemia
|   | - testing_data
|   |   | - C-NMC_test_final_phase_data(With no lable)
|   |
|   | - training_data
|   |   | - fold_0
|   |   |   | - all
|   |   |   | - hem
|   |   |
|   |   | - fold_1
|   |   |   | - all
|   |   |   | - hem
|   |   |
|   |   | - fold_2
|   |   |   | - all
|   |   |   | - hem
|   |
|   | - validation_data
|   |   | - C-NMC_test_prelim_phase_data
|   |   | - C-NMC_test_prelim_phase_data_labels.csv
```

โครงสร้างไฟล์จาก Kaggle

และเราจะนำข้อมูลเหล่านี้มาจัดเรียงใหม่

Preferred file structure

```
| - root_dir
|   | - train
|   |   | - all
|   |   | - hem
|   |
|   | - validation
|   |   | - all
|   |   | - hem
|   |
|   | - test
|   |   | - all
|   |   | - hem
```

โครงสร้างFolder ข้อมูลที่ต้องการ

เรามาเริ่มด้วย import library ที่ต้องใช้ในเตรียมข้อมูลและสร้างโมเดล

```
import torch
import pandas as pd
import glob
import os
import os.path as op
import shutil
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
import pytorch_lightning as pl
import torch.nn.functional as F
import torch.nn as nn
import glob

from torchvision import datasets, models, transforms
import torchvision.transforms as T
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from torchmetrics import Accuracy
from pytorch_lightning.callbacks import ModelCheckpoint
from torchmetrics.classification import Accuracy
from PIL import Image
from torch.utils.data import WeightedRandomSampler
from IPython.display import display
from ipywidgets import FileUpload
```

จากโครงสร้างไฟล์ที่เราโหลดมา เพื่อให้ได้มาซึ่ง Path ของ Folder เราจึงสร้าง Function find_deepest_paths() เพื่อให้ได้มาซึ่ง Path ที่เราต้องการ จาก Folder training_data

```
def find_deepest_paths(folder_path):
    max_depth = 0
    deepest_paths = []
    for root, dirs, files in os.walk(folder_path):
        current_depth = root.count(os.sep)

        if current_depth > max_depth:
            max_depth = current_depth
            deepest_paths = [root.replace('\\', '/')]
        elif current_depth == max_depth:
            deepest_paths.append(root.replace('\\', '/'))
    return deepest_paths
```

```

folder_path = 'C-NMC_Leukemia/training_data' #ใส่ path folder ที่เราทำการdownload ไว้
deepest_paths = find_deepest_paths(folder_path)
print("Deepest Paths:")
for path in deepest_paths:
    print(path)

```

Deepest Paths:

```

C-NMC_Leukemia/training_data/fold_0/all
C-NMC_Leukemia/training_data/fold_0/hem
C-NMC_Leukemia/training_data/fold_1/all
C-NMC_Leukemia/training_data/fold_1/hem
C-NMC_Leukemia/training_data/fold_2/all
C-NMC_Leukemia/training_data/fold_2/hem

```

ผลลัพธ์ ที่เก็บไว้ในตัวแปร deepest_paths

สร้างฟังก์ชัน เพื่อนำไฟล์รูปภาพไปเก็บใน List all_list , hem_list เพื่อที่จะนำเก็บเข้าไปใส่ใน data frame ในภายหลัง

```

def create_glob_patterns(file_paths):
    all_list = []
    hem_list = []
    for path in file_paths:
        _, _, fold, category = path.split('/')
        glob_pattern = f'{path}/*.bmp'
        paths_for_category = glob.glob(glob_pattern)
        if 'all' in path:
            all_list.extend(paths_for_category)
        elif 'hem' in path:
            hem_list.extend(paths_for_category)
    return all_list, hem_list

all_list, hem_list = create_glob_patterns(deepest_paths)
# Print the results
print("All List:", len(all_list))
print("Hem List:", len(hem_list))

```

จัดการไฟล์ Validation_data ในส่วนนี้ข้อมูลได้ถูกจัดอยู่รูปแบบ CSV และ Map รูปภาพใน Folder C-NMC_test_prelim_phase_data

Patient_ID	new_names	labels
UID_57_29_1_all.bmp	1.bmp	1
UID_57_22_2_all.bmp	2.bmp	1
UID_57_31_3_all.bmp	3.bmp	1
UID_H49_35_1_hem.bmp	4.bmp	0
UID_58_6_13_all.bmp	5.bmp	1
UID_57_8_11_all.bmp	6.bmp	1
UID_H49_29_2_hem.bmp	7.bmp	0
UID_H30_6_2_hem.bmp	8.bmp	0
UID_58_2_1_all.bmp	9.bmp	1
UID_54_35_3_all.bmp	10.bmp	1
UID_57_21_5_all.bmp	11.bmp	1
UID_65_3_1_all.bmp	12.bmp	1
UID_H34_17_1_hem.bmp	13.bmp	0
UID_H39_18_2_hem.bmp	14.bmp	0
UID_66_29_1_all.bmp	15.bmp	1
UID_57_8_9_all.bmp	16.bmp	1

ตัวอย่างข้อมูลไฟล์ CSV

นำ validation data นำมาแปลงเก็บไว้ใน path all_v_list , hem_v_list

```
valid_data=pd.read_csv('C-NMC_Leukemia/validation_data/C-NMC_test_prelim_phase_data_labels.csv')
all_v = valid_data[valid_data[labels]==1]
hem_v = valid_data[valid_data[labels]==0]
all_v_list = ['C-NMC_Leukemia/validation_data/C-NMC_test_prelim_phase_data/' + i for i in list(all_v.new_names)]
hem_v_list = ['C-NMC_Leukemia/validation_data/C-NMC_test_prelim_phase_data/' + i for i in list(hem_v.new_names)]
```

นำ List มารวมกัน ใน all_list และ hem_list

```
# Merge all data into one data path list
all_list.extend(all_v_list)
hem_list.extend(hem_v_list)
print("All List:", len(all_list))
print("Hem List:", len(hem_list))
```

หลังจากนั้นเราจะเป็น pathlist ทั้งหมดไว้ใน Data frame

df_all : เก็บข้อมูลรูปภาพเซลล์ที่มีโรค

df_hem : เก็บข้อมูลรูปภาพเซลล์ที่ปกติ

```
df_all = pd.DataFrame({'path': all_list, 'label': 1})
df_hem = pd.DataFrame({'path': hem_list, 'label': 0})
df_all['path'] = df_all['path'].str.replace('\\', '/', regex=False)
df_hem['path'] = df_hem['path'].str.replace('\\', '/', regex=False)
```

```
df_all['id'] = df_all['path'].apply(lambda x: x.split('/')[-1].split('.')[0])
df_hem['id'] = df_hem['path'].apply(lambda x: x.split('/')[-1].split('.')[0])
```

จะได้ดังภาพนี้ ที่ประกอบด้วย path label id

path : path ที่ใช้เข้าถึงไฟล์รูปภาพนั้นๆ

label : รูปภาพดังกล่าวอยู่ใน Class อะไร

Id : ชื่อไฟล์รูปภาพ

	path	label	id
0	C-NMC_Leukemia/training_data/fold_0/hem/UID_H1...	0	UID_H11_10_1_hem
1	C-NMC_Leukemia/training_data/fold_0/hem/UID_H1...	0	UID_H11_10_2_hem
2	C-NMC_Leukemia/training_data/fold_0/hem/UID_H1...	0	UID_H11_10_3_hem
3	C-NMC_Leukemia/training_data/fold_0/hem/UID_H1...	0	UID_H11_11_1_hem
4	C-NMC_Leukemia/training_data/fold_0/hem/UID_H1...	0	UID_H11_11_2_hem
...
4680	C-NMC_Leukemia/validation_data/C-NMC_test_prel...	0	1842
4681	C-NMC_Leukemia/validation_data/C-NMC_test_prel...	0	1845
4682	C-NMC_Leukemia/validation_data/C-NMC_test_prel...	0	1850
4683	C-NMC_Leukemia/validation_data/C-NMC_test_prel...	0	1858
4684	C-NMC_Leukemia/validation_data/C-NMC_test_prel...	0	1865

รูปภาพ Data frame

เราได้แบ่ง train_size = 2880 validate_size_720 และ test_size = 403

ซึ่งจะนำข้อมูลจาก df_all c และ df_hem แบ่งไปยัง dataframe ดังต่อไปนี้ df_hem_train

df_hem_validate

df_hem_test

df_all_train

df_all_validate

df_all_test

```
train_size = 2880
validate_size = 720
test_size = 403
# Create the splits
df_all_train = df_all.iloc[:train_size]
df_all_validate = df_all.iloc[train_size:train_size+validate_size]
df_all_test = df_all.iloc[train_size+validate_size:train_size+validate_size+test_size+200]
df_hem_train = df_hem.iloc[:train_size]
df_hem_validate = df_hem.iloc[train_size:train_size+validate_size]
df_hem_test = df_hem.iloc[train_size+validate_size:train_size+validate_size+test_size]
# Print the shapes of the new dataframes
print("df_hem_train shape:", df_hem_train.shape)
print("df_hem_validate shape:", df_hem_validate.shape)
print("df_hem_test shape:", df_hem_test.shape)
# Print the shapes of the new dataframes
print("df_all_train shape:", df_all_train.shape)
print("df_all_validate shape:", df_all_validate.shape)
print("df_all_test shape:", df_all_test.shape)
```

```
df_hem_train shape: (2880, 3)
df_hem_validate shape: (720, 3)
df_hem_test shape: (403, 3)
df_all_train shape: (2880, 3)
df_all_validate shape: (720, 3)
df_all_test shape: (603, 3)
```

Shape ของ dataframe ที่ถูกแบ่งออกมา

ต่อมานำ dataframe มาต่อกันเพื่อให้เหลือเพียง df_train df_test และ df_validate


```
# Concatenate dataframes along rows (axis=0)
df_train = pd.concat([df_all_train, df_hem_train], ignore_index=True)
df_test = pd.concat([df_all_test, df_hem_test], ignore_index=True)
df_validate = pd.concat([df_all_validate, df_hem_validate], ignore_index=True)
```

```
# Display the concatenated dataframe
print(df_train.shape)
```

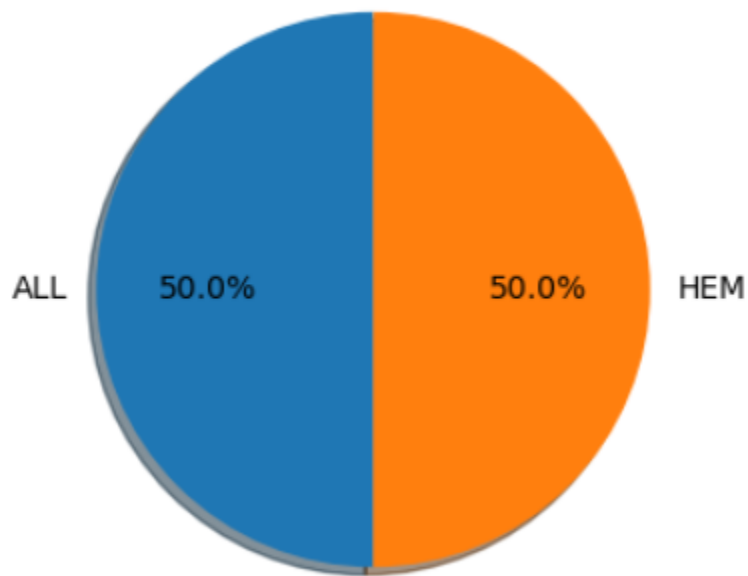
df_train

	path	label	id
0	C-NMC_Leukemia/training_data/fold_0/all/UID_11...	1	UID_11_10_1_all
1	C-NMC_Leukemia/training_data/fold_0/all/UID_11...	1	UID_11_11_1_all
2	C-NMC_Leukemia/training_data/fold_0/all/UID_11...	1	UID_11_11_2_all
3	C-NMC_Leukemia/training_data/fold_0/all/UID_11...	1	UID_11_11_3_all
4	C-NMC_Leukemia/training_data/fold_0/all/UID_11...	1	UID_11_12_1_all
...
5755	C-NMC_Leukemia/training_data/fold_2/hem/UID_H2...	0	UID_H23_6_4_hem
5756	C-NMC_Leukemia/training_data/fold_2/hem/UID_H2...	0	UID_H23_6_5_hem
5757	C-NMC_Leukemia/training_data/fold_2/hem/UID_H2...	0	UID_H23_6_6_hem
5758	C-NMC_Leukemia/training_data/fold_2/hem/UID_H2...	0	UID_H23_6_7_hem
5759	C-NMC_Leukemia/training_data/fold_2/hem/UID_H2...	0	UID_H23_6_8_hem

5760 rows x 3 columns
df_train

ทำการแสดงรูป ในรูปแบบ pie chart และจะเห็นว่าในชุดข้อมูล train นั้นมีจำนวนข้อมูล All และ Hem เท่าๆกัน ซึ่งถือเป็นสิ่งที่ดีเวลาทำการเทรนโมเดล

```
labels = ['ALL', 'HEM']
plt.figure(figsize=(4,4))
plt.pie(df_train.groupby('label').size(), labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)
plt.show()
```



ขั้นตอนสุดท้ายเราจะคัดลอกข้อมูลทั้งหมดมาแบ่งให้อยู่ในโครงสร้างที่เราได้ออกแบบไว้เพื่อง่ายต่อการใช้งานและเข้าถึงในอนาคต

```
root_dir = "data/bloodcell/"
# Iterate over each DataFrame and its corresponding folder name
for df, folder_name in zip([df_train, df_validate, df_test], ["train", "validation", "test"]):
    for _, r in df.iterrows():
        # Determine the subfolder based on the label value
        subfolder = "hem" if r.label == 0 else "all"
        # Create subfolder if it doesn't exist
        d = op.join(root_dir, folder_name, subfolder)
        if not op.exists(d):
            os.makedirs(d)
        # Check if the file already exists in the destination
        destination_path = op.join(d, f"{r.id}.bmp")
        if not op.exists(destination_path):
            # Copy the image to the appropriate subfolder with a new filename
            shutil.copy(r.path, destination_path)
```

```
# file structure of data/bloodcell should look like this
|
| - root_dir
|   |
|   | - train
|   |   |
|   |   | - all
|   |   | - hem
|   |
|   | - validation
|   |   |
|   |   | - all
|   |   | - hem
|   |
|   | - test
|   |   |
|   |   | - all
|   |   | - hem
```

โครงสร้างไฟล์ข้อมูลภาพหลังจากRun code ส่วนบน

```
folder_path = "data/bloodcell/"
new_deepest_paths = find_deepest_paths(folder_path)
print("Deepest Paths:")
for path in new_deepest_paths:
    print(path)
```

```
Deepest Paths:
data/bloodcell/test/all
data/bloodcell/test/hem
data/bloodcell/train/all
data/bloodcell/train/hem
data/bloodcell/validation/all
data/bloodcell/validation/hem
```

print_folder_counts() จะทำการแสดงจำนวนแสดงจำนวนไฟล์ทั้งหมดที่มีอยู่ในโฟลเดอร์เพื่อดูว่าจำนวนไฟล์ที่ถูกลดออกมาถูกต้อง

```
# Print the number of files in 'hem' and 'all' subfolders for each dataset
def print_folder_counts(root_dir, dataset_names=["train", "validation", "test"]):
    for folder_name in dataset_names:
        hem_folder = op.join(root_dir, folder_name, "hem")
        all_folder = op.join(root_dir, folder_name, "all")
        hem_count = len(os.listdir(hem_folder))
        all_count = len(os.listdir(all_folder))
        print(f"For {folder_name} dataset:")
        print(f"Number of files in 'hem' folder: {hem_count}")
        print(f"Number of files in 'all' folder: {all_count}")
```

```
print("=" * 30)
```

```
root_directory = "data/bloodcell/"  
print_folder_counts(root_directory)
```

```
For train dataset:  
Number of files in 'hem' folder: 2880  
Number of files in 'all' folder: 2880  
=====  
For validation dataset:  
Number of files in 'hem' folder: 720  
Number of files in 'all' folder: 720  
=====  
For test dataset:  
Number of files in 'hem' folder: 403  
Number of files in 'all' folder: 603  
=====
```

จำนวนไฟล์รูปภาพในแต่ละไฟล์ถูกต้อง

Image classification with Pytorch lightning

ในการเทรนโมเดลจำแนกรูปภาพการใช้ GPU(Graphics processing unit) ในการประมวลผลด้านกราฟฟิก นั้นจะมีประสิทธิภาพ มากกว่า CPU เสมอ

เรามาเริ่มจากเช็คว่ามี GPU ที่สามารถใช้งานได้หรือไม่

```
gpu = torch.cuda.is_available()  
print("GPU available:", gpu)
```

```
GPU available: True
```

กระบวนการที่เราต้องทำต่อไปคือการทำพร็โพรเซส โดยปกติที่เรานิยมใช้คือ `Resize` รูปภาพข้อมูลให้เป็นขนาดเดียวกัน เนื่องจากว่าขนาดรูปภาพที่มากับชุดข้อมูลนั้น มีค่าขนาด $450 * 450$ เหมือนกันทุกภาพกันั้นจึงไม่จำเป็นต้อง `Resize` รูปภาพ `Normalize` สำหรับการทำให้ normalize รูปภาพและ `ToTensor` สำหรับปรับรูปให้กลายเป็นชนิด `tensor` สำหรับเทรนโมเดลด้วย PyTorch ตัวอย่างการใช้งานจะเป็นดังนี้หลังจากนั้นเราเข้าถึงรูปภาพด้วย `datasets.ImageFolder` และตั้ง

transform ตามที่เราได้เขียนไว้ และเราใช้ ฟังก์ชัน DataLoader เพื่อทำการเรียกชุดข้อมูลที่จะป้อนเข้าสู่โมเดล ซึ่งจะจำกัดจำนวนในการป้อนข้อมูลในแต่ละครั้งด้วย batch_size และ ใช้กระบวนการย่อย กระบวนการในการโหลดชุดข้อมูลด้วย num_workers

```
train_transform = transforms.Compose([
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
    transforms.ToTensor()
])
val_transform = transforms.Compose([
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
    transforms.ToTensor()
])
train_data = datasets.ImageFolder("data/bloodcell/train", transform=train_transform)
val_data = datasets.ImageFolder("data/bloodcell/validation", transform=val_transform)
train_loader = DataLoader(train_data, batch_size=32, shuffle=True, num_workers=15)
val_loader = DataLoader(val_data, batch_size=32, shuffle=False, num_workers=15)
```

เก็บ classes ใน classes และ จำนวน class ใน n_classes

```
classes = train_data.classes
n_classes = len(classes)
```

ขั้นตอนต่อไปเป็นการสร้าง

Class LeukemiaResNet เป็นโมเดลการเรียนรู้ของเครื่องสำหรับจำแนกเซลล์เม็ดเลือดออกเป็น 2 สถานะ (ปกติหรือมะเร็ง)

Class LeukemiaResNet

- init(): กำหนดค่าเริ่มต้นของโมเดล
- n_classes: จำนวนสถานะของเซลล์เม็ดเลือด

- backbone: สถาปัตยกรรมของโมเดล โดยเราได้ใช้resnet18 ที่ได้ทำการ pretrain มาแล้ว
- entropy_loss: ฟังก์ชันสูญเสียที่ใช้สำหรับการเทรน
- accuracy: ตัววัดประสิทธิภาพที่ใช้สำหรับการประเมิน

Def forward()

- ส่งผ่านข้อมูลอินพุตไปยังโมเดลและส่งคืนเอาต์พุต

Def training_step()

- คำนวณการสูญเสียและประสิทธิภาพสำหรับชุดข้อมูลการเทรน
- บันทึกการสูญเสียและประสิทธิภาพ

Def validation_step()

- คำนวณการสูญเสียและประสิทธิภาพสำหรับชุดข้อมูลการตรวจสอบ
- บันทึกการสูญเสียและประสิทธิภาพ

Def test_step()

- คำนวณการสูญเสียและประสิทธิภาพสำหรับชุดข้อมูลทดสอบ
- บันทึกการสูญเสียและประสิทธิภาพ

Def configure_optimizers()

- กำหนดตัวปรับแต่งสำหรับการเทรน

คำอธิบายโดยละเอียด

Class LeukemiaResNet

- init():
- n_classes: จำนวนสถานะของเซลล์เม็ดเลือด กำหนดค่าเริ่มต้นเป็น 2
- backbone: สถาปัตยกรรมของโมเดล ใช้สถาปัตยกรรม ResNet18 ที่มี pre-trained weights
- entropy_loss: ฟังก์ชันสูญเสียที่ใช้สำหรับการเทรนใช้ CrossEntropyLoss ซึ่งเป็นฟังก์ชันสูญเสียที่ได้รับความนิยมสำหรับงานจำแนก
- accuracy: ตัววัดประสิทธิภาพที่ใช้สำหรับการประเมิน ใช้ Accuracy ซึ่งเป็นตัววัดประสิทธิภาพที่ได้รับความนิยมสำหรับงานจำแนก

Def forward()

- ส่งผ่านข้อมูลอินพุตไปยังโมเดล
- โมเดลจะดำเนินการตามขั้นตอนต่อไปนี้:
- แปลงข้อมูลอินพุตเป็นรูปแบบที่เหมาะสมสำหรับโมเดล
- ประมวลผลข้อมูลอินพุตผ่านโมเดล
- ส่งคืนเอาต์พุตของโมเดล

Def training_step()

- คำนวณการสูญเสียและประสิทธิภาพสำหรับชุดข้อมูลการเทรน
- การสูญเสียคำนวณโดยใช้ฟังก์ชันสูญเสียที่ระบุไว้ใน `init()`
- ประสิทธิภาพคำนวณโดยใช้ตัววัดประสิทธิภาพที่ระบุไว้ใน `init()`
- การสูญเสียและประสิทธิภาพจะถูกบันทึกลงในล็อกสำหรับติดตามความคืบหน้าของการเทรน

Def `validation_step()`

- คำนวณการสูญเสียและประสิทธิภาพสำหรับชุดข้อมูลการตรวจสอบ
- การสูญเสียและประสิทธิภาพจะถูกบันทึกลงในล็อกสำหรับติดตามความคืบหน้าของการเทรน

Def `test_step()`

- คำนวณการสูญเสียและประสิทธิภาพสำหรับชุดข้อมูลทดสอบ
- การสูญเสียและประสิทธิภาพจะถูกบันทึกลงในล็อกสำหรับติดตามความคืบหน้าของการเทรน

Def `configure_optimizers()`

- กำหนดตัวปรับแต่งสำหรับการเทรน
- ตัวปรับแต่งใช้อัลกอริธึม AdamW ซึ่งเป็นอัลกอริธึมการปรับแต่งที่ได้รับความนิยมสำหรับงานการเรียนรู้ของเครื่อง
- ตัวปรับแต่งจะถูกกำหนดค่าด้วยอัตราการเรียนรู้เริ่มต้นที่ $1e-3$
- ตัวปรับแต่งจะติดตามการสูญเสียของชุดข้อมูลการตรวจสอบเพื่อปรับอัตราการเรียนรู้


```

class LeukemiaResNet(pl.LightningModule):
    def __init__(self, n_classes=2):
        super(LeukemiaResNet, self).__init__()
        # จำนวนของสถานะของเซลล์เม็ดเลือด 2 สถานะ
        self.n_classes = n_classes
        # ใช้สถาปัตยกรรม resnet18; เปลี่ยน layer สุดท้าย/////
        self.backbone = models.resnet18(pretrained=True)
        for param in self.backbone.parameters():
            param.requires_grad = False
        # เปลี่ยน fc layer เป็น output ขนาด 2
        self.backbone.fc = torch.nn.Linear(self.backbone.fc.in_features, n_classes)
        self.entropy_loss = nn.CrossEntropyLoss()
        self.accuracy = Accuracy(task="binary")

    def forward(self, x):
        preds = self.backbone(x)
        return preds

    def training_step(self, batch, batch_idx):
        x, y = batch
        logits = self.backbone(x)
        loss = self.entropy_loss(logits, y)
        y_pred = torch.argmax(logits, dim=1)
        self.log("train_loss", loss)
        self.log("train_acc", self.accuracy(y_pred, y))
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        logits = self.backbone(x)
        loss = self.entropy_loss(logits, y)
        y_pred = torch.argmax(logits, dim=1)
        self.log("val_loss", loss)
        self.log("val_acc", self.accuracy(y_pred, y))
        return loss

    def test_step(self, batch, batch_idx):
        x, y = batch
        logits = self.backbone(x)
        loss = self.entropy_loss(logits, y)
        y_pred = torch.argmax(logits, dim=1)
        self.log("test_loss", loss)
        self.log("test_acc", self.accuracy(y_pred, y))
        return loss

    def configure_optimizers(self):
        self.optimizer = torch.optim.AdamW(self.parameters(), lr=1e-3)
        return {
            "optimizer": self.optimizer,

```

```
"monitor": "val_loss"  
}
```

กำหนดโมเดลLeukemiaResNetเข้าตัวแปลmodel

```
model =LeukemiaResNet(n_classes=n_classes)
```

ต่อไปเราใช้ ฟังก์ชันModelCheckpoint

โค้ดนี้กำหนดตัวติดตามการตรวจสอบสำหรับโมเดล LeukemiaResNet ตัวติดตามการตรวจสอบจะบันทึกโมเดลที่ดีที่สุดที่พบระหว่างการเทรน ซึ่งสามารถช่วยในการกู้คืนโมเดลที่มีประสิทธิภาพดีที่สุดได้

พารามิเตอร์ของตัวติดตามการตรวจสอบ

- dirpath: ไดร็กทอรีที่จะบันทึกโมเดล
- filename: รูปแบบชื่อไฟล์ที่จะใช้สำหรับโมเดลที่บันทึกไว้
- save_top_k: จำนวนโมเดลที่ดีที่สุดที่จะบันทึก
- verbose: ระดับการแจ้งเตือนที่จะใช้
- monitor: ปริมาณที่ตัวติดตามการตรวจสอบจะติดตาม
- mode: ทิศทางที่โมเดลที่ดีที่สุดควรได้รับการบันทึก (min หรือ max)
- dirpath="./checkpoints/Leukemia/" กำหนดpathที่จะบันทึกโมเดล ตัวติดตามการตรวจสอบจะสร้างโฟลเดอร์ “checkpoints/Leukemia” หากโฟลเดอร์นี้ไม่มี
- ไดร็กทอรี checkpoints/Leukemia

- filename="resnet18 — {epoch:02d}-{val_acc:.2f}-{val_loss:.2f}" กำหนดรูปแบบชื่อไฟล์ที่จะใช้สำหรับโมเดลที่บันทึกไว้ รูปแบบนี้จะรวมข้อมูลต่อไปนี้:
- ชื่อของสถาปัตยกรรมโมเดล (resnet18)
- หมายเลข epoch ของโมเดล
- ค่าความถูกต้องของโมเดลบนชุดข้อมูลการตรวจสอบ
- ค่าการสูญเสียของโมเดลบนชุดข้อมูลการตรวจสอบ
- save_top_k=1 กำหนดจำนวนโมเดลที่ดีที่สุดที่จะบันทึก ตัวติดตามการตรวจสอบจะบันทึกโมเดลที่ดีที่สุด 1 โมเดลเท่านั้น
- verbose=True กำหนดระดับการแจ้งเตือนที่จะใช้ ตัวติดตามการตรวจสอบจะส่งข้อความแจ้งเตือนเมื่อบันทึกโมเดล
- monitor="val_loss" กำหนดปริมาณที่ตัวติดตามการตรวจสอบจะติดตาม ตัวติดตามการตรวจสอบจะบันทึกโมเดลที่ดีที่สุดเมื่อค่าการสูญเสียของโมเดลบนชุดข้อมูลการตรวจสอบลดลง
- mode="min" กำหนดทิศทางที่โมเดลที่ดีที่สุดควรได้รับการบันทึก ตัวติดตามการตรวจสอบจะบันทึกโมเดลที่ดีที่สุดเมื่อค่าการสูญเสียของโมเดลลดลง

```
checkpoint_callback = ModelCheckpoint(
    dirpath="/checkpoints/Leukemia/",
    filename="resnet18-{epoch:02d}-{val_acc:.2f}-{val_loss:.2f}",
    save_top_k=1,
    verbose=True,
    monitor="val_loss",
    mode="min",
)
```

ต่อมาเราจะสร้างPytorcho lightning Trainer object โดยได้ เซ็ตพารามิเตอร์ `max_epochs = 20` ซึ่งให้เทรนโมเดลมากที่สุด 20 epochs

devices = 1 กำหนดให้โมเดลเทรนใน 1 device

accelerator = "gpu" เราให้โมเดลเทรนใน GPU ซึ่งมีประสิทธิภาพมากกว่า CPU

callbacks = [checkpoint_callback] จะถูกใช้ในการจัดเก็บ snapshots ของโมเดลขนาดเทรนโมเดล ตามที่เราได้เซตเอาไว้ และสุดท้ายเราได้ทำการเทรนโมเดลด้วยฟังก์ชัน fit

```
trainer = pl.Trainer(max_epochs=20, devices=1, accelerator="gpu", callbacks=[checkpoint_callback])
trainer.fit(model, train_dataloaders=train_loader, val_dataloaders=val_loader)
```

และระหว่างเทรนจะมีผลลัพธ์การเทรนดังนี้

```
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

| Name      | Type      | Params
-----|-----|-----
0 | backbone  | ResNet    | 11.2 M
1 | entropy_loss | CrossEntropyLoss | 0
2 | accuracy  | BinaryAccuracy | 0

1.0 K   Trainable params
11.2 M   Non-trainable params
11.2 M   Total params
44.710   Total estimated model params size (MB)
C:\Users\USER\anaconda3\lib\site-packages\pytorch_lightning\trainer\connectors\data_connector.py:436: Consider setting `persistent_workers=True` in
`val_dataloader` to speed up the dataloader worker initialization.
C:\Users\USER\anaconda3\lib\site-packages\pytorch_lightning\trainer\connectors\data_connector.py:436: Consider setting `persistent_workers=True` in
`train_dataloader` to speed up the dataloader worker initialization.
Epoch 19: 100%
180/180 [01:46<00:00, 1.69it/s, v_num=29]
Epoch 0, global step 180: `val_loss` reached 0.62877 (best 0.62877), saving model to 'C:\Users\USER\BU
Deeplearning\checkpoints\Leukemia\resnet34-epoch=00-val_acc=0.64-val_loss=0.63-v1.ckpt' as top 1
Epoch 1, global step 360: `val_loss` was not in top 1
Epoch 2, global step 540: `val_loss` was not in top 1
Epoch 3, global step 720: `val_loss` reached 0.61176 (best 0.61176), saving model to 'C:\Users\USER\BU
Deeplearning\checkpoints\Leukemia\resnet34-epoch=03-val_acc=0.68-val_loss=0.61.ckpt' as top 1
Epoch 4, global step 900: `val_loss` was not in top 1
```

```

Epoch 5, global step 1080: 'val_loss' was not in top 1
Epoch 6, global step 1260: 'val_loss' was not in top 1
Epoch 7, global step 1440: 'val_loss' was not in top 1
Epoch 8, global step 1620: 'val_loss' was not in top 1
Epoch 9, global step 1800: 'val_loss' reached 0.61054 (best 0.61054), saving model to 'C:\\Users\\USER\\BU
Deeplearning\\checkpoints\\Leukemia\\vesnet34-epoch=09-val_acc=0.70-val_loss=0.61ckpt' as top 1
Epoch 10, global step 1980: 'val_loss' was not in top 1
Epoch 11, global step 2160: 'val_loss' was not in top 1
Epoch 12, global step 2340: 'val_loss' was not in top 1
Epoch 13, global step 2520: 'val_loss' was not in top 1
Epoch 14, global step 2700: 'val_loss' was not in top 1
Epoch 15, global step 2880: 'val_loss' was not in top 1
Epoch 16, global step 3060: 'val_loss' was not in top 1
Epoch 17, global step 3240: 'val_loss' was not in top 1
Epoch 18, global step 3420: 'val_loss' was not in top 1
Epoch 19, global step 3600: 'val_loss' was not in top 1
`Trainer.fit` stopped: `max_epochs=20` reached.

```

จากการเทรนข้อมูล โมเดลที่ดีที่สุด เดินขึ้นใน epoch ที่ 9 โดยมี epoch=09 val_acc=0.70 และ val_loss=0.61

Testing model

ขั้นตอนการเทสมีความคล้ายกันกับตอนเทรน โดนเริ่มจากการ transform data เลือกข้อโฟเดอร์ข้อมูลที่ต้องการเทส และ

ใช้ DataLoader

```

test_transform = transforms.Compose([
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    transforms.ToTensor(),
])
test_data = datasets.ImageFolder("data/bloodcell/test", transform=test_transform)
test_loader = DataLoader(test_data, batch_size=32, shuffle=False, num_workers=15)

```

checkpoint_path ใส่ patch ของโมเดลที่เราต้องการใช้งาน

test_data ใส่โฟลเดอร์ไฟล์ที่เราต้องการทดสอบ

```
checkpoint_path = 'checkpoints/Leukemia/resnet34-epoch=09-val_acc=0.70-val_loss=0.61.ckpt'
trained_model = LeukemiaResNet.load_from_checkpoint(checkpoint_path, strict=False)
test_data = datasets.ImageFolder("data/bloodcell/test", transform=test_transform)
test_loader = DataLoader(test_data, batch_size=32) # replace with your test dataloader
trainer_test = pl.Trainer(accelerator="gpu")
trainer_test.test(model, test_loader)
```

```
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
```

Testing DataLoader 0: 100%  32/32 [00:03<00:00, 8.07it/s]

Test metric	DataLoader 0
test_acc	0.6550695896148682
test_loss	0.9235817193984985

```
[{'test_loss': 0.9235817193984985, 'test_acc': 0.6550695896148682}]
```

จากการทดสอบแสดงให้เห็นว่าโมเดลมีความแม่นยำในการทำนายอยู่ที่ 65%

ผู้อ่านสามารถลองทดสอบ แอปพลิเคชันที่ถูก Deploy ผ่าน Link : <https://demo-allidb.streamlit.app/>