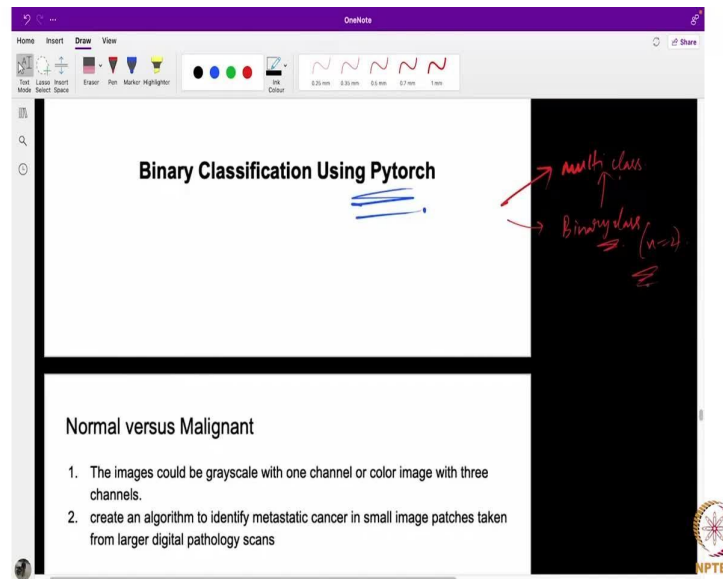


**Medical Image Analysis**  
**Professor Ganapathy Krishnamurthi**  
**Department of Engineering Design**  
**Indian Institute of Technology Madras**  
**Lecture 48**

**Classification Demo in Pytorch**

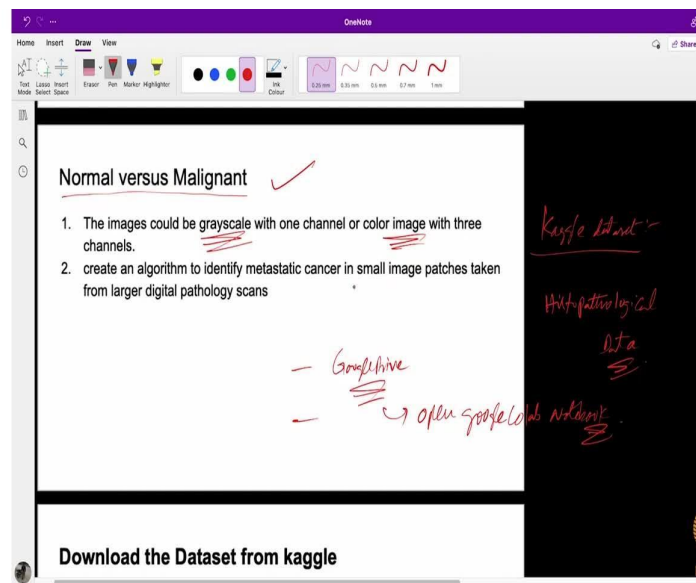
(Refer Slide Time: 00:15)



I am Ram Krishna Gallori, TA of Ganapathy sir. So, today we are going to discuss about binary classification task using Pytorch framework. Usually, the classification can be, we can call this one as multiclass or you can say, restrict yourselves to binary class. Even binary class can also be thought of as multiclass where class number is equal to 2.

That is it we will be having several images like in related to a number of classes. And we just have to train a model in such a way that whenever you give a new image to this particular model, that should identify as a particular class, that is the whole target of this entire demo.

(Refer Slide Time: 1:18)



For this one, you are just using downloaded some Kaggle data set, Kaggle data sets that containing histopathological data. Here, I will show you where exactly this data can be downloaded, and how to download this data and how this data can be downloaded to the Google Drive and from that particular directory in this particular Google Drive, we can work in Google colab notebook.

Google colab notebook, you can practice this entire code, whatever I am going to do now, basically, in this problem, we are just identifying the images either normal versus malignant, this is the basic task. This one can be this whatever code we are going to develop that can be useful for grayscale images or even color images can be given as inputs and create an algorithm to identify metastatic cancer in small image patches taken from other digital pathology scans.

(Refer Slide Time: 2:33)

The screenshot shows a OneNote application window. The main content area has a title "Download the Dataset from kaggle" and a list of five steps: 1. Create a kaggle account. 2. Search for the dataset you want to download. 3. Download the json file on to your local drive. 4. Create a directory in the google drive. 5. Upload the .json file into the directory. To the right of the list, there are handwritten notes in red ink: "Histopathology" is circled with an arrow pointing to step 2; "100,000 x 100,000" is written with an arrow pointing to step 3; "4GB" is written with an arrow pointing to step 3; "CNN" is circled with a checkmark; and "Tissue", "Pathology", and "Image" are written vertically. Below the list, there is a section titled "Key Topics" with a bullet point "Exploring the dataset".

**Download the Dataset from kaggle**

1. Create a kaggle account.
2. Search for the dataset you want to download.
3. Download the json file on to your local drive.
4. Create a directory in the google drive.
5. Upload the .json file into the directory.

**Key Topics**

- Exploring the dataset

The screenshot shows a Google search page. The Google logo is at the top, followed by a search bar with the text "Search Google or type a URL". Below the search bar are several shortcuts: YouTube, Gmail, Maps, Breast\_Cancer\_M..., Programming, NDA, YouTube, Classification-of..., Medical Data Sci..., and BratS2020 Datas... At the bottom right, there is a "Customise Chrome" button and an NPTEL logo.

Google

Search Google or type a URL

YouTube Gmail Maps Breast\_Cancer\_M... Programming NDA YouTube Classification-of... Medical Data Sci... BratS2020 Datas...

(4) Feed Facebook - L... AJO Online Shopp... Add shortcut

Customise Chrome NPTEL

Binary\_Classific... NPTEL\_Demo... GitHub - Packt... PyTorch-Compu... PyTorch-Compu... PyTorch-Compu... Histopathologic... Kaggle Your Hi...

Search

Newsfeed

Mohd Azam • Follow  
ran a notebook 5 days ago

outlier\_removal\_using\_z  
Python Notebook on **Real-World Quality**  
28s to run • 79 lines • 20 views • 10 visualizations

Novice

- ✓ Run 1 notebook or script
- ✓ Make 1 competition submission
- Make 1 comment
- ✓ Give 1 upvote

Your Competitions

Titanic - Machine Learning fo...

Your Notebooks

- ✓ P VGG16 Transfer Learning - Py...
- ✓ P VGG16 Transfer Learning - Py...
- ✓ P 100% accuracy Decision Tree...
- ✓ P 100% accuracy Decision Tree...
- ✓ P Vision Transformer (ViT): Tut...
- notebookdbcd9d71d
- notebook744402914

NPTEL

Binary\_Classific... NPTEL\_Demo... GitHub - Packt... PyTorch-Compu... PyTorch-Compu... PyTorch-Compu... Histopathologic... Kaggle Your Hi...

Search

Histopathologic Cancer Detection  
Identify metastatic tissue in histopathologic scans of lymph node sections.  
Kaggle • 1.4k teams • 9 years ago

Overview Data Code Discussion Leaderboard Rules Team My Submissions Late Submission

Overview

Description

Evaluation

Prizes

Timeline

NPTEL

Binary\_Classific... NPTEL\_Demo... GitHub - Packt... PyTorch-Compu... PyTorch-Compu... PyTorch-Compu... Histopathologic... Search Kaggle

Search

Histopathologic Cancer Detection

Searching for Histopathologic Cancer Detection within

Comments 674 Notebooks 430 Topics 88 Datasets 7 Competitions 1

Filter by

Date

Viewed By You

Creator

Dataset Size

Dataset File Types

1,200 Results Sort by: Relevancy

Competition

Histopathologic Cancer Detection

Playground

3 years ago • 1143 teams

Histopathologic Cancer Detection

Notebook

Histopathologic Cancer Detection using CNNs

by Abhinand

3 years ago • 1h to run • Python • 54

Histopathologic Cancer Detection using CNNs

Notebook

A complete ML pipeline (Fast.ai)

by Jari Juvenen

4 years ago • 8h to run • Python • 505

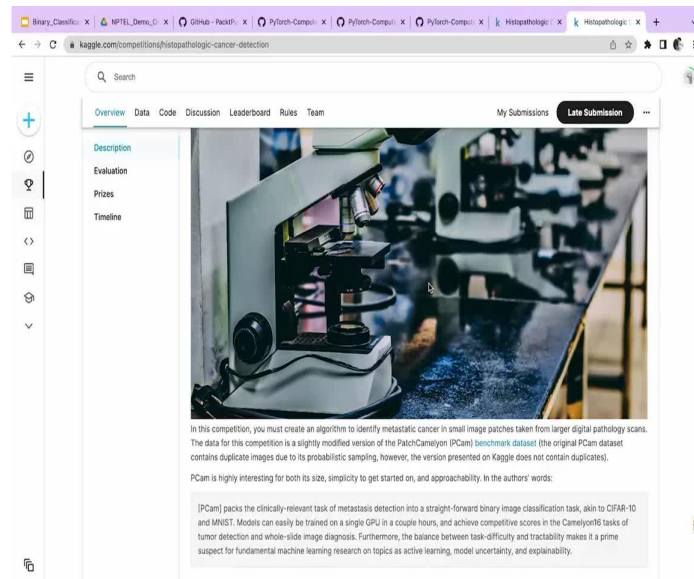
1, 100 return img, img, ax = plt.subplots(2, 2, figsize=(10, 8)) fig.savefig('Cropped\_Histopathologic')

Notebook

Histopathologic Cancer Detection

NPTEL





Before that, let me give you some idea what exactly I mean by histopathology? People who suffer from cancer mostly they will undergo some diagnosis, CT or MRI or some regular diagnostic scans. After the scanning that will identify that maybe there are a chance of a cancer so that they just take biopsy from this particular tissue sample wherever they find, suppose if there is a breast cancer, they will take a tissue from the particular breast area where exactly it has been identified as chances of cancer.

So, after taking this particular tissue part, it will be taken to pathological lab where in the particular pathological lab, they are going to make it into several slices and after making it into several slices, they will keep it under the microscope, under the microscope they will examine the particular histopathology image.

Basically histopathology means histamine tissue, histamine tissue pathology means disease, so we call tissue related diseases, I think but histopathology. Just open a Kaggle account just go and type this one kaggle dot com. Suppose, if you are a new user, it will ask you to create an account. After creating this account just type this one like histopathological cancer detection in the search bar histopathological cancer detection.

Here you can identify this particular data set where histopathological cancer detection, you can observe here, these microscopes, under these microscopes this tissue slides this will be kept under glass slides and the glass slides will be taken into observation. After this since the evolvement of digital scanner, these glass slides containing tissues, tissue slices will be scanned and the new histopathology level images will be developed.

(Refer Slide Time: 5:50)

OneNote

Home Insert Draw View

Pen Highlighter Eraser Ink Color

0.25 mm 0.35 mm 0.5 mm 0.7 mm 1 mm

## Key Topics

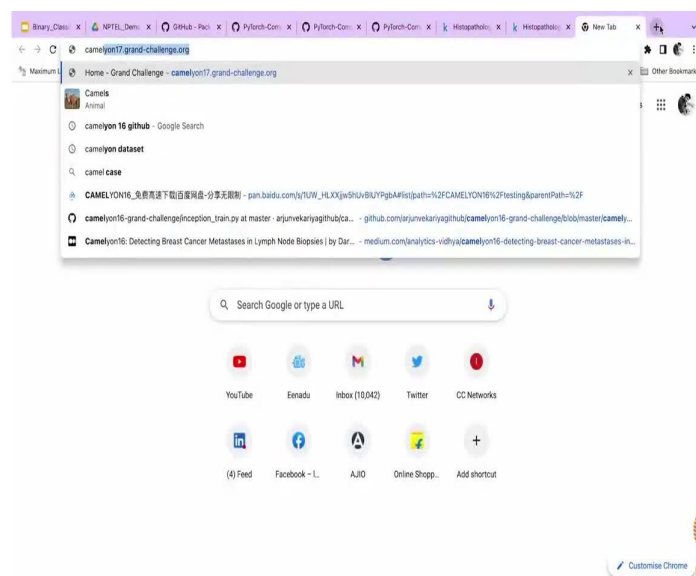
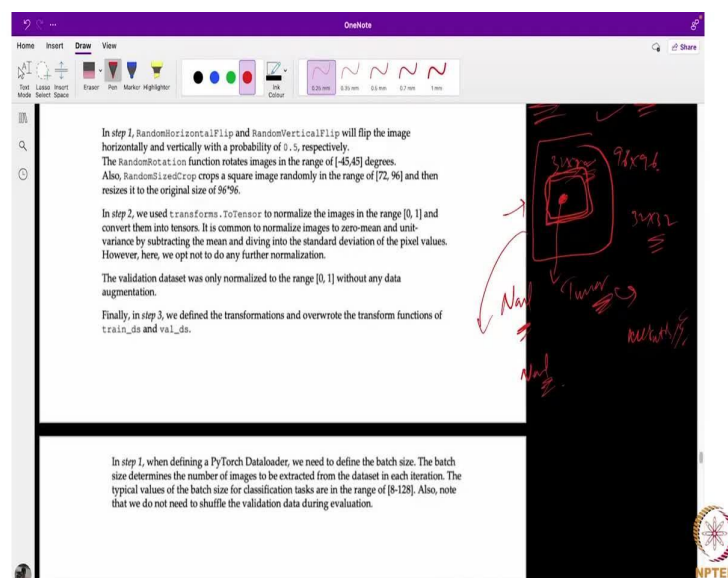
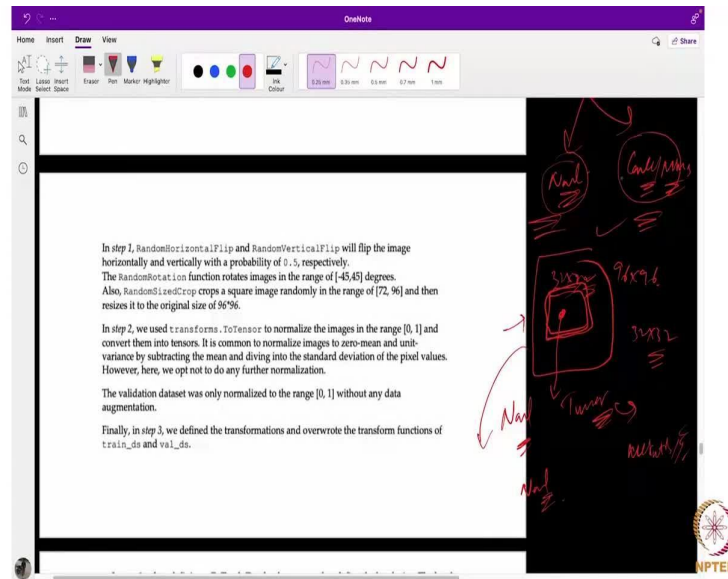
- Exploring the dataset
- Creating a custom dataset
- Splitting the dataset
- Transforming the data
- Creating Data Loaders
- Building the classification model
- Defining the loss function
- Defining the optimizer
- Training and evaluation of the model
- Deploying the model
- Model inference on test data

Hand-drawn diagram on the right side of the slide:

```

graph TD
    Data((Data)) --> Model[Model]
    Model --> Loss[Loss]
    Loss --> Training[Training and evaluation]
    Training --> Deployment[Deployment]
  
```

The diagram is annotated with '9.6x9.6', '10x10', and '10x10' next to different stages of the model. The word 'Data' is circled in red.




Binary\_Class x NPTEL\_Demo x GitHub - P... x PyTorch-Cor... x PyTorch-Cor... x PyTorch-Cor... x Histopatho... x Histopatho... x Home - Gran... x

camelyon17.grand-challenge.org

Grand Challenge Challenges Algorithms

Help Sign In Register

Challenges / CAMELYON17 / Home



Info Leaderboard

Home Background Rules Data Evaluation Organisers Workshop Email organizers

<https://camelyon17.grand-challenge.org/background/>

## Overview

The CAMELYON17 challenge is still open for submissions!

Built on the success of its predecessor, CAMELYON17 is the second grand challenge in pathology organised by the Diagnostic Image Analysis Group (DIAG) and Department of Pathology of the Radboud University Medical Center (Radboudumc) in Nijmegen, The Netherlands.

The goal of this challenge is to evaluate new and existing algorithms for automated detection and classification of breast cancer metastases in whole-slide images of histological lymph node sections. This task has high clinical relevance and would normally require extensive microscopic assessment by pathologists. The presence of metastases in lymph nodes has therapeutic implications for breast cancer patients. Therefore, an automated solution would hold great promise to reduce the workload of pathologists while at the same time reduce the subjectivity in diagnosis.

NPTEL

Binary\_Class x NPTEL\_Demo x GitHub - P... x PyTorch-Cor... x PyTorch-Cor... x PyTorch-Cor... x Histopatho... x Histopatho... x CC Data - Gran... x

camelyon17.grand-challenge.org/Cont...

Grand Challenge Challenges Algorithms

Help Sign In Register

## Accessing the data

The whole-slide images provided in this challenge are standard TIFF files. Standard libraries like `OpenSlide` can be used to open and read these files. We used `ASAP` to prepare the data. Its `multiresolutionimageinterface` C++ library and python package provides an easy to use interface for accessing pixel data in TIFF files efficiently. Only 3 simple steps are necessary to be able to use it in python:

1. Download and install ASAP.
2. Configure your `PYTHONPATH` environment variable to contain the `.bin` directory path.
3. Import `multiresolutionimageinterface` to your python module.

A few things to know about the library and the TIFF image format:

- The TIFF contains multiple down-sampled versions of the original image. The highest resolution is on level 0.
- Pixel values can be read in patches from any available level.
- Patch indexing is done in column, row number.
- Regardless of which level a patch is read from the indexing is done on level 0.

The following python code snippet loads a TIFF file and reads a 300 pixel wide, 200 pixel high image patch starting at the (568, 732) XY coordinate on level 2:

```
import multiresolutionimageinterface as mir
reader = mir.MultiresolutionImageReader()
or_image = reader.open('camelyon17/centre_8/patient_000_node_8.tif')
level = 2
dx = or_image.getLevelDownsampleLevel()
image_patch = or_image.getPatch(int(568 * dx), int(732 * dx), 300, 200, level)
```

## Annotations

The annotations were made in `ASAP`. The annotation ROIs are polygons that are stored as an ordered list of vertex (X, Y) pixel coordinates on level 0 of the multi-resolution images.

The following python code snippet converts an annotation into a mask file:

NPTEL

Binary\_Class x NPTEL\_Demo x GitHub - P... x PyTorch-Cor... x PyTorch-Cor... x PyTorch-Cor... x Histopatho... x Histopatho... x CC Data - Gran... x New Tab x

camelyon16.github

camelyon16.github - Google Search

- camelyon16 dataset
- camelyon1
- camelyon17 dataset
- camelyon16 dataset download
- camelyon17 wilds
- CAMELYON16: 免费高速下载百度网盘-分享无限制 - pan.baidu.com/s/LUW\_HLXXgv9Nj-BUYPgaA#list/path=/32FCAMELYON16%2Ftesting&pageNum=32F
- camelyon16-grand-challenge/inception\_train.py at master · arjunekariyagithub/camelyon16-grand-challenge · GitHub

Search Google or type a URL

YouTube Eemadu Inbox (1,042) Twitter CC Networks

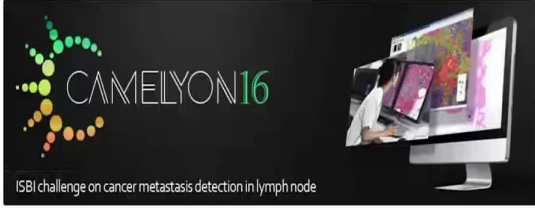
(4) Feed Facebook - L. AJO Online Shopp... Add shortcut

Customise Chrome NPTEL

Grand Challenge Challenges Algorithms

Help Sign In Register

Challenges / CAMELYON16 / Data



ISBI challenge on cancer metastasis detection in lymph node

**Info**

- Home
- Background
- Rules
- Register
- Data**
- Evaluation
- Submit
- Results

The CAMELYON16 challenge has ended in November 2016  
PLEASE CHECK OUT CAMELYON17:  
<https://camedyon17.grand-challenge.org>

The data in this challenge contains a total of 400 whole-slide images (WSIs) of sentinel lymph node from two independent datasets collected in Radboud University Medical Center (Nijmegen, the Netherlands), and the University Medical Center Utrecht (Utrecht, the Netherlands).

**The training dataset**

Waiting for cache...

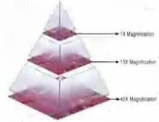
NPTEL

Grand Challenge Challenges Algorithms

Help Sign In Register

The test dataset consists of 130 WSIs which are collected from both Universities.


### Visualizing the images and annotations



Whole-slide images are generally stored in a multi-resolution pyramid

structure. Image files contain multiple downsampled versions of the original image. Each image in the pyramid is stored as a series of tiles, to facilitate rapid retrieval of subregions of the image. Reading these images using standard image tools or libraries is a challenge because these tools are typically designed for images that can comfortably be uncompressed into RAM or a swap file. OpenSlide is a C library that provides a simple interface to read WSIs of different formats. Automated Slide Analysis Platform (ASAP) is an open source platform for visualizing, annotating and automatically analyzing whole-slide histopathology images. ASAP is built on top of several well-developed open source packages like OpenSlide, Qt and OpenCV. We strongly recommend the participants to use this platform for visualizing the slides and viewing the annotations. You can download ASAP from GitHub.

Example of a metastatic region:

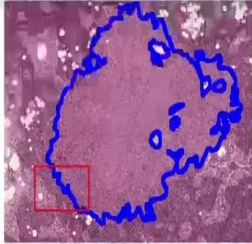
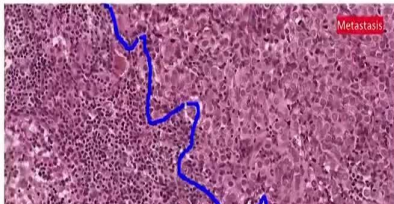


Waiting for cache...

NPTEL

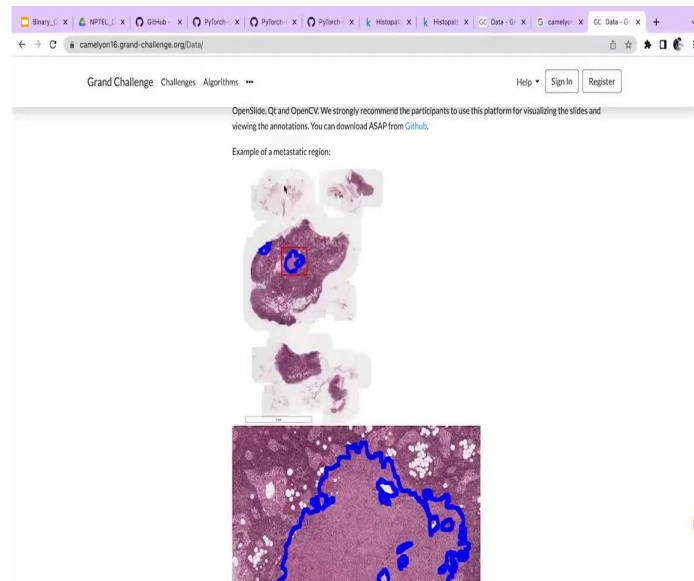
Grand Challenge Challenges Algorithms

Help Sign In Register

Metastasis

NPTEL



Most of the times they will come up with something called PatchCamelyon data where PatchCamelyon data is nothing but it has been extracted patches from camelyon16 data. Camelyon16 data I will show you camelyon17 grand challenge you can see here data, go with this particular data here you can see these histopathology images are like large slide images, so these kinds of images you can see like very huge images.

From this particular camelyon challenge the donor the data named it as PatchCamelyon data it is kind of a go to data like you can measure so many things in related to medical image analysis where instead of from the center 100×100,000 dataset they have extracted patch images like small patches that will show you the entire dataset, patch images were these patch images are like 96×96, you can see they have extracted patches from this entire slide image containing 100,000 by 100,000 pixels.

Now several patches of size 96×96 has been taken out and they have generated this new dataset called PatchCamelyon. In this particular Kaggle competition they have again made some changes into this PatchCamelyon data set, PCam data set where most of the... in this particular Kaggle competition the data set which we are going to see most of the times the center data contains only normal images in cancer images.

If you type in Kaggle you can identify this like this you can see here the data description, you can see here there are 2 folders containing train and test folder. Here you can see test image, I have downloaded the entire data, suppose see here you can see images like this that is the dataset we are going to talk about and they have divided the dataset into something like suppose by taking all these 96 patches.



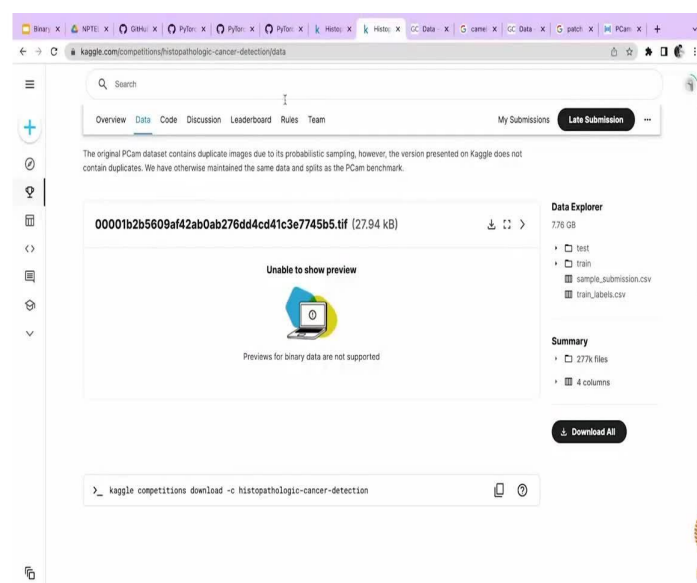
They are converting into something like normal images and cancer metastasis image, where this normal images are declared, suppose if you take some  $96 \times 96$  size image and if you take in between  $32 \times 32$  pixel, that inset is one in case in the center pixels like  $32 \times 32$  pixels in the center pixel if they identify it as a tumor pixel, tumor related pixel then they are considering this one as a metastasis or cancer related image.

I mean, cancer belongs to metastasis. Whereas suppose if this particular belongs pixel belongs to something like a normal only then the center patch has been termed as normal image. In short, this kaggle data whichever data set we have I am going to show for the coding it has been downloaded from or has been extracted from PatchCamelyon data.

Where in this particular data we are having several images all the images are containing only  $96 \times 96$  size where for each  $96 \times 96$  image they are going to name it as normal images and metastasis image. Metastasis is cancer related image.

Metastasis is not like benign it is really cancer only where in this particular pixel if they identify in the  $96 \times 96$  patch in between  $32 \times 32$  if they find out that this particular pixel belongs to normal image then we they have named it as a normal. Suppose this particular pixel belongs to tumor or metastasis, then they have termed it as an enter patch as attach patch. So this is how this has been taken out.

(Refer Slide Time: 11:32)



Google Drive interface showing the 'NPTEL\_Demo\_On\_Binary\_Classification' folder. The folder contains the following files:

Name	Owner	Last modified
train	me	Sep 18, 2022
test	me	Sep 18, 2022
Untitled0.ipynb	me	Sep 18, 2022
sample_submission.csv	me	Dec 12, 2019
kaggle.json	me	Sep 18, 2022
histopathologic-cancer-detection.zip	me	Sep 18, 2022

Storage: 83.59 GB used

Colab interface showing the 'CO' logo and a small red dot, indicating the notebook is loading or running.

```

import os

os.environ['KAGGLE_CONFIG_DIR'] = '/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification'

!kaggle competitions download -c histopathologic-cancer-detection
histopathologic-cancer-detection.zip: Skipping, found more recently modified local copy (use --force to force download)

! unzip \*.zip && rm \*.zip

import os

list_images = os.listdir('/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train')

len(list_images)

109228

import pandas as pd

path_csv = '/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train_labels.csv'
  
```



Google Drive interface showing a folder named "NPTEL\_Demo\_On\_Binary\_Classification". The folder contains the following files:

Name	Owner	Last modified
train	me	Sep 18, 2022
test	me	Sep 18, 2022
Untitled0.ipynb	me	Sep 18, 2022
sample_submission.csv	me	Dec 12, 2019
kaggle.json	me	Sep 18, 2022
histopathologic-cancer-detection.zip	me	Sep 18, 2022

Storage: 83.59 GB used

Kaggle interface showing the "histopathologic-cancer-detection" dataset. The dataset is a CSV file named "sample\_submission.csv" (2.47 MB). The dataset is loaded and displayed in the Data Explorer.

**Data Explorer**

7.76 GB

- test
- train
- sample\_submission.csv
- train\_labels.csv

**Summary**

- 277k files
- 4 columns

[Download All](#)

Terminal output:

```
> kaggle competitions download -c histopathologic-cancer-detection
```

Kaggle profile page for KALLURI RAMAKRISHNA. The profile shows the user's name, location (Chennai, Tamil Nadu, India), and a bio (Research Scholar at IIT MADRAS). The profile is marked as "Competitions Novice".

**KALLURI RAMAKRISHNA**

Research Scholar at IIT MADRAS  
Chennai, Tamil Nadu, India  
Joined 2 years ago · last seen in the past day

[Home](#) [Competitions](#) [Datasets](#) [Code \(22\)](#) [Discussion](#) [Followers](#) [Notifications](#) [Account](#)

[Edit Public Profile](#)

**User Name**

Your username cannot be changed

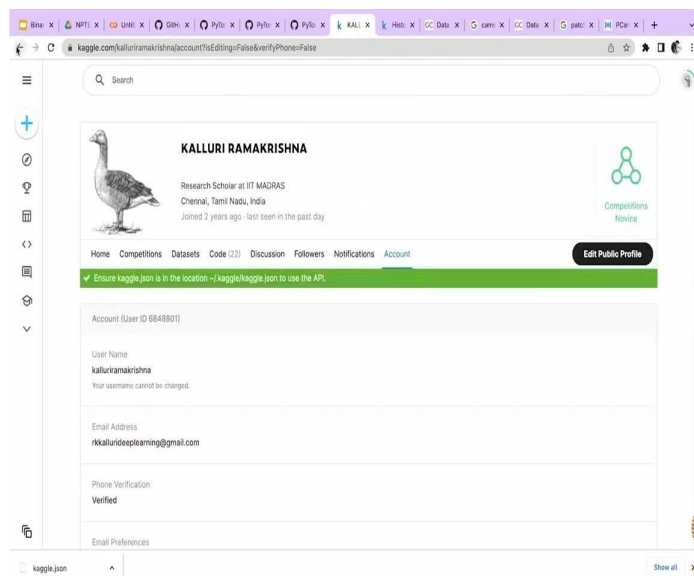
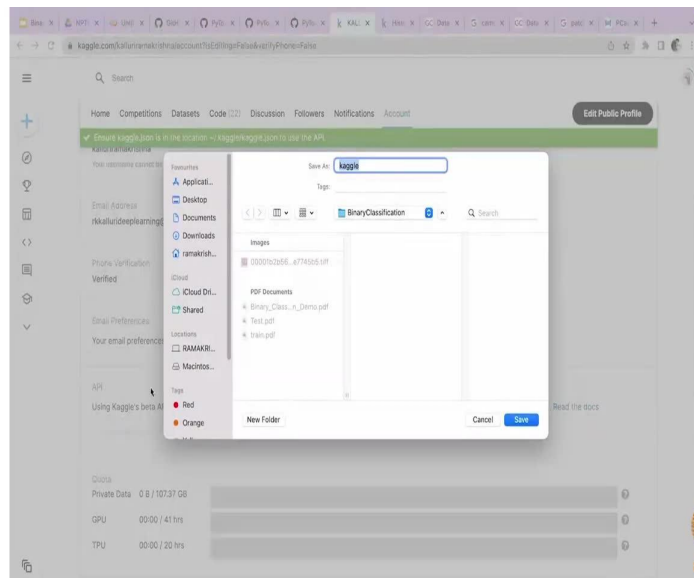
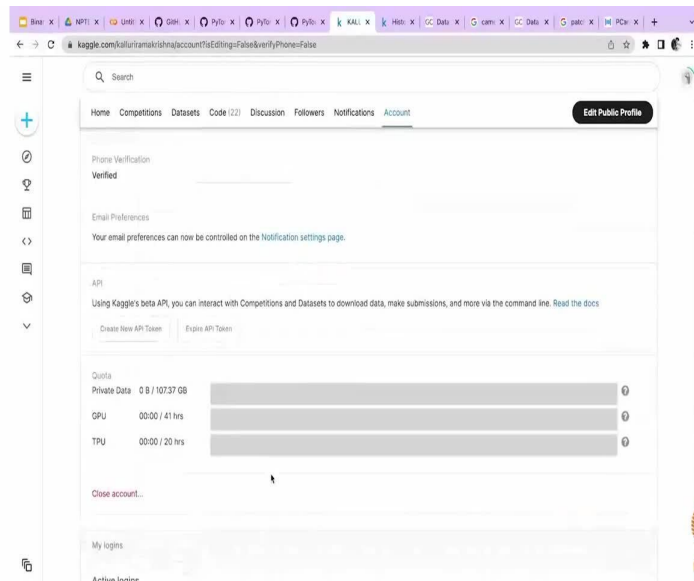
**Email Address**

**Phone Verification**

Not verified +

**Email Preferences**

Your email preferences can now be controlled on the [Notification settings page](#).





```
Untitled0.ipynb
os.environ["KAGGLE_CONFIG_DIR"] = "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification"

!cd "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification"

!kaggle competitions download -c histopathologic-cancer-detection
histopathologic-cancer-detection.zip: Skipping, found more recently modified local copy (use --force to force download)

!unzip *.zip && rm *.zip

import os

list_images = os.listdir('/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train')

len(list_images)

109208

import pandas as pd
```

```
Untitled0.ipynb
os.environ["KAGGLE_CONFIG_DIR"] = "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification"

!cd "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification"

!kaggle competitions download -c histopathologic-cancer-detection
histopathologic-cancer-detection.zip: Skipping, found more recently modified local copy (use --force to force download)

!unzip *.zip && rm *.zip

import os

list_images = os.listdir('/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train')

len(list_images)

109208

import pandas as pd
```

```
Untitled0.ipynb
import pandas as pd

path2train = "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train_labels.csv"
labels_df = pd.read_csv(path2train)
labels_df.head()

print(labels_df['label'].value_counts())

!matplotlib inline
labels_df['label'].hist()

import matplotlib.pyplot as plt
from PIL import Image, ImageDraw
import numpy as np
import os
!matplotlib inline

# data is stored here
path2train = "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train"

# show images in gray-scale, if you want color change it to True
color=False

# get ids for malignant images
malignant_ids = labels_df[labels_df['label'] == 1]['id'].values
```

```
import os

os.environ['KAGGLE_CONFIG_DIR'] = '/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification'

!cd '/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification'

!pwd

!ls -l '/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification'

!kaggle competitions download -c histopathologic-cancer-detection
histopathologic-cancer-detection.zip: Skipping, found more recently modified local copy (use --force to force download)

!unzip '*.zip as rm*.zip'

import os

!ls -l

!ls -l '/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train'

!ls -l

109268
```

sample\_submission.csv (2.47 MB)

7.76 GB

2 of 2 columns

id	label
57458 unique values	0
802e62d822a323f01b155 d62653d889f7d2c8d5	0
9559b02d086c5c3246 6c9b6aff889639f2727	0
2486f78880e6b0c1f25 80c1f32f799eb76914	0
2c35657d312364e024e a0847126f73a7481edf	0
143702e67ca91c518ac3 d2e6d4e9a3731c41f8	0
7254b6e6cc08b958a 2c76d730a3f3a2c7a83	0
a0838786531648868f	0

Summary

Home Competitions Datasets Code (22) Discussion Followers Notifications Account

API

Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. [Read the docs](#)

Create New API Token Expires API Token

Quota

Private Data 0 B / 107.37 GB

GPU 00:00 / 41 hrs

TPU 00:00 / 20 hrs

Close account...

My logins

Active logins

Google email  
rkallurideplearning@gmail.com

My linked accounts

The screenshot shows a Jupyter Notebook with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'Untitled0.ipynb', 'histopathologic-cancer...', 'kaggle.json', 'sample\_submission.csv', 'NPTEL\_Medical\_Image\_A...', 'New\_np161\_breast', 'NoniocalMeans', 'PMRF\_DOCUMENTS', 'PROGRAMMING', 'PROJECT\_NEW', 'Personality\_Development', 'PHD', 'Placement\_Preparation', and 'Project\_Guidance'. The code editor shows the following code:

```
import os

os.environ['KAGGLE_CONFIG_DIR'] = "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification"

! kaggle competitions download -c histopathologic-cancer-detection

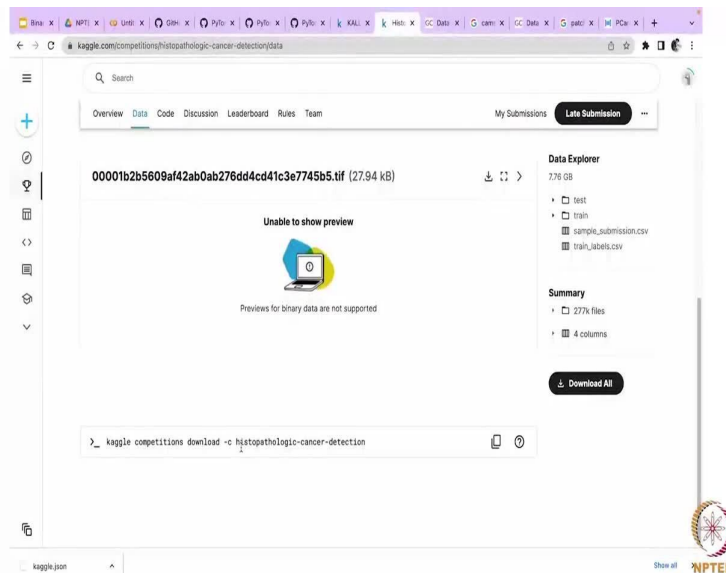
! unzip \*.zip && rm *.zip

import os

list_images = os.listdir('/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train')

len(list_images)
```

The output of the last cell is 109208.



The screenshot shows a Jupyter Notebook with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'Untitled0.ipynb', 'histopathologic-cancer...', 'kaggle.json', 'sample\_submission.csv', 'NPTEL\_Medical\_Image\_A...', 'New\_np161\_breast', 'NoniocalMeans', 'PMRF\_DOCUMENTS', 'PROGRAMMING', 'PROJECT\_NEW', 'Personality\_Development', 'PHD', 'Placement\_Preparation', and 'Project\_Guidance'. The code editor shows the following code:

```
! kaggle competitions download -c histopathologic-cancer-detection

! unzip \*.zip && rm *.zip

import os

list_images = os.listdir('/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train')

len(list_images)
```

The output of the last cell is 109208.

```
import pandas as pd

path_csv = "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train_labels.csv"
labels_df = pd.read_csv(path_csv)
labels_df.head()

print(labels_df['label'].value_counts())

! matplotlib inline
labels_df['label'].hist()
```

The output of the last cell is 109208.

Now first of all, let us go and download this data. Suppose in your Google Drive, just create a particular folder and in this particular folder, you will let these files assume that these files are not here, I just have only Kaggle json file. where exactly assume that I do not have anything else here. I just have only this particular file. This I am saying this all this so that you can directly work on Google Drive itself using this Google colab no need to do any installations in your local computer and everything.

To download the center data set that is available in Kaggle you can directly download into NPTEL where this data set can be seen. Like suppose see train folder is there, test folder is there sample submission CSV. Now if you go to this Kaggle competition, where you have data, here you can see test folder is there train folder is there, train underscore labels dot CSV is also there. All these files will be downloaded, do not worry. I will show you how to download this entire data. First of all, create the Kaggle account.

After creating this Kaggle account, you can go here. Here you can see this account. After going here you can see here, create new API token. Once you click on this one, you will get a Kaggle json file just download this file into a local drive, I mean in your computer after saving this file, after saving this file, whatever I asked you to create a particular folder in this Google colab. Upload this Kaggle json file into this tray.

So assume that we have this Google colab file and only Kaggle json file, after creating this one. My job is to now download this data to do this one, just go ahead and copy this particular API command. Just copy this API command to the clipboard. Then go to this file just type these things.

First import OS next os dot Kaggle directory it is more or less like where exactly you have stored this entire... suppose here I am using here in this drive without any convenient board ways. It is taking so much time... just refresh here, after typing on this mount in Google drive next to the particular directory here I have named it as a NPTEL demo on binary classification, NPTEL demo on binary classification just open this particular directory.

Assume that test train these folders are not present here only you have Untitled, this Python notebook, Jupyter Notebook and only these Kaggle json files are there. This histopathology cancer containing dot zip file is not there sample submission file is not there, these 2 folders are also not that just assume that one.

Out of the suppose I want to this Kaggle environment where exactly I store this Kaggle json file I just go to this particular folder path and you can see the copy path, just type the path here, after giving the path go and type this particular present working directory to CD indicates change directory after changing the directory to it will convert to present working directory if you type that one, you can see this one.

Next our job is to just download this Kaggle data set whatever I asked you to type sorry to copy here. I have already told you to copy this particular command, API command and type that particular command here it is with respect to you. Like competitions download. After this, you will get to see it takes some time to download this data. After downloading the data you can this particular file at least a pathological cancer detection.

That is what this file is. Go and see your drive, you can see histopathological cancer detection that will be placed here. Assume that while I run this program, we have only this particular notebook and only Kaggle json file after running this particular line, this Kaggle competition download C histopathological cancer detection then you can see that this particular zip file will be created there.

After that my job is to just extract this file like histopathological cancer detection dot jpg, so zip file, so that once you extract that one, you will get to see train file test file and submission dot CSV and under this like train levels underscore CSV, all these files will be generated there. So to do that one, what I am doing here, just this is the command I will use just take this one unzip.

Since we are already present in this particular directory only like this folder, so it will go and find whichever zip files are there and the difference will be extracted. It takes some time, just have some patience and run this particular line into your Google colab notebook. Then it is done. Next, after that, you can see all these files if you refresh this folder.

Once the extraction of this default is done, you can see train test, sample submission and train CSV also, labels of the train. Like this is how whatever files I have here, all these things will be provided there hope everybody got it. Just I will repeat the steps just to go into create a Kaggle account. Once creating this Kaggle account, just go and type this one in the search bar like histopathological cancer detection in the search bar.

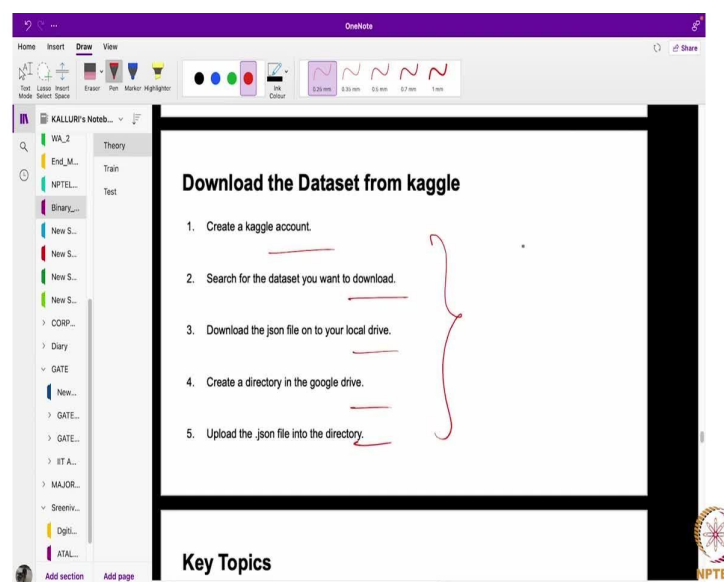


Once you do that one, go and download your Kaggle json file, to download that one just go to this account. After going to this account, here you can see create new API token. Once you take this one, you can see this json file will be downloaded onto your local computer. Once it is downloaded just go and create a folder in Google Drive and upload that particular Kaggle json file into this Google Drive.

After that, you can go to your note... open your Google colab notebook there just follow whatever instructions I have taught there are one more thing to notice is just bring this particular API command. Maybe it gets varied from one competition to another competition. So depending on here, in this particular competition, you can see this particular command here. Just copy a command and upload here. Sorry, just paste this command line here. And you can wait for something.

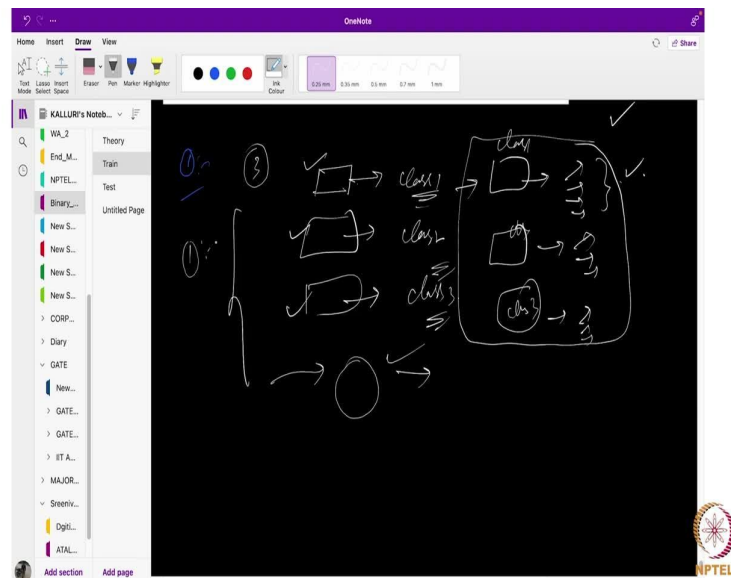
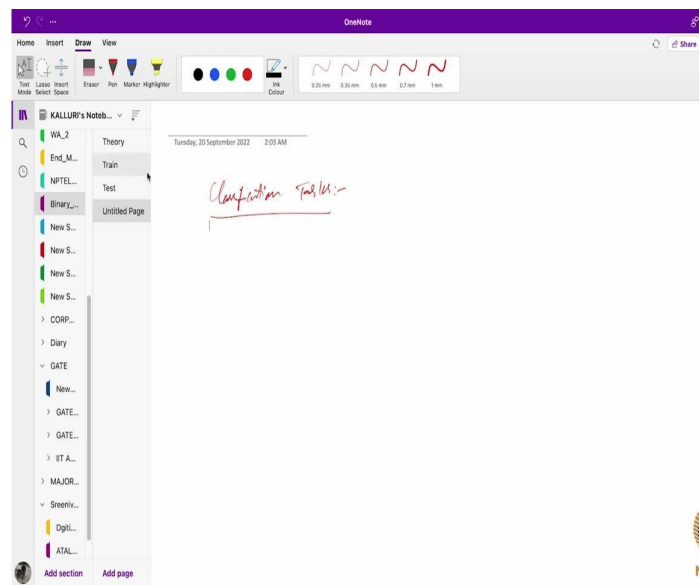
This histopathological that zip will be downloaded and after that it is your responsibility to extract this particular file. Once you extract that file, you can see that these kinds of folders are here, I hope everybody got it. Till now we are clear that we can directly bring the data from Kaggle notebook to sorry Kaggle data set can be directly taken into your Google Drive after taking into Google Drive. Now, the dataset has been downloaded. Now our job is to develop algorithm to classify entire images whatever is given to us.

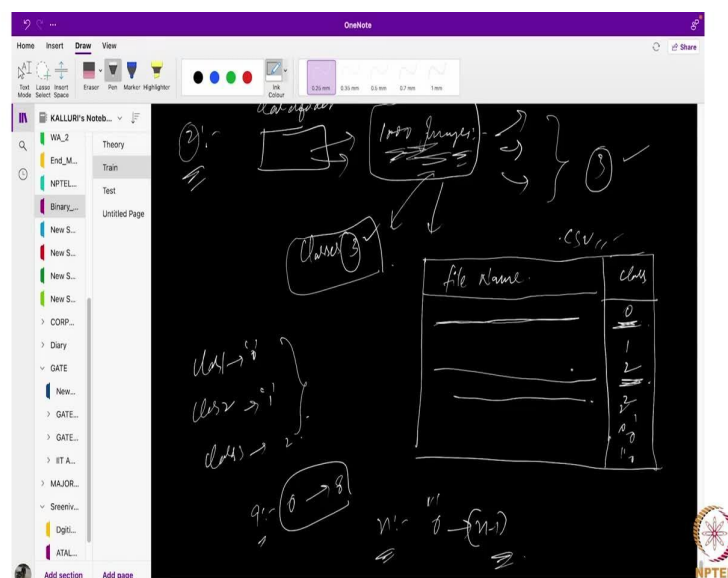
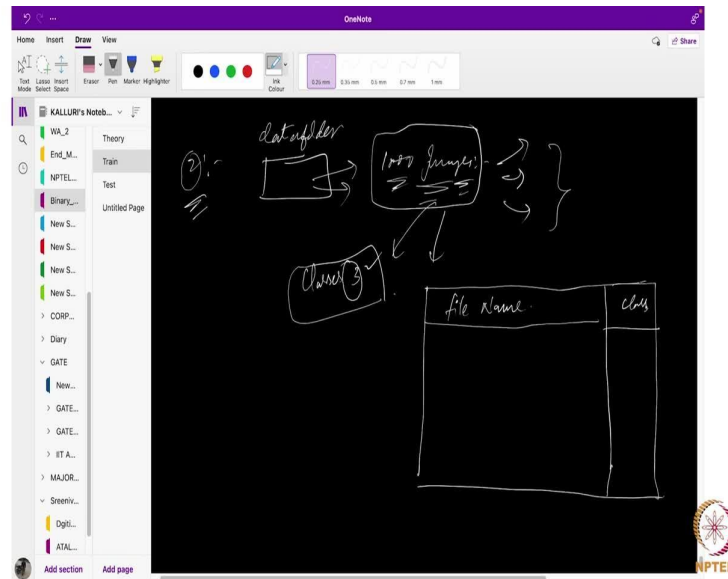
(Refer Slide Time: 20:41)



Till now we discussed about how to download Kaggle data set and creating a folder and how to load the entire Kaggle dataset into our Google Drive.

(Refer Slide Time: 21:01)





Now, let me tell you how they usually, how exactly they made classification tasks in general. There are 2 ways usually the agenda cross presents X exactly in what way problems are given. In that I have some 3 classes then I have a folder containing all the images related to class one. Similarly we have another folder containing all the images that are related to class 2 and another folder that contains all the images that are related to class 3.

Sometimes they will give the class name as a folder name here assume that here suppose I have a folder name is class one only and in side this folder, I will keep all the images that are related to class one. Similarly, in the second folder, I will keep all the images related to class 2. Similarly, I will keep all the class 3 images in a folder. This is how usually classification data problems are given.

Likewise, we will be given 3 folder containing 3 classes of data set and they will ask us to prepare a model that will you take the new input and predict that particular class that is how this is one way of defining the problem. Second thing is we will be given an entire folder assume that it is a data folder. In this particular folder assume that there are 1000 images.

Out of these 1000 images each and every image belongs to a particular class, assume that we have 3 classes, assume that I have 3 classes. That means, I will have here all the 1000 images, every image belongs to one of these classes, in a single folder, we have the entire training data set, I mean all the images where every individual image belongs to one of these particular classes.

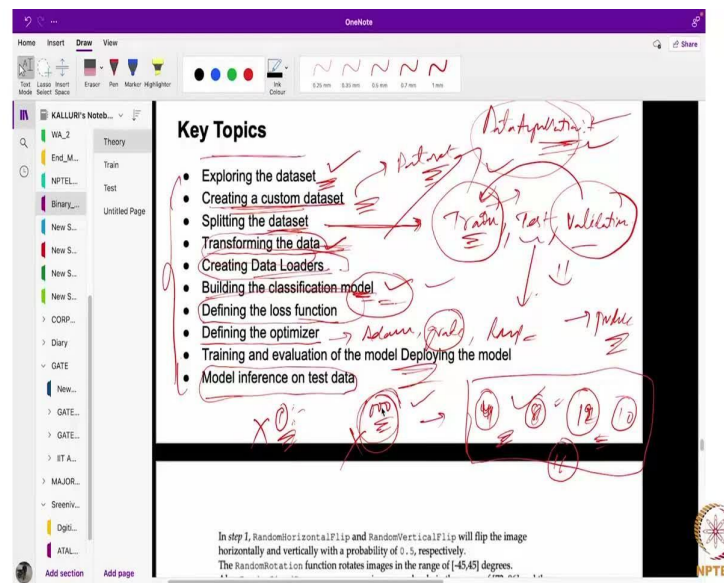
Apart from that, we will also be given a CSV file or an Excel file that contents like this. Here on the left side name of the image, I mean file name of the image and here we will be given class. In general, all the Pytorch frameworks or TensorFlow frameworks mostly assume that class 1 belongs to 0 index class 2 belongs to 1 index similarly, class 3 belongs to suppose I have some 9 classes at the time I will have 0 to 8 numbers.

Suppose  $n$  is the my class number then class index or class belongs to 0 to  $n-1$ , that is so, suppose in this accelerator CSV file, you can see that you can see the image name here image name of the file here and to which class in particular class belongs to this particular image belongs to. Let us assume that here I have a folder containing 1000 images out of a single I will take one image here, that particular file name will be provided here.

At the same time class also will be provided here. Assume that here in this discussion I am talking about 3 classes. So, all the column values will be something like 1 0 1 2 or 21001020 likewise lecture we will be having file names of the images. This is the usual way in which you will be given classification problems this is where we are you will be having each folder containing images, all the images.

I mean class 1 images will actually a particular folder classroom 2 belongs to particular folder classroom 3 belongs to a particular folder. Similarly, here you will be having a single folder containing all the images and at the same time you will be having some CSV file where the CSV file contains file name like imagine and the particular category to which it belongs. This is how the data set is provided either this way or this way. We will get back to how exactly we should plan this particular classification problem.

Refer Slide Time: 25:57)



In general, any deep learning task especially like this classification task, we are just going to follow these steps, these are the key steps that should be followed by anyone by solving this particular classification problem first one is exploring the datasets. Second one is creating custom data set, splitting the dataset splitting the data set means, you just have to split the data into train data, test data, validation data.

Because we will train the model using this train data, while training the model, we have to tune the hyper parameters in such a way that model should not be trapped by either overfitting problem or underfitting problems that is where while training the model we will use validation data by using training data and validation data, we will come up with a particular deep learning model.

Once the entire training is done, we use this test data to produce the generalization about this particular model of what error this particular model works. Next thing is first thing is exploring the dataset. Second thing is creating a custom data set, I will tell you what exactly this means mean by this one, custom data set, splitting the data set, transforming the data set. In general, while we are doing deep learning problems.

Most of the times models are prone to overfitting problems to tackle this one we are going to apply data augmentation tasks that means suppose I have some 100 images, I just want to make them images into something like 1000 images, I will use some basic image processing

techniques, where I will apply those image processing techniques on these images and I will generate new images.

Likewise, whatever data I have, I can extend it to some more data that is the main goal of data augmentation using this transformation transforming the data you can use that augmentation step. Next thing is creating DataLoaders. In general, I will tell you each and every step just instead of suppose I have some 1000 data points, instead of feeding these all these 1000 data points into the model we just provide this data in terms of some chunks like 4 every time, like 8 images every time, like 12 images every time or 10 images or 16.

Likewise we divide it to, we will divide the entire data set into several batches we call, instead of because GPU will be having some limitations to deal the data memory. Usually people go for either feeding each and every data point or all the data points at an instant, but these 2 both these strategies are bad, you just have to go to feed the data in terms of batches, this is very good step.

Next thing is building the classification model this is where our CNN algorithms we will use how to design here we have just used a single model sequential model. Next thing is defining the loss function what exact loss function we will use like stochastic gradient or Adam RRMS prop something like that.

While defining the optimizer we will use something like Adam or gradient descent or stochastic gradient descent or RMS prop like that. When defining the loss function, depending on the problem, if I am dealing with a linear regression problem, I will go for mean squared error loss value. Now, when it comes to classification problems, I will go with either binary cross entropy or cross entropy loss.

Binary cross entropy is supposed you wear 2 glasses cross entropy in general for m classes. Next thing is training and evaluation of the model deploying the model. I will train the model and I will deploy the model. Next thing is that last step is model inference on test data, whatever test data we have, we will use that model, I will feed that data into the model and everything will be done.

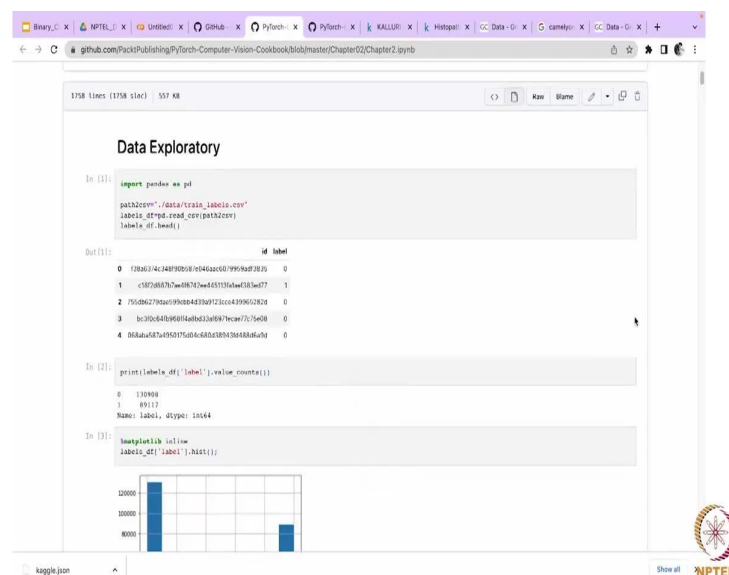
I will generate the testing error that is it. This is the whole program, this is the 4 steps. Like first we have to export the data set, second thing is creating a custom data set, splitting the data set train, test validation transforming the data most of the times belongs to data

augmentation task, creating DataLoaders, we have to load the data in terms of batches instead of loading the entire data at an instant.

Next we will go for designing the building classification model. Third thing, next thing is like defining the loss function like if depending on the task, it is a classification task we will go for cross entropy or something like binary cross entropy. Now, coming to optimizer we will go for Adam or gradient decent or something like RMS prop.

Now, coming back to training and evolution of the deploying model. We will train the model using the train data and validation DataLoaders. Next thing is model inference on the test data then we will test the model using test data set. This is the whole idea of the center project.

(Refer Slide Time: 31:20)



Google Drive interface showing the contents of a folder named "NPTEL\_Demo\_On\_Binary\_Classification".

Name	Owner	Last modified
train	me	Sep 18, 2022
test	me	Sep 18, 2022
Untitled0.ipynb	me	1:35 AM
sample_submission.csv	me	Dec 12, 2019
kaggle.json	me	Sep 18, 2022
histopathologic-cancer-detection.zip	me	Sep 18, 2022

Storage: 83.59 GB used

Colab interface showing the code for loading and displaying a histopathologic image.

```

import matplotlib.pyplot as plt
import PIL
import Image, ImageDraw
import numpy as np
import os
matplotlib.interactive(True)

# data is stored here
path_train = "/content/drive/MyDrive/NPTEL_Demo_On_Binary_Classification/train"

# show images in gray-scale, if you want color change it to True
color = False

# get ids for malignant images
malignant_ids = labels_df[labels_df['label'] == 1].values

plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.imshow(img)
plt.show()

for i, id in enumerate(malignant_ids):
    full_filename = os.path.join(path_train, id + ".tif")

    # load image
    img = Image.open(full_filename)

    # draw a 12x12 rectangle
    draw = ImageDraw.Draw(img)
    draw.rectangle([12, 12, 64, 64], outline="green")
  
```

Execution completed at 1:35 AM

GitHub interface showing the code for data exploration.

```

import pandas as pd

path_csv = "/data/train_labels.csv"
labels_df = pd.read_csv(path_csv)

# Data Exploration

# Get the first 5 rows
labels_df.head()

# Print the shape of the dataset
labels_df.shape

# Print the unique values of the 'label' column
labels_df['label'].value_counts()

# Print the data types of the columns
labels_df.dtypes
  
```

Execution completed at 1:35 AM



Binary\_C X NPTEL X Untitled X GitHub X PyTorch X KALLURI X Histopari X CC Data-G X camelyon X CC Data-G X

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter2.ipynb

### Data Exploratory

```
In [1]: import pandas as pd

path2train = "/data/train_labels.csv"
labels_df = pd.read_csv(path2train)
labels_df.head()
```

```
Out[1]:
```

	id	label
0	f3ba6374c348f9d08b6d4a4ac6b7959baf3835	0
1	c1972887b7a498742ae44373f4af383ac77	1
2	755db6278d999b0a43baf23c0e43996326d	0
3	bc30c645d968f4ab0d3a687acae7a79e08	0
4	00bab07b495075d4c685d894364886a5d	0

```
In [2]: print(labels_df['label'].value_counts())
```

```
0    130488
1     89117
Name: label, dtype: int64
```

```
In [3]: import matplotlib
labels_df['label'].hist()
```

kaggle.json

Show all NPTEL

Binary\_C X NPTEL X Untitled X GitHub X PyTorch X KALLURI X Histopari X CC Data-G X camelyon X CC Data-G X

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter2.ipynb

```
# show images in gray-scale, if you want color change it to True
color=False

# get ids for malignant images
malignant_ids = labels_df[labels_df['label']==1]['id'].values

plt.rcParams['figure.figsize'] = (16.0, 16.0)
plt.subplots_adjust(wspace=0, hspace=0)
rows, cols = 1, 1
for i, id in enumerate(malignant_ids[:rows*cols]):
    full_filename = os.path.join(path2train, id + '.tif')

    # load image
    img = image.open(full_filename)

    # draw a 32x32 rectangle
    draw = ImageDraw.Draw(img)
    draw.rectangle([12, 12], (64, 64), outline='green')

    plt.subplot(rows, cols, i+1)
    if color is True:
        plt.imshow(np.array(img))
    else:
        plt.imshow(np.array(img)[:,:64,:64], cmap='gray')
    plt.axis('off')
```

kaggle.json

Show all NPTEL

Binary\_C X NPTEL X Untitled X GitHub X PyTorch X KALLURI X Histopari X CC Data-G X camelyon X CC Data-G X

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter2.ipynb

```
In [4]: import matplotlib.pyplot as plt
draw = plt.imshow(imgs, cmap=draw)
import numpy as np
import os
import matplotlib

# data is stored here
path2train = "/data/train/"

# show images in gray-scale, if you want color change it to True
color=False

# get ids for malignant images
malignant_ids = labels_df[labels_df['label']==1]['id'].values

plt.rcParams['figure.figsize'] = (16.0, 16.0)
plt.subplots_adjust(wspace=0, hspace=0)
rows, cols = 1, 1
for i, id in enumerate(malignant_ids[:rows*cols]):
    full_filename = os.path.join(path2train, id + '.tif')

    # load image
    img = image.open(full_filename)

    # draw a 32x32 rectangle
    draw = ImageDraw.Draw(img)
    draw.rectangle([12, 12], (64, 64), outline='green')

    plt.subplot(rows, cols, i+1)
    if color is True:
        plt.imshow(np.array(img))
    else:
        plt.imshow(np.array(img)[:,:64,:64], cmap='gray')
    plt.axis('off')
```

kaggle.json

Show all NPTEL

Binary\_C X NPTEL X Untitled X GitHub X PyTorch X PyTorch X KALLURI X Histopath X CC Data-G X Camelyon X CC Data-G X

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter2.ipynb

```
# show images in gray-scale, if you want color change it to True
color=False

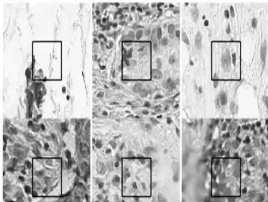
# get ids for malignant images
malignantIds = labels_df.loc[labels_df['label']=='1']['id'].values

plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
rows, ncols=3,3
for i, id in enumerate(malignantIds[:rows*ncols]):
    full_filename = os.path.join(path_train, id + ".tif")

    # load image
    img = image.open_full_filename(full_filename)

    # draw a 2x2 rectangle
    draw = ImageDraw.Draw(img)
    draw.rectangle([115, 12], [64, 64], outline="green")

    plt.subplot(rows, ncols, i+1)
    if color is True:
        plt.imshow(np.array(img))
    else:
        plt.imshow(np.array(img)[:,:,0], cmap="gray")
    plt.axis('off')
```



kaggle.json

Show all NPTEL

Binary\_C X NPTEL X Untitled X GitHub X PyTorch X PyTorch X KALLURI X Histopath X CC Data-G X Camelyon X CC Data-G X

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter2.ipynb

```
# show images in gray-scale, if you want color change it to True
color=False

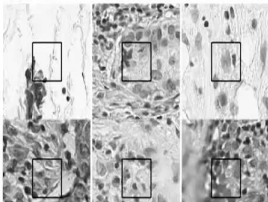
# get ids for malignant images
malignantIds = labels_df.loc[labels_df['label']=='1']['id'].values

plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
rows, ncols=3,3
for i, id in enumerate(malignantIds[:rows*ncols]):
    full_filename = os.path.join(path_train, id + ".tif")

    # load image
    img = image.open_full_filename(full_filename)

    # draw a 2x2 rectangle
    draw = ImageDraw.Draw(img)
    draw.rectangle([115, 12], [64, 64], outline="green")

    plt.subplot(rows, ncols, i+1)
    if color is True:
        plt.imshow(np.array(img))
    else:
        plt.imshow(np.array(img)[:,:,0], cmap="gray")
    plt.axis('off')
```



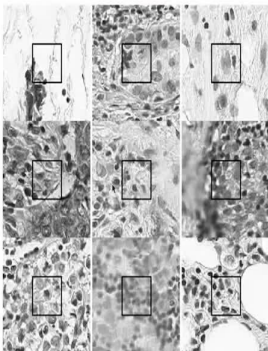
kaggle.json

Show all NPTEL

Binary\_C X NPTEL X Untitled X GitHub X PyTorch X PyTorch X KALLURI X Histopath X CC Data-G X Camelyon X CC Data-G X

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter2.ipynb

```
plt.imshow(np.array(img)[:,:,0], cmap="gray")
plt.axis('off')
```

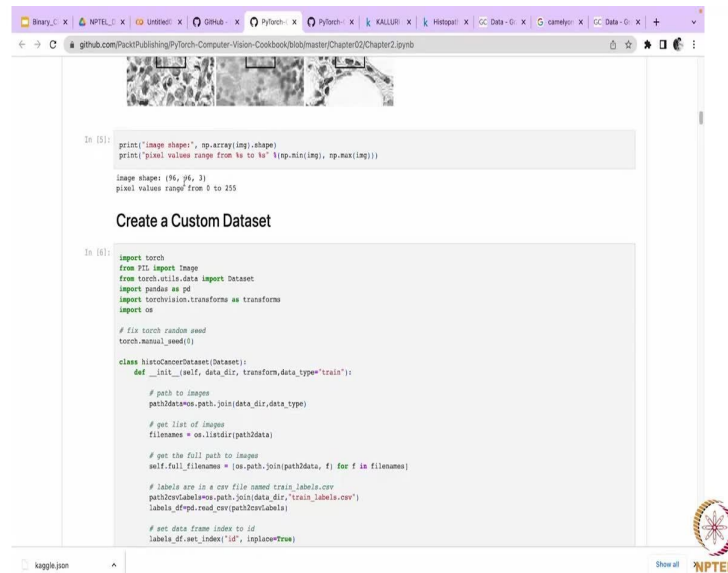


```
In [5]:
print("image shape:", np.array(img).shape)
print("pixel values range from %s to %s" % (np.min(img), np.max(img)))

image shape: (64, 64, 3)
pixel values range from 0 to 255
```

kaggle.json

Show all NPTEL



```

In [5]:
print('image shapes', np.array(img).shape)
print('pixel values range from %s to %s' % (np.min(img), np.max(img)))

image shapes: (96, 96, 3)
pixel values range from 0 to 255

Create a Custom Dataset

In [6]:
import torch
from PIL import Image
from torch.utils.data import Dataset
import pandas as pd
import torchvision.transforms as transforms
import os

# fix torch random seed
torch.manual_seed(1)

class CustomDataset(Dataset):
    def __init__(self, data_dir, transform, data_type='train'):
        # path to images
        path_images = os.path.join(data_dir, data_type)

        # get list of images
        filenames = os.listdir(path_images)

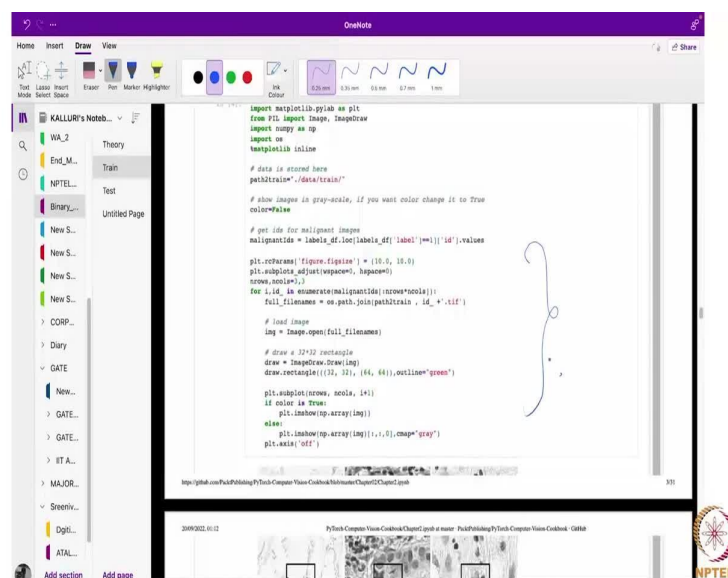
        # get the full path to images
        self.full_filenames = [os.path.join(path_images, f) for f in filenames]

        # labels are in a csv file named train_labels.csv
        path_csv_labels = os.path.join(data_dir, 'train_labels.csv')
        labels_df = pd.read_csv(path_csv_labels)

        # get data frame index to id
        labels_df['id'] = labels_df.index

kaggle.json

```



```

import matplotlib.pyplot as plt
from PIL import Image, ImageDraw
import numpy as np
import os
import matplotlib

# data is stored here
path_train = './data/train/'

# show images in gray-scale, if you want color change it to True
color=False

# get ids for malignant images
malignant_ids = labels_df[labels_df['label']=='1']['id'].values

plt.subplots(figsize=(10, 10))
plt.subplots_adjust(wspace=0.5, hspace=0.5)
rows, cols = 2, 2
for i, id in enumerate(malignant_ids):
    full_filename = os.path.join(path_train, id + '.tif')

    # load image
    img = Image.open(full_filename)

    # draw a 2x2 rectangle
    draw = ImageDraw.Draw(img)
    draw.rectangle([10, 10, 60, 60], outline='green')

    plt.subplot(rows, cols, i+1)
    if color is True:
        plt.imshow(np.array(img))
    else:
        plt.imshow(np.array(img)[:, :, 0], cmap='gray')
    plt.axis('off')

```

Now let us get back to the code. In your particular Google Drive, just open a Google colab notebook like this. Our main job is to first give a path of this particular folder. Suppose here everything, every step I will describe here. First thing is data exploration step where in this particular data exploration step I just imported pandas document. Since in the train labels dot CSV file I will be given the labels of each and every image.

So, what I do is I just use these commands like path to CSV file. Suppose assume that here in this Google colab file I am using just go to the stage like suppose to give the path data to CSV file just go to the file name and use the copy path there you can take this one. That is the enter training code, here you can see just this is to just load the dataset and you can see here we are given the file name accordingly will be given the... to each class, suppose if it is normal it is 0 if it is binary we can say this one is 1.

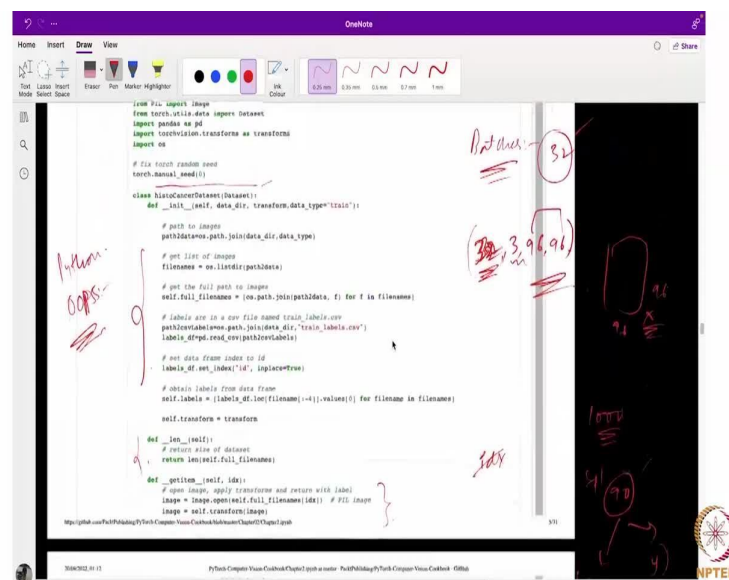
Next thing is I am counting how many 0's are there and how many class ones are there, like 0 classes or normal class containing images and one class containing images. We just have done the histogram plot 0's containing some these many number and one is containing the 8000 something you can see here 8900 something, 8 - 9000 sorry.

Now coming back to here, since I have the folder full of images, I just want to enter into this particular folder and draw the images here. Import matplotlib image draw, you just have to go through each and every line and usually to draw the images or to visualize the images we will go for matplotlib. You can see here in the code if you see you can see the gray images here, but in the code you have to write something like...

See instead of color is equal to false if I type here as true, you can generate all the color images here. Here I am just exploring the training data set containing how the images are looking like. All the images are in dot tif format. Similarly, we are just observing for a single image if you want to generate color images just take instead of false just true. Then you can see the images in color.

Next thing is image shape. Image shape if you observe  $96 \times 96 \times 3$  where as the maximum value of the pixel is like 255, minimum values of the pixel is 0 because it is in integer format.

(Refer Slide Time: 35:13)



```
from cv2 import imgproc
from cv2 import imgproc
import pandas as pd
import cv2
import matplotlib.pyplot as plt
import os

# fix torch random seed
torch.manual_seed(1)

class ImageDataset(torch.utils.data.Dataset):
    def __init__(self, data_dir, transform, data_type='train'):
        self.data_dir = data_dir
        self.transform = transform

        # path to images
        path_to_images = os.path.join(data_dir, data_type)

        # get list of images
        filenames = os.listdir(path_to_images)

        # get the full path to images
        self.full_filenames = [os.path.join(path_to_images, f) for f in filenames]

        # labels are in a csv file named train_labels.csv
        path_to_labels = os.path.join(data_dir, 'train_labels.csv')
        labels_df = pd.read_csv(path_to_labels)

        # set data frame index to id
        labels_df.set_index('id', inplace=True)

        # obtain labels from data frame
        self.labels = [labels_df.loc[filename].values[0] for filename in self.full_filenames]

        self.transform = transform

    def __len__(self):
        # return size of dataset
        return len(self.full_filenames)

    def __getitem__(self, idx):
        # open image, apply transform and return with label
        image = imgproc.imread(self.full_filenames[idx]) # PIL image
        image = self.transform(image)

        label = self.labels[idx]
```

Handwritten notes in red ink on the right side of the code:

- Not done
- 3, 96, 96
- 1000



```
from torch.utils.data import random_split
len_histo=len(histo_dataset)
len_train=int(0.8*len_histo)
len_val=len_histo-len_train
train_ds,val_ds=random_split(histo_dataset,[len_train,len_val])
print('train dataset length', len(train_ds))
print('validation dataset length', len(val_ds))

train_dataset = torch.utils.data.DataLoader(train_ds)
validation_dataset = torch.utils.data.DataLoader(val_ds)

train_dataset.length()
validation_dataset.length()

for x,y in train_ds:
    print(x.shape,y)
    break

torch.Size([3, 96, 96]) 0

for x,y in val_ds:
    print(x.shape,y)
    break

torch.Size([3, 96, 96]) 0

from torchvision import utils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
np.random.seed(0)

def show(img,color=False):
    # convert tensor to numpy array
    ximg = img.numpy()

    # Convert to B*W*C shape
    ximg = ximg.transpose(1,2,0)

    if color=False:
        ximg = ximg[0]
        plt.imshow(ximg,interpolation='nearest',cmap='gray')
    else:
        # display images
        plt.imshow(ximg,interpolation='nearest')
        plt.title('label: %s'%str(y))

grid_size=4
rnd_idx=np.random.randint(0,len(train_ds),grid_size)
print('image indices:',rnd_idx)

x_grid=train[train_ds][0] for i in rnd_idx
y_grid=train[train_ds][1] for i in rnd_idx

x_grid=torch.utils.make_grid(x_grid_train,ncol=4,padding=2)
print(x_grid_train.shape)

plt.figure(figsize=(10,5))
show(x_grid_train,y_grid_train)

image_indices = [43547 173485 117952 152315]
torch.Size([3, 100, 394])

label [0.0.0.1]
```

```
for x,y in val_ds:
    print(x.shape,y)
    break

torch.Size([3, 96, 96]) 0

In [13]:
from torchvision import utils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
np.random.seed(0)

def show(img,color=False):
    # convert tensor to numpy array
    ximg = img.numpy()

    # Convert to B*W*C shape
    ximg = ximg.transpose(1,2,0)

    if color=False:
        ximg = ximg[0]
        plt.imshow(ximg,interpolation='nearest',cmap='gray')
    else:
        # display images
        plt.imshow(ximg,interpolation='nearest')
        plt.title('label: %s'%str(y))

grid_size=4
rnd_idx=np.random.randint(0,len(train_ds),grid_size)
print('image indices:',rnd_idx)

x_grid=train[train_ds][0] for i in rnd_idx
y_grid=train[train_ds][1] for i in rnd_idx

x_grid=torch.utils.make_grid(x_grid_train,ncol=4,padding=2)
print(x_grid_train.shape)

plt.figure(figsize=(10,5))
show(x_grid_train,y_grid_train)

image_indices = [43547 173485 117952 152315]
torch.Size([3, 100, 394])

label [0.0.0.1]
```

```
print('image indices:',rnd_idx)
x_grid=train[train_ds][0] for i in rnd_idx
y_grid=train[train_ds][1] for i in rnd_idx
x_grid=torch.utils.make_grid(x_grid_train,ncol=4,padding=2)
print(x_grid_train.shape)

plt.figure(figsize=(10,5))
show(x_grid_train,y_grid_train)

image_indices = [43547 173485 117952 152315]
torch.Size([3, 100, 394])

label [0.0.0.1]
```

```
In [14]:
grid_size=4
rnd_idx=np.random.randint(0,len(val_ds),grid_size)
print('image indices:',rnd_idx)
x_grid=val[val_ds][0] for i in rnd_idx
y_grid=val[val_ds][1] for i in rnd_idx
x_grid=torch.utils.make_grid(x_grid_val,ncol=4,padding=2)
print(x_grid_val.shape)

show(x_grid_val,y_grid_val)

image_indices = [30403 32103 41993 20757]
torch.Size([3, 100, 394])

label [0.1.0.0]
```

OneNote

Home Insert Draw View

Test Less Short Mode Select Space

Eraser Pen Marker Highlighter Ink Colour

0.20 mm 0.30 mm 0.5 mm 0.7 mm 1 mm

```

In [12]: for x, y in val_d:
        print(x.shape, y)
        break

torch.Size([3, 96, 96]) 0

In [13]: from torchvision import utils
import numpy as np
import matplotlib.pyplot as plt
matplotlib.use('agg')
np.random.seed(0)

def show_img, color=False:
    # convert tensor to numpy array
    nimg = img.numpy()

    # Convert to HWC shape
    nimg = nimg.transpose((1, 2, 0))

    if color==False:
        nimg = nimg.astype('uint8')
        plt.imshow(nimg, interpolation='nearest', cmap='gray')
    else:
        # display image
        plt.imshow(nimg, interpolation='nearest')
        plt.title('val')
        plt.show()

grid_x_val =
rand_idx = np.random.randint(0, len(train_d), grid_x_val)
print('image indices:', rand_idx)
x_grid_train = train_d[rand_idx]
y_grid_train = train_d[rand_idx]
x_grid_val = val_d[rand_idx]
y_grid_val = val_d[rand_idx]
x_grid_train = make_grid(x_grid_train, nrow=4, padding=5)
print(x_grid_train.shape)
plt.rcParams['figure.figsize'] = (10, 5)
show_img(x_grid_train, y_grid_train)

```

100%  
hist. dist. → train is 80%  
val is 20%

To view  
for all train data

NPTEL

OneNote

Home Insert Draw View

Test Less Short Mode Select Space

Eraser Pen Marker Highlighter Ink Colour

0.20 mm 0.30 mm 0.5 mm 0.7 mm 1 mm

```

In [14]: grid_x_val =
rand_idx = np.random.randint(0, len(val_d), grid_x_val)
print('image indices:', rand_idx)
x_grid_val = val_d[rand_idx]
y_grid_val = val_d[rand_idx]
x_grid_val = make_grid(x_grid_val, nrow=4, padding=5)
print(x_grid_val.shape)
show_img(x_grid_val, y_grid_val)

Image indices: [2043 32103 41993 20757]
torch.Size([4, 192, 192])
label: [0.1 0.0]

```

Transform Data

2019/02/01 12:12 PyTorch Computer Vision Cookbook Chapter 02: Data Loading

```

In [15]: train_transformer = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomRotation(45),
        transforms.RandomResizedCrop(96, scale=(0.8, 1.0), ratio=(1.0, 1.0)),
        transforms.ToTensor()])

val_transformer = transforms.Compose([transforms.ToTensor()])

# overwrite the transform functions
train_d.transform = train_transformer
val_d.transform = val_transformer

```

NPTEL

OneNote

Home Insert Draw View

Test Less Short Mode Select Space

Eraser Pen Marker Highlighter Ink Colour

0.20 mm 0.30 mm 0.5 mm 0.7 mm 1 mm

```

In [16]: train_transformer = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomRotation(45),
        transforms.RandomResizedCrop(96, scale=(0.8, 1.0), ratio=(1.0, 1.0)),
        transforms.ToTensor()])

val_transformer = transforms.Compose([transforms.ToTensor()])

# overwrite the transform functions
train_d.transform = train_transformer
val_d.transform = val_transformer

Create DataLoader

In [18]: from torch.utils.data import DataLoader
train_d = DataLoader(train_d, batch_size=32, shuffle=True)
val_d = DataLoader(val_d, batch_size=4, shuffle=False)

In [19]: # extract a batch from training data
for x, y in train_d:
    print(x.shape)
    print(y.shape)
    break

torch.Size([32, 3, 96, 96])
torch.Size([32])

In [20]: # extract a batch from validation data
for x, y in val_d:
    print(x.shape)
    print(y.shape)
    break

```

train data  
val data  
Test data

NPTEL



**Create Dataloader**

```

In [18]: from torch.utils.data import DataLoader
train_dataloader = DataLoader(train_data_loader, batch_size=32, shuffle=True)
val_dataloader = DataLoader(val_data_loader, batch_size=32, shuffle=False)

In [19]: # extract a batch from train data
for x, y in train_dataloader:
    print(x.shape)
    print(y.shape)
    break

torch.Size([32, 3, 96, 96])
torch.Size([32])

In [20]: # extract a batch from validation data
for x, y in val_dataloader:
    print(x.shape)
    print(y.shape)
    break

```

*Handwritten notes:* True - shuffle, False - no shuffle, 32, torch.Size([32, 3, 96, 96])

**Building Classification Model**

**dumb baselines**

```

In [21]: # get labels for validation dataset
y_val = y for x, y in val_dataloader

def accuracy(labels, out):
    return np.sum(out == labels) / float(len(labels))

# accuracy all zero predictions
acc_all_zero = accuracy(y_val, np.zeros_like(y_val))

# accuracy all ones predictions
acc_all_ones = accuracy(y_val, np.ones_like(y_val))

# accuracy random predictions
acc_random = accuracy(y_val, np.random.randint(0, size=len(y_val)))

print("accuracy random predictions: %.2f" % acc_random)
print("accuracy all zero predictions: %.2f" % acc_all_zero)
print("accuracy all ones predictions: %.2f" % acc_all_ones)

accuracy random predictions: 0.50
accuracy all zero predictions: 0.40
accuracy all ones predictions: 0.40

find Output size

In [22]: import torch.nn as nn

def findConvOutputSize(*args, **kwargs):
    # get conv arguments
    kernel_size = kwargs['kernel_size']
    stride = kwargs['stride']
    padding = kwargs['padding']
    dilation = kwargs['dilation']
    groups = kwargs['groups']
    bias = kwargs['bias']
    return findConvOutputSize(*args, **kwargs)

```

*Handwritten note:* Data Loader

Next thing is custom data Set I will explain it to you we are done with this one. Creating custom data set this is very important. As I already told you while feeding to the deep learning models, I just have to feed the model in terms of batches. Suppose assume that the batch size is 80, let me check what batch size they have used here.

32 they have used, further so assume that I am using batch size of 32 that means, I will feed the model 32 images at an instant (32,3,96,96). This is how I do 32 3, 96, so 32 images containing 3 channels where height and width of each image is length (96,96). This is how I plan.

So, to that means, I will first create a dataset class, this is very important, very very important, where it contains all the directories from where you have to obtain the images and



everything. Here to understand this one you guys have to learn something like oops concept in Python, go and study about this object oriented programming.

Here you will get the length of the data, here you will get, get item means, suppose you have a given index of some suppose assume that I have images of some 1000 images, if I given some random index of something like 1 or 2 or some 90, it will obtain both image and its label, image x y it is labeled, both will be provided as an output if you create this particular data set.

Whenever you are using Pytorch framework, I request everyone to please be familiar with this creating a data set and this dataset class and DataLoader, these are very much important. By using this enter class we can get image and label, see here image and label at a particular index. Suppose assume that I have given is the DataLoader I have given as something like 32 images I want, so it will go and grab all the 32 images at some random indexes.

This is how the data set and DataLoader will work, because in CNN models in computer region often we are not only, in computer vision and the entire deep learning domain we are going to feed the data to the models in terms of batches. Now, to each and every image that are in this particular data set, I just want to apply this transformer ToTensor where earlier I have shown that to 0 to 255 these will be now converted to 0 to 1 range.

This is where this operation will be helpful, just go to the data set directory like where exactly I have kept the train data, see here, after creating this custom data set. Now our job is to go and use something like this here this is the train data path to train, here just instead of here data underscore directory just going to give you the train data format. Later you can histo cancer data set it is given, this is very important step, data train.

Now this will be treated as a train data set class. Now, histo data set you can assume that that length is the entire training images, how many number of images are... while running in the Google colab it takes some time, please have some patience till it gets completed, have some patience here. You can see the length of the data set is like 220,025 images. Now see here, hist data underscore name.

That means at some random index name I am just obtaining the image comma label. The image shape is 3,96,96 and its output is like tensor 0. And here they are printing out

something like Sorry. Over here they are printing just image shape and torch minimum, remember that this data set class, preparing this data set class is very important.

In any Pytorch program, irrespective of the test data or dealing with the video data or images data, irrespective of the data set you deal this is very important, creating the data set class in Pytorch DataLoader is very important. Next once that particular data set class is created, it is your responsibility now to split the data into train and validation pot.

Because extensively they have given the test dataset for us that is why restricting to this histo data set class it took 2 things only, otherwise you can convert if you are not given the test dataset you can even convert this one into train dataset, validation data set and test dataset. See here out of 100 points I am just converting into 80 percentage to train and validation I am just giving 20 percent.

That is the code we will give, splitted the dataset. Here you can see train dataset containing around 176,000, here 44005 like 20 percent this data set for validation data. On train dataset we are just taking the loop like dataset observe here 3 and 3 comma one image and its class, similarly from validation dataset also we are getting something like (3,96,96) 0.

After that we are just taking the images from data set, train dataset and, I will show you a notebook. This is almost normal, after creating the data set class we are just printing out the first single image shape and it is labeled correspondingly. Now coming back to this one, I just want to just create a grid containing images related to train dataset and the corresponding labels here.

To get the output like this I have written the code like this, this code is useful for to images related to train DS, train underscore DS. Whatever DS I have created like hist DS, I just have converted into train DSN value this is what I am talking about where it contains 80 percent is data, like this is 100 percentage, 80 percentage and 20 percentage. This is how they have planned, very important.

Similarly for the validation also they are bringing some images like 4 images and their labels 0 first image belongs to 0 like 0 means normal this is like metastasis image, remaining 2 images are also normal only, just for visualization purposes nothing else. Now coming back to transformations what exists transformers I am using on the train data, validation data and test data.

Always remember that we use this data augmentation steps only on the train DS files only, train dataset, but not on the validation data set and on the test dataset, always remember this one. We use this data transformation, except like suppose we are converting 0 to 255 into 0 to 1 this step is. Other than this any other for image processing, we will not do for validation DSM, test DSN, please keep this one in mind.

See here for train transformer like data augmentation they are using all these steps, whereas for validation they just use only 0 to 255 to convert it into 0 to 1. Apart from this everything else is common. Train DS transform we have fed this one. Now coming back to I mean what sort of data augmentation stuff that are required here?

Validation we are restricting to only 2 tensor operations, whereas for train transform we will do all the things like 2 tensors and random horizontal, random vertical, everything. Now, coming back to creating a DataLoader. Here we are just importing the from torch, you can download that DataLoader here giving the data set batches 32, we are doing this shuffling. For training DataLoader we use the shuffle.

Shuffle is equal to true whereas for validation and test, we need not go for... we need to keep the option as shuffle as true, so shuffle we will keep as false for both validation and test dataset classes. See here in creating the dataset after creating the dataset class, we can obtain for every index we will get only image and neighborhood whereas after creating this DataLoader see here DataLoader train DS batch size underscore 32.

Here we are given the output shape of the images like 32,9. So, 32,3,96,96 and 32, 32 indicates here is 32 era, I mean containing 32 numbers size of this particular torch tensor where it contains a classes of each and every image that are often have 32. Similarly, for validation data they have obtained this shape.

Here for the validation DataLoader they have used the batch as 64. Till now it should be very clear. Once you are clear with this one in dealing any problem with respect to computer vision, it is your responsibility to create this data set class and data set loader. Please do practice how to do the data set class and data set loader. It depends on how the data set has been provided. Good.

(Refer Slide Time: 46:15)



**Building the classification model**

In this section, we will build a model for our binary classification task. Our model is comprised of four convolutional neural networks (CNNs) and two fully connected layers, as shown in the following diagram:

**Defining the loss function**

In this section, we will build a model for our binary classification task. Our model is comprised of four convolutional neural networks (CNNs) and two fully connected layers, as shown in the following diagram:

**Defining the loss function**

Output Activation	Number of Outputs	Loss Function
None	1	nn.BCEWithLogitsLoss

See here for this particular task, I am going to create convolution block one convolutional block 2 convolutional block 3 convolution block 4; like I will have 4 CNN layers, and 2 fully connected layers like these are also called as dense layers, dense layer 1, dense layer 2. Remember this one to build the entire classification model whatever I am going to bring, sorry, build I am going to use 4 CNN layers and 2 fully connected layers.

Now where exactly we have to focus I will tell you. Here I am feeding the input with size 3,96,96. It is the size of the image. Here depending on the kernel, depending on the number of filters we will call them as convolution layer 1, convolutional layer 2, convolution layer 3, convolution layer 4. If you observe the size of the image gets reduced whereas the depth of the image will increase.

Like initially we will have 3 channel image, at the after convolution 4 layer we will have several channels and this size of the height and width of the images will reduce, width of the images will reduce, please keep this point in mind. After convolution layer 4 we will have something shape like this, using that I have a shape of some 64,9, not 96, something like 24,24.

This is the image size whereas this is nothing but the channels. So, total number of pixels that can be obtained from here is 64 into, each individual image contains 24 into 24 pixels. Likewise, we have 64 images, sorry, channels. Channels means feature arms here. All these pixels now I convert this entire thing into a single vector that is where we will use the operations called flatten.

Entire 3 dimensional thing has been converted into a single dimensional vector that particular vector after converting that into a single dimensional vector, assume that I have these many number of pixels 64, 64 some n number of pixels. So, I will convert them to a single feature vector. Now I will feed this one into another dense layer. Assume that I have some 100 nodes here.

The next layer I will keep some time, yet the output I am keeping here 2 nodes, because I am dealing with 2 classes. Please keep the point in mind. In case you are going to solve a multiclass like 3 classes or 4 classes, then you have to keep it the last layer like 3 or 4 depending on the number of classes.

Please remember at the end of this convolution layer, you will have these many number of pixels in this particular case, not the 64 I am just assuming, not some n number, I will multiply the, I will make this computation and you are allowed to enter into a feature vector now will feed the feature vector into the dense layer one next that dense layer to, the another dense layer and then I am feeding to last layer.

This is how the whole structure is, this entire thing I will show you how to do in the Pytorch. Whatever model I am using here is like sequential model.

(Refer Slide Time: 51.03)

**Defining the loss function**

Output Activation	Number of Outputs	Loss Function
None	1	nn.BCEWithLogitsLoss
Sigmoid	1	nn.BCELoss
None	2	nn.CrossEntropyLoss
log_softmax	2	nn.NLLLoss

We recommend using the `log_softmax` function as it is easier to expand to multi-class classification. PyTorch combines the log and softmax operations into one function due to numerical stability and speed.

Please keep this point in your mind while defining any classification problem when dealing any classification problem. This is very important depending number of suppose if I use the output activation... these 3 are very important what output activation I am using how many number of outputs I am using, what sort of last function we should use. Since Pytorch has provided all these things by default, in this code I am, in general people say at the end please keep a softmax activation function and compute the cross entropy loss.

But here Pytorch provide something like log underscore softmax function and negative log likelihood loss. Just fix to this one depending on the dissenter code, whatever in going to show you the code will be applicable to even multiclass classification also. Log softmax and 2 can be replaced by number of the classes need to log likelihood loss, let us assume these things. Till now we are there, validation, transformation done, create loader is done, building the classification model.

(Refer Slide Time: 52:10)

classification. PyTorch combines the log and softmax operations into one function due to numerical stability and speed.

*Accuracy?*

## Activation functions and Loss Computation

We use `log_softmax` as the output and `nn.NLLLoss` as the negative log-likelihood loss. An important argument in defining the loss function to pay attention to is the reduction, which specifies the reduction to apply to the output. There are three options to choose from: none, sum, and mean. We choose `reduction=sum` so that the output loss will be summed. Since we will process the data in batches, this will return the sum of loss values per batch of data.

In step 2, we calculate the loss using an example with `n=8` samples and `c=2` classes.

In step 3, we compute the gradients for the example in step 2. Later, we will use the backward method to compute the gradients of the loss with respect to the model parameters.

## Building Classification Model

### dumb baselines

```
In [21]: # get labels for validation dataset
y_val=y for _,y in val_ds

def accuracy(labels, out):
    return np.sum(out==labels)/float(len(labels))

# accuracy all zero predictions
acc_all_zero=accuracy(y_val,np.zeros_like(y_val))

# accuracy all ones predictions
acc_all_ones=accuracy(y_val,np.ones_like(y_val))

# accuracy random predictions
acc_random=accuracy(y_val,np.random.randint(2,size=len(y_val)))

print('accuracy random predictions: %.2f' %acc_random)
print('accuracy all zero predictions: %.2f' %acc_all_zero)
print('accuracy all one predictions: %.2f' %acc_all_ones)

accuracy random predictions: 0.50
accuracy all zero predictions: 0.40
accuracy all one predictions: 0.40
```

### find Output size

```
In [22]: import torch.nn as nn

def findConv2dOutputSize(in_W,in_H,in_C,conv_pool):
    # get conv arguments
    kernel_size=conv.kernel_size
    stride=conv.stride
    padding=conv.padding
    dilation=conv.dilation

    # Ref: https://pytorch.org/docs/stable/nn.html
```

```
# accuracy all ones predictions
acc_all_ones=accuracy(y_val,np.ones_like(y_val))

# accuracy random predictions
acc_random=accuracy(y_val,np.random.randint(2,size=len(y_val)))

print('accuracy random predictions: %.2f' %acc_random)
print('accuracy all zero predictions: %.2f' %acc_all_zero)
print('accuracy all one predictions: %.2f' %acc_all_ones)

accuracy random predictions: 0.50
accuracy all zero predictions: 0.40
accuracy all one predictions: 0.40
```

### find Output size

```
In [22]: import torch.nn as nn

def findConv2dOutputSize(in_W,in_H,in_C,conv_pool):
    # get conv arguments
    kernel_size=conv.kernel_size
    stride=conv.stride
    padding=conv.padding
    dilation=conv.dilation

    # Ref: https://pytorch.org/docs/stable/nn.html
    W_out=np.floor((in_W+2*padding[0]-dilation[0]*kernel_size[0]-1)/stride[0]+1)
    W_out=np.floor((in_H+2*padding[1]-dilation[1]*kernel_size[1]-1)/stride[1]+1)

    if pool:
        W_out/=pool
        W_out/=pool
    return int(W_out),int(H_out)

# example
conv1 = nn.Conv2d(3, 8, kernel_size=3)
h,w=findConv2dOutputSize(19,19,conv1)
print(h,w)

47 47
```

## Define Model



```

print('accuracy all one predictions: %.2f' % acc_all_one)

accuracy random predictions: 0.30
accuracy all zero predictions: 0.40
accuracy all one predictions: 0.40

find Output size

In [22]:
import torch.nn as nn

def findConv2dOutShape(H,W_in,conv_pool=2):
    # get conv arguments
    kernel_size=conv.kernel_size
    stride=conv.stride
    padding=conv.padding
    dilation=conv.dilation

    # Ref: https://pytorch.org/docs/stable/nn.html
    H_out=conv.floor((H_in+padding[0]-dilation[0]*kernel_size[0]-1)//stride[0]+1)
    W_out=conv.floor((W_in+padding[1]-dilation[1]*kernel_size[1]-1)//stride[1]+1)

    if pool:
        H_out//=pool
        W_out//=pool
    return int(H_out),int(W_out)

# example
conv1 = nn.Conv2d(1, 8, kernel_size=3)
h,w=findConv2dOutShape(16,16,conv1)
print(h,w)

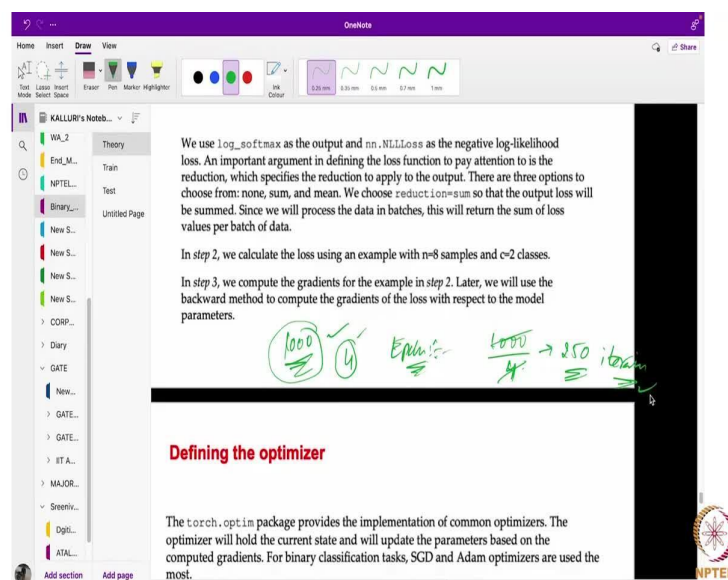
47 47

Define Model

In [23]:
import torch.nn as nn
import torch.nn.functional as F

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

```

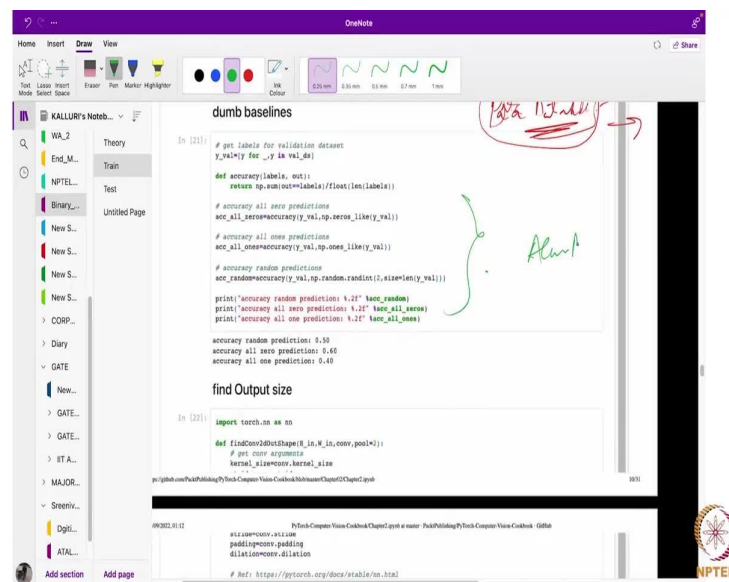


While designing this classification model mostly metrics like accuracy and loss, accuracy and loss. Here are some dumb baselines they help prepared to find the accuracy and accuracy. This function they have created to compute the accuracy. Suppose I am feeding some 32 batch data, 32 sizes as a batch data and for all the 32 images, I will get some outputs like 32. 32 outputs I will get and I will compare these 32 outputs to my original target value and I will compute this accuracy metric.

This is all accuracy metric is computed like number of predictions hold divided by total number of predictions. Before that, let me tell you what is meant by epoch. Assume that I have some 1000 data points in total and that size is... I just assume that size is something like 4, 1 epoch means training has to be done for the entire data set. That means model has to be trained for all on this 1000 images.

This is the usual step, likewise they will do next iteration I will give for next 4 images, I will compute the loss and accuracy metric. Likewise I will do for all these 25 iterations. So, that entire data set has been trained once. For training the entire data once batch size mean like iterations. Next thing is I will obtain the laws for each and every batch, every, here I have made some functions to compute the accuracy at the same time and loss metric by using the for every batch.

(Refer Slide Time: 55:17)



find Output size

```
In [22]: import torch.nn as nn

def findConv2dOutShape(H_in, W_in, conv, pool):
    # get conv arguments
    kernel_size=conv.kernel_size
    ...
```

```
PyTorch Computer Vision Cookbook Chapter 2.pyth at master · PyTorch/pytorch · GitHub
```

```
#!/usr/bin/env python
import sys
import torch
import torch.nn as nn
import torch.nn.functional as F

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

Define Model

```
In [23]: import torch.nn as nn
import torch.nn.functional as F

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

comprised of four convolutional neural networks (CNNs) and two fully connected layers, as shown in the following diagram:

Defining the loss function

Output Activation	Number of Outputs	Loss Function
None	1	nn.BCEWithLogitsLoss
Sigmoid	1	nn.BCELoss

THRU Output Size

```
In [22]: import torch.nn as nn

def findConv2dOutShape(H_in, W_in, conv, pool):
    # get conv arguments
    kernel_size=conv.kernel_size
    ...
```

```
PyTorch Computer Vision Cookbook Chapter 2.pyth at master · PyTorch/pytorch · GitHub
```

```
#!/usr/bin/env python
import sys
import torch
import torch.nn as nn
import torch.nn.functional as F

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

Define Model

```
In [23]: import torch.nn as nn
import torch.nn.functional as F

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

Handwritten notes:  $96 \times 96$ ,  $8, 8, 96, 96$

This is what they are doing here, just dumb baseline to prepare the accuracy metric, accuracy metric. Similarly as I told you to compute this value, to compute this value like a single feature vector I told him something like n, suppose assume (64,24,24), likewise values are came out, 64 indicates channel, 24 for all these pixels can be converted into feature vector.

So, what I am doing is I just have made a function, I just have made a function depending on the for every problem we need not computer this one, by default a function has been developed so, that you can generate this output. Likewise, suppose this output will be taken out and I will multiply this value and it will be converted into this one.

Instead of our manual computation I just made a function with this particular code, see here it gives the output after each and every convolution layer. These formulas will be there, let us go and find out, not an issue. Assume that for a single convolution layer if 3 channel input and 8 kernel, sorry, kernel size is 3 and I am giving some 8 kernels that means output will be containing 8 comma input; image size is something like  $96 \times 96$ .

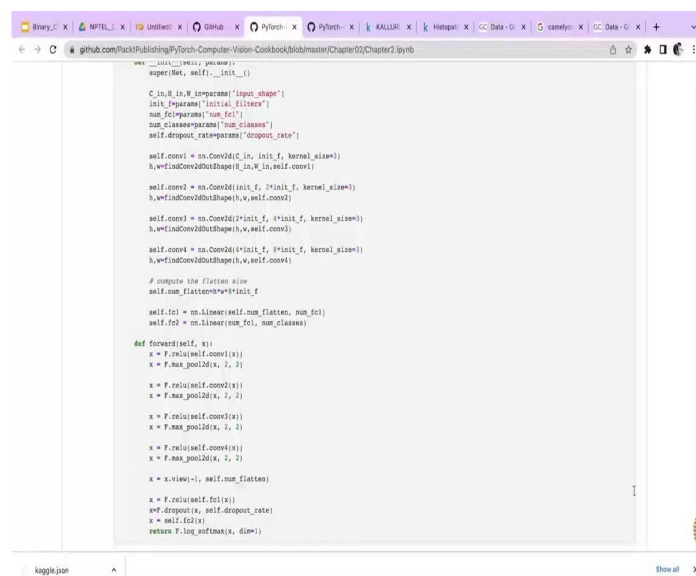
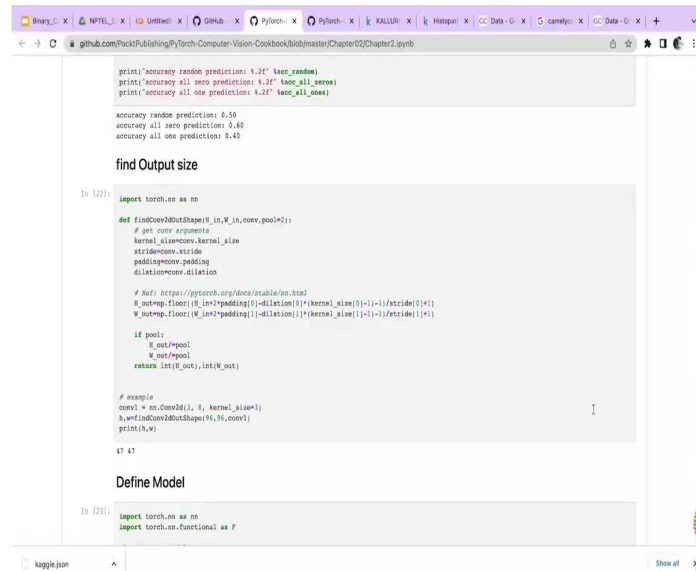
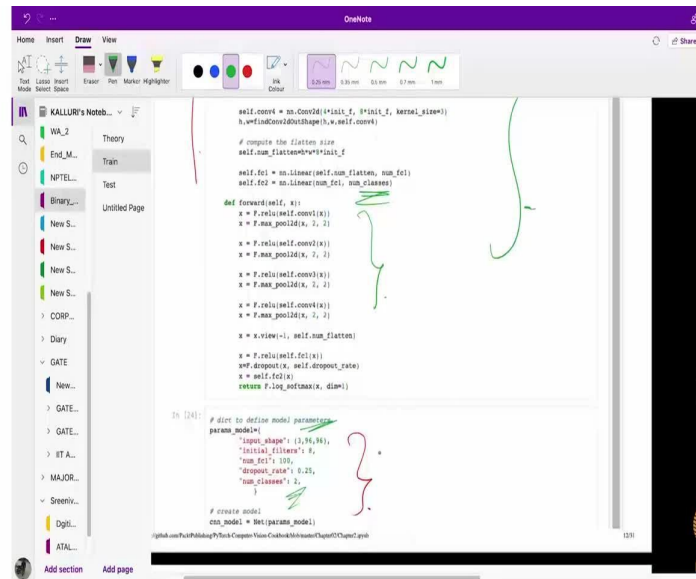
If I feed this image into this particular convolution, which takes 3 layers and outputs 8 layers; the size of the entire thing 8,96,96, this is the output. Here they just have printed out only height and width of this particular shape of this particular convolution here. In summary, this function usually calculates the output shape after each and every convolutional layer.

(Refer Slide Time: 57.21)

The screenshot displays a Jupyter Notebook environment with the following components:

- Top Toolbar:** Includes icons for file operations (Save, Open, Print, etc.), a color palette, and a ruler.
- Left Sidebar:** Contains a file explorer showing a directory structure with files like 'WA\_2', 'End\_M...', 'NPTEL...', 'Binary...', 'New S...', 'New S...', 'New S...', 'CORP...', 'Diary', 'GATE', 'GATE...', 'GATE...', 'IT A...', 'MAJOR...', 'Screeni...', 'Digi...', and 'ATAL...'. Below it is a table of contents with sections like 'Theory', 'Test', and 'Untitled Page'.
- Main Notebook Area:**
  - Cell 1:** Imports necessary modules and defines a custom `Dataset` and `DataLoader` class. It includes handwritten notes 'oof/sr' and 'Hyperm.' with an arrow pointing to the `learning_rate` parameter.
  - Cell 2:** Defines a `CNN` class with `__init__` and `forward` methods. It includes handwritten notes 'oof/sr' and 'Hyperm.' with an arrow pointing to the `learning_rate` parameter.
  - Cell 3:** Defines a `train` function that iterates over the dataset and performs training steps.
- Bottom Status Bar:** Shows the current file name 'Add section' and the page number 'Add page'.

[illegible]



```
Binary.C X | UFTEL.C X | GitHub X | PyTorch X | KALLAR X | Histopals X | CC Data - G X | Cansely X | HUST X  
github.com/PacktPublishing/Torch-Computer-Vision-Cookbook/blob/master/Chapter02/chapter2.ipynb
```

```
x = F.relu(self.conv2(x))  
x = F.max_pool2d(x, 2, 2)  
  
x = F.relu(self.conv3(x))  
x = F.max_pool2d(x, 2, 2)  
  
x = F.relu(self.conv4(x))  
x = F.max_pool2d(x, 2, 2)  
  
x = x.view(-1, self.num_features)  
  
x = F.relu(self.fc1(x))  
w_dropouts = self.dropout_rate  
x = self.fc2(x)  
return F.log_softmax(x, dim=1)
```

```
In [24]: # dict to define model parameters  
params_model={  
    "input_shape": (3,96,96),  
    "initial_filters": 8,  
    "num_fol": 100,  
    "dropout_rate": 0.25,  
    "num_classes": 2,  
}
```

```
# create model  
cnn_model = Net(params_model)
```

```
In [25]: # move model to cuda/gpu device  
(if torch.cuda.is_available())  
device = torch.device("cuda")  
cnn_model=cnn_model.to(device)
```

```
In [26]: print(cnn_model)
```

```
Net(  
  (conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1))  
  (conv2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))  
  (conv3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))  
  (conv4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))  
  (fc1): Linear(in_features=1024, out_features=100, bias=True)  
  (fc2): Linear(in_features=100, out_features=2, bias=True)
```

```
Binary: X | Untitled: X | GitHub: X | PyTorch: X | PyTorch: X | KALLURI: X | Hsistop: X | CC Data - G: X | Catelyn: X | CC Data - G: X | +
github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter2_1.ipynb
```

```
x = F.relu(self.conv2(x))
x = F.max_pool2d(x, 2, 2)

x = F.relu(self.conv3(x))
x = F.max_pool2d(x, 2, 2)

x = x.view(-1, self.num_floattens)

x = F.relu(self.fc1(x))
x=F.dropout(x, self.dropout_rate)
x = self.fc2(x)
return F.log_softmax(x, dim=1)
```

```
In [24]: # dict to define model parameters
params_model={
    "input_shape": (1,96,96),
    "initial_filters": 8,
    "num_fc1": 100,
    "dropout_rate": 0.25,
    "num_classes": 5,
}

# create model
cnn_model = Net(params_model)
```

```
In [25]: # move model to cuda/gpu device
if torch.cuda.is_available():
    device = torch.device("cuda")
    cnn_model=cnn_model.to(device)
```

```
In [26]: print(cnn_model)

Net(
  (conv1): Conv2d(3, 8, kernel_size=[3, 3], stride=[1, 1])
  (conv2): Conv2d(8, 16, kernel_size=[3, 3], stride=[1, 1])
  (conv3): Conv2d(16, 32, kernel_size=[3, 3], stride=[1, 1])
  (conv4): Conv2d(32, 64, kernel_size=[3, 3], stride=[1, 1])
  (fc1): Linear(in_features=320, out_features=100, bias=True)
  (fc2): Linear(in_features=100, out_features=5, bias=True)
)
```

```
In [27]: print(next(cnn_model.parameters()).device)
```

Parameters has taken and it puts a number of filters and first dense layer, how many nodes we have to keep, similarly, number of classes dropout rate. Dropout rate is usually we will use for regularization, in this one 1 declaration is also a concept we to use to tackle overfitting problem.

Likewise, I have made this entire thing you can keep the center code as it is even if you solve with the multiclass classification even you can increase the layers also wherever you want, but the core remains same, mostly few blocks will be changed, something like suppose if I add another convolutional layer, instead of convolution 4, I will add more layer.

Similarly, I will add activation functions like this, but at the end instead of number of classes here I just have to give depending here I am dealing with the 2 classes that is why I kept 2. Suppose if you are dealing with 4 classes just replace this value with 4, please keep this point in mind. And suppose here I am using 3,96,96 as input shape but whatever size you are dealing with the image you can keep that sizes.

Please keep all these points in your mind, this is where you have to make changes. And initial filters are something like for after giving the input image how many filters I am using. Like how many I will use. And a fully connected layer like dense layer 100 this is how we have planned this dictionary I am sending only for the sake of hyper parameter tuning.

All these things have done we know we have defined the model within the Torch framework. See here we just find the output size of every convolutional layer node model has been defined with respect to... see here. At the end we are using log underscore softmax activation function. Now coming back to in case you are using GPU, you provided the GPU general test use this command to we have to load the model into device, whichever device you use, suppose if you are dealing with CPU, you have to use the device CPU if you are using GPU.

You have to use this GPU device or a device we can call that is over this one. Next thing is what are the parameters like coda 0? Next thing just checking only from torch summary just you have to install the torch summary, from this you can see suppose if I give inputs as 3,96,96 what are the output shifts after each and every layer. See here after linear 6 we are having 2.

What are given as linear layers? See here after doing this flatten operation there used FC1 and fully connected layer 2, it contains, it takes input is suppose 100 and 2. Suppose if I want to



add one more dense layer I will use something like this one and FC 2 something like that small changes, minute changes you have to do, you have to be familiar with that one.

Now, we are done with the defining the models. Model and we have verified these are the outputs after each and every layer, after first convolutional layer the output is 100. Now, that 100 nodes will be fed, will have 2. At the end whatever at the output layer whatever maximum activation function logs after that is we design the model.

(Refer Slide Time: 62:19)

The screenshot shows a OneNote page with the title "Loss function". The code defines a loss function using `nn.NLLLoss` and includes a training loop. Handwritten annotations in red and green ink are present:

- A green bracket on the left side of the code block, spanning from the `loss_func` definition to the `loss.backward()` call.
- Red handwritten notes:  $(32, 3, 96, 96)$  with a downward arrow,  $(32, 2)$  with a downward arrow,  $(32, 4)$  with a downward arrow, and  $(32, 0.150)$  with a downward arrow.

```
In [29]: loss_func = nn.NLLLoss(reduction='sum')

In [30]: # fixed random seed
torch.manual_seed(0)

n, n_ch, 3
y = torch.randn(n, c, requires_grad=True)
log_p = nn.LogSoftmax(dim=1)
y_out=log_p(y)
print(y_out.shape)

target = torch.randn(n, size*(n,))
print(target.shape)

loss = loss_func(y_out, target)
print(loss.item())

torch.Size([8, 2])
torch.Size([8])
5.26899542992676

In [31]: loss.backward()
print(y.data)

tensor([[[-1.1258, -1.1524],
         [-0.2204, -0.4339],
         [ 0.8487,  0.6920],
         [-0.3165, -0.1152],
         [ 0.3223, -1.2433],
         [ 0.3505,  0.3081],
         [ 0.1198,  1.2377],
         [ 1.1168, -0.2473]])])
```

The screenshot shows the same OneNote page as above, but with different handwritten annotations:

- A green bracket on the left side of the code block, spanning from the `loss_func` definition to the `loss.backward()` call.
- Red handwritten notes: a checkmark next to the `loss_func` definition and a green checkmark next to the `loss.backward()` call.

```
In [29]: loss_func = nn.NLLLoss(reduction='sum')

In [30]: # fixed random seed
torch.manual_seed(0)

n, n_ch, 3
y = torch.randn(n, c, requires_grad=True)
log_p = nn.LogSoftmax(dim=1)
y_out=log_p(y)
print(y_out.shape)

target = torch.randn(n, size*(n,))
print(target.shape)

loss = loss_func(y_out, target)
print(loss.item())

torch.Size([8, 2])
torch.Size([8])
5.26899542992676

In [31]: loss.backward()
print(y.data)

tensor([[[-1.1258, -1.1524],
         [-0.2204, -0.4339],
         [ 0.8487,  0.6920],
         [-0.3165, -0.1152],
         [ 0.3223, -1.2433],
         [ 0.3505,  0.3081],
         [ 0.1198,  1.2377],
         [ 1.1168, -0.2473]])])
```

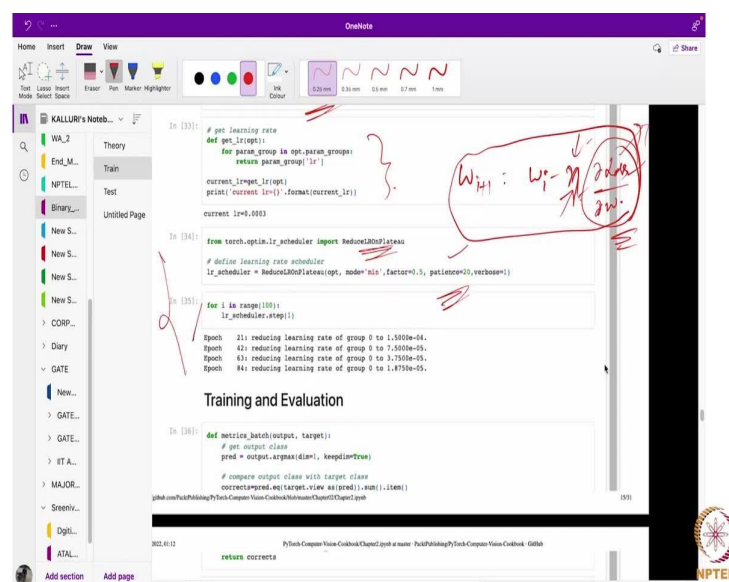
Now, coming back to loss function, while the training is going on we have to observe the last value. For the loss function they have used the negative log likelihood loss, please keep this

one in mind. This one is verify is very easy, not that difficult, here just assuming that we are having some practice like how is the loss function has appeared.

Suppose I give you input as 32 and we are having 2 classes that was 32,3,96,96 I have given as input and we will get output as 32 since we have 2 classes I will get 2, this is how the shape will be if you have 4 classes you will get output as 32,4.

Later I will use something like `arg max` that particular index value will be given as class. Suppose 32,2 it will be converted into 32 containing even 1 0 1 1 0 0. Depending on the index value which has more value, that is where we will use `arg max`, this is how we do it. Loss function, they have used negative likelihood loss and the sum is reduction, reduction means after every batch we are summing up that one.

(Refer Slide Time: 63:44)



Optimizer they have used this Adam optimizer. Second thing is we just want to get to the learning rate, yet the while the code is running at any given instant if you want what is the current learning rate you just have to use this function. One more thing we have to provide learning rate schedule also, suppose there is no improvement in the model valuation loss after several epochs also.

At that time, we just either go, if you observe the basic form or something like weight implies, weight minus learning rate into delta loss whole divided by the parameters, this is the basic equation. What we do here is we either we change the loss function, either learning rate

or we have these things. Suppose if you do not observe any much improvement, these gradient variables at the time go ahead and change this learning rate.

So, what we do is we use some validation loss as a parameter. Validation loss as a parameter to check whether the model is improving or not. So, if it is not improving after every 10 verbose patience number indicates after those many number of posts, just reduce the learning rate, this is what the functionality of this particular learning rate scheduler.

(Refer Slide Time: 64:59)

```
In [36]: def metric_batch(output, target):  
# get output class  
pred = output.argmax(dim=1, keepdim=True)  
# compare output class with target class  
correctpred.eq(target.view_as(pred)).sum().item()  
github.com/PacktPublishing/PyTorch-Cookbook/Chapter02/Chapter02.ipynb  
13/11  
bala  
Arun  
K302.01.02 PyTorch-Cookbook/Chapter02.ipynb a maver PacktPublishing/PyTorch-Cookbook/Chapter02  
return corrects  
In [37]: n_out=2  
output = torch.randn(1, c, requires_grad=True)  
print(output)  
print(output.shape)  
#target = torch.randn(c, size=n_out)  
target = torch.ones(n_out, dtype=torch.long)  
print(target.shape)  
metric_batch(output, target)  
tensor([[ 0.4681, -0.1577],  
[ 0.4437, 0.2660],  
[ 0.1665, 0.8744],  
[-0.1435, -0.1114],  
[ 0.9318, 1.2590],  
[ 2.0050, 0.0537],  
[ 0.4381, -0.4128],  
[ 0.4381, -0.4128],  
[-0.8411, -2.3160]], requires_grad=True)  
torch.Size([8, 2])  
Out [37]: 3  
In [38]: def loss_batch(loss_func, output, target, opt=None):  
# new train
```

```
[-0.1435, -0.1114],  
[ 0.9318, 1.2590],  
[ 2.0050, 0.0537],  
[ 0.4381, -0.4128],  
[-0.8411, -2.3160]], requires_grad=True)  
torch.Size([8, 2])  
Out [37]: 3  
In [38]: def loss_batch(loss_func, output, target, opt=None):  
# get loss  
loss = loss_func(output, target)  
# get performance metric  
metric_b = metric_batch(output, target)  
if opt is not None:  
opt.zero_grad()  
loss.backward()  
opt.step()  
return loss.item(), metric_b  
14/11  
Arun  
K302.01.02 PyTorch-Cookbook/Chapter02.ipynb a maver PacktPublishing/PyTorch-Cookbook/Chapter02  
# define device as a global variable  
device = torch.device('cpu')  
def loss_epoch(model, loss_func, dataset_d1, sanity_check=False, opt=None):  
running_loss=0  
running_metric=0  
len_data=len(dataset_d1.dataset)  
for xh, yh in dataset_d1:  
# move batch to device  
xb=xb.to(device)  
yb=yb.to(device)  
# get model output
```



```
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.backward(loss)
        opt.step()

    return loss.item(), metric_b

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

def loss_epoch(model, loss_func, dataset_d1, sanity_check=False, opt=None):
    running_loss = 0
    running_metric = 0
    len_data = len(dataset_d1.dataset)

    for xb, yb in dataset_d1:
        # 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

In [39]:
def train_val(model, params):
    # extract model parameters
    num_epochs = params["num_epochs"]
    loss_func = params["loss_func"]
    opt = params["optimizer"]
    train_d1 = params["train_d1"]
    val_d1 = params["val_d1"]
    sanity_check = params["sanity_check"]
    lr_scheduler = params["lr_scheduler"]
    path_weights = params["path_weights"]

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

    # history 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 = lr_scheduler.get_lr()[0]
        print(f"Epoch {epoch+1}, current lr={current_lr:.4f}, num_epochs={num_epochs}, current_lr={current_lr}")

        # train model on training dataset
        model.train()
        train_loss, train_metric = loss_epoch(model, loss_func, train_d1, 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()
```

```
In [39]:
# define device as a global variable
device = torch.device("cuda")

def loss_epoch(model, loss_func, dataset_d1, sanity_check=False, opt=None):
    running_loss = 0
    running_metric = 0
    len_data = len(dataset_d1.dataset)

    for xb, yb in dataset_d1:
        # 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

In [40]:
def train_val(model, params):
    # extract model parameters
    num_epochs = params["num_epochs"]
    loss_func = params["loss_func"]
    opt = params["optimizer"]
    train_d1 = params["train_d1"]
    val_d1 = params["val_d1"]
    sanity_check = params["sanity_check"]
    lr_scheduler = params["lr_scheduler"]
    path_weights = params["path_weights"]

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

    # history 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 = lr_scheduler.get_lr()[0]
        print(f"Epoch {epoch+1}, current lr={current_lr:.4f}, num_epochs={num_epochs}, current_lr={current_lr}")

        # train model on training dataset
        model.train()
        train_loss, train_metric = loss_epoch(model, loss_func, train_d1, 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()
```

```
In [40]:
def train_val(model, params):
    # extract model parameters
    num_epochs = params["num_epochs"]
    loss_func = params["loss_func"]
    opt = params["optimizer"]
    train_d1 = params["train_d1"]
    val_d1 = params["val_d1"]
    sanity_check = params["sanity_check"]
    lr_scheduler = params["lr_scheduler"]
    path_weights = params["path_weights"]

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

    # history 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 = lr_scheduler.get_lr()[0]
        print(f"Epoch {epoch+1}, current lr={current_lr:.4f}, num_epochs={num_epochs}, current_lr={current_lr}")

        # train model on training dataset
        model.train()
        train_loss, train_metric = loss_epoch(model, loss_func, train_d1, 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()
```



[illegible]

The screenshot shows a video player interface with a top toolbar containing icons for Home, Insert, Draw, and View. Below the toolbar is a drawing palette with various tools like Lasso, Line, Mark, Eraser, Pen, Marker, Highlighter, and a color selection tool. The video content displays a code editor with Python code for training a PyTorch model. The code includes a loop for training epochs, data loading, model training, and validation. A 'sanity check' is implemented to ensure the model's performance on the validation set improves over time. The video player shows a progress bar at 16/51 and a timestamp of 20:08:22.01.12. The code is as follows:

```
for epoch in range(num_epochs):  
    # get current learning rate  
    current_lr = lr_scheduler.get_lr()[0]  
    print('Epoch: %d (%d), current lr: %f' % (epoch, num_epochs - 1, current_lr))  
  
    # train model on training dataset  
    model.train()  
    train_loss, train_metric = loss_epoch(model, loss_func, training_loader, 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_loader, 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(), path_weights)  
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(val_loss)  
if current_lr != get_lr(opt):  
    print('Loading best model weights')
```



The screenshot shows a Jupyter Notebook interface with a terminal window at the bottom. The terminal output displays the training progress of a neural network over 50 epochs. The output is as follows:

```
Epoch 1/50, current lr=0.0015
train loss: 0.00124, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 11/50, current lr=0.0015
train loss: 0.00129, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 17/50, current lr=0.0015
train loss: 0.00119, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 24/50, current lr=0.0015
train loss: 0.00123, dev loss: 0.00094, accuracy: 0.50
-----
Epoch 30/50, current lr=0.0015
train loss: 0.00124, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 36/50, current lr=0.0015
train loss: 0.00132, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 41/50, current lr=0.0015
train loss: 0.00119, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 46/50, current lr=0.0015
train loss: 0.00116, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 49/50, current lr=0.0015
train loss: 0.00122, dev loss: 0.00095, accuracy: 0.50
-----
Epoch 50/50, current lr=0.00075
train loss: 0.00121, dev loss: 0.00094, accuracy: 0.50
-----
Epoch 51/50, current lr=0.00015
train loss: 0.00175, dev loss: 0.00095, accuracy: 0.50
-----
Epoch 52/50, current lr=0.00015
train loss: 0.00123, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 53/50, current lr=0.00015
train loss: 0.00129, dev loss: 0.00093, accuracy: 0.50
-----
Epoch 54/50, current lr=0.00015
train loss: 0.00128, dev loss: 0.00094, accuracy: 0.50
-----
Epoch 55/50, current lr=0.00015
train loss: 0.00114, dev loss: 0.00094, accuracy: 0.50
-----
Epoch 56/50, current lr=0.00015
train loss: 0.00113, dev loss: 0.00093, accuracy: 0.50
```



```
patchweights": ".\\model\\weights.pt",
}

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

Epoch 0/99, current lr=0.0003
Copied best model weights!
train loss: 0.00125, dev loss: 0.00124, accuracy: 0.95
-----
Epoch 1/99, current lr=0.0003
Copied best model weights!
train loss: 0.00125, dev loss: 0.00123, accuracy: 0.95
-----
Epoch 2/99, current lr=0.0003
Copied best model weights!
train loss: 0.00127, dev loss: 0.00114, accuracy: 0.95
-----
Epoch 3/99, current lr=0.0003
Copied best model weights!
train loss: 0.00127, dev loss: 0.00112, accuracy: 0.96
-----
Epoch 4/99, current lr=0.0003
Copied best model weights!
train loss: 0.00127, dev loss: 0.00109, accuracy: 0.97
-----
Epoch 5/99, current lr=0.0003
Copied best model weights!
train loss: 0.00128, dev loss: 0.00105, accuracy: 0.99
-----
Epoch 6/99, current lr=0.0003
Copied best model weights!
train loss: 0.00124, dev loss: 0.00099, accuracy: 0.99
-----
Epoch 7/99, current lr=0.0003
Copied best model weights!
train loss: 0.00129, dev loss: 0.00094, accuracy: 0.99
-----
Epoch 8/99, current lr=0.0003
Copied best model weights!
train loss: 0.00124, dev loss: 0.00095, accuracy: 0.99
-----
Epoch 9/99, current lr=0.0003
Copied best model weights!
train loss: 0.00122, dev loss: 0.00093, accuracy: 0.99
-----
Epoch 10/99, current lr=0.0003
```

```
# history 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_dataloader, 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_dataloader, sanity_check)

    # store best model
    if val_loss < best_loss:
        best_loss = val_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_dataloader, 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(), patchweights)
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(val_loss)
if current_lr != get_lr(opt):
    print('Loading best model weights!')
    model.load_state_dict(best_model_wts)

print('train loss: %.4f, dev loss: %.4f, 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
```



```
def train_and_validate_the_model(cnn_model, loss_hist_metric_hist_train_val(cnn_model, param_train):

    Epoch 6/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00129, dev loss: 0.001024, accuracy: 0.05
    -----
    Epoch 7/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00125, dev loss: 0.001021, accuracy: 0.05
    -----
    Epoch 8/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00127, dev loss: 0.001016, accuracy: 0.05
    -----
    Epoch 9/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00127, dev loss: 0.001012, accuracy: 0.06
    -----
    Epoch 10/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00127, dev loss: 0.001009, accuracy: 0.07
    -----
    Epoch 11/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00126, dev loss: 0.001005, accuracy: 0.09
    -----
    Epoch 12/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00126, dev loss: 0.000999, accuracy: 0.09
    -----
    Epoch 13/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00125, dev loss: 0.000994, accuracy: 0.09
    -----
    Epoch 14/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00126, dev loss: 0.000985, accuracy: 0.09
    -----
    Epoch 15/99, current lr=0.0003
    Copied best model weights!
    train loss: 0.00122, dev loss: 0.000983, accuracy: 0.09
    -----

# Train and validate the model
cnn_model, loss_hist_metric_hist_train_val(cnn_model, param_train)
```

```
Epoch 16/99, current lr=0.0003
Copied best model weights!
train loss: 0.00122, dev loss: 0.000953, accuracy: 0.09
-----
Epoch 17/99, current lr=0.0003
Copied best model weights!
train loss: 0.00121, dev loss: 0.000948, accuracy: 0.09
-----
Epoch 18/99, current lr=0.0003
Copied best model weights!
train loss: 0.00116, dev loss: 0.000938, accuracy: 0.09
-----
Epoch 19/99, current lr=0.0003
Copied best model weights!
train loss: 0.00108, dev loss: 0.000930, accuracy: 0.09
-----
Epoch 20/99, current lr=0.0003
Copied best model weights!
train loss: 0.00133, dev loss: 0.000961, accuracy: 0.09
-----
Epoch 21/99, current lr=0.0003
Copied best model weights!
train loss: 0.00138, dev loss: 0.000959, accuracy: 0.09
-----
Epoch 22/99, current lr=0.0003
Copied best model weights!
train loss: 0.00123, dev loss: 0.000962, accuracy: 0.09
-----

Epoch 23/99, current lr=0.0003
train loss: 0.00119, dev loss: 0.000957, accuracy: 0.09
-----
Epoch 24/99, current lr=0.0003
train loss: 0.00126, dev loss: 0.000941, accuracy: 0.09
-----
Epoch 25/99, current lr=0.0003
train loss: 0.00119, dev loss: 0.000947, accuracy: 0.09
-----
Epoch 26/99, current lr=0.0003
train loss: 0.00125, dev loss: 0.000955, accuracy: 0.09
-----
Epoch 27/99, current lr=0.0003
train loss: 0.00120, dev loss: 0.000947, accuracy: 0.09
-----
```

```
# 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(), path(weights))
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(val_loss)
if current_lr != get_lr(opt):
    print("Loading best model weights!")
    model.load_state_dict(best_model_wts)

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

In [42]: import copy

loss_func = nn.L1Loss(reduction="sum")
opt = optim.Adam(cnn_model.parameters(), lr=1e-4)
lr_scheduler = ReduceLROnPlateau(opt, mode='min', factor=0.5, patience=20, verbose=1)

param_train = {
    "num_epochs": 100,
    "optimizer": opt,
    "loss_func": loss_func,
    "train_dl": train_dl,
    "val_dl": val_dl,
    "sanity_check": True,
    "lr_scheduler": lr_scheduler,
```

OneNote

Home Insert Draw View

Test Lesson Insert Mode Select Space Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.50 mm 0.5 mm 0.7 mm 1 mm

KALLURI's Notebooks

WA\_2

End\_M...

NPTEL

Binary...

New S...

New S...

New S...

New S...

CORP...

Diary

GATE

New...

GATE...

GATE...

IT A...

MAJOR...

Sreeniv...

Dgitl...

ATAL...

Add section Add page

Theory

Train

Test

Untitled Page

```

Epoch 87/99, current lr=3.75e-05
train loss: 0.00122, dev loss: 0.000954, accuracy: 0.69
Epoch 88/99, current lr=3.75e-05
train loss: 0.00122, dev loss: 0.000954, accuracy: 0.69
Epoch 89/99, current lr=3.75e-05
train loss: 0.00107, dev loss: 0.000953, accuracy: 0.69
Epoch 90/99, current lr=3.75e-05
train loss: 0.00129, dev loss: 0.000974, accuracy: 0.69
Epoch 91/99, current lr=3.75e-05
train loss: 0.00129, dev loss: 0.000955, accuracy: 0.69
Epoch 92/99, current lr=3.75e-05
train loss: 0.00104, dev loss: 0.000966, accuracy: 0.69
Epoch 93/99, current lr=3.75e-05
train loss: 0.00139, dev loss: 0.000955, accuracy: 0.69

```

26/1

NPTEL

OneNote

Home Insert Draw View

Test Lesson Insert Mode Select Space Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.50 mm 0.5 mm 0.7 mm 1 mm

KALLURI's Notebooks

WA\_2

End\_M...

NPTEL

Binary...

New S...

New S...

New S...

New S...

CORP...

Diary

GATE

New...

GATE...

GATE...

IT A...

MAJOR...

Sreeniv...

Dgitl...

ATAL...

Add section Add page

Theory

Train

Test

Untitled Page

```

plt.plot(range(1,num_epochs+1),metric_val["train"],label="train")
plt.plot(range(1,num_epochs+1),metric_val["val"],label="val")
plt.ylabel("Accuracy")
plt.xlabel("Training Epochs")
plt.legend()
plt.show()

```

Train-Val Loss

PyTorch Computer Vision Cookbook/Chapter2/Chapter2.ipynb

PyTorch Computer Vision Cookbook/Chapter2/Chapter2.ipynb at master · PyTorch/PyTorch Computer Vision Cookbook · GitHub

Train-Val Accuracy

27/1

NPTEL

OneNote

Home Insert Draw View

Test Lesson Insert Mode Select Space Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.50 mm 0.5 mm 0.7 mm 1 mm

KALLURI's Notebooks

WA\_2

End\_M...

NPTEL

Binary...

New S...

New S...

New S...

New S...

CORP...

Diary

GATE

New...

GATE...

GATE...

IT A...

MAJOR...

Sreeniv...

Dgitl...

ATAL...

Add section Add page

Theory

Train

Test

Untitled Page

```

loss_func = nn.Loss(reduction='sum')
opt = optim.Adam(model.parameters(), lr=3e-4)

```

28/1

NPTEL

PyTorch Computer Vision Cookbook Chapter 2: Epochs at a glance - PyTorch Publishing PyTorch Computer Vision Cookbook - GitHub

```

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--epochs', type=int, default=10)
    parser.add_argument('--lr', type=float, default=0.001)
    parser.add_argument('--batch-size', type=int, default=64)
    parser.add_argument('--device', type=str, default='cuda')
    args = parser.parse_args()

    # Create the model, loss function, optimizer, data loader, and trainer
    model = torchvision.models.resnet18(pretrained=True)
    loss_fn = nn.LossFunction()
    optimizer = optim.Adam(model.parameters())
    data_loader = torch.utils.data.DataLoader(
        ImageFolder(root_dir, transform=transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ]))
    )

    # Train and validate the model
    trainer = GradientDescentTrainer(model=model, loss_fn=loss_fn, optimizer=optimizer, data_loader=data_loader)
    trainer.train(epochs=args.epochs)

    # Print the best model weights
    print('Best model weights: %s' % trainer.best_model_weights)

    # Print the training progress
    print('Training progress:')
    for epoch in range(1, args.epochs + 1):
        train_loss, train_acc = trainer.train_epoch()
        val_loss, val_acc = trainer.validate_epoch()
        print('Epoch %d, current lr=%f, train loss=%f, dev loss=%f, accuracy=%f' % (epoch, lr, train_loss, val_loss, val_acc))
    
```

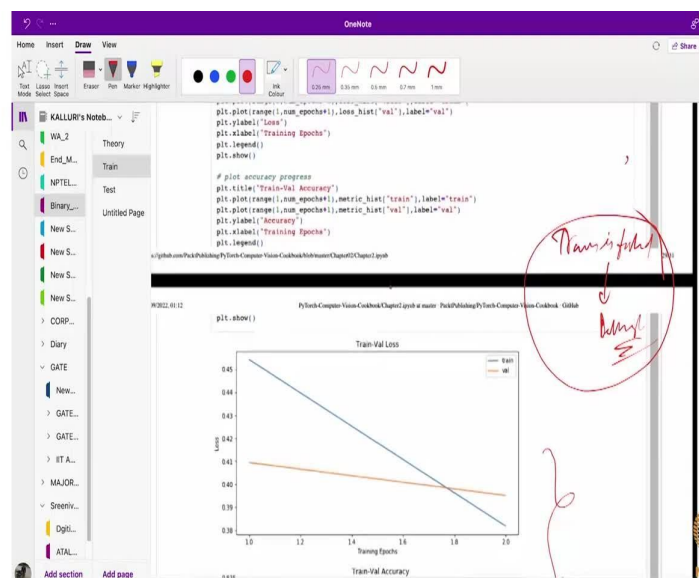
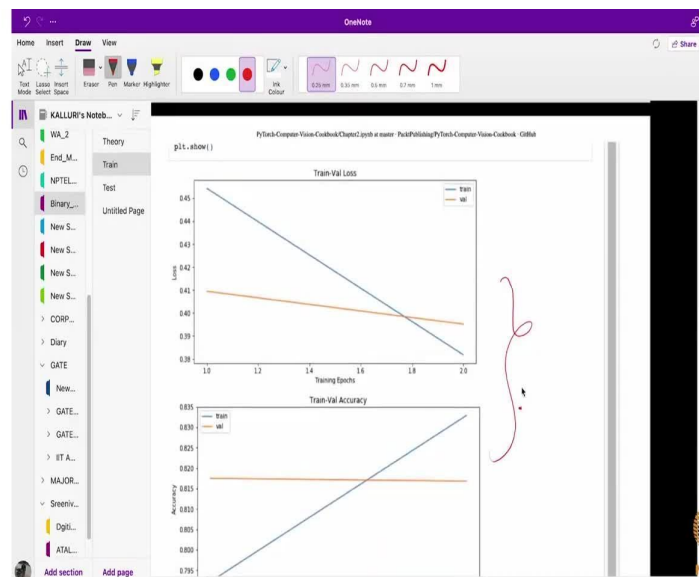
Epoch 1/1, current lr=0.001  
Copied best model weights!  
train loss: 0.454370, dev loss: 0.409489, accuracy: 81.75  
-----  
Epoch 1/1, current lr=0.001  
Copied best model weights!  
train loss: 0.381784, dev loss: 0.395194, accuracy: 81.68  
-----

# Train-Validation Progress  
num\_epochs, num\_epochs\_train, num\_epochs\_val, num\_epochs\_test, num\_epochs\_validate

# plot loss progress  
plt.title('Train-Val Loss')  
plt.plot(range(1, num\_epochs+1), loss\_hist['train'], label='train')  
plt.plot(range(1, num\_epochs+1), loss\_hist['val'], label='val')  
plt.xlabel('Loss')  
plt.ylabel('Training Epochs')  
plt.legend()  
plt.show()

# plot accuracy progress  
plt.title('Train-Val Accuracy')  
plt.plot(range(1, num\_epochs+1), metric\_hist['train'], label='train')  
plt.plot(range(1, num\_epochs+1), metric\_hist['val'], label='val')  
plt.xlabel('Accuracy')  
plt.ylabel('Training Epochs')  
plt.legend()  
plt.show()

*Handwritten notes:*  
Epochs →  
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z  
Dish



Now coming back to training and evaluation, here in training and evaluation what we do is as I told you, we just made a function like for every batch I will compute the accuracy metric. Similarly, I will for every batch using this function I will compute the loss, using this function we are going to obtain the accuracy of a single batch. Similarly, we are going to obtain the loss for the single batch.

Once these 2 are done since device is CUDA if it is available. Now, our job is to compute the losses for every epoch, epoch means number of batches, sum of number of all number of batches. Here we are computing for every epoch I am computing the loss, for every epoch I am computing the accuracy.

Last computation loss of batch 1, loss of batch 2, likewise loss of batch 2 and like that coming to accuracy, accuracy of the batch 1 accuracy of batch 2 all these things will be summed up at the end we just divide with the length of the data. Suppose assume I have some 1000 data points in the entire data set I will take up these average, this average that is how we will get for every epoch the summary note I can say.

For every epoch what I get is loss value, average value, average loss value similarly, your average accuracy metric for every epoch, please keep these points in mind. Now, coming back to training. Loss function is done defining optimizer is done here we are using Adam optimizer and learning rate scheduler also I am using to if you do not see any improvement in the model training.

Now coming back to training and evaluation, this is one batch loss, similarly accuracy for batch here they are computing the loss for the entire epoch, loss accuracy for the entire epoch. Now coming back to the training and valuation. This is very important, please keep this one in mind what we are doing is as I told you earlier, we just sent the parameters only here.

Number of epochs, loss function, kind of loss function, kind of optimizer and DataLoader of the trained DataLoader, validation DataLoader and sanity check, we use sanity check, suppose I have made the model, instead for every suppose 100 epochs I have kept, assume that I have made the model.

And I am running the model for 100 epochs; that means in each and every epoch assume that there are 250 iterations, like 100 epoch I am running, in each epoch, in epoch 1 feeding data

suppose assume the data set points are like 1000, like 4 is the batch that size likewise, it will do for 250 times.

Suppose, since model training without disturbing I am just checking out whether the code is running well or not. That is why I kept the sanity check. So, you should have going for all these number 250 iterations, I am just checking that for all the 100 epochs I will go to check for first batch and if it is given the value and break the loop there.

So instead of going for the remaining 250 iterations. Likewise I will do for entire epoch, like 100 epochs at the 100th epoch, again I can do like I can first batch if it is updated then I will break this loop, just checking. If I keep the sanity check value as true that means it will go to a single epoch and it will compute all the stuff like 250 iteration it will do... it will compute the average loss, average accuracy and will give output as it will return those values.

Likewise you will do for all the epochs. Sanity check false we keep that is just to check whether the model is running or not well and giving the desired outputs, if you keep sanity check is equal to that is like a true training where you will wait for entire epoch training, entire epoch training like means all the iterations. Now coming back to how exactly they have plan this coding or this training loop.

I hope everybody is understanding, if you have still any questions, you can approach me in the discussion forums. See here what I am doing is at the end of the training after path 2 weights, after the model... once the model is completely trained I will save the weights of the model that is where we are going to do the pathway, path of the model.

Next thing is loss history train, at the end earlier while discussing linear regression them also I told you at the end we are going to draw the graphs like validation and training loss. See here to generate these kinds of graphs we are going to use these values. Loss history, metric history, this is the accuracy metric, this is loss metric, please keep this in mind.

After every epoch whatever the average loss value I got I will keep this one as a first entity similarly, accuracy also will keep in the second entity, validation loss also validation accuracy also like measurement. Best model weights, keep the model static. To load the index in a model I am just here assume that epoch number and it will give the current learning rate like outputs will be like this, this is how it is.

To generate while the model is training your job is to keep these values straight, trace these values. Current learning rate whether best model rates are coupled or not, this is depends on the validation loss. I will show you where exactly the condition is used. Sometimes as I told you earlier we are using the learning rate schedule, we are reducing the learning rate, there is no much improvement in the model.

At that point of time we once the learning rate is reduced I will use that strategy or the use the model rates, what were the best model rates till that time. Suppose after 50 epoch the model has reduced the learning rate to some another value, at that point of time till the 50 epochs I am going to save the best model rates till the 50th epoch.

Once that is done, I mean, this learning rate is off reduced at that point of time this model is again loaded with the best model rates, from there I will start the training with the new learning rate, that is the whole strategy. I will show you in the code where exactly I have written. See here it is our responsibility to keep the model in terms of train, train loss, train metric using this function.

This function is definitely above and after that train loss and train metric will be taken into this one. Similarly, now coming back to model level, in this mode we do not send any optimizer and all that stuff, at that time we want all the no back propagation, nothing we should do, no update of weights and everything, we just feed some data in terms of batches in validation loss and I will take the output.

I will compare the loss and everything that is what we should do, other than that nothing else. I will trace here validation loss and validation accuracy, assume that till the time assume that I have my initial value of the some best loss, validation losses I mean if I need. Suppose after several epoch, after 2 or 3 epochs, my valuation loss is less than this infinity.

That means it is a better validation loss, at the end of the day you need very less validation loss. So, at that point of time, I will just save the weights of that particular model. After again some, assume that first epoch we can see that validation loss is less than infinity. So, they are copying the weights, similarly after first epoch, suppose at that time, whatever validation loss I have, I will swap that value to, I will assign that value to best validation loss.

After second epoch also I will get the validation loss and I will compare this validation loss with the earlier validation loss, which I have stored earlier when the best, so that is the logic I

have used. Suppose if the new validation loss is less than the previous validation loss again I will save the weights because those are very good weights.

Similarly, see here copied best weights, copied best weights, copied best weights, copied best weights, copied best weights, copied best weights, after a certain number of epochs you cannot see that one. See here till 18 epoch it is good after that there is no such printing statement that means, there is no much improvement in the validation loss compared to earlier validation loss, that is the logic I have used.

Next thing is, I am explaining each and every segment so that you will get to know what is the importance of each and every thing. See here that is all I am doing here, next thing is learning rate scheduler step. Suppose the new validation loss after several epochs also there is no much improvement in the validation loss, here I have given the patience as 20.

It will wait up to 20 epoch, if that validation loss is not improving at that time they will go to go and reduce the values to some lesser learning rate, here I have some of the learning rate previous learning rate then at the same time I am going to load the best model weights. See I will show you, loss see here, in some of the cases we can see that best model weights loading also.

Likewise, we will do the training where by using the validation loss, by using the validation loss, I am just checking out my how exactly my model is good or not and saving the best model weights at every instant while reducing the learning rate also, what I am doing is I am loading the model with the best model weight till the number of epoch and I will again start the new training.

It happens automatically. That is how the entire code has been written. At the end of the training you will have losses for each and every training loss list. Similarly, you will have training loss, sorry validation loss list, training accuracy, similarly, validation accuracy using these 4 lists you can generate the graphs, that is the whole idea and at the same time you will get the model weights also stored.

See in the training loop of the training loop the model will be given as output, written like CNN model, loss history, metric history everything will be assigned. Metric in terms of metric means accuracy only here that is how we will use. See here what they have done is sanity check is true that means they are just checking going to each and every epoch and first do the training for first batch and update the weights.

Similarly, they will do for the next batch in the next epoch likewise, they are restricting themselves to first batch only. Next the original training starts, see here, only this is a sanity check only. And after the sanity check they are plotting the plot, usually we will assume the plots to be something like this. Suppose if it is valid... training loss, validation loss should be something like this curve.

This is where the point of interest, please keep that point in mind. This is just like that, some basic differences will be there, main plot is . Now, coming back to see here sanity check false that means, we are now allowing the entire model to train in a proper way, it will go to a epoch one and it will generate, it will compute the loss for first batch, in metric for first batch, go for second batch computed the loss and metric.

Go for third batch, compute the loss and metric, go to the batch 4, likewise you will do for every batch and computed the entire loss, you can see like this. Now you will obtain the graphs for both training loss and validation loss. Next your job is to once this is done, you will, you can save the models. Once this is done what is the step? Now the training is done. Suppose assume 100 epoch, I have done for 100 epoch.

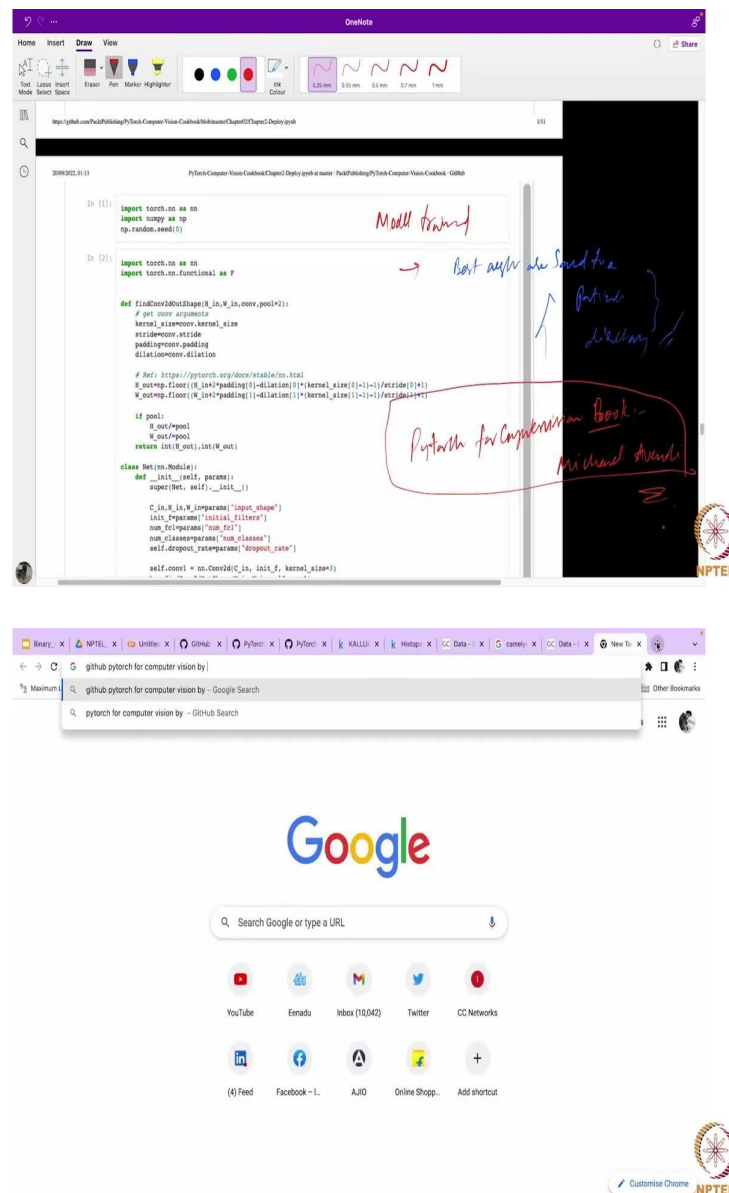
Sorry, you see here they have generated only for 2 epoch, you can keep this value as 100 and train this model. After 100 epochs you will get to see lower and see the best model weights and those model weights can be used for testing purpose. You can change these values, number of epochs or law, what we say, mostly this one, learning rate scheduler also, go to learning residual function that is the starting value I think.

You can even change that value also, learning rate scheduler you can see here, you can change this value also. Like optimizer opt they have given I think, optimizer is equal opt, you can change that value also. These are the things we call as hyper parameters we will tune the model. Till now we have done the training part. This is how the code has been done.

This is for sanity check, this graph, whereas these 2 graphs are for after entire training, but there it is still done to 2 epochs only that is why it is showing like this, but you can train this model for 100 epochs and do some more depending on your capacity or competition usually that you have, you can use this stuff. Now, we can say training is finished and we have saved the best model weights. Next we will concentrate on testing of the model after deploying.



(Refer Slide Time: 81: 48)





Browser tabs: NPTEL, Unlabeled, GitHub, PyTorch, KALI, Histo, CC Data, camr, CC Data, GitHub, GitHub.

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook

Product · Solutions · Open Source · Pricing

Search [ ] Sign in Sign up

PacktPublishing / PyTorch-Computer-Vision-Cookbook Public

Notifications Fork 147 Star 261

Code Issues Pull requests Actions Security Insights

master 1 branch 0 tags

Go to file Code

About

PyTorch Computer Vision Cookbook, Published by Packt

Readme MIT license 261 stars 10 watching 147 forks

Releases

No releases published

Packages

No packages published

Contributors 4

ManikandanKurup-Packt Mar1

Packt-ITService

File	Commit	Time
Packt-ITService remove \$5 campaign	811bfc	on 21 Jan 2021 21 commits
Chapter01	Code files uploaded	3 years ago
Chapter02	Code files uploaded	3 years ago
Chapter03	Code files uploaded	3 years ago
Chapter04	fixed vertical flip function bug in Ch4	2 years ago
Chapter05	Code files uploaded	3 years ago
Chapter06	Chapter 6: bug fixed CUDA memory error	2 years ago
Chapter07	Code files uploaded	3 years ago
Chapter08	Code files uploaded	3 years ago
Chapter09	Code files added	3 years ago
Chapter10	Rename Chapter11.lynb to Chapter10.lynb	3 years ago
LICENSE	Initial commit	3 years ago
README.md	remove \$5 campaign	2 years ago
cv_pytorch.txt	Files uploaded	3 years ago
cv_pytorch.yml	Files uploaded	3 years ago

Browser tabs: NPTEL, Unlabeled, GitHub, PyTorch, KALI, Histo, CC Data, camr, CC Data, GitHub, GitHub.

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook

README.md

use several different algorithms for different CV problems such as classification, detection, segmentation, and more using Pytorch. Packed with best practices in training and deployment of CV applications.

This book covers the following exciting features:

- Develop, train and deploy deep learning algorithms using PyTorch 1.x
- Understand how to fine-tune and change hyperparameters to train deep learning algorithms
- Perform various CV tasks such as classification, detection, and segmentation
- Implement a neural style transfer network based on CNNs and pre-trained models
- Generate new images and implement adversarial attacks using GANs
- Implement video classification models based on RNN, LSTM, and 3D-CNN
- Discover best practices for training and deploying deep learning algorithms for CV applications

If you feel this book is for you, get your copy today!

### Instructions and Navigations

All of the code is organized into folders.

The code will look like the following:

```
# define a tensor with specific data type
x = torch.ones(2, 2, dtype=torch.int8)
print(x)
print(x.dtype)
tensor([[[, ],
        [, ]], dtype=torch.int8)
torch.int8
```

Browser tabs: NPTEL, Unlabeled, GitHub, PyTorch, KALI, Histo, CC Data, camr, CC Data, GitHub, GitHub.

github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook

PyTorch Computer Vision Cookbook, Published by Packt

Readme MIT license 261 stars 10 watching 147 forks

Releases

No releases published

Packages

No packages published

Contributors 4

ManikandanKurup-Packt Mar1

Packt-ITService

caojie5231 Joseph Sunil

mnauf Muhammad Naufal

File	Commit	Time
Packt-ITService remove \$5 campaign	811bfc	on 21 Jan 2021 21 commits
Chapter01	Code files uploaded	3 years ago
Chapter02	Code files uploaded	3 years ago
Chapter03	Code files uploaded	3 years ago
Chapter04	fixed vertical flip function bug in Ch4	2 years ago
Chapter05	Code files uploaded	3 years ago
Chapter06	Chapter 6: bug fixed CUDA memory error	2 years ago
Chapter07	Code files uploaded	3 years ago
Chapter08	Code files uploaded	3 years ago
Chapter09	Code files added	3 years ago
Chapter10	Rename Chapter11.lynb to Chapter10.lynb	3 years ago
LICENSE	Initial commit	3 years ago
README.md	remove \$5 campaign	2 years ago
cv_pytorch.txt	Files uploaded	3 years ago
cv_pytorch.yml	Files uploaded	3 years ago
errata.md	Update errata.md	2 years ago

### PyTorch-Computer-Vision-Cookbook

This is the code repository for PyTorch Computer Vision Cookbook, published by Packt.

PyTorch Computer Vision Cookbook

Languages

- Jupyter Notebook 98.7%
- Python 1.3%

[illegible]

The screenshot shows the OneNote application with a code editor. The code is a Python implementation of a 2D convolution layer using PyTorch. The code is annotated with red handwritten notes and red circles. The notes include 'f' for the input feature map, 'k' for the kernel, 's' for the stride, 'p' for the padding, and 'd' for the dilation. The code also includes a 'main' function that tests the module with a 10x10x1 input tensor and a 3x3x1 kernel, resulting in a 7x7x1 output tensor.

```

13 177
import torch.nn as nn
import torch.nn.functional as F

def conv2d(torch.nn.Module):
    # get our arguments
    kernel_size=kernel_size
    stride=stride
    padding=padding
    dilation=dilation

    # Ref: https://pytorch.org/docs/stable/nn.html
    w_out=nn.Conv2d(in_channels=1, out_channels=1, kernel_size=1, stride=1, padding=1, dilation=1)
    w_out=nn.Conv2d(in_channels=1, out_channels=1, kernel_size=1, stride=1, padding=1, dilation=1)

    if pool:
        w_out=nn.MaxPool2d(2, stride=2, padding=0, dilation=1)
        return torch.nn.Sequential(w_out, w_out)

    class Conv2d(torch.nn.Module):
        def __init__(self, in_channels, out_channels, kernel_size, stride, padding, dilation):
            super(Conv2d, self).__init__()

            C_in=torch.nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=kernel_size, stride=stride, padding=padding, dilation=dilation)
            self.conv1=C_in
            self.conv2=C_in
            self.conv3=C_in
            self.conv4=C_in
            self.conv5=C_in
            self.conv6=C_in
            self.conv7=C_in
            self.conv8=C_in
            self.conv9=C_in
            self.conv10=C_in
            self.conv11=C_in
            self.conv12=C_in
            self.conv13=C_in
            self.conv14=C_in
            self.conv15=C_in
            self.conv16=C_in
            self.conv17=C_in
            self.conv18=C_in
            self.conv19=C_in
            self.conv20=C_in
            self.conv21=C_in
            self.conv22=C_in
            self.conv23=C_in
            self.conv24=C_in
            self.conv25=C_in
            self.conv26=C_in
            self.conv27=C_in
            self.conv28=C_in
            self.conv29=C_in
            self.conv30=C_in
            self.conv31=C_in
            self.conv32=C_in
            self.conv33=C_in
            self.conv34=C_in
            self.conv35=C_in
            self.conv36=C_in
            self.conv37=C_in
            self.conv38=C_in
            self.conv39=C_in
            self.conv40=C_in
            self.conv41=C_in
            self.conv42=C_in
            self.conv43=C_in
            self.conv44=C_in
            self.conv45=C_in
            self.conv46=C_in
            self.conv47=C_in
            self.conv48=C_in
            self.conv49=C_in
            self.conv50=C_in
            self.conv51=C_in
            self.conv52=C_in
            self.conv53=C_in
            self.conv54=C_in
            self.conv55=C_in
            self.conv56=C_in
            self.conv57=C_in
            self.conv58=C_in
            self.conv59=C_in
            self.conv60=C_in
            self.conv61=C_in
            self.conv62=C_in
            self.conv63=C_in
            self.conv64=C_in
            self.conv65=C_in
            self.conv66=C_in
            self.conv67=C_in
            self.conv68=C_in
            self.conv69=C_in
            self.conv70=C_in
            self.conv71=C_in
            self.conv72=C_in
            self.conv73=C_in
            self.conv74=C_in
            self.conv75=C_in
            self.conv76=C_in
            self.conv77=C_in
            self.conv78=C_in
            self.conv79=C_in
            self.conv80=C_in
            self.conv81=C_in
            self.conv82=C_in
            self.conv83=C_in
            self.conv84=C_in
            self.conv85=C_in
            self.conv86=C_in
            self.conv87=C_in
            self.conv88=C_in
            self.conv89=C_in
            self.conv90=C_in
            self.conv91=C_in
            self.conv92=C_in
            self.conv93=C_in
            self.conv94=C_in
            self.conv95=C_in
            self.conv96=C_in
            self.conv97=C_in
            self.conv98=C_in
            self.conv99=C_in
            self.conv100=C_in
            self.conv101=C_in
            self.conv102=C_in
            self.conv103=C_in
            self.conv104=C_in
            self.conv105=C_in
            self.conv106=C_in
            self.conv107=C_in
            self.conv108=C_in
            self.conv109=C_in
            self.conv110=C_in
            self.conv111=C_in
            self.conv112=C_in
            self.conv113=C_in
            self.conv114=C_in
            self.conv115=C_in
            self.conv116=C_in
            self.conv117=C_in
            self.conv118=C_in
            self.conv119=C_in
            self.conv120=C_in
            self.conv121=C_in
            self.conv122=C_in
            self.conv123=C_in
            self.conv124=C_in
            self.conv125=C_in
            self.conv126=C_in
            self.conv127=C_in
            self.conv128=C_in
            self.conv129=C_in
            self.conv130=C_in
            self.conv131=C_in
            self.conv132=C_in
            self.conv133=C_in
            self.conv134=C_in
            self.conv135=C_in
            self.conv136=C_in
            self.conv137=C_in
            self.conv138=C_in
            self.conv139=C_in
            self.conv140=C_in
            self.conv141=C_in
            self.conv142=C_in
            self.conv143=C_in
            self.conv144=C_in
            self.conv145=C_in
            self.conv146=C_in
            self.conv147=C_in
            self.conv148=C_in
            self.conv149=C_in
            self.conv150=C_in
            self.conv151=C_in
            self.conv152=C_in
            self.conv153=C_in
            self.conv154=C_in
            self.conv155=C_in
            self.conv156=C_in
            self.conv157=C_in
            self.conv158=C_in
            self.conv159=C_in
            self.conv160=C_in
            self.conv161=C_in
            self.conv162=C_in
            self.conv163=C_in
            self.conv164=C_in
            self.conv165=C_in
            self.conv166=C_in
            self.conv167=C_in
            self.conv168=C_in
            self.conv169=C_in
            self.conv170=C_in
            self.conv171=C_in
            self.conv172=C_in
            self.conv173=C_in
            self.conv174=C_in
            self.conv175=C_in
            self.conv176=C_in
            self.conv177=C_in
            self.conv178=C_in
            self.conv179=C_in
            self.conv180=C_in
            self.conv181=C_in
            self.conv182=C_in
            self.conv183=C_in
            self.conv184=C_in
            self.conv185=C_in
            self.conv186=C_in
            self.conv187=C_in
            self.conv188=C_in
            self.conv189=C_in
            self.conv190=C_in
            self.conv191=C_in
            self.conv192=C_in
            self.conv193=C_in
            self.conv194=C_in
            self.conv195=C_in
            self.conv196=C_in
            self.conv197=C_in
            self.conv198=C_in
            self.conv199=C_in
            self.conv200=C_in
            self.conv201=C_in
            self.conv202=C_in
            self.conv203=C_in
            self.conv204=C_in
            self.conv205=C_in
            self.conv206=C_in
            self.conv207=C_in
            self.conv208=C_in
            self.conv209=C_in
            self.conv210=C_in
            self.conv211=C_in
            self.conv212=C_in
            self.conv213=C_in
            self.conv214=C_in
            self.conv215=C_in
            self.conv216=C_in
            self.conv217=C_in
            self.conv218=C_in
            self.conv219=C_in
            self.conv220=C_in
            self.conv221=C_in
            self.conv222=C_in
            self.conv223=C_in
            self.conv224=C_in
            self.conv225=C_in
            self.conv226=C_in
            self.conv227=C_in
            self.conv228=C_in
            self.conv229=C_in
            self.conv230=C_in
            self.conv231=C_in
            self.conv232=C_in
            self.conv233=C_in
            self.conv234=C_in
            self.conv235=C_in
            self.conv236=C_in
            self.conv237=C_in
            self.conv238=C_in
            self.conv239=C_in
            self.conv240=C_in
            self.conv241=C_in
            self.conv242=C_in
            self.conv243=C_in
            self.conv244=C_in
            self.conv245=C_in
            self.conv246=C_in
            self.conv247=C_in
            self.conv248=C_in
            self.conv249=C_in
            self.conv250=C_in
            self.conv251=C_in
            self.conv252=C_in
            self.conv253=C_in
            self.conv254=C_in
            self.conv255=C_in
            self.conv256=C_in
            self.conv257=C_in
            self.conv258=C_in
            self.conv259=C_in
            self.conv260=C_in
            self.conv261=C_in
            self.conv262=C_in
            self.conv263=C_in
            self.conv264=C_in
            self.conv265=C_in
            self.conv266=C_in
            self.conv267=C_in
            self.conv268=C_in
            self.conv269=C_in
            self.conv270=C_in
            self.conv271=C_in
            self.conv272=C_in
            self.conv273=C_in
            self.conv274=C_in
            self.conv275=C_in
            self.conv276=C_in
            self.conv277=C_in
            self.conv278=C_in
            self.conv279=C_in
            self.conv280=C_in
            self.conv281=C_in
            self.conv282=C_in
            self.conv283=C_in
            self.conv284=C_in
            self.conv285=C_in
            self.conv286=C_in
            self.conv287=C_in
            self.conv288=C_in
            self.conv289=C_in
            self.conv290=C_in
            self.conv291=C_in
            self.conv292=C_in
            self.conv293=C_in
            self.conv294=C_in
            self.conv295=C_in
            self.conv296=C_in
            self.conv297=C_in
            self.conv298=C_in
            self.conv299=C_in
            self.conv300=C_in
            self.conv301=C_in
            self.conv302=C_in
            self.conv303=C_in
            self.conv304=C_in
            self.conv305=C_in
            self.conv306=C_in
            self.conv307=C_in
            self.conv308=C_in
            self.conv309=C_in
            self.conv310=C_in
            self.conv311=C_in
            self.conv312=C_in
            self.conv313=C_in
            self.conv314=C_in
            self.conv315=C_in
            self.conv316=C_in
            self.conv317=C_in
            self.conv318=C_in
            self.conv319=C_in
            self.conv320=C_in
            self.conv321=C_in
            self.conv322=C_in
            self.conv323=C_in
            self.conv324=C_in
            self.conv325=C_in
            self.conv326=C_in
            self.conv327=C_in
            self.conv328=C_in
            self.conv329=C_in
            self.conv330=C_in
            self.conv331=C_in
            self.conv332=C_in
            self.conv333=C_in
            self.conv334=C_in
            self.conv335=C_in
            self.conv336=C_in
            self.conv337=C_in
            self.conv338=C_in
            self.conv339=C_in
            self.conv340=C_in
            self.conv341=C_in
            self.conv342=C_in
            self.conv343=C_in
            self.conv344=C_in
            self.conv345=C_in
            self.conv346=C_in
            self.conv347=C_in
            self.conv348=C_in
            self.conv349=C_in
            self.conv350=C_in
            self.conv351=C_in
            self.conv352=C_in
            self.conv353=C_in
            self.conv354=C_in
            self.conv355=C_in
            self.conv356=C_in
            self.conv357=C_in
            self.conv358=C_in
            self.conv359=C_in
            self.conv360=C_in
            self.conv361=C_in
            self.conv362=C_in
            self.conv363=C_in
            self.conv364=C_in
            self.conv365=C_in
            self.conv366=C_in
            self.conv367=C_in
            self.conv368=C_in
            self.conv369=C_in
            self.conv370=C_in
            self.conv371=C_in
            self.conv372=C_in
            self.conv373=C_in
            self.conv374=C_in
            self.conv375=C_in
            self.conv376=C_in
            self.conv377=C_in
            self.conv378=C_in
            self.conv379=C_in
            self.conv380=C_in
            self.conv381=C_in
            self.conv382=C_in
            self.conv383=C_in
            self.conv384=C_in
            self.conv385=C_in
            self.conv386=C_in
            self.conv387=C_in
            self.conv388=C_in
            self.conv389=C_in
            self.conv390=C_in
            self.conv391=C_in
            self.conv392=C_in
            self.conv393=C_in
            self.conv394=C_in
            self.conv395=C_in
            self.conv396=C_in
            self.conv397=C_in
            self.conv398=C_in
            self.conv399=C_in
            self.conv400=C_in
            self.conv401=C_in
            self.conv402
```

The screenshot shows a Google Colab notebook with the following code:

```

W_out=torch.zeros(
    return init(W_out),list(W_out)

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

        C_in,W_in=params["input_shape"]
        init_params["initial_filters"]
        num_fc=params["num_fc"]
        num_linear=params["num_linear"]
        self.dropout_rate=params["dropout_rate"]

        self.conv1 = nn.Conv2d(C_in, init_f, kernel_size=
        h_w=initConv2dShape(W_in,W_in,self.conv1)

        self.conv2 = nn.Conv2d(init_f, 2*init_f, kernel_size=
        h_w=initConv2dShape(h_w,self.conv2)

        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, kernel_size=
        h_w=initConv2dShape(h_w,self.pool1)

        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, kernel_size=
        h_w=initConv2dShape(h_w,self.pool2)

        self.fc1 = nn.Linear(self.conv2.out_channels, num_fc)
        self.fc2 = nn.Linear(self.fc1.out_channels, num_linear)
        self.linear = nn.Linear(num_linear, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)

        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)

        x = F.relu(self.pool1(x))
        x = F.max_pool2d(x, 2, 2)

        x = F.relu(self.pool2(x))
        x = F.max_pool2d(x, 2, 2)

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.linear(x)
        return x

```

The notebook interface includes a toolbar with drawing tools and a red handwritten mark on the right side.



The image consists of two screenshots of a OneNote application, showing Python code for model deployment. The top screenshot shows the initial code with handwritten annotations. The bottom screenshot shows the code after modifications, with more annotations and a red checkmark.

**Top Screenshot:**

```
conv1 = Conv2d(3, 64, kernel_size=3, stride=1, bias=False)
fc1 = Linear(16, 1000, bias=True)
fc2 = Linear(1000, 1000, bias=True)

# save model to cuda/gpu device
if torch.cuda.is_available():
    device = torch.device("cuda")
    net_model = net_model.to(device)

In [6]:

In [7]:
import time
def deploy_model(model, dataset, device, num_classes=1, sanity_check=False):
    len_data=len(dataset)
    # initialize output tensor on CPU due to GPU memory limits
    y_out=torch.zeros(len_data,num_classes)
    # initialize ground truth on CPU due to GPU memory limits
    y_true=torch.zeros(len_data, dtype="uint8")
    # move model to device
    model=model.to(device)
    elapsed_times=[]
    with torch.no_grad():
        for i in range(len_data):
            x,y=dataset[i]
            y_out[i]=model(x.unsqueeze(0)).to(device)
            elapsed_time=time.time()-start
            elapsed_times.append(elapsed_time)
            if sanity_check is True:
                break
    inference_time=mean(elapsed_times)*1000
    print("Average inference time per image on %s: %.2f ms" % (device, inference_time))

In [8]:
import torch
from PIL import Image
from torch.utils.data import Dataset
import pandas as pd
import torchvision.transforms as transforms
import os
```

**Bottom Screenshot:**

```
In [7]:
import time
def deploy_model(model, dataset, device, num_classes=1, sanity_check=False):
    len_data=len(dataset)
    # initialize output tensor on CPU due to GPU memory limits
    y_out=torch.zeros(len_data,num_classes)
    # initialize ground truth on CPU due to GPU memory limits
    y_true=torch.zeros(len_data, dtype="uint8")
    # move model to device
    model=model.to(device)
    elapsed_times=[]
    with torch.no_grad():
        for i in range(len_data):
            x,y=dataset[i]
            y_out[i]=model(x.unsqueeze(0)).to(device)
            elapsed_time=time.time()-start
            elapsed_times.append(elapsed_time)
            if sanity_check is True:
                break
    inference_time=mean(elapsed_times)*1000
    print("Average inference time per image on %s: %.2f ms" % (device, inference_time))
    return y_out.unsqueeze(0).to(device)

In [8]:
import torch
from PIL import Image
from torch.utils.data import Dataset
import pandas as pd
import torchvision.transforms as transforms
import os
```

Now, let us focus on best way to say what we have done is model has been trained, model trained. And the best weights are saved to a particular, later in sense one folder I have kept loading models in my Google Drive where the entire directory is present, there the model will be saved.

We forgot to mention all these codes and everything what books and Jupyter books I am using are from that is a by Pytorch for computer vision book, pytorch for computer vision book by Michael Avendi. Michael Avendi he basically works in medical image analysis, all his the Jupyter Notebooks belongs to this particular textbook or already kept in Google, I will show you.

If you go to this one the chapter 2, these notebooks have taken, not only for this one, this will gives you something like a basic understanding, you can go and use the Jupyter Notebooks depending on the binary class within a multiclassificaton and something like segmentation tasks, everything is provided in this particular book, where you can use the codes for your work, you can modify those codes according to your usage also. Just for the sake of information I am saying.

Coming back to deployment stage of testing phase. We have to before loading the model weights we have save the model in terms of state dict. In Torch it is available. It is the kind of strategy we will use in terms of this particular format. Whatever I have developed before training the model, the same thing I have done here like calculate the output share for every after every convolution block.

Later, you can see model has been defined, convolution layer, convolutional layers, usually in terms of layers we call convolutional layer and fully dense layers. This max pooling layer and activation function on the layer, all these things cannot be called as layers, only these convolutional layers, max pool operation everything is done, model A has been developed and the output layer and we will see a loss function like loss, sorry function like log underscore softmax activation.

I have already discuss this earlier. Next thing the parameters here see input shape is this one, here we have removed the optimizer we have removed, number of epochs we have removed because we are not doing training here we are just testing, suppose if I use input images and should predict the output like or to which class this particular image belongs to which class.

That means if I give the input as this one it should give me the output as a label that it will have to 0 or 1 that is the whole idea. So dropout rate number of flicker interlace and number of classes. Model net of parameters model see here, wherever I have saved the models, I just have to give the path from here, this particular path can give you the path dot pt file.

So, the models will be loaded using this one. Now, the model has been loaded with the best model weights, best model weights, weights that are obtained through those are obtained through training, please keep this point in mind. Now, please keep the model in the evaluation model not on clean mode, that is whatever device you have, suppose GPU please use this command.

Now coming to deployment, this function they have created just to obtain the number of sorry, predict the output in terms of numpy format for the given any image, it predicts the output. Similarly ground truth value also, for the data set, we will send the test dataset or the test data set will send the test dataset.

It just computes the average time, it has taken to compute the inference to compute the value see here. Depending on the data set, In of the data set will take and depending on the number of images and we will just obtaining the output.

(Refer Slide Time: 87:26)

```

In [8]:
import torch
from PIL import Image
from torch.utils.data import Dataset
import pandas as pd
import torchvision.transforms as transforms
import os

# fix torch random seed
torch.manual_seed(1)

class HistoCancerDataset(Dataset):
    def __init__(self, data_dir, transform, data_type="train"):
        # path to images
        path_to_images = os.path.join(data_dir, data_type)

        # get a list of images
        self.filesname = os.listdir(path_to_images)

        # get the full path to images
        self.full_filenames = [os.path.join(path_to_images, f) for f in self.filesname]

        # labels are in a csv file named train_labels.csv
        csv_filename = data_dir + "train_labels.csv"
        path_to_labels = os.path.join(data_dir, csv_filename)
        labels_df = pd.read_csv(path_to_labels)

        # set data frame index to id
        self.labels = labels_df.set_index("id").to_dict()["label"]

    def __len__(self):
        # return size of dataset
        return len(self.full_filenames)

    def __getitem__(self, idx):
        # open image, apply transform and return with label
        image = Image.open(self.full_filenames[idx]) # PIL image
        image = self.transform(image)
        return image, self.labels[idx]

In [9]:
import torchvision.transforms as transforms
data_transformer = transforms.Compose([transforms.ToTensor()])

In [10]:
data_dir = "/data/"
histo_dataset = HistoCancerDataset(data_dir, data_transformer, "train")
print(len(histo_dataset))

229025

In [11]:
from torch.utils.data import random_split
len_hist = len(histo_dataset)
len_train = int(0.8 * len_hist)
len_val = len_hist - len_train

train_ds, val_ds = random_split(histo_dataset, [len_train, len_val])

print('train dataset length:', len(train_ds))
print('validation dataset length:', len(val_ds))

train dataset length: 176020
validation dataset length: 44005

In [12]:

```



```
histo_dataset = histoCancerDataset(data_dir, data_transformer, "train")
print(len(histo_dataset))

220925

In [11]:
from torch.utils.data import random_split
len_histo=len(histo_dataset)
len_train=int(0.7*len_histo)
len_val=len_histo-len_train

train_ds,val_ds=random_split(histo_dataset,[len_train,len_val])

print('train dataset length:', len(train_ds))
print('validation dataset length:', len(val_ds))

train_dataset_length=13440
validation_dataset_length=8480

https://github.com/PyTorch/PyTorch-Custom-Vision-Cookbook/blob/master/Chapter02/Chapter02_Demos.ipynb

20092021.01.13 PyTorch-Custom-Vision-Cookbook/Chapter02_Demos.ipynb at master · PyTorch/PyTorch-Custom-Vision-Cookbook · GitHub

# deploy model
y_out,y_gt=deploy_model(cnn_model,val_ds,device=device,sanity_check=False)
print(y_out.shape,y_gt.shape)

average inference time per image on cuda: 0.70 ms
(44005, 2) (44005, 2)

Accuracy

In [13]:
from sklearn.metrics import accuracy_score

# get predictions
y_pred = np.argmax(y_out,axis=1)
print(y_pred.shape,y_gt.shape)

# compute accuracy
accuracy_score(y_pred,y_gt)
print('accuracy: %.2f' %acc)
```

```
train_dataset_length=13440
validation_dataset_length=8480

In [12]:
https://github.com/PyTorch/PyTorch-Custom-Vision-Cookbook/blob/master/Chapter02/Chapter02_Demos.ipynb

20092021.01.13 PyTorch-Custom-Vision-Cookbook/Chapter02_Demos.ipynb at master · PyTorch/PyTorch-Custom-Vision-Cookbook · GitHub

# deploy model
y_out,y_gt=deploy_model(cnn_model,val_ds,device=device,sanity_check=False)
print(y_out.shape,y_gt.shape)

average inference time per image on cuda: 0.70 ms
(44005, 2) (44005, 2)

Accuracy

In [13]:
from sklearn.metrics import accuracy_score

# get predictions
y_pred = np.argmax(y_out,axis=1)
print(y_pred.shape,y_gt.shape)

# compute accuracy
accuracy_score(y_pred,y_gt)
print('accuracy: %.2f' %acc)

(44005, 2) (44005, 2)
accuracy: 0.94

Deploy on CPU

In [14]:
# deploy model on cpu
device_cpu = torch.device("cpu")
y_out,y_gt=deploy_model(cnn_model,val_ds,device=device_cpu,sanity_check=False)
print(y_out.shape,y_gt.shape)

average inference time per image on cpu: 2.24 ms
(44005, 2) (44005, 2)

Model Inference on Test Data
```

```
self.transformer = transformer

def __len__(self):
    # return size of dataset
    return len(self.full_filenames)

def __getitem__(self, idx):
    # use image, apply transform and return with label
    image = image_ops[self.full_filenames[idx]] # PIL image
    image = self.transform(image)
    return image, self.labels[idx]

In [18]:
import torchvision.transforms as transforms
data_transformer = transforms.Compose([transformer.Softmax()])

In [19]:
data_dir = "../data/"
histo_dataset = histoCancerDataset(data_dir, data_transformer, "train")
print(len(histo_dataset))

220925

In [11]:
from torch.utils.data import random_split
len_histo=len(histo_dataset)
len_train=int(0.7*len_histo)
len_val=len_histo-len_train

train_ds,val_ds=random_split(histo_dataset,[len_train,len_val])

print('train dataset length:', len(train_ds))
print('validation dataset length:', len(val_ds))

train_dataset_length=13440
validation_dataset_length=8480

https://github.com/PyTorch/PyTorch-Custom-Vision-Cookbook/blob/master/Chapter02/Chapter02_Demos.ipynb

20092021.01.13 PyTorch-Custom-Vision-Cookbook/Chapter02_Demos.ipynb at master · PyTorch/PyTorch-Custom-Vision-Cookbook · GitHub
```



Maybe we can test later on whenever test data set is provided. Here we are just using the validation data set only just to predict the outputs at the comparing with the ground truths maybe that is why, train dataset length is this one. And validation data set length is like 176000 and 44 maybe I will use these things now. Deploy models, CNN model, validation dataset.

Now we have send the entire validation dataset like all these numbers. Now, if you observe the output shape, it is outputting the values likes (44005,2) that means because we have 2 what we say 2 classes. That is why it indicates, suppose you have 3 classes, it indicates 3, for every data point for every prediction, you will have every prediction you have 3 points and on that we use the argmax function it is very helpful.

argmax function is very helpful and we regularly use. Please make yourself comfortable with how to use this argmax value. On this one you can see now we have suppose, I will give you example, let us say 0.5, sorry, 0.6 or some 0.4 this class belongs to 0 this class belongs to 1, at that point of time and I will np dot argmax will return the value as 0.

In case the values are like this 0.4 and 0.6 at that case, np dot orgmax will send the index value as 1. I mean, it sends that index value in which the maximum value occur. After that from this value (44005,2) now the uptrend values (44005) because using RNS we have predicted the levels. Here they have done for and they have computed the accuracy also, the accuracy value is 94 percentage that means, they have tested the model on validation data set only till now.

I told you earlier test data set, we will go for test data set later, but first we will compute the loss and the accuracy on the weight test, sorry, validation data only, validation data set has been given as an input and we have obtained the output and this is giving me a compared with ground truth values and now the output is 95%. This one we have done on the GPU device I think. Now, they are comparing with the values on CPU also, there is no much difference and no need to discuss also this stuff.

(Refer Slide Time: 91:51)

average inference time per image on cpu: 2.24 ms (44005, 2) (44005, 1)

Model Inference on Test Data

In [151]:

```
path32rev='./data/test_label1.csv'
label1_df=pd.read_csv(path32rev)
label1_df.head()
```

http://bit.ly/2PwPbYgYrTorch-Cmpu-Vision-Cookbook/Chapter01/Chapter01-Deploy-cpu

2019-02-01 15:11 PyTorch-Cmpu-Vision-Cookbook/Chapter01-Deploy-cpu on name: torch-Publishing/PyTorch-Cmpu-Vision-Cookbook: GitHub

Out[151]:

	label
0	0b2ea2822a6271b1b7a5620513aa3999c32c0c5
1	9559092a0c566c5c3446c90d6f9839c392737
2	24846738880a2ebc6125dc0c7127299dc76814
3	1c36657312964e9284ac684772093a768df
4	145782a75ea1c316ac3a5a549a3721a3106

In [161]:

```
data_dir='./data'
histio_test = histioCancerDataset(data_dir, data_transformer, data_type='test')
print(len(histio_test))

57458
```

In [171]:

```
y_test_out_=_deploy_model(cnn_model,histio_test, device, sanity_check=False)

y_test_pred=np.argmax(y_test_out,axis=1)
print(y_test_pred.shape)
```

average inference time per image on cuda: 0.49 ms (57458, 1) (57458, 1)

[illegible]

OneNote

Home Insert Draw View

Test Lines Short Mode Select Space

Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.50 mm 0.7 mm 1 mm

```

plt.imshow(imging_tr, interpolation='nearest', cmap='gray')

# display images
plt.imshow(imging_tr, interpolation='nearest')
plt.title('test')

grid_size=4
rnd_idx=np.random.randint(0, len(histo_test), grid_size)
print('image indices:', rnd_idx)

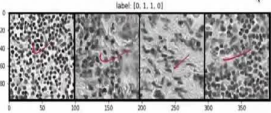
x_grid_test=[histo_test[i][0] for i in range(grid_size)]
y_grid_test=[histo_test[i][1] for i in range(grid_size)]

x_grid_test=utils.make_grid(x_grid_test, nrow=4, padding=2)
print(x_grid_test.shape)

plt.rcParams['figure.figsize'] = (10, 5)
show(x_grid_test, y_grid_test)

image_indices = [ 2732 43567 42613 52416]
torch.Size([3, 100, 394])
label [0.1 1.0]

```



Create Submission

```

In [19]:
print(y_test_out.shape)
cancer_grade = sp.exp_y_test_out[:, 1]
print(cancer_grade.shape)

(57458, 2)

```

[https://github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter02\\_Display.ipynb](https://github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter02_Display.ipynb)

NPTEL

OneNote

Home Insert Draw View

Test Lines Short Mode Select Space

Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.50 mm 0.7 mm 1 mm

[https://github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter02\\_Display.ipynb](https://github.com/PacktPublishing/PyTorch-Computer-Vision-Cookbook/blob/master/Chapter02/Chapter02_Display.ipynb)

```

20093022.81.13 PyTorch Computer Vision Cookbook Chapter02_Display.ipynb at master · PacktPublishing/PyTorch-Computer-Vision-Cookbook · GitHub

imging_tremping_tr[:, :, 0]
plt.imshow(imging_tr, interpolation='nearest', cmap='gray')

# display images
plt.imshow(imging_tr, interpolation='nearest')
plt.title('label: "test")

grid_size=4
rnd_idx=np.random.randint(0, len(histo_test), grid_size)
print('image indices:', rnd_idx)

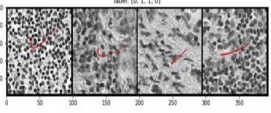
x_grid_test=[histo_test[i][0] for i in range(grid_size)]
y_grid_test=[histo_test[i][1] for i in range(grid_size)]

x_grid_test=utils.make_grid(x_grid_test, nrow=4, padding=2)
print(x_grid_test.shape)

plt.rcParams['figure.figsize'] = (10, 5)
show(x_grid_test, y_grid_test)

image_indices = [ 2732 43567 42613 52416]
torch.Size([3, 100, 394])
label [0.1 1.0]

```



Create Submission

```

In [19]:
print(y_test_out.shape)
cancer_grade = sp.exp_y_test_out[:, 1]

```

NPTEL

OneNote

Home Insert Draw View

Test Lines Short Mode Select Space

Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.50 mm 0.7 mm 1 mm

KALLURI's Noteb...

WA\_2

End\_M...

NPTEL

Binary...

Untitled Page

20093022.81.13 PyTorch Computer Vision Cookbook Chapter02\_Display.ipynb at master · PacktPublishing/PyTorch-Computer-Vision-Cookbook · GitHub

```

(57458, 2)

In [20]:
# get test id's from the sample_submission.csv
path2sampleId = "../data/" + "sample_submission.csv"

# read sample submission
sample_df = pd.read_csv(path2sampleId)

# get id column
ids_list = list(sample_df.id)

# convert predictions to list
pred_list = [p for p in cancer_preds]

# create a dict of id and prediction
pred_dic = dict(zip(ids_list, pred_list))

# re-order predictions to match sample_submission.csv
pred_list_sub = [pred_dic[id_] for id_ in ids_list]

# create convert to data frame
submission_df = pd.DataFrame({'id': ids_list, 'label': pred_list_sub})

```

Create Submission

NPTEL

Coming back to model inference on test dataset since on the test data set, we do not have any ground truth values or the original values of the book ports to which class it belongs. So, we just have to predict the outputs and have to similarly send the data suppose this is how it is been given just made the test dataset class here histo data set, this is out and deploy the model and you can send `np.argmax` and you got that this one also.

And you can, note it you need this function. After this you can even compute the accuracy also how we have computed on the validation data set. But to do so, we need ground truth, right. So, we do not have, so we cannot compare those values. But we can predict the values. I mean, what class it has given like 0 or 1 or everything.

So, suppose in case we have 5 7 4 5 8 testaments are here, now we have 5 4 5 7 4 5 8 output array containing a single dimension where it all the values of 0 1 0 1 1 1 0 like to each class particular it belongs. We are just predicting using our model that is it nothing else. Here you can see all the original images, maybe you can see why they are grey they have just replace this color value with false.

See here if color W equal to false they have written this one else this one. That is the basic difference. This is how we will train the entire classification model using the basic sequential network I have used here. You can even go for transfer learning where the model will converge faster that is it, you just go and verify on the internet and check out the details. You guys need not concentrate on this one this condition is more than enough. Thank you so much for the lecture.

(Refer Slide Time: 94:00)

The screenshot shows a OneNote page titled "Normal versus Malignant". The left sidebar lists various notebooks and sections, including "KALLURI's Notes", "WA\_2", "End\_M...", "NPTEL...", "Binary...", and several "New S..." entries. The main content area has a title "Normal versus Malignant" with a red underline. Below the title are two numbered points: "1. The images could be grayscale with one channel or color image with three channels." and "2. create an algorithm to identify metastatic cancer in small image patches taken from larger digital pathology scans". A handwritten diagram in red ink shows a square labeled "input" with an arrow pointing to a circle containing "0" and "1", with "neg" written next to the "1". The bottom of the slide has the title "Download the Dataset from kaggle" and a single point: "1. Create a kaggle account." The NPTEL logo is visible in the bottom right corner.

### Normal versus Malignant

1. The images could be grayscale with one channel or color image with three channels.
2. create an algorithm to identify metastatic cancer in small image patches taken from larger digital pathology scans

input → 0 → neg  
          1 → neg

### Download the Dataset from kaggle

1. Create a kaggle account.

The screenshot shows a OneNote page titled "Download the Dataset from kaggle". The left sidebar is identical to the previous slide. The main content area has a title "Download the Dataset from kaggle" with a red underline. Below the title are five numbered points: "1. Create a kaggle account.", "2. Search for the dataset you want to download.", "3. Download the json file on to your local drive.", "4. Create a directory in the google drive.", and "5. Upload the .json file into the directory." A handwritten diagram in red ink shows a circle with a checkmark, with arrows pointing to the first and second points. The bottom of the slide has the NPTEL logo.

### Download the Dataset from kaggle

1. Create a kaggle account.
2. Search for the dataset you want to download.
3. Download the json file on to your local drive.
4. Create a directory in the google drive.
5. Upload the .json file into the directory.

OneNote

Home Insert Draw View

Test Lasso Select Space Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.5 mm 0.7 mm 1 mm

KALLURI's Notebooks

WA\_2 Theory  
End\_M... Train  
NPTEL Test  
Binary... Untitled Page  
New S...  
New S...  
New S...  
New S...  
CORP...  
Diary  
GATE  
New...  
GATE...  
GATE...  
ET A...  
MAJOR...  
Sreeniv...  
Dgit...  
ATAL...

## Key Topics

- Exploring the dataset
- Creating a custom dataset
- Splitting the dataset
- Transforming the data
- Creating Data Loaders
- Building the classification model
- Defining the loss function
- Defining the optimizer
- Training and evaluation of the model
- Model inference on test data

NPTEL

In step 1, RandomHorizontalFlip and RandomVerticalFlip will flip the image horizontally and vertically with a probability of 0.5, respectively.

OneNote

Home Insert Draw View

Test Lasso Select Space Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.5 mm 0.7 mm 1 mm

KALLURI's Notebooks

WA\_2 Theory  
End\_M... Train  
NPTEL Test  
Binary... Untitled Page  
New S...  
New S...  
New S...  
New S...  
CORP...  
Diary  
GATE  
New...  
GATE...  
GATE...  
ET A...  
MAJOR...  
Sreeniv...  
Dgit...  
ATAL...

## Building the classification model

In this section, we will build a model for our binary classification task. Our model is comprised of four convolutional neural networks (CNNs) and two fully connected layers, as shown in the following diagram:

### Defining the loss function

NPTEL

OneNote

Home Insert Draw View

Test Lasso Select Space Eraser Pen Marker Highlighter Ink Colour 0.25 mm 0.5 mm 0.7 mm 1 mm

KALLURI's Notebooks

WA\_2 Theory  
End\_M... Train  
NPTEL Test  
Binary... Untitled Page  
New S...  
New S...  
New S...  
New S...  
CORP...  
Diary  
GATE  
New...  
GATE...  
GATE...  
ET A...  
MAJOR...  
Sreeniv...  
Dgit...  
ATAL...

### Defining the loss function

Output Activation	Number of Outputs	Loss Function
None	1	nn.BCEWithLogitsLoss
Sigmoid	1	nn.BCELoss
None	2	nn.CrossEntropyLoss
log_softmax	2	nn.NLLLoss

We recommend using the log\_softmax function as it is easier to expand to multi-class classification. PyTorch combines the log and softmax operations into one function due to numerical stability and speed.

### Activation functions and Loss Computation

NPTEL



One more thing, one more thing is like in the entire project, what we have done I will have a summary note will give you. An idea, the basic program is our task is to suppose if you are given any 96×96 patch image to the trained model, it should give the output either as a 0 or 1. I mean, either it belongs to normal image or it belongs to malignant image. This is what does.

For that we have to develop the code using the Pytorch framework. That is downloaded using the first thing where I have using the Kaggle, competition I am using and download the data set using these following steps. Later these are the keys topics one has to go through to design the model. After that we want to design this model like convolution neural network.

Next this is very important step, like how many nodes you have to keep depends on the entire for whichever I have given now this one can be useful for multiclass also, but at the same time in the training level phase instead of 0 1 2, assume that you have 3 classes then we will have labels like 0 1 2 also. So when you have what less than 0 1 2 3 levels. That is how we should prefer. That this entire code can be used for your problems whichever you use and optimize the definition and everything all these things are done. Thank you so much.