part2

March 27, 2024

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

[2]:
```python
data = pd.read_csv("data/published_images.csv")
data.head()
```

[2]:
```
                                   uuid  \
0  00007f61-4922-417b-8f27-893ea328206c
1  0000bd8c-39de-4453-b55d-5e28a9beed38
2  0001668a-dd1c-48e8-9267-b6d1697d43c8
3  00032658-8a7a-44e3-8bb8-df8c172f521d
4  0003d4e4-d7fd-4835-8d27-1e9e20672e39

                                             iiifurl  \
0  https://api.nga.gov/iiif/00007f61-4922-417b-8f…
1  https://api.nga.gov/iiif/0000bd8c-39de-4453-b5…
2  https://api.nga.gov/iiif/0001668a-dd1c-48e8-92…
3  https://api.nga.gov/iiif/00032658-8a7a-44e3-8b…
4  https://api.nga.gov/iiif/0003d4e4-d7fd-4835-8d…

                                        iiifthumburl viewtype  sequence  \
0  https://api.nga.gov/iiif/00007f61-4922-417b-8f…  primary       0.0
1  https://api.nga.gov/iiif/0000bd8c-39de-4453-b5…  primary       0.0
2  https://api.nga.gov/iiif/0001668a-dd1c-48e8-92…  primary       0.0
3  https://api.nga.gov/iiif/00032658-8a7a-44e3-8b…  primary       0.0
4  https://api.nga.gov/iiif/0003d4e4-d7fd-4835-8d…  primary       0.0

   width  height  maxpixels                 created                modified  \
0   3365    4332        NaN  2013-07-05 15:41:08-04  2023-07-27 12:06:38-04
1   3500    4688        NaN  2013-08-05 14:31:59-04  2023-07-27 12:11:57-04
2   3446    4448        NaN  2014-01-02 14:50:50-05  2023-07-27 12:39:11-04
3   2674    3798        NaN  2010-10-13 15:37:25-04  2023-07-27 15:51:54-04
4   3000    2648      640.0  2014-11-19 14:24:42-05  2023-11-07 14:13:17-05

   depictstmsobjectid assistivetext
0               17387           NaN
1               19245           NaN
2               23830           NaN
3                 713           NaN
```

```
4              71457          NaN
```

[3]:
```python
objects = pd.read_csv("data/objects.csv")
objects.head()
```

```
/tmp/ipykernel_18622/1759643199.py:1: DtypeWarning: Columns (29) have mixed
types. Specify dtype option on import or set low_memory=False.
  objects = pd.read_csv("data/objects.csv")
```

[3]:
```
   objectid  accessioned accessionnum  locationid  \
0         0            1    1937.1.2.c         NaN
1         1            1      1937.1.3         NaN
2        17            1     1937.1.15         NaN
3        70            1     1937.1.63      8469.0
4        25            1     1937.1.23      8388.0

                               title displaydate  beginyear  endyear  \
0                  Saint James Major     c. 1310     1310.0   1310.0
1  Saint Paul and a Group of Worshippers    1333     1333.0   1333.0
2                 Matteo Olivieri (?)       1430s     1430.0   1440.0
3        An Old Woman Dozing over a Book  c. 1655     1655.0   1655.0
4            Profile Portrait of a Lady   c. 1410     1410.0   1410.0

  visualbrowsertimespan                                      medium  \
0          1300 to 1400                            tempera on panel
1          1300 to 1400                            tempera on panel
2          1401 to 1500  tempera (and oil?) on panel transferred to canvas
3          1651 to 1700                               oil on canvas
4          1401 to 1500                                oil on panel

   … parentid isvirtual departmentabbr portfolio series volume watermarks  \
0  …     34.0         0          CIS-R       NaN    NaN    NaN        NaN
1  …      NaN         0          CIS-R       NaN    NaN    NaN        NaN
2  …      NaN         0          CIS-R       NaN    NaN    NaN        NaN
3  …      NaN         0          CNE-B       NaN    NaN    NaN        NaN
4  …      NaN         0          CNE-R       NaN    NaN    NaN        NaN

    lastdetectedmodification wikidataid customprinturl
0   2023-05-09 17:01:03.48-04  Q20172973            NaN
1  2024-01-26 22:01:48.797-05  Q20173083            NaN
2  2023-09-15 22:01:37.343-04  Q20173485            NaN
3   2023-05-09 17:01:03.48-04  Q20177396            NaN
4   2023-05-09 17:01:03.48-04   Q3937690            NaN

[5 rows x 30 columns]
```

[4]:
```
data = data.merge(objects[["objectid", "visualbrowsertimespan"]],␣
↪left_on="depictstmsobjectid", right_on="objectid", how="left")
data.head()
```

[4]:
```
                                   uuid  \
0  00007f61-4922-417b-8f27-893ea328206c
1  0000bd8c-39de-4453-b55d-5e28a9beed38
2  0001668a-dd1c-48e8-9267-b6d1697d43c8
3  00032658-8a7a-44e3-8bb8-df8c172f521d
4  0003d4e4-d7fd-4835-8d27-1e9e20672e39

                                             iiifurl  \
0  https://api.nga.gov/iiif/00007f61-4922-417b-8f…
1  https://api.nga.gov/iiif/0000bd8c-39de-4453-b5…
2  https://api.nga.gov/iiif/0001668a-dd1c-48e8-92…
3  https://api.nga.gov/iiif/00032658-8a7a-44e3-8b…
4  https://api.nga.gov/iiif/0003d4e4-d7fd-4835-8d…

                                        iiifthumburl viewtype  sequence  \
0  https://api.nga.gov/iiif/00007f61-4922-417b-8f…  primary       0.0
1  https://api.nga.gov/iiif/0000bd8c-39de-4453-b5…  primary       0.0
2  https://api.nga.gov/iiif/0001668a-dd1c-48e8-92…  primary       0.0
3  https://api.nga.gov/iiif/00032658-8a7a-44e3-8b…  primary       0.0
4  https://api.nga.gov/iiif/0003d4e4-d7fd-4835-8d…  primary       0.0

   width  height  maxpixels                 created                modified  \
0   3365    4332        NaN  2013-07-05 15:41:08-04  2023-07-27 12:06:38-04
1   3500    4688        NaN  2013-08-05 14:31:59-04  2023-07-27 12:11:57-04
2   3446    4448        NaN  2014-01-02 14:50:50-05  2023-07-27 12:39:11-04
3   2674    3798        NaN  2010-10-13 15:37:25-04  2023-07-27 15:51:54-04
4   3000    2648      640.0  2014-11-19 14:24:42-05  2023-11-07 14:13:17-05

   depictstmsobjectid assistivetext  objectid visualbrowsertimespan
0               17387           NaN   17387.0          1926 to 1950
1               19245           NaN   19245.0          1926 to 1950
2               23830           NaN   23830.0          1926 to 1950
3                 713           NaN     713.0          1501 to 1550
4               71457           NaN   71457.0          1976 to 2000
```

[5]:
```
print(f"Number of rows before dropping NaN values: {data.shape[0]}")
data.dropna(subset=['visualbrowsertimespan'], inplace=True)
print(f"Number of rows after dropping NaN values: {data.shape[0]}")
```

```
Number of rows before dropping NaN values: 116251
Number of rows after dropping NaN values: 116093
```

[6]:
```python
import glob

image_files = glob.glob("data/images/*.jpg")
num_images = len(image_files)

print(f"Number of images in 'data/images/' directory: {num_images}")
```

```
Number of images in 'data/images/' directory: 116162
```

[7]:
```python
import os

# Filter out the rows where the image file doesn't exist
data = data[data['uuid'].apply(lambda x: os.path.exists(f'data/images/{x}.
  ↪jpg'))]
```

[8]:
```python
task_data = data[['uuid', 'visualbrowsertimespan']]
task_data.to_csv("data/task_data.csv", index=False)
```

[9]:
```python
time_spans = task_data['visualbrowsertimespan'].unique()
time_span_mappping = {time_span: i for i, time_span in enumerate(time_spans)}
print(time_span_mappping)
```

```
{'1926 to 1950': 0, '1501 to 1550': 1, '1976 to 2000': 2, '1601 to 1650': 3,
'1951 to 1975': 4, '1876 to 1900': 5, '1826 to 1850': 6, '1401 to 1500': 7,
'1300 to 1400': 8, '1751 to 1775': 9, '1801 to 1825': 10, '1851 to 1875': 11,
'1901 to 1925': 12, '2001 to present': 13, '1726 to 1750': 14, '1651 to 1700':
15, '1776 to 1800': 16, '1551 to 1600': 17, '1701 to 1725': 18, 'before 1300':
19}
```

[10]:
```python
task_data['visualbrowsertimespan'] = task_data['visualbrowsertimespan'].
  ↪map(time_span_mappping)
task_data.head()
```

```
/tmp/ipykernel_18622/55327113.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  task_data['visualbrowsertimespan'] =
task_data['visualbrowsertimespan'].map(time_span_mappping)
```

[10]:

	uuid	visualbrowsertimespan
0	00007f61-4922-417b-8f27-893ea328206c	0
1	0000bd8c-39de-4453-b55d-5e28a9beed38	0
2	0001668a-dd1c-48e8-9267-b6d1697d43c8	0
3	00032658-8a7a-44e3-8bb8-df8c172f521d	1
4	0003d4e4-d7fd-4835-8d27-1e9e20672e39	2

[11]:
```python
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
train_data, test_data = train_test_split(task_data, test_size=0.2,␣
↪random_state=42)

# Print the shapes of the training and test sets
print("Training data shape:", train_data.shape)
print("Test data shape:", test_data.shape)
```

```
Training data shape: (92803, 2)
Test data shape: (23201, 2)
```

[12]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.tensorboard import SummaryWriter
import numpy as np
import matplotlib.pyplot as plt
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
from torchvision import models
from tqdm import tqdm
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
from torch.nn import DataParallel

import warnings
warnings.filterwarnings("ignore")
```

[13]:
```python
class ImageDataset(Dataset):
    def __init__(self, data, image_dir, transform=None):
        self.data = data
        self.image_dir = image_dir
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        img_name = f"{self.image_dir}/{self.data.iloc[idx, 0]}.jpg"
        image = Image.open(img_name)
        label = self.data.iloc[idx, 1]

        if self.transform:
            image = self.transform(image)
```

```
        return image, label
```

[14]:
```
class ImageClassifier(nn.Module):
    def __init__(self, num_classes):
        super(ImageClassifier, self).__init__()
        self.model = models.resnet50(pretrained=True)
        in_features = self.model.fc.in_features
        self.model.fc = nn.Linear(in_features, num_classes)

    def forward(self, x):
        return self.model(x)
```

[15]:
```
train_transform = transforms.Compose([
            transforms.RandomRotation(360),
            transforms.RandomHorizontalFlip(),
            transforms.RandomAffine(degrees=(-10, 10), translate=(0.05, 0.05),␣
 ↪scale=(0.95, 1.05), shear=5),
            transforms.RandomPerspective(distortion_scale=0.1, p=0.1),
            transforms.GaussianBlur(kernel_size=(3, 3), sigma=(0.1, 0.2)),
            transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.
 ↪1, hue=0.1),
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ])

test_transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
        ])

train_dataset = ImageDataset(train_data, "data/images",␣
 ↪transform=train_transform)
test_dataset = ImageDataset(test_data, "data/images", transform=test_transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = ImageClassifier(num_classes=len(time_spans)).to(device)
model = DataParallel(model)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
summary = SummaryWriter()
iterations = 0
```

```python
def train(model, train_loader, criterion, optimizer, device, epoch, summary,␣
 ↪iterations):
    model.train()
    running_loss = 0.0
    for i, data in enumerate(tqdm(train_loader)):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        summary.add_scalar("train_loss", loss.item(), iterations)
        iterations += 1
        running_loss += loss.item()
    epoch_loss = running_loss / len(train_loader)
    print(f"Train Loss: {epoch_loss}")
    return model, iterations

def test(model, test_loader, criterion, device, epoch, summary):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for data in tqdm(test_loader):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.item()

            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(test_loader)
    accuracy = 100 * correct / total
    print(f"Test Loss: {epoch_loss}, Accuracy: {accuracy}")
    summary.add_scalar("test_loss", epoch_loss, epoch)
    summary.add_scalar("test_accuracy", accuracy, epoch)
    return accuracy
```

[16]:
```python
best_accuracy = 0
for epoch in range(10):
    print(f"Epoch {epoch + 1}")
    model, iterations = train(model, train_loader, criterion, optimizer,␣
 ↪device, epoch, summary, iterations)
    accuracy = test(model, test_loader, criterion, device, epoch, summary)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        torch.save(model.state_dict(), "model.pth")
```

```
Epoch 1

100%|      | 2901/2901 [15:52<00:00,  3.05it/s]

Train Loss: 1.690137814834421

100%|      | 726/726 [01:31<00:00,  7.93it/s]

Test Loss: 1.6270704830973601, Accuracy: 49.32977026852291
Epoch 2

100%|      | 2901/2901 [15:13<00:00,  3.17it/s]

Train Loss: 1.4655975448760605

100%|      | 726/726 [01:29<00:00,  8.12it/s]

Test Loss: 1.6822010135026675, Accuracy: 50.592646868669455
Epoch 3

100%|      | 2901/2901 [15:13<00:00,  3.17it/s]

Train Loss: 1.3722520290017743

100%|      | 726/726 [01:35<00:00,  7.59it/s]

Test Loss: 1.4545212095351916, Accuracy: 54.30369380630145
Epoch 4

100%|      | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 1.308250688450454

100%|      | 726/726 [01:33<00:00,  7.79it/s]

Test Loss: 1.5210858398725178, Accuracy: 54.217490625404075
Epoch 5

100%|      | 2901/2901 [15:09<00:00,  3.19it/s]

Train Loss: 1.2609052630960016

100%|      | 726/726 [01:32<00:00,  7.83it/s]

Test Loss: 1.3968077205100993, Accuracy: 55.92000344812723
Epoch 6

100%|      | 2901/2901 [15:20<00:00,  3.15it/s]
```

```
Train Loss: 1.2195870250062835

100%|      | 726/726 [01:32<00:00,  7.81it/s]

Test Loss: 1.3554734793740528, Accuracy: 56.99754320934443
Epoch 7

100%|      | 2901/2901 [15:15<00:00,  3.17it/s]

Train Loss: 1.184016942443703

100%|      | 726/726 [01:31<00:00,  7.93it/s]

Test Loss: 1.3625999040511685, Accuracy: 57.954398517305286
Epoch 8

100%|      | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 1.153340318035726

100%|      | 726/726 [01:30<00:00,  8.06it/s]

Test Loss: 1.343617481141051, Accuracy: 57.967328994439896
Epoch 9

100%|      | 2901/2901 [15:20<00:00,  3.15it/s]

Train Loss: 1.1244382328388816

100%|      | 726/726 [01:31<00:00,  7.90it/s]

Test Loss: 1.279041744132344, Accuracy: 59.18710400413775
Epoch 10

100%|      | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 1.0983684018475153

100%|      | 726/726 [01:31<00:00,  7.89it/s]

Test Loss: 1.3050892327606842, Accuracy: 59.29485798025947
```

[18]:

```python
for epoch in range(30):
    print(f"Epoch {epoch + 1}")
    model, iterations = train(model, train_loader, criterion, optimizer,␣
 ↪device, epoch, summary, iterations)
    accuracy = test(model, test_loader, criterion, device, epoch, summary)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        torch.save(model.state_dict(), "model.pth")
```

```
Epoch 1

100%|      | 2901/2901 [15:17<00:00,  3.16it/s]

Train Loss: 1.0731199352402638

100%|      | 726/726 [01:33<00:00,  7.79it/s]
```

```
Test Loss: 1.3640705223911065, Accuracy: 58.84660143959312
Epoch 2

100%|      | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 1.0487641955564369

100%|      | 726/726 [01:31<00:00,  7.97it/s]

Test Loss: 1.360556701983302, Accuracy: 58.600922374035605
Epoch 3

100%|      | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 1.026329971066593

100%|      | 726/726 [01:30<00:00,  8.04it/s]

Test Loss: 1.3139927555132176, Accuracy: 59.45864402396448
Epoch 4

100%|      | 2901/2901 [15:14<00:00,  3.17it/s]

Train Loss: 1.0068358256668108

100%|      | 726/726 [01:31<00:00,  7.95it/s]

Test Loss: 1.2765662414804306, Accuracy: 60.82927460023275
Epoch 5

100%|      | 2901/2901 [15:17<00:00,  3.16it/s]

Train Loss: 0.9853862111464076

100%|      | 726/726 [01:31<00:00,  7.93it/s]

Test Loss: 1.3261786331486767, Accuracy: 60.4629110814189
Epoch 6

100%|      | 2901/2901 [15:14<00:00,  3.17it/s]

Train Loss: 0.9690948955480989

100%|      | 726/726 [01:31<00:00,  7.98it/s]

Test Loss: 1.311826701458164, Accuracy: 60.863755872591696
Epoch 7

100%|      | 2901/2901 [15:13<00:00,  3.17it/s]

Train Loss: 0.9511676182709082

100%|      | 726/726 [01:31<00:00,  7.92it/s]

Test Loss: 1.3074272146484411, Accuracy: 61.4154562303349
Epoch 8

100%|      | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 0.9290971909256402
```

```
100%|     | 726/726 [01:30<00:00,  8.01it/s]

Test Loss: 1.2541883173231907, Accuracy: 61.56200163786044
Epoch 9

100%|     | 2901/2901 [15:17<00:00,  3.16it/s]

Train Loss: 0.9111072075588873

100%|     | 726/726 [01:31<00:00,  7.92it/s]

Test Loss: 1.3668394812867661, Accuracy: 60.949959053489074
Epoch 10

100%|     | 2901/2901 [15:09<00:00,  3.19it/s]

Train Loss: 0.8960632865735475

100%|     | 726/726 [01:32<00:00,  7.85it/s]

Test Loss: 1.3644437264066098, Accuracy: 60.678419033662344
Epoch 11

100%|     | 2901/2901 [15:15<00:00,  3.17it/s]

Train Loss: 0.8810253038832913

100%|     | 726/726 [01:33<00:00,  7.76it/s]

Test Loss: 1.3553913240590372, Accuracy: 60.71721046506616
Epoch 12

100%|     | 2901/2901 [15:20<00:00,  3.15it/s]

Train Loss: 0.8620052225155815

100%|     | 726/726 [01:30<00:00,  8.01it/s]

Test Loss: 1.287482999244669, Accuracy: 61.958536269988365
Epoch 13

100%|     | 2901/2901 [15:09<00:00,  3.19it/s]

Train Loss: 0.845609387511921

100%|     | 726/726 [01:32<00:00,  7.83it/s]

Test Loss: 1.4490600350228222, Accuracy: 60.35946726434205
Epoch 14

100%|     | 2901/2901 [15:15<00:00,  3.17it/s]

Train Loss: 0.8290279850348322

100%|     | 726/726 [01:36<00:00,  7.53it/s]

Test Loss: 1.3484457275591606, Accuracy: 62.26886772121891
Epoch 15

100%|     | 2901/2901 [15:12<00:00,  3.18it/s]
```

```
Train Loss: 0.8145340183548827

100%|     | 726/726 [01:34<00:00,  7.67it/s]

Test Loss: 1.294868934909831, Accuracy: 63.130899530192664
Epoch 16

100%|     | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.7965203226639952

100%|     | 726/726 [01:33<00:00,  7.78it/s]

Test Loss: 1.3278530860324864, Accuracy: 62.29472867548812
Epoch 17

100%|     | 2901/2901 [15:15<00:00,  3.17it/s]

Train Loss: 0.7809749720056317

100%|     | 726/726 [01:31<00:00,  7.94it/s]

Test Loss: 1.3749777219019645, Accuracy: 62.16542390414206
Epoch 18

100%|     | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 0.7681955712411125

100%|     | 726/726 [01:31<00:00,  7.91it/s]

Test Loss: 1.375300587044603, Accuracy: 61.81630102150769
Epoch 19

100%|     | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.7506349904501861

100%|     | 726/726 [01:31<00:00,  7.90it/s]

Test Loss: 1.4458212725428181, Accuracy: 63.393819231929655
Epoch 20

100%|     | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.7362199015388896

100%|     | 726/726 [01:32<00:00,  7.86it/s]

Test Loss: 1.4028292325820477, Accuracy: 62.863669669410804
Epoch 21

100%|     | 2901/2901 [15:10<00:00,  3.18it/s]

Train Loss: 0.7200810335825657

100%|     | 726/726 [01:33<00:00,  7.77it/s]

Test Loss: 1.426847750490362, Accuracy: 62.69557346666092
Epoch 22
```

```
100%|      | 2901/2901 [15:16<00:00,  3.17it/s]

Train Loss: 0.7063657639458935

100%|      | 726/726 [01:35<00:00,  7.63it/s]

Test Loss: 1.4502108799031943, Accuracy: 62.28610835739839
Epoch 23

100%|      | 2901/2901 [15:13<00:00,  3.17it/s]

Train Loss: 0.6888736269517999

100%|      | 726/726 [01:32<00:00,  7.85it/s]

Test Loss: 1.4406594608194572, Accuracy: 62.79901728373777
Epoch 24

100%|      | 2901/2901 [15:14<00:00,  3.17it/s]

Train Loss: 0.6791201755430155

100%|      | 726/726 [01:31<00:00,  7.95it/s]

Test Loss: 1.5414045637591482, Accuracy: 62.53178742295591
Epoch 25

100%|      | 2901/2901 [15:17<00:00,  3.16it/s]

Train Loss: 0.66138602537416

100%|      | 726/726 [01:30<00:00,  7.98it/s]

Test Loss: 1.4996483213891667, Accuracy: 63.0015947588466
Epoch 26

100%|      | 2901/2901 [15:16<00:00,  3.17it/s]

Train Loss: 0.6498614293014292

100%|      | 726/726 [01:31<00:00,  7.98it/s]

Test Loss: 1.5028258647882577, Accuracy: 62.863669669410804
Epoch 27

100%|      | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.6370408145344204

100%|      | 726/726 [01:31<00:00,  7.97it/s]

Test Loss: 1.5096620853897953, Accuracy: 62.72143442093013
Epoch 28

100%|      | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.623176160859511

100%|      | 726/726 [01:30<00:00,  8.05it/s]
```

```
Test Loss: 1.5724494867745182, Accuracy: 62.54471790009051
Epoch 29

100%|      | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.6076582055160071

100%|      | 726/726 [01:30<00:00,  8.02it/s]

Test Loss: 1.641739511210728, Accuracy: 61.9671565880781
Epoch 30

100%|      | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 0.5941098445563512

100%|      | 726/726 [01:31<00:00,  7.92it/s]

Test Loss: 1.5748681583680397, Accuracy: 62.574889013404594
```

[19]:

```python
for epoch in range(30):
    print(f"Epoch {epoch + 1}")
    model, iterations = train(model, train_loader, criterion, optimizer,␣
 ↪device, epoch, summary, iterations)
    accuracy = test(model, test_loader, criterion, device, epoch, summary)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        torch.save(model.state_dict(), "model.pth")
```

```
Epoch 1

100%|      | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.5841842094029858

100%|      | 726/726 [01:31<00:00,  7.94it/s]

Test Loss: 1.574741728583315, Accuracy: 63.10934873496832
Epoch 2

100%|      | 2901/2901 [15:17<00:00,  3.16it/s]

Train Loss: 0.5704419449952831

100%|      | 726/726 [01:31<00:00,  7.93it/s]

Test Loss: 1.6124020591172963, Accuracy: 63.21710271109004
Epoch 3

100%|      | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.5575335187897194

100%|      | 726/726 [01:30<00:00,  8.03it/s]
```

```
Test Loss: 1.6218958239253232, Accuracy: 63.46709193569243
Epoch 4

100%|      | 2901/2901 [15:07<00:00,  3.20it/s]

Train Loss: 0.5435561933925505

100%|      | 726/726 [01:30<00:00,  8.02it/s]

Test Loss: 1.6054723077240727, Accuracy: 63.2774449377182
Epoch 5

100%|      | 2901/2901 [15:30<00:00,  3.12it/s]

Train Loss: 0.5321687847550512

100%|      | 726/726 [01:36<00:00,  7.55it/s]

Test Loss: 1.654402204288924, Accuracy: 62.587819490539204
Epoch 6

100%|      | 2901/2901 [15:15<00:00,  3.17it/s]

Train Loss: 0.5229305098207603

100%|      | 726/726 [01:31<00:00,  7.93it/s]

Test Loss: 1.7675459911487483, Accuracy: 62.574889013404594
Epoch 7

100%|      | 2901/2901 [15:27<00:00,  3.13it/s]

Train Loss: 0.5102772527971131

100%|      | 726/726 [01:33<00:00,  7.79it/s]

Test Loss: 1.75020331283903, Accuracy: 62.80332744278264
Epoch 8

100%|      | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.5003935639373766

100%|      | 726/726 [01:31<00:00,  7.92it/s]

Test Loss: 1.7361015405290383, Accuracy: 62.850739192276194
Epoch 9

100%|      | 2901/2901 [15:17<00:00,  3.16it/s]

Train Loss: 0.48540374629972555

100%|      | 726/726 [01:30<00:00,  7.99it/s]

Test Loss: 1.6879902401164215, Accuracy: 63.316236369122024
Epoch 10

100%|      | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 0.4764779852186347
```

```
100%|     | 726/726 [01:31<00:00,  7.90it/s]

Test Loss: 1.764539520230565, Accuracy: 62.2430067669497
Epoch 11

100%|     | 2901/2901 [15:11<00:00,  3.18it/s]

Train Loss: 0.46535194102226557

100%|     | 726/726 [01:33<00:00,  7.73it/s]

Test Loss: 1.729557144006627, Accuracy: 62.967113486487655
Epoch 12

100%|     | 2901/2901 [15:14<00:00,  3.17it/s]

Train Loss: 0.45684047246217646

100%|     | 726/726 [01:29<00:00,  8.07it/s]

Test Loss: 1.8249542707007778, Accuracy: 62.36369122020603
Epoch 13

100%|     | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.44226976285275404

100%|     | 726/726 [01:31<00:00,  7.95it/s]

Test Loss: 1.6902453749550932, Accuracy: 63.65242877462178
Epoch 14

100%|     | 2901/2901 [15:10<00:00,  3.19it/s]

Train Loss: 0.4369557541311917

100%|     | 726/726 [01:30<00:00,  8.01it/s]

Test Loss: 1.838951381441647, Accuracy: 61.83354165768717
Epoch 15

100%|     | 2901/2901 [15:18<00:00,  3.16it/s]

Train Loss: 0.43023958494543246

100%|     | 726/726 [01:32<00:00,  7.89it/s]

Test Loss: 1.8583025800130912, Accuracy: 62.52316710486617
Epoch 16

100%|     | 2901/2901 [15:13<00:00,  3.17it/s]

Train Loss: 0.4164440942474251

100%|     | 726/726 [01:31<00:00,  7.97it/s]

Test Loss: 1.8575779050999108, Accuracy: 63.26020430153873
Epoch 17

100%|     | 2901/2901 [15:21<00:00,  3.15it/s]
```

```
Train Loss: 0.4069580955675494

100%|     | 726/726 [01:29<00:00,  8.07it/s]

Test Loss: 1.9286355537570212, Accuracy: 62.337830265936816
Epoch 18

100%|     | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.3958651038237951

100%|     | 726/726 [01:32<00:00,  7.81it/s]

Test Loss: 1.9415915090354678, Accuracy: 62.842118874186454
Epoch 19

100%|     | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.38871323414498754

100%|     | 726/726 [01:36<00:00,  7.56it/s]

Test Loss: 1.9688646134474, Accuracy: 63.00590491789147
Epoch 20

100%|     | 2901/2901 [15:16<00:00,  3.16it/s]

Train Loss: 0.37920100117209493

100%|     | 726/726 [01:30<00:00,  8.04it/s]

Test Loss: 2.0810656687116493, Accuracy: 61.65251497780268
Epoch 21

100%|     | 2901/2901 [15:14<00:00,  3.17it/s]

Train Loss: 0.37289904823583886

100%|     | 726/726 [01:31<00:00,  7.91it/s]

Test Loss: 1.855735930775808, Accuracy: 62.5188569458213
Epoch 22

100%|     | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.3641731788329485

100%|     | 726/726 [01:32<00:00,  7.88it/s]

Test Loss: 2.043836662920382, Accuracy: 62.37231153829576
Epoch 23

100%|     | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.35301943851319606

100%|     | 726/726 [01:30<00:00,  8.00it/s]

Test Loss: 2.0979382489133145, Accuracy: 63.75587259169863
Epoch 24
```

```
100%|     | 2901/2901 [15:12<00:00,  3.18it/s]

Train Loss: 0.348063370968629

100%|     | 726/726 [01:30<00:00,  8.02it/s]

Test Loss: 1.9430203689852366, Accuracy: 64.07482436101893
Epoch 25

100%|     | 2901/2901 [15:15<00:00,  3.17it/s]

Train Loss: 0.3407567353576925

100%|     | 726/726 [01:30<00:00,  8.06it/s]

Test Loss: 2.115215538726406, Accuracy: 62.35507090211629
Epoch 26

100%|     | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.33419982180184266

100%|     | 726/726 [01:30<00:00,  8.04it/s]

Test Loss: 2.039065026547298, Accuracy: 63.333477005301496
Epoch 27

100%|     | 2901/2901 [15:10<00:00,  3.19it/s]

Train Loss: 0.32518909942976404

100%|     | 726/726 [01:31<00:00,  7.98it/s]

Test Loss: 2.0296253873132803, Accuracy: 63.24727382440412
Epoch 28

100%|     | 2901/2901 [15:13<00:00,  3.18it/s]

Train Loss: 0.3185634356260464

100%|     | 726/726 [01:30<00:00,  8.04it/s]

Test Loss: 2.1329430344183584, Accuracy: 62.09215120037929
Epoch 29

100%|     | 2901/2901 [15:09<00:00,  3.19it/s]

Train Loss: 0.3135327958443768

100%|     | 726/726 [01:33<00:00,  7.72it/s]

Test Loss: 2.167870124591612, Accuracy: 61.93698547476402
Epoch 30

100%|     | 2901/2901 [15:16<00:00,  3.16it/s]

Train Loss: 0.30488706763316253

100%|     | 726/726 [01:33<00:00,  7.74it/s]

Test Loss: 2.082339142974833, Accuracy: 63.59639670703849
```

[20]:
```python
# load best model and check accuracy
model.load_state_dict(torch.load("model.pth"))
test(model, test_loader, criterion, device, 0, summary)
```

```
100%|          | 726/726 [01:34<00:00,  7.70it/s]
```

```
Test Loss: 1.9430203689852366, Accuracy: 64.07482436101893
```

[20]:
```
64.07482436101893
```

[ ]:
```python

```