2472525m

Discussion cell:

Model architecture:

I initially created a simple convolutional neural network using a couple convolutional layers and a set of fully connected layers. The goal was to get this model to work then expand on it. Once it was working, it got better than random chance accuracy on the test set (55%), which is an excellent sign that completing the image classification task successfully (at least somewhat anyway). After this, I took inspiration from the layout of the layers of the VGG-16 model: blocks of convolutional layers where each conv layer is separated by a ReLu layer and each block terminated with a MaxPool layer, the whole model then ending with a fully connected layer. After setting up the new architecture and calculating the size of the tensor at each layer, the new model achieved 77.5% accuracy on the test data without hyperparameter tuning.

Data augmentation and preprocessing:

I decided that I could augment the data by applying a set of random transformations to each image and then appending these transformed images to the dataset. This is to increase the size of the training dataset, as having a larger dataset will help smaller, simpler models train better. Additionally, seeing transformed data should help avoid overfitting on the training data. I apply these random transformations to the dataset twice and then concatenate the 3 datasets together, so that the original dataset is still being trained on too. Some of the data transforms used are random flips and rotations and random crops to 50 x 50 pixels. These should help add some basic augmentations to the dataset.

Hyperparameter tuning:

For the parameters to be tuned, I chose the learning rate and momentum for the optimiser and batch size for the data loader. The learning rate is being tuned on a logarithmic scale between 1e-6 and 0.4, the momentum on a linear scale between 0 and 1 and the batch size chosen from one of 1, 2, 4 or 8. After twenty trials with the Ax library, the best parameters apparently performed worse than the original parameters I had chosen at the start. After a quick train by changing the batch_size by hand, I found that the best parameters are {'lr': 0.001, 'momentum': 0.9, 'batch_size': 2}. After training and evaluating a model with these parameters, we get an accuracy on the test set of 79.8%.

Future work:

In the future, I would like to experiment with other optimisers, namely Adam, and improve dataset augmentation by balancing the dataset across its labels. Additionally, I'd like to experiment with training with a learning rate schedule to alter the learning rate during training.

Deep Learning 2023 - Coursework

Classifying Plankton!

The aim of this coursework will be for you to design a deep learning architecture to predict identify plankton species from images.

Your aim is to design a model that, when given a new image of a plankton speciment would return to which species it belongs to.

You are free to use any architecture you prefer, from what we have seen in class. You can decide to use unsupervised pre-training of only supervised end-to-end training - the approach you choose is your choice.

Hand-in date: Thursday 16th of March before 4:30pm (on Moodle)

Steps & Hints

- First, look at the data. What are the different classes? How different are they? What type of transformations for your data augmentation do you think would be acceptable here?.
- You will note that it is *very* imbalanced (large differences in number of samples between classes) --- this will be one challenge to look for.
- Also, note that the dataset is rather small (hint: you will need to think about data augmentation!).
- Second, try and load the data and separate into training, validation and test set (or better, use cross-validation)
- Write a DataLoader class for the data (Hint: you will want to put the data augmentation in the data loader).
- Think about pre-processing of the input? The output? Normalisation or not? Data augmentation? Which one?
- Design a network for the task. What layers? How many? Do you want to use an Autoencoder for unsupervised pre-training?
- Choose a loss function for your network
- Select optimiser and training parameters (batch size, learning rate)

- Optimise your model, and tune hyperparameters (especially learning rate, momentum etc)
- Analyse the results on the test data. How to measure success? Which classes are recognised well, which are not? Is there confusion between some classes? Look at failure cases.
- If time allows, go back to drawing board and try a more complex, or better, model.
- Explain your thought process, justify your choices and discuss the results!

Submission

- submit TWO files on Moodle:
  - **your notebook**: use `File -> download .ipynb` to download the notebook file locally from colab.
  - **a PDF file** of your notebook's output as you see it: use `File -> print` to generate a PDF.
- your notebook must clearly contains separate cells for:
  - setting up your model and data loader
  - training your model from data
  - loading your pretrained model (eg, from github/gitlab)
  - testing your model on test data.
- The training cells must be disabled by a flag, such that when running *run all* on your notebook it does
  - load the data
  - load your model
  - apply the model to the test data
  - analyse and display the results and accuracy
- In addition provide markup cell:
  - containing your student number at the top
  - to describe and motivate your design choices: architecture, pre-processing, training regime
  - to analyse, describe and comment on your results
  - to provide some discussion on what you think are the limitations of your solution and what could be future work
- **Note that you must put your trained model on a github so that your code can download it.**

Assessment criteria

- In order to get a pass mark, you will need to demonstrate that you have designed and trained a deep NN to solve the problem, using sensible approach and reasonable efforts to tune hyper-parameters. You have analysed the results. It is NOT necessary to have any level of accuracy (a network that predicts poorly will always yield a pass mark if it is designed, tuned and analysed sensibly).
- In order to get a good mark, you will show good understanding of the approach and provide a working solution.
- in order to get a high mark, you will demonstrate a working approach of gradual improvement between different versions of your solution.
- bonus marks for attempting something original if well motivated - even if it does not yield increased performance.
- bonus marks for getting high performance, and some more points are to grab for getting the best performance in the class.

Notes

- make sure to clearly set aside training, validation and test sets to ensure proper setting of all hyperparameters.
- I recommend to start with small models that can be easier to train to set a baseline performance before attempting more complex one.
- Be mindful of the time!

Data

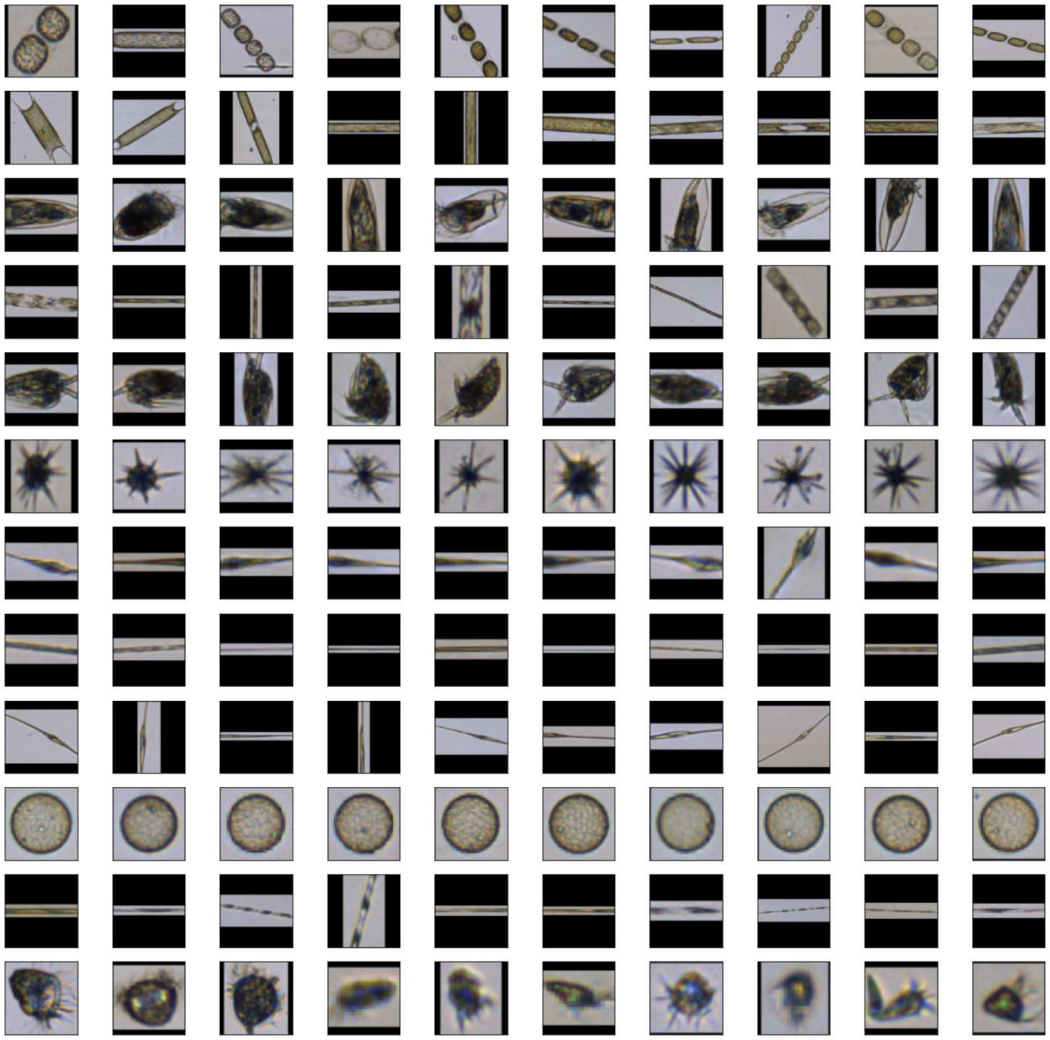
The following cells will show you how to download the data and view it.

```
import os
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
import collections

# Loading the data
# we will use wget to get the archive
!wget --no-check-certificate "https://www.dropbox.com/s/v2udcnt98miwwrq/plankton.pt?dl=1" -O plankton.pt
```

```
--2023-03-21 16:18:30--  https://www.dropbox.com/s/v2udcnt98miwwrq/plankton.pt?dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.81.18, 2620:100:6031:18::a27d:5112
Connecting to www.dropbox.com (www.dropbox.com)|162.125.81.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /s/dl/v2udcnt98miwwrq/plankton.pt [following]
--2023-03-21 16:18:31--  https://www.dropbox.com/s/dl/v2udcnt98miwwrq/plankton.pt
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc11e6ceb9aeb1bb39657a89f85a.dl.dropboxusercontent.com/cd/0/get/B4rBLXAqdBmqJmftoDh0hXv9gXYF8QO8f40wIxa1gDtasWu6inFPO_
--2023-03-21 16:18:31--  https://uc11e6ceb9aeb1bb39657a89f85a.dl.dropboxusercontent.com/cd/0/get/B4rBLXAqdBmqJmftoDh0hXv9gXYF8QO8f40wIxa
Resolving uc11e6ceb9aeb1bb39657a89f85a.dl.dropboxusercontent.com (uc11e6ceb9aeb1bb39657a89f85a.dl.dropboxusercontent.com)... 162.125.81.
Connecting to uc11e6ceb9aeb1bb39657a89f85a.dl.dropboxusercontent.com (uc11e6ceb9aeb1bb39657a89f85a.dl.dropboxusercontent.com)|162.125.81
HTTP request sent, awaiting response... 200 OK
Length: 194047719 (185M) [application/binary]
Saving to: ‘plankton.pt’

plankton.pt         100%[===================>] 185.06M  12.2MB/s    in 18s

2023-03-21 16:18:50 (10.2 MB/s) - ‘plankton.pt’ saved [194047719/194047719]
```

```
data = torch.load('plankton.pt')

# get the number of different classes
classes = data['labels'].unique()
nclasses = len(classes)
print('The classes in this dataset are: ')
print(classes)

# display the number of instances per class:
print('\nAnd the numbers of examples per class are: ')
print( pd.Series(data['labels']).value_counts() )

# we now print some examples from each class for visualisation
fig = plt.figure(figsize=(20,20))

n = 10 # number of examples to show per class

for i in range(nclasses):
  idx = data['labels'] == classes[i]
  imgs = data['images'][idx,...]
  for j in range(n):
    ax = plt.subplot(nclasses,n,i*n+j+1)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax.imshow( imgs[j,...].permute(1, 2, 0) ) # note the permute because tensorflow puts the channel as the first dimension whereas matplotli
plt.show()
```

```
The classes in this dataset are:
tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.])

And the numbers of examples per class are:
2.0     257
8.0     235
7.0     219
10.0    157
11.0    135
0.0     134
3.0     110
6.0      92
9.0      76
4.0      70
5.0      67
1.0      65
dtype: int64
```



- First, look at the data. What are the different classes? How different are they? What type of transformations for your data augmentation do you think would be acceptable here?.
- You will note that it is very imbalanced (large differences in number of samples between classes) --- this will be one challenge to look for.
- Also, note that the dataset is rather small (hint: you will need to think about data augmentation!).
- Second, try and load the data and separate into training, validation and test set (or better, use cross-validation)
- Write a DataLoader class for the data (Hint: you will want to put the data augmentation in the data loader).
- Think about pre-processing of the input? The output? Normalisation or not? Data augmentation? Which one?
- Design a network for the task. What layers? How many? Do you want to use an Autoencoder for unsupervised pre-training?
- Choose a loss function for your network
- Select optimiser and training parameters (batch size, learning rate)
- Optimise your model, and tune hyperparameters (especially learning rate, momentum etc)

- Analyse the results on the test data. How to measure success? Which classes are recognised well, which are not? Is there confusion between some classes? Look at failure cases.
- If time allows, go back to drawing board and try a more complex, or better, model.
- Explain your thought process, justify your choices and discuss the results!

CW section

```
# Imports
import collections
import os
from collections import Counter

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from torch.utils.data import ConcatDataset, DataLoader, Dataset, Subset
from torchvision import datasets
from torchvision.transforms import ToTensor
```

Setting up model and data loader

```
class Net(nn.Module):  # Results: accuracy of 55% after 3 epochs but would overfit heavily after 4 epochs
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            # input: 3 x 100 x 100
            nn.Conv2d(3, 32, kernel_size=3, padding=1),  # 100 x 100 x 32
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),  # 100 x 100 x 64
            nn.MaxPool2d(2, 2),  # 50 x 50 x 64
            nn.Flatten(),

            nn.Linear(50 * 50 * 64, 128),
            nn.Linear(128, 32),
            nn.Linear(32, 12)
        )

    def forward(self, x):
        return self.network(x)


# A VGG-16 inspired model but much smaller
class Net(nn.Module):  # Results: accuracy of 77.5% after 10 epochs, no major signs of overfitting
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            # input: 3 x 100 x 100
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),  # 100 x 100 x 64
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            #  50 x 50 x 64
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            #  25 x 25 x 128

            nn.Flatten(),
```

```
            nn.Linear(25 * 25 * 128, 128),
            nn.ReLU(),
            nn.Linear(128, 32),
            nn.ReLU(),
            nn.Linear(32, 12)
        )

    def forward(self, x):
        return self.network(x)


class CustomImageDataset(Dataset):
    def __init__(self, labels, images, transform=None, target_transform=None):
        self.img_labels = labels
        self.img_dir = images
        self.transform = transform
        self.target_transform = target_transform

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

    def __getitem__(self, idx):
        return self.img_dir[idx], self.img_labels[idx]


def load_data(data):

    original_dataset = CustomImageDataset(data['labels'].long(), data['images'], transform=transforms.ToTensor())

    transform = transforms.Compose(
        [
            transforms.RandomCrop(size=(50, 50)),
            transforms.RandomHorizontalFlip(p=0.5),
            transforms.RandomVerticalFlip(p=0.5),
            transforms.RandomRotation((0, 180)),
            transforms.ToTensor(),
        ]
    )

    transformed_dataset1 = CustomImageDataset(data['labels'].long(), data['images'], transform=transform)
    transformed_dataset2 = CustomImageDataset(data['labels'].long(), data['images'], transform=transform)

    concat_dataset = ConcatDataset([original_dataset, transformed_dataset1, transformed_dataset2])
    dataset_labels = torch.cat([original_dataset.img_labels, transformed_dataset1.img_labels, transformed_dataset2.img_labels], axis=0)

    # Split twice to get train, val and test splits
    train_indices, val_test_indices, _, val_test_labels = train_test_split(
        range(len(dataset_labels)),
        dataset_labels,
        stratify=dataset_labels,
        test_size=0.4,
        random_state=42
    )

    val_indices, test_indices, _, _ = train_test_split(
        val_test_indices,
        val_test_labels,
        stratify=val_test_labels,
        test_size=0.5,
        random_state=42
    )

    train_split = Subset(concat_dataset, train_indices)

    val_split = Subset(concat_dataset, val_indices)

    test_split = Subset(concat_dataset, test_indices)

    return train_split, val_split, test_split


#split into balanced training, validation, testing set
train_set, val_set, test_set = load_data(data)

train_loader = DataLoader(train_set, batch_size=1, shuffle=False)

val_loader = DataLoader(val_set, batch_size=1, shuffle=False)
```

```
test_loader = DataLoader(test_set, batch_size=1, shuffle=False)

# print(data['images'][0].shape)

# print(len(train_loader), len(data['labels'])*0.6*3)
# print(len(val_loader), len(data['labels'])*0.2*3)
# print(len(test_loader), len(data['labels'])*0.2*3)

# print(Counter([int(x) for x in data['labels']]))

# train_counter = []
# for image, label in train_loader:
#     train_counter.append((int(label[0])))
# print(Counter(train_counter))

# val_counter = []
# for image, label in val_loader:
#     val_counter.append((int(label[0])))
# print(Counter(val_counter))

# test_counter = []
# for image, label in test_loader:
#     test_counter.append((int(label[0])))
# print(Counter(test_counter))
```

Training model from data

```
# Training

TRAIN = False

def train_loop(net, data, config, device='cpu'):

    train_set, _, _ = load_data(data)
    train_loader = DataLoader(train_set, batch_size=config.get("batch_size", 2), shuffle=False)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=config.get("lr", 0.001), momentum=config.get("momentum", 0.9))

    #train the network
    for epoch in range(10):  # loop over the dataset multiple times

        running_loss = 0.0
        for i, data2 in enumerate(train_loader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data2
            inputs, labels = inputs.to(device), labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999:    # print every 2000 mini-batches
                print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
                running_loss = 0.0

        # Validation loss
        val_loss = 0.0
        val_steps = 0
        total = 0
        correct = 0
        for i, data3 in enumerate(val_loader, 0):
            with torch.no_grad():
                inputs, labels = data3
                inputs, labels = inputs.to(device), labels.to(device)
```

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

                loss = criterion(outputs, labels)
                val_loss += loss.cpu().numpy()
                val_steps += 1
        print(f'Epoch {epoch} validation loss={val_loss}, validation accuracy={correct/total}')
    print('Finished Training')

    # Ran on my local machine due to collab restricting GPU usage then uplaoaded to github
    torch.save((net.state_dict(), optimizer.state_dict()), "checkpoint")

if TRAIN:
    net = Net()
    device = "cpu"
    if torch.cuda.is_available():
        device = "cuda:0"
        if torch.cuda.device_count() > 1:
            net = nn.parallel.DataParallel(net)
    net.to(device)
    train_loop(net, train_loader, None, device)


def val_accuracy(net, data, device: str = "cpu"):
    _, valset, _ = load_data(data)

    val_loader = DataLoader(valset, batch_size=1, shuffle=False)

    correct = 0
    total = 0
    with torch.no_grad():
        for data in val_loader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return correct / total

def test_accuracy(net, data, device: str = "cpu"):
    _, _, testset = load_data(data)

    testloader = DataLoader(testset, batch_size=1, shuffle=False)

    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return correct / total
```

Load model from dropbox (github has a file limit size)

```
!wget --no-check-certificate "https://www.dropbox.com/s/832pwsxy5nw69yq/checkpoint?dl=0" -O checkpoint
```

```
--2023-03-21 16:18:56--  https://www.dropbox.com/s/832pwsxy5nw69yq/checkpoint?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.81.18, 2620:100:6031:18::a27d:5112
Connecting to www.dropbox.com (www.dropbox.com)|162.125.81.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /s/raw/832pwsxy5nw69yq/checkpoint [following]
--2023-03-21 16:18:57--  https://www.dropbox.com/s/raw/832pwsxy5nw69yq/checkpoint
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
```

```
Location: https://uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com/cd/0/inline/B4oXCKaOSrVsa7PCAacu16kfGPFiTeLEWUV3Tg7TNobmt19vUEg
--2023-03-21 16:18:57--  https://uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com/cd/0/inline/B4oXCKaOSrVsa7PCAacu16kfGPFiTeLEWUV3
Resolving uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com (uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com)... 162.125.81.
Connecting to uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com (uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com)|162.125.81
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/B4qudoI0exHLaWOllj6_kmQSyU8rYxDZqtfaGFuwZCqADmg3LKW93iPmrsPSyNgDg-6ZDHsXCc_9dtb7ARn4nSdpEePmPkhyR-oUzWTZlPFC4UI_
--2023-03-21 16:18:58--  https://uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com/cd/0/inline2/B4qudoI0exHLaWOllj6_kmQSyU8rYxDZqtf
Reusing existing connection to uc66000cab9d28c1ad0ed19d9914.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 84047467 (80M) [application/octet-stream]
Saving to: 'checkpoint'

checkpoint          100%[===================>]  80.15M  11.9MB/s    in 7.4s

2023-03-21 16:19:06 (10.9 MB/s) - 'checkpoint' saved [84047467/84047467]
```

```
LR = 0.001
MOMENTUM = 0.9
BATCH_SIZE = 8

# Evaluate
model = Net()

device = "cpu"
if torch.cuda.is_available():
    device = "cuda:0"
    if torch.cuda.device_count() > 1:
        net = nn.parallel.DataParallel(model)
model.to(device)

optimiser = optim.SGD(model.parameters(), lr=LR, momentum=MOMENTUM)

# Load state dicts
model_state, optimizer_state = torch.load("checkpoint", map_location=torch.device(device))
model.load_state_dict(model_state)
optimiser.load_state_dict(optimizer_state)

model.eval()

device = "cpu"
if torch.cuda.is_available():
    device = "cuda:0"
model.to(device)

print(test_accuracy(model, data, device))
```

```
0.7981462409886715
```

Hyperparameter tuning

```
!pip3 install ax-platform
```

```
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.9/dist-packages (from notebook>=4.4.1->widgetsnbextension~
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.9/dist-packages (from notebook>=4.4.1->widgetsnbextension~
Requirement already satisfied: wcwidth in /usr/local/lib/python3.9/dist-packages (from prompt-toolkit<2.1.0,>=2.0.0->ipython>=4.0.0->
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.9/dist-packages (from pexpect->ipython>=4.0.0->ipywidgets->a
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.9/dist-packages (from jupyter-core>=4.6.1->notebook>=4.4.1
Requirement already satisfied: notebook-shim>=0.1.0 in /usr/local/lib/python3.9/dist-packages (from nbclassic>=0.4.7->notebook>=4.4.1
Requirement already satisfied: jupyter-server>=1.8 in /usr/local/lib/python3.9/dist-packages (from nbclassic>=0.4.7->notebook>=4.4.1-
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widge
Requirement already satisfied: bleach in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextensi
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgets
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbexte
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->wi
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbexten
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widg
Requirement already satisfied: defusedxml in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbext
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->wid
Requirement already satisfied: lxml in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.9/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsn
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.9/dist-packages (from nbformat->notebook>=4.4.1->widgetsnbex
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.9/dist-packages (from nbformat->notebook>=4.4.1->widgetsnbext
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.9/dist-packages (from argon2-cffi->notebook>=4.4.1->wid
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/local/lib/python3.9/dist-packages (from jsonsche
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.9/dist-packages (from jsonschema>=2.6->nbformat->notebook>=4.4
Requirement already satisfied: websocket-client in /usr/local/lib/python3.9/dist-packages (from jupyter-server>=1.8->nbclassic>=0.4.7
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from jupyter-server>=1.8->nbclassic>=0.4.7-
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from argon2-cffi-bindings->argon2-cffi->noteboo
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.9/dist-packages (from beautifulsoup4->nbconvert>=5->notebook>=
Requirement already satisfied: webencodings in /usr/local/lib/python3.9/dist-packages (from bleach->nbconvert>=5->notebook>=4.4.1->wi
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.9/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbc
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.9/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclas
Requirement already satisfied: pycparser in /usr/local/lib/python3.9/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cf
Installing collected packages: pyro-api, typeguard, jedi, pyro-ppl, linear-operator, gpytorch, botorch, ax-platform
```

```
from ax.service.managed_loop import optimize


def train_evaluate(config):
    net = Net()
    device = "cpu"
    if torch.cuda.is_available():
        device = "cuda:0"
        if torch.cuda.device_count() > 1:
            net = nn.parallel.DataParallel(net)
    net.to(device)
    train_loop(net, data, config, device)
    return val_accuracy(net, data, device)


best_parameters, values, experiment, model = optimize(
    parameters=[
        {"name": "lr", "type": "range", "bounds": [1e-6, 0.4], "log_scale": True},
        {"name": "momentum", "type": "range", "bounds": [0.0, 1.0]},
        {"name": "batch_size", "type": "choice", "values": [1, 2, 4, 8]},
    ],
    evaluation_function=train_evaluate,
    objective_name='accuracy',
)

print(best_parameters)
```

```
[INFO 03-21 16:19:33] ax.service.utils.instantiation: Inferred value type of ParameterTy
[INFO 03-21 16:19:33] ax.service.utils.instantiation: Inferred value type of ParameterTy
[INFO 03-21 16:19:33] ax.service.utils.instantiation: Inferred value type of ParameterTy
/usr/local/lib/python3.9/dist-packages/ax/core/parameter.py:492: UserWarning: `is_ordere
  warn(
/usr/local/lib/python3.9/dist-packages/ax/core/parameter.py:492: UserWarning: `sort_valu
  warn(
[INFO 03-21 16:19:33] ax.service.utils.instantiation: Created search space: SearchSpace(
[INFO 03-21 16:19:33] ax.modelbridge.dispatch_utils: Using Models.GPEI since there are m
[INFO 03-21 16:19:33] ax.modelbridge.dispatch_utils: Calculating the number of remaining
[INFO 03-21 16:19:33] ax.modelbridge.dispatch_utils: calculated num_initialization_trial
[INFO 03-21 16:19:33] ax.modelbridge.dispatch_utils: num_completed_initialization_trials
[INFO 03-21 16:19:33] ax.modelbridge.dispatch_utils: Using Bayesian Optimization generat
[INFO 03-21 16:19:33] ax.service.managed_loop: Started full optimization with 20 steps.
[INFO 03-21 16:19:33] ax.service.managed_loop: Running optimization trial 1...
Epoch 0 validation loss=2428.053390264511, validation accuracy=0.08350515463917525
Epoch 1 validation loss=2426.083875656128, validation accuracy=0.08350515463917525
Epoch 2 validation loss=2424.218227624893, validation accuracy=0.08350515463917525
Epoch 3 validation loss=2422.3714225292206, validation accuracy=0.08350515463917525
Epoch 4 validation loss=2420.545978784561, validation accuracy=0.08350515463917525
Epoch 5 validation loss=2418.7000715732574, validation accuracy=0.08350515463917525
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-11-8c7180df8327> in <module>
     14
     15
---> 16 best_parameters, values, experiment, model = optimize(
     17     parameters=[
```

1m 13s completed at 16:20