Student No. 2403967

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
import pandas as pd
import torch.optim as optim
from  torch.utils.data import WeightedRandomSampler
import torchvision.transforms as T

from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from torch.utils.data import random_split
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset


from tqdm import tqdm
from sklearn.model_selection import train_test_split
import plotly.express as px
from IPython.display import Image

# 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
```

```
--2023-03-14 15:25:15--  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]
--2023-03-14 15:25:15--  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://ucf99cb87f90a41c7c788e853250.dl.dropboxusercontent.com/cd/0/get/B4OyCBBJGWEK_PCyXnyO6JOLxNydPsjrjND8KWfWIwxQjlZ_LAHDw5S1xDdDEP5
--2023-03-14 15:25:16--  https://ucf99cb87f90a41c7c788e853250.dl.dropboxusercontent.com/cd/0/get/B4OyCBBJGWEK_PCyXnyO6JOLxNydPsjrjND8KWfWIwxQjlZ_
Resolving ucf99cb87f90a41c7c788e853250.dl.dropboxusercontent.com (ucf99cb87f90a41c7c788e853250.dl.dropboxusercontent.com)... 162.125.81.15, 2620:
Connecting to ucf99cb87f90a41c7c788e853250.dl.dropboxusercontent.com (ucf99cb87f90a41c7c788e853250.dl.dropboxusercontent.com)|162.125.81.15|:443.
HTTP request sent, awaiting response... 200 OK
Length: 194047719 (185M) [application/binary]
Saving to: ‘plankton.pt’

plankton.pt         100%[===================>] 185.06M  12.3MB/s    in 16s

2023-03-14 15:25:32 (11.9 MB/s) - ‘plankton.pt’ saved [194047719/194047719]
```

```
!git clone https://github.com/buttsaad909/Deeplearning.git
```

```
Cloning into 'Deeplearning'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 18.05 MiB | 9.35 MiB/s, done.
```

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

# make sure you use the GPU (btw check your runtime is a GPU in colab)
use_cuda = True
device = torch.device("cuda" if (use_cuda and torch.cuda.is_available()) else "cpu")
torch.manual_seed(1)
```

```
# 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 matplotlib expects
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
```

Augmentation

Due to a small size of the dataset, I wasn't able to produce desireable results by using the default dataset. The maximum accuracy that I could achieve was from a range of 25-35%. Upon doing data augmentation, I was able to improve the accuracy results massively by augmenting the dataset to train the model.

Every image in the dataset was flipped horizontally and vertically and as well parts of it removed and as randomly rotated three times (each). The result of this augmentation resulted in producing results of the model to 91% from somewhere around 25% earlier. The size of the dataset increased from 1617 to 14553 which shows the impact of the augmentation process.


```
length_images = len(data['images'])

augmentation = {}
augmentation["images"] = list()
augmentation["labels"] = list()

for i in range(length_images):

    for j in range(9):
      augmentation["labels"].append(data['labels'][i])

    image_tensor = data['images'][i]
    augmentation["images"].append(image_tensor)

    transformation = T.Compose([T.ToPILImage(), T.ToTensor(), T.RandomVerticalFlip(p=1.0)])
    augmentation["images"].append(transformation(image_tensor))

    transformation = T.Compose([T.ToPILImage(), T.ToTensor(), T.RandomHorizontalFlip(p=1.0)])
    augmentation["images"].append(transformation(image_tensor))

    transformation = T.Compose([T.ToPILImage(), T.ToTensor(), T.RandomErasing(p=1.0)])
    augmentation["images"].append(transformation(image_tensor))
    augmentation["images"].append(transformation(image_tensor))
    augmentation["images"].append(transformation(image_tensor))

    transformation = T.Compose([T.ToPILImage(), T.ToTensor(), T.transforms.RandomRotation((0,360))])
    augmentation["images"].append(transformation(image_tensor))
    augmentation["images"].append(transformation(image_tensor))
    augmentation["images"].append(transformation(image_tensor))

augmentation['images'] = torch.stack(augmentation['images'])
augmentation['labels'] = torch.stack(augmentation['labels'])

print("Augmented Data Size Images:", len(augmentation['images']))
print("Augmented Data Size Labels:", len(augmentation['labels']))
```

```
Augmented Data Size Images: 14553
Augmented Data Size Labels: 14553
```

Simple splitting of the augmented dataset by using train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(augmentation['images'], augmentation['labels'], stratify=augmentation['labels'], test_size=0.2, ran

print(X_train.shape)
print(X_test.shape)
```

```
torch.Size([11642, 3, 100, 100])
torch.Size([2911, 3, 100, 100])
```

Used WeightedRandomSampler to deal with the imbalance in my dataset

```
class CustomDataset(torch.utils.data.Dataset):

   def __init__(self, images, labels):
    self.img = images
    self.lab = labels.to(torch.int64)
```

```
    self.transform = transformation

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

   def __getitem__(self, idx):
    images = self.img[idx]
    labels = self.lab[idx]

    return images, labels
```

```
weights_assigned = 1. / pd.Series(y_train.numpy()).groupby(y_train.numpy()).count().to_numpy()

WeightedSampler = WeightedRandomSampler(torch.from_numpy(np.array([weights_assigned[ind] for ind in y_train.numpy().astype(int)])),
                                len(torch.from_numpy(np.array([weights_assigned[ind] for ind in y_train.numpy().astype(int)]))))

trainloader = DataLoader(CustomDataset(X_train, y_train), sampler = WeightedSampler, batch_size=64)

testloader = DataLoader(CustomDataset(X_test, y_test), shuffle=True, batch_size=64)
```

The CNN Model

The CNN model designed below is a classical neural network what has four convolutional layers followed by a batch normalization later, ReLU activation function and a max_pooling opeartion.

The result of these layers is then flattened and passed through, followed by a ReLU function.

At the end, the output is passed through a log softmax activation function to get the class probabilites.

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, padding=1, stride=1, kernel_size=3)
        self.norm1 = nn.BatchNorm2d(16)
        self.fc1 = nn.Linear(128*6*6, 512)
        self.conv2 = nn.Conv2d(16, 32, padding=1, stride=1, kernel_size=3)
        self.norm2 = nn.BatchNorm2d(32)
        self.fc2 = nn.Linear(512, 64)
        self.conv3 = nn.Conv2d(32, 64, padding=1, stride=1, kernel_size=3)
        self.norm3 = nn.BatchNorm2d(64)
        self.fc3 = nn.Linear(64, 32)
        self.conv4 = nn.Conv2d(64, 128, padding=1, stride=1, kernel_size=3)
        self.norm4 = nn.BatchNorm2d(128)
        self.fc4 = nn.Linear(32, 12)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = self.norm1(x)
        x = nn.functional.max_pool2d(x,2,2)
        x = nn.functional.relu(self.conv2(x))
        x = self.norm2(x)
        x = nn.functional.max_pool2d(x,2,2)
        x = nn.functional.relu(self.conv3(x))
        x = self.norm3(x)
        x = nn.functional.max_pool2d(x,2,2)
        x = nn.functional.relu(self.conv4(x))
        x = self.norm4(x)
        x = nn.functional.max_pool2d(x,2,2)
        x = torch.flatten(x, 1)
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        x = nn.functional.relu(self.fc3(x))
        x = nn.functional.relu(self.fc4(x))

        return nn.functional.log_softmax(x, dim=-1)
```

Training

The training cline code is commented out!

```
epoch_print_gap = 1

def training_loop(n_epochs, optimizer, model, device, loss_fn, train_loader):
    model = model.to(device)
    for epoch in range(1, n_epochs + 1):
        loss_train = 0.0
        for imgs, labels in train_loader:
            outputs = model(imgs.to(device))
            loss = loss_fn(outputs, labels.to(device))
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```
            loss_train += loss.item()

        if epoch == 1 or epoch % epoch_print_gap == 0:
            print(f"Epoch: {epoch}, Training Loss: {loss_train}")

    torch.save(model.state_dict(), "CNN_model.pt")
```

```
CNN_model = CNN()

n_epochs = 100
optimizer = optim.Adam(CNN_model.parameters(), lr=0.001, weight_decay = 0.0001)
loss_fn = nn.CrossEntropyLoss()

#training_loop(n_epochs, optimizer, CNN_model, device, loss_fn, train_loader = trainloader,)
```

```
Epoch: 43, Training Loss: 8.576297196763335
Epoch: 44, Training Loss: 9.834531297266949
Epoch: 45, Training Loss: 8.307802464347333
Epoch: 46, Training Loss: 8.289942772127688
Epoch: 47, Training Loss: 5.042136246396694
Epoch: 48, Training Loss: 7.181732599448878
Epoch: 49, Training Loss: 11.017585594439879
Epoch: 50, Training Loss: 5.073284218786284
Epoch: 51, Training Loss: 7.073612255771877
Epoch: 52, Training Loss: 9.738849431509152
Epoch: 53, Training Loss: 8.571253203961533
Epoch: 54, Training Loss: 6.810456865932792
Epoch: 55, Training Loss: 7.08089028991526
Epoch: 56, Training Loss: 6.812677628506208
Epoch: 57, Training Loss: 7.571069644254749
Epoch: 58, Training Loss: 5.832178319469676
Epoch: 59, Training Loss: 5.381850018107798
Epoch: 60, Training Loss: 8.471844837709796
Epoch: 61, Training Loss: 6.813156825315673
Epoch: 62, Training Loss: 6.359225913591217
Epoch: 63, Training Loss: 6.754005993832834
Epoch: 64, Training Loss: 6.721496259968262
Epoch: 65, Training Loss: 6.843918298953213
Epoch: 66, Training Loss: 5.558923243967001
Epoch: 67, Training Loss: 7.579405911732465
Epoch: 68, Training Loss: 5.058684810239356
Epoch: 69, Training Loss: 7.60222296405118
Epoch: 70, Training Loss: 6.379202913216432
Epoch: 71, Training Loss: 6.390372179215774
Epoch: 72, Training Loss: 7.410402858542511
Epoch: 73, Training Loss: 6.422440703492612
Epoch: 74, Training Loss: 4.526186316550593
Epoch: 75, Training Loss: 6.280468989541987
Epoch: 76, Training Loss: 8.379014480364276
Epoch: 77, Training Loss: 4.657284480927046
Epoch: 78, Training Loss: 5.53824737292598
Epoch: 79, Training Loss: 7.781758251629071
Epoch: 80, Training Loss: 5.896814293548232
Epoch: 81, Training Loss: 5.158180358703248
Epoch: 82, Training Loss: 5.975570944807259
Epoch: 83, Training Loss: 4.574200690083671
Epoch: 84, Training Loss: 6.462786762291216
Epoch: 85, Training Loss: 5.4333722406299785
Epoch: 86, Training Loss: 7.107376050378662
Epoch: 87, Training Loss: 7.79775684721244
Epoch: 88, Training Loss: 6.886025117026293
Epoch: 89, Training Loss: 5.447480453120079
Epoch: 90, Training Loss: 5.68633808841696
Epoch: 91, Training Loss: 5.336107751558302
Epoch: 92, Training Loss: 6.093457053939346
Epoch: 93, Training Loss: 4.742233165161451
Epoch: 94, Training Loss: 4.844317783637962
Epoch: 95, Training Loss: 6.163464996046969
Epoch: 96, Training Loss: 7.4806505557498895
Epoch: 97, Training Loss: 5.756280324145337
Epoch: 98, Training Loss: 4.524660238661454
Epoch: 99, Training Loss: 5.842256819683826
Epoch: 100, Training Loss: 6.540311889781151
```

Testing

The model produces 91% on the test set, which is something I am happy with.

```
def test_loop(model, device, test_loader):

    model.eval()
    model = model.to(device)
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
```

```
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

```
CNN_model = CNN()
CNN_model.load_state_dict(torch.load("Deeplearning/CNN_model.pt"))
test_loop(CNN_model, device, test_loader = testloader)
```

```
Test set: Average loss: 0.4304, Accuracy: 2636/2911 (91%)
```

Design Choices

- Upon building a basic two layer convolutional neural network, followed by max pooling and then followed by two fully connected linear layers. By using this architecture and using un-augmented dataset, the accuracy that I achieved ranged from 25-35%.
- After that, I performed data augmentation where I flipped, rotated and removed some pieces from the data which helped to imporove the accuracy to more than 60%.
- Upon seeing improvements, I added 2 more layers of convolutional network, used batch normalization, and used ReLU activation functions which bumped by the accuracy to 80+%.
- Finally, I hypertuned the parameter for learning rate, weighted decay and n_epochs to settle for 0.001 for learning decay, 0.0001 weight_decay and n_epochs for 100 to get the 91% accuracy.