Student Number: 2461352

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!

Loading Data

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 torchvision
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 11:01:24--  https://www.dropbox.com/s/v2udcnt98miwwrq/plankton.pt?dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.80.18, 2620:100:6031:18::a27d:5112
Connecting to www.dropbox.com (www.dropbox.com)|162.125.80.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /s/dl/v2udcnt98miwwrq/plankton.pt [following]
--2023-03-21 11:01:24--  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://ucc1e764b2f409e7a9d6a9c18b8a.dl.dropboxusercontent.com/cd/0/get/B4popvNOcP6CJ5S2qs_7Q6cybHcV4SBBwfPloZzLiXaaRXX9
--2023-03-21 11:01:25--  https://ucc1e764b2f409e7a9d6a9c18b8a.dl.dropboxusercontent.com/cd/0/get/B4popvNOcP6CJ5S2qs_7Q6cybHcV4SBBw
Resolving ucc1e764b2f409e7a9d6a9c18b8a.dl.dropboxusercontent.com (ucc1e764b2f409e7a9d6a9c18b8a.dl.dropboxusercontent.com)... 162.12
Connecting to ucc1e764b2f409e7a9d6a9c18b8a.dl.dropboxusercontent.com (ucc1e764b2f409e7a9d6a9c18b8a.dl.dropboxusercontent.com)|162.1
HTTP request sent, awaiting response... 200 OK
Length: 194047719 (185M) [application/binary]
Saving to: ‘plankton.pt’

plankton.pt         100%[===================>] 185.06M  16.8MB/s    in 11s

2023-03-21 11:01:37 (16.1 MB/s) - ‘plankton.pt’ saved [194047719/194047719]
```

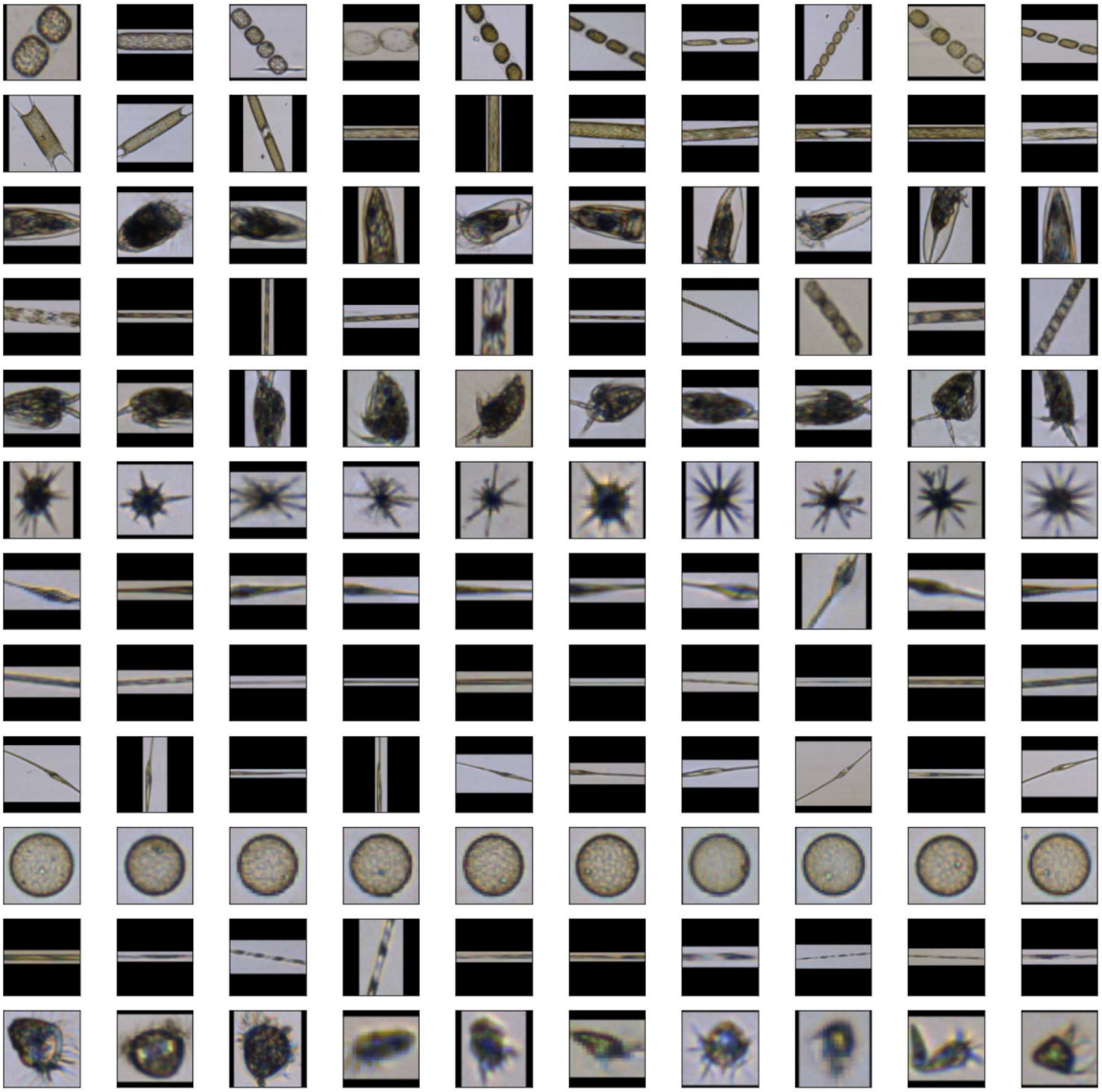
```
torch.manual_seed(12345)

use_cuda = True
device = torch.device("cuda" if (use_cuda and torch.cuda.is_available()) else "cpu")
device
```

```
device(type='cuda')
```

```
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 matp
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
```



Load data

In this section, I create a custom dataset object which:

- Supports train/test splits
- SUpports **normalisation** (subtract entire dataset mean and divide by standard deviation)
  - I decided this was appropriate as it would remove all the "noise", focusing on the sections of the images that were actually different (what the CNN needs to detect in order to classify)
- Supports upsampling using data augmentation
  - I chose to use scaled cropping and rotation as the augmentations, as the model should be scale and rotation invariant
- If upsampling, the dataset will generate augmented examples which **balance the dataset** (e.g. the final dataset will have near equal numbers of items in each class). This should help the model to make balanced predictions and not be far stronger or majority classes than others

```
from torch.utils.data import Dataset
from sklearn.model_selection import train_test_split

class PlanktonDataset(Dataset):

    def generate_augmented_data(self, data, augmented_size):

      # Create tensors
      self.augmented_images = torch.zeros([augmented_size, 3, 100, 100], dtype=data["images"].dtype)
      self.augmented_labels = torch.zeros([augmented_size], dtype=torch.int64)

      self.augmented_images[:len(self.train_labels),:,:,:] = self.train_images.detach().clone()
      self.augmented_labels[:len(self.train_labels)] = self.train_labels.detach().clone()

      all_samples = []

      # Get number of items per class (including original dataset)
      samples_per_class = augmented_size // nclasses

      # Get array of original data to sample for balanced augmentation
      for i in range(nclasses):
        idx = torch.where(self.train_labels == classes[i])[0]
        samples = np.random.choice(idx, size=(samples_per_class-len(idx)))
        all_samples.append(samples)
      all_samples = np.concatenate(all_samples)

      # Reaugment all samples
      i = len(self.train_labels)
      for s in all_samples:
        self.augmented_images[i] = self.augmentation(self.train_images[s].detach().clone())
        self.augmented_labels[i] = self.train_labels[s].detach().clone()
        i += 1

      #Randomly sample to fill up gap (if we can't have an exactly even split of classes)
      while i < len(self.augmented_labels):
        sample = np.random.randint(0, len(self.train_labels) - 1)
        self.augmented_images[i] = self.augmentation(self.train_images[sample].detach().clone())
        self.augmented_labels[i] = self.train_labels[sample].detach().clone()
        i += 1

    def __init__(self, data, normalise=False, augmented_size=0, train=True):

      self.labels = data["labels"].to(torch.int64).detach().clone()
      self.images = data["images"].detach().clone()

      self.training = train

      if normalise:
        image_means = self.images.mean([0,2,3])
        image_stds = self.images.std([0,2,3])

        for i in range(3):
          self.images[:, i, :, :] -= image_means[i]
          self.images[:, i, :, :] /= image_stds[i]

      # Split train and test data
      self.train_images, self.test_images, self.train_labels, self.test_labels = train_test_split(self.images, self.label
      self.use_augmented_data = (augmented_size - len(self.train_labels) > 0)

      if self.use_augmented_data:
        self.augmentation = torchvision.transforms.Compose([
            torchvision.transforms.RandomRotation(180),
            torchvision.transforms.RandomResizedCrop(size=100, scale=(0.9, 1.0))
        ])
        self.generate_augmented_data(data, augmented_size)
```

```
    def __len__(self):
        if not self.training:
          return len(self.test_labels)
        if self.use_augmented_data:
          return len(self.augmented_labels)
        return len(self.train_labels)

    def __getitem__(self, idx):
        if not self.training:
          return self.test_images[idx], self.test_labels[idx]
        if self.use_augmented_data:
          return self.augmented_images[idx], self.augmented_labels[idx]
        return self.train_images[idx], self.train_labels[idx]


# load datasets
train_set = PlanktonDataset(data=data, train=True)
norm_train_set = PlanktonDataset(data=data, normalise=True, train=True)
aug_train_set = PlanktonDataset(data=data, normalise=True, augmented_size=10000, train=True)

test_set = PlanktonDataset(data=data, train=False)
norm_test_set = PlanktonDataset(data=data, normalise=True, train=False)
# set up a loader object to use minibatches
loader = torch.utils.data.DataLoader(train_set, batch_size = 16, shuffle=True)
norm_loader = torch.utils.data.DataLoader(norm_train_set, batch_size = 16, shuffle=True)
aug_loader = torch.utils.data.DataLoader(aug_train_set, batch_size = 16, shuffle=True)
# also set up a loader for test data, using a single batch
test_loader = torch.utils.data.DataLoader(test_set)
norm_test_loader = torch.utils.data.DataLoader(norm_test_set)
```

Model training code

Code fom lab materials (but including gradient clipping) to train the models

```
import datetime
epoch_print_gap=1

def training_loop(n_epochs, optimizer, model, device, loss_fn, train_loader, silent=False):
    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()
            torch.nn.utils.clip_grad_norm_(model.parameters(),  max_norm=10)
            optimizer.step()

            loss_train += loss.item()

        if not silent and (epoch == 1 or epoch % epoch_print_gap == 0):
            print('{} Epoch {}, Training loss {}'.format(
                datetime.datetime.now(), epoch, float(loss_train)))

def test_loop(model, device, loss_fn, 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 += loss_fn(output, target).item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            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)))
```

Training models

Initial model

As the baseline model, I chose to use a simple convolutional neural network. It is composed of:

- A convolution layer which increases number of channels from 3 to 10
- A ReLU activation layer
- A 2x2 max pooling (to decrease the embedding size)
- A second convolution layer which increases number of channels from 10 to 20
- Another ReLU activation layer
- Another 2x2 max pooling
- A flattening (for use in the linear layers)
- 2 Linear layers to convert the convolutions to a 12 value output
- A log softmax to calculate the likelihood of each class

Later on, we will experiment with changing the values, architecture and adding some regularisation

```
# convolutional model
noHidden = 256
lr = 0.01

def get_model():
  return nn.Sequential(collections.OrderedDict([
          ("Convolution", nn.Conv2d(3, 10, kernel_size=13)),
          ("Activation", nn.ReLU()),
          ("Pool", nn.MaxPool2d((2,2), (2,2))),
          ("Convolution 2", nn.Conv2d(10, 20, kernel_size=9)),
          ("Activation", nn.ReLU()),
          ("Pool 2", nn.MaxPool2d((2,2), (2,2))),
          ("Flatten", nn.Flatten()),
          ("Linear", nn.Linear(6480, noHidden)),
          ("Activation", nn.ReLU()),
          ("Linear 2", nn.Linear(noHidden, 12)),
          ("Softmax", nn.LogSoftmax(dim=1))
        ]))

# generate model
model_conv = get_model()
# set optimiser and loss function
optimizer_conv = torch.optim.SGD(model_conv.parameters(), lr=lr)
loss_fn = nn.CrossEntropyLoss()
```

Baseline Accuracies:

These accuracies can be used to view how the model is improved over time. We will calculate the accuracy using the original data, the normalised original data and an augmented balanced dataset of size 10,000

```
n_epochs = 35
# generate model
model_conv = get_model()
# set optimiser
optimizer_conv = torch.optim.SGD(model_conv.parameters(), lr=lr)
# train the CNN
training_loop(
    n_epochs = n_epochs,
    optimizer = optimizer_conv,
    model = model_conv,
    device = device,
    loss_fn = loss_fn,
    train_loader = loader,
)
test_loop(model = model_conv, device = device, loss_fn = loss_fn, test_loader = test_loader)
```

```
2023-03-21 11:03:04.666422 Epoch 1, Training loss 165.37348985671997
2023-03-21 11:03:05.263041 Epoch 2, Training loss 129.60642421245575
2023-03-21 11:03:05.840369 Epoch 3, Training loss 120.3417357802391
2023-03-21 11:03:06.410581 Epoch 4, Training loss 114.72453314065933
2023-03-21 11:03:06.986583 Epoch 5, Training loss 112.26184570789337
2023-03-21 11:03:07.563533 Epoch 6, Training loss 107.186572432518
2023-03-21 11:03:08.134601 Epoch 7, Training loss 103.95222771167755
2023-03-21 11:03:08.711767 Epoch 8, Training loss 101.63842570781708
2023-03-21 11:03:09.286525 Epoch 9, Training loss 97.53432315587997
2023-03-21 11:03:09.857667 Epoch 10, Training loss 96.50364571809769
2023-03-21 11:03:10.428686 Epoch 11, Training loss 92.54740864038467
2023-03-21 11:03:11.000976 Epoch 12, Training loss 89.57467633485794
2023-03-21 11:03:11.572221 Epoch 13, Training loss 86.21093687415123
2023-03-21 11:03:12.148581 Epoch 14, Training loss 82.49578642845154
```

```
2023-03-21 11:03:12.726289 Epoch 15, Training loss 80.91675847768784
2023-03-21 11:03:13.300850 Epoch 16, Training loss 78.31165745854378
2023-03-21 11:03:13.880755 Epoch 17, Training loss 75.16463389992714
2023-03-21 11:03:14.456553 Epoch 18, Training loss 72.81843250989914
2023-03-21 11:03:15.029518 Epoch 19, Training loss 69.0445711016655
2023-03-21 11:03:15.612886 Epoch 20, Training loss 66.56029731035233
2023-03-21 11:03:16.220250 Epoch 21, Training loss 64.34334263205528
2023-03-21 11:03:16.837912 Epoch 22, Training loss 62.65328671038151
2023-03-21 11:03:17.455984 Epoch 23, Training loss 60.56069599092007
2023-03-21 11:03:18.046211 Epoch 24, Training loss 58.06434512138367
2023-03-21 11:03:18.626706 Epoch 25, Training loss 57.350107967853546
2023-03-21 11:03:19.205130 Epoch 26, Training loss 53.22575789690018
2023-03-21 11:03:19.794826 Epoch 27, Training loss 52.3888136446476
2023-03-21 11:03:20.373624 Epoch 28, Training loss 50.595063269138336
2023-03-21 11:03:20.958178 Epoch 29, Training loss 48.00672824680805
2023-03-21 11:03:21.539896 Epoch 30, Training loss 46.70724618434906
2023-03-21 11:03:22.122956 Epoch 31, Training loss 46.248865500092506
2023-03-21 11:03:22.712117 Epoch 32, Training loss 42.71952448785305
2023-03-21 11:03:23.296542 Epoch 33, Training loss 41.792680971324444
2023-03-21 11:03:23.883111 Epoch 34, Training loss 39.978246197104454
2023-03-21 11:03:24.467121 Epoch 35, Training loss 38.536164440214634

Test set: Average loss: 1.7505, Accuracy: 174/324 (54%)
```

```
# generate model
model_conv = get_model()
# set optimiser
optimizer_conv = torch.optim.SGD(model_conv.parameters(), lr=lr)
# train the CNN
training_loop(
    n_epochs = n_epochs,
    optimizer = optimizer_conv,
    model = model_conv,
    device = device,
    loss_fn = loss_fn,
    train_loader = norm_loader,
)
test_loop(model = model_conv, device = device, loss_fn = loss_fn, test_loader = norm_test_loader)
```

```
2023-03-21 11:03:41.964548 Epoch 1, Training loss 147.80387890338898
2023-03-21 11:03:42.548140 Epoch 2, Training loss 117.71940004825592
2023-03-21 11:03:43.132262 Epoch 3, Training loss 104.65522772073746
2023-03-21 11:03:43.770779 Epoch 4, Training loss 97.68602514266968
2023-03-21 11:03:44.557169 Epoch 5, Training loss 91.54905426502228
2023-03-21 11:03:45.443656 Epoch 6, Training loss 84.8468134701252
2023-03-21 11:03:46.106190 Epoch 7, Training loss 82.38323622941971
2023-03-21 11:03:46.694685 Epoch 8, Training loss 76.99865141510963
2023-03-21 11:03:47.277604 Epoch 9, Training loss 73.72032085061073
2023-03-21 11:03:47.866057 Epoch 10, Training loss 69.67052125930786
2023-03-21 11:03:48.449728 Epoch 11, Training loss 64.3704945743084
2023-03-21 11:03:49.039496 Epoch 12, Training loss 60.40803091228008
2023-03-21 11:03:49.630238 Epoch 13, Training loss 58.16985096037388
2023-03-21 11:03:50.404320 Epoch 14, Training loss 53.900026232004166
2023-03-21 11:03:51.746361 Epoch 15, Training loss 50.64354032278061
2023-03-21 11:03:53.146733 Epoch 16, Training loss 50.41144624352455
2023-03-21 11:03:54.046206 Epoch 17, Training loss 47.04776990413666
2023-03-21 11:03:54.634822 Epoch 18, Training loss 45.65203730762005
2023-03-21 11:03:55.227356 Epoch 19, Training loss 43.14748574793339
2023-03-21 11:03:55.833016 Epoch 20, Training loss 38.797262236475945
2023-03-21 11:03:56.591915 Epoch 21, Training loss 37.17128784954548
2023-03-21 11:03:57.392859 Epoch 22, Training loss 35.71130508556962
2023-03-21 11:03:58.059888 Epoch 23, Training loss 32.17001165449619
2023-03-21 11:03:58.651079 Epoch 24, Training loss 30.99953742325306
2023-03-21 11:03:59.245259 Epoch 25, Training loss 30.028542704880238
2023-03-21 11:03:59.839731 Epoch 26, Training loss 28.766520779579878
2023-03-21 11:04:00.436519 Epoch 27, Training loss 27.088940788060427
2023-03-21 11:04:01.029337 Epoch 28, Training loss 25.90709651261568
2023-03-21 11:04:01.622686 Epoch 29, Training loss 25.27094940468669
2023-03-21 11:04:02.211903 Epoch 30, Training loss 22.887812864035368
2023-03-21 11:04:02.800392 Epoch 31, Training loss 22.255528569221497
2023-03-21 11:04:03.411296 Epoch 32, Training loss 20.179976988583803
2023-03-21 11:04:04.023305 Epoch 33, Training loss 19.943205313757062
2023-03-21 11:04:04.649355 Epoch 34, Training loss 17.536104798316956
2023-03-21 11:04:05.265682 Epoch 35, Training loss 18.995347995311022

Test set: Average loss: 2.2864, Accuracy: 183/324 (56%)
```

```
# generate model
model_conv = get_model()
# set optimiser
optimizer_conv = torch.optim.SGD(model_conv.parameters(), lr=lr)
# train the CNN
training_loop(
    n_epochs = n_epochs,
```

```
    optimizer = optimizer_conv,
    model = model_conv,
    device = device,
    loss_fn = loss_fn,
    train_loader = aug_loader,
)
test_loop(model = model_conv, device = device, loss_fn = loss_fn, test_loader = norm_test_loader)
```

```
2023-03-21 11:04:10.085699 Epoch 1, Training loss 873.3106174468994
2023-03-21 11:04:15.390707 Epoch 2, Training loss 709.3962269723415
2023-03-21 11:04:20.707500 Epoch 3, Training loss 650.1392314136028
2023-03-21 11:04:25.229187 Epoch 4, Training loss 610.598666369915
2023-03-21 11:04:30.396828 Epoch 5, Training loss 576.9133334904909
2023-03-21 11:04:34.884861 Epoch 6, Training loss 548.3805478364229
2023-03-21 11:04:39.429866 Epoch 7, Training loss 531.194458335638
2023-03-21 11:04:44.175791 Epoch 8, Training loss 506.7088025957346
2023-03-21 11:04:49.330043 Epoch 9, Training loss 481.64419716596603
2023-03-21 11:04:53.998806 Epoch 10, Training loss 463.20808976888657
2023-03-21 11:04:58.720680 Epoch 11, Training loss 443.57610085606575
2023-03-21 11:05:03.438693 Epoch 12, Training loss 427.6065788269043
2023-03-21 11:05:08.116718 Epoch 13, Training loss 403.56546944379807
2023-03-21 11:05:12.886408 Epoch 14, Training loss 385.4828147739172
2023-03-21 11:05:17.425599 Epoch 15, Training loss 371.0687774568796
2023-03-21 11:05:21.963164 Epoch 16, Training loss 345.99568378180265
2023-03-21 11:05:26.453779 Epoch 17, Training loss 335.349942214787
2023-03-21 11:05:31.073806 Epoch 18, Training loss 313.08566799014807
2023-03-21 11:05:35.576419 Epoch 19, Training loss 297.24348609894514
2023-03-21 11:05:40.052912 Epoch 20, Training loss 278.5901518948376
2023-03-21 11:05:44.625792 Epoch 21, Training loss 271.4993350841105
2023-03-21 11:05:49.125247 Epoch 22, Training loss 253.0196265913546
2023-03-21 11:05:53.636099 Epoch 23, Training loss 243.90917060151696
2023-03-21 11:05:58.168121 Epoch 24, Training loss 224.68384223617613
2023-03-21 11:06:02.641694 Epoch 25, Training loss 208.55302365869284
2023-03-21 11:06:07.224565 Epoch 26, Training loss 196.63923583645374
2023-03-21 11:06:11.783057 Epoch 27, Training loss 189.90496662072837
2023-03-21 11:06:16.905302 Epoch 28, Training loss 177.6258527431637
2023-03-21 11:06:22.343461 Epoch 29, Training loss 165.26938586961478
2023-03-21 11:06:27.169583 Epoch 30, Training loss 154.60298346704803
2023-03-21 11:06:32.278387 Epoch 31, Training loss 144.23081833892502
2023-03-21 11:06:37.070613 Epoch 32, Training loss 137.26688921253663
2023-03-21 11:06:41.900820 Epoch 33, Training loss 127.76749277091585
2023-03-21 11:06:47.210663 Epoch 34, Training loss 113.04029629161232
2023-03-21 11:06:51.998378 Epoch 35, Training loss 108.09219831429073

Test set: Average loss: 2.1409, Accuracy: 194/324 (60%)
```

- No normalisation, no augmentation: 172/324 (54%)
- Normalisation, no augmentation: 183/324 (56%)
- Normalisation, augmentation: 194/324 (60%)

Model optimisation

To save time and resources, model variations were ran for 15 epochs on the non-augmented normalised dataset, and individual variations were tried sequentially and effective variations (which improved accuracy) are kept. The baseline accuracy was 174/324 (54%). The following variations were tried, with boldface indicating variations which improved accuracy:

- **Increasing number of channels from 5 and 10 to 16 and 32, and doubling number of intermediate hidden layers in the linear section to 256 (190/324, 59%)**
- Adding an extra convolutional layer of 24 channels between the two current convolutional layers
- Adding an extra linear layer with 4096 hidden outputs between the two current linear layers
- Switching the order of the pooling and activation layers
- Adding a dropout layer after the second pooling layer (with p=0.5 and p=0.3)
- Reducing the kernel size of the convolutions and increasing the number of channels
- Increasing the number of convolution channels without decreasing the kernel size
- Increasing the number of intermediate layers in the linear section to 512 and 1024
- Removing the second pooling layer
- Adding a third convolution layer with no pooling between 2nd and 3rd convolution layers
- Adding a dropout layer before the second pooling layer (p=0.5 and p=0.3
- **Removing the second linear layer (192/324, 59%)**

Shown below is the new model after the changes are made:

```
def get_improved_model():
  return nn.Sequential(collections.OrderedDict([
          ("Convolution", nn.Conv2d(3, 16, kernel_size=13)),
          ("Activation", nn.ReLU()),
          ("Pool", nn.MaxPool2d((2,2), (2,2))),
```

```
            ("Convolution 2", nn.Conv2d(16, 32, kernel_size=9)),
            ("Activation", nn.ReLU()),
            ("Dropout", nn.Dropout2d(p=0.3)),
            ("Pool 2", nn.MaxPool2d((2,2), (2,2))),
            ("Flatten", nn.Flatten()),
            ("Linear", nn.Linear(10368, 12)),
            ("Softmax", nn.LogSoftmax(dim=1))
          ]))

# generate model
model_conv = get_improved_model()
# set optimiser and loss function
optimizer_conv = torch.optim.SGD(model_conv.parameters(), lr=lr)


n_epochs = 15

# train the CNN
training_loop(
    n_epochs = n_epochs,
    optimizer = optimizer_conv,
    model = model_conv,
    device = device,
    loss_fn = loss_fn,
    train_loader = norm_loader,
)
test_loop(model = model_conv, device = device, loss_fn = loss_fn, test_loader = norm_test_loader)
```

```
2023-03-21 11:48:25.598566 Epoch 1, Training loss 148.72956454753876
2023-03-21 11:48:26.152703 Epoch 2, Training loss 122.06906306743622
2023-03-21 11:48:26.926241 Epoch 3, Training loss 110.42716437578201
2023-03-21 11:48:27.505732 Epoch 4, Training loss 102.52837532758713
2023-03-21 11:48:28.068027 Epoch 5, Training loss 94.39941036701202
2023-03-21 11:48:28.617319 Epoch 6, Training loss 89.77071017026901
2023-03-21 11:48:29.203554 Epoch 7, Training loss 82.45360189676285
2023-03-21 11:48:29.862849 Epoch 8, Training loss 79.523333132267
2023-03-21 11:48:30.450224 Epoch 9, Training loss 77.32567435503006
2023-03-21 11:48:31.116634 Epoch 10, Training loss 69.39973539113998
2023-03-21 11:48:31.718055 Epoch 11, Training loss 64.87508192658424
2023-03-21 11:48:32.354585 Epoch 12, Training loss 63.188399225473404
2023-03-21 11:48:33.244818 Epoch 13, Training loss 60.473756581544876
2023-03-21 11:48:34.100831 Epoch 14, Training loss 55.70406310260296
2023-03-21 11:48:34.971969 Epoch 15, Training loss 52.61545006930828

Test set: Average loss: 1.4765, Accuracy: 192/324 (59%)
```

Let's try the new model on the augmented dataset (of size 10,000)

```
n_epochs = 35

# generate model
model_conv = get_improved_model()
# set optimiser and loss function
optimizer_conv = torch.optim.SGD(model_conv.parameters(), lr=lr)

# train the CNN
training_loop(
    n_epochs = n_epochs,
    optimizer = optimizer_conv,
    model = model_conv,
    device = device,
    loss_fn = loss_fn,
    train_loader = aug_loader,
)
test_loop(model = model_conv, device = device, loss_fn = loss_fn, test_loader = norm_test_loader)
```

```
2023-03-21 11:50:10.668702 Epoch 1, Training loss 918.6292443871498
2023-03-21 11:50:15.178726 Epoch 2, Training loss 765.7356795668602
2023-03-21 11:50:20.281459 Epoch 3, Training loss 696.8755466938019
2023-03-21 11:50:29.156873 Epoch 5, Training loss 610.6885731816292
2023-03-21 11:50:33.557231 Epoch 6, Training loss 577.210551828146
2023-03-21 11:50:37.856169 Epoch 7, Training loss 544.4997192621231
2023-03-21 11:50:42.198327 Epoch 8, Training loss 518.1945608556271
2023-03-21 11:50:46.576216 Epoch 9, Training loss 485.94902999699116
2023-03-21 11:50:50.843413 Epoch 10, Training loss 454.9268746525049
2023-03-21 11:50:55.200232 Epoch 11, Training loss 429.0802566111088
2023-03-21 11:50:59.438629 Epoch 12, Training loss 402.3796516805887
2023-03-21 11:51:03.673239 Epoch 13, Training loss 385.209616381675
2023-03-21 11:51:08.007439 Epoch 14, Training loss 362.6026476472616
2023-03-21 11:51:12.233071 Epoch 15, Training loss 343.0990045107901
2023-03-21 11:51:16.483208 Epoch 16, Training loss 318.68985355459154
2023-03-21 11:51:20.820894 Epoch 17, Training loss 308.08665109053254
2023-03-21 11:51:25.069999 Epoch 18, Training loss 286.56529126502573
```

```
2023-03-21 11:51:29.350547 Epoch 19, Training loss 270.2818593326956
2023-03-21 11:51:33.682160 Epoch 20, Training loss 259.03691862151027
2023-03-21 11:51:37.947833 Epoch 21, Training loss 243.07106925547123
2023-03-21 11:51:42.257688 Epoch 22, Training loss 223.7336306218058
2023-03-21 11:51:46.553584 Epoch 23, Training loss 217.36651736870408
2023-03-21 11:51:50.806492 Epoch 24, Training loss 210.90323145501316
2023-03-21 11:51:55.167011 Epoch 25, Training loss 209.5093921170337
2023-03-21 11:51:59.425205 Epoch 26, Training loss 184.9453620761633
2023-03-21 11:52:03.690084 Epoch 27, Training loss 178.2112844986841
2023-03-21 11:52:08.268328 Epoch 28, Training loss 173.45631172903813
2023-03-21 11:52:12.563716 Epoch 29, Training loss 170.38446395564824
2023-03-21 11:52:16.813829 Epoch 30, Training loss 152.82636367017403
2023-03-21 11:52:21.165060 Epoch 31, Training loss 144.4502615169622
2023-03-21 11:52:25.445183 Epoch 32, Training loss 150.0628118130844
2023-03-21 11:52:29.730688 Epoch 33, Training loss 141.1230579309631
2023-03-21 11:52:34.053040 Epoch 34, Training loss 134.1894664191641
2023-03-21 11:52:38.288625 Epoch 35, Training loss 128.35975532396697

Test set: Average loss: 2.0636, Accuracy: 206/324 (64%)
```

We have a new best accuracy of 206/324 (64%)!

Hyperparameter optimisation

We are going to optimise the learning rate and momentum using k_fold cross validation with 10 folds, using 10 epochs to save resources (this may result in less optimal hyperparameters but there is not enough time to run each trial for 35 epochs).

```
!pip3 install ax-platform

from ax.plot.trace import optimization_trace_single_method
from ax.service.managed_loop import optimize
from ax.utils.tutorials.cnn_utils import train, evaluate
```

```
Collecting jedi>=0.10
  Downloading jedi-0.18.2-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 77.7 MB/s eta 0:00:00
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.9/dist-packages (from ipython>=4.0.0->ipywidgets->ax-p
Requirement already satisfied: prompt-toolkit<2.1.0,>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from ipython>=4.0.0->ipyw
Requirement already satisfied: pickleshare in /usr/local/lib/python3.9/dist-packages (from ipython>=4.0.0->ipywidgets->ax-platfo
Requirement already satisfied: pexpect in /usr/local/lib/python3.9/dist-packages (from ipython>=4.0.0->ipywidgets->ax-platform)
Requirement already satisfied: decorator in /usr/local/lib/python3.9/dist-packages (from ipython>=4.0.0->ipywidgets->ax-platform
```

```
Requirement already satisfied: webencodings in /usr/local/lib/python3.9/dist-packages (from bleach->nbconvert>=5->notebook>=4.4.
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.9/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->n
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.9/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8
Requirement already satisfied: pycparser in /usr/local/lib/python3.9/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argo
Installing collected packages: pyro-api, typeguard, jedi, pyro-ppl, linear-operator, gpytorch, botorch, ax-platform
Successfully installed ax-platform-0.3.1 botorch-0.8.3 gpytorch-1.9.1 jedi-0.18.2 linear-operator-0.3.0 pyro-api-0.1.2 pyro-ppl
```

```
from sklearn.model_selection import KFold
k_folds = 10
n_epochs = 10
dtype = torch.float

# K-Fold validation adapted from https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-k-fold-cross-valida
def train_evaluate(parameterization):
    kfold = KFold(n_splits=k_folds, shuffle=True)
    total_acc = 0
    # Split into k train-test folds
    for fold, (train_ids, test_ids) in enumerate(kfold.split(aug_train_set)):
      # Load data from folds
      train_subsampler = torch.utils.data.SubsetRandomSampler(train_ids)
      test_subsampler = torch.utils.data.SubsetRandomSampler(test_ids)

      train_loader = torch.utils.data.DataLoader(
                     aug_train_set,
                     batch_size=32, sampler=train_subsampler)
      test_loader = torch.utils.data.DataLoader(
                     aug_train_set, sampler=test_subsampler)
      # Train model
      net = get_improved_model()
      optimizer = torch.optim.SGD(net.parameters(), lr=parameterization["lr"], momentum=parameterization["momentum"])

      training_loop(
          n_epochs = n_epochs,
          optimizer = optimizer,
          model = net,
          device = device,
          loss_fn = loss_fn,
          train_loader = train_loader,
          silent=True
      )
      # Get average fold accuracy
      total_acc += evaluate(
          net=net,
          data_loader=test_loader,
          dtype=dtype,
          device=device,
      )
    return total_acc / k_folds
```

```
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]},
    ],
    evaluation_function=train_evaluate,
    objective_name='accuracy',
)

best_parameters
```

```
[INFO 03-21 12:28:09] ax.service.utils.instantiation: Inferred value type of ParameterType.FLOAT for parameter lr. If that is not t
[INFO 03-21 12:28:09] ax.service.utils.instantiation: Inferred value type of ParameterType.FLOAT for parameter momentum. If that is
[INFO 03-21 12:28:09] ax.service.utils.instantiation: Created search space: SearchSpace(parameters=[RangeParameter(name='lr', param
[INFO 03-21 12:28:09] ax.modelbridge.dispatch_utils: Using Models.GPEI since there are more ordered parameters than there are categ
[INFO 03-21 12:28:09] ax.modelbridge.dispatch_utils: Calculating the number of remaining initialization trials based on num_initial
[INFO 03-21 12:28:09] ax.modelbridge.dispatch_utils: calculated num_initialization_trials=5
[INFO 03-21 12:28:09] ax.modelbridge.dispatch_utils: num_completed_initialization_trials=0 num_remaining_initialization_trials=5
[INFO 03-21 12:28:09] ax.modelbridge.dispatch_utils: Using Bayesian Optimization generation strategy: GenerationStrategy(name='Sobo
[INFO 03-21 12:28:09] ax.service.managed_loop: Started full optimization with 20 steps.
[INFO 03-21 12:28:09] ax.service.managed_loop: Running optimization trial 1...
[INFO 03-21 12:32:09] ax.service.managed_loop: Running optimization trial 2...
[INFO 03-21 12:36:04] ax.service.managed_loop: Running optimization trial 3...
[INFO 03-21 12:39:59] ax.service.managed_loop: Running optimization trial 4...
[INFO 03-21 12:43:54] ax.service.managed_loop: Running optimization trial 5...
[INFO 03-21 12:47:50] ax.service.managed_loop: Running optimization trial 6...
[INFO 03-21 12:51:44] ax.service.managed_loop: Running optimization trial 7...
[INFO 03-21 12:55:39] ax.service.managed_loop: Running optimization trial 8...
[INFO 03-21 12:59:35] ax.service.managed_loop: Running optimization trial 9...
[INFO 03-21 13:03:27] ax.service.managed_loop: Running optimization trial 10...
[INFO 03-21 13:07:22] ax.service.managed_loop: Running optimization trial 11...
[INFO 03-21 13:11:17] ax.service.managed_loop: Running optimization trial 12...
[INFO 03-21 13:15:12] ax.service.managed_loop: Running optimization trial 13...
[INFO 03-21 13:19:07] ax.service.managed_loop: Running optimization trial 14...
[INFO 03-21 13:23:03] ax.service.managed_loop: Running optimization trial 15...
```

```
[INFO 03-21 13:23:03] ax.modelbridge.base: Untransformed parameter 0.40000000000000013 greater than upper bound 0.4, clamping
[INFO 03-21 13:26:55] ax.service.managed_loop: Running optimization trial 16...
[INFO 03-21 13:30:50] ax.service.managed_loop: Running optimization trial 17...
[INFO 03-21 13:34:46] ax.service.managed_loop: Running optimization trial 18...
[INFO 03-21 13:38:40] ax.service.managed_loop: Running optimization trial 19...
[INFO 03-21 13:42:35] ax.service.managed_loop: Running optimization trial 20...
/usr/local/lib/python3.9/dist-packages/botorch/models/utils/assorted.py:201: InputDataWarning: Input data is not standardized. Plea
  warnings.warn(msg, InputDataWarning)
{'lr': 0.0027344855796973054, 'momentum': 0.6780177914469011}
```

The best parameters found are:

- Learning rate: 0.0027344855796973054
- Momentum: 0.6780177914469011

Final result:

```
n_epochs = 35

# generate model
model_conv = get_improved_model()
# set optimiser and loss function
optimizer_conv = torch.optim.SGD(model_conv.parameters(), lr=best_parameters["lr"], momentum=best_parameters["momentum"])

# train the CNN
training_loop(
    n_epochs = n_epochs,
    optimizer = optimizer_conv,
    model = model_conv,
    device = device,
    loss_fn = loss_fn,
    train_loader = aug_loader,
)
test_loop(model = model_conv, device = device, loss_fn = loss_fn, test_loader = norm_test_loader)
torch.save(model_conv, "models/cnn.pt")
```

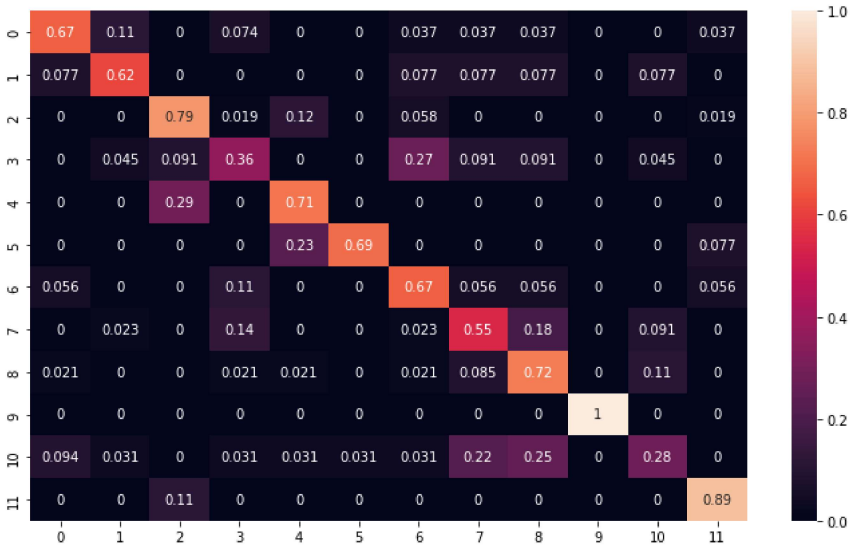
```
2023-03-21 14:10:43.801899 Epoch 1, Training loss 894.9363679289818
2023-03-21 14:10:48.053964 Epoch 2, Training loss 741.7179484963417
2023-03-21 14:10:52.362696 Epoch 3, Training loss 682.4286262094975
2023-03-21 14:10:56.769422 Epoch 4, Training loss 623.0643245279789
2023-03-21 14:11:01.070406 Epoch 5, Training loss 580.0111924260855
2023-03-21 14:11:05.402874 Epoch 6, Training loss 540.3949448168278
2023-03-21 14:11:09.837619 Epoch 7, Training loss 501.3995380550623
2023-03-21 14:11:14.185303 Epoch 8, Training loss 477.98770452290773
2023-03-21 14:11:18.537025 Epoch 9, Training loss 454.23105420172215
2023-03-21 14:11:22.960954 Epoch 10, Training loss 422.36643827706575
2023-03-21 14:11:27.229837 Epoch 11, Training loss 402.8574287071824
2023-03-21 14:11:31.535342 Epoch 12, Training loss 387.6705908663571
2023-03-21 14:11:35.938411 Epoch 13, Training loss 361.32423889636993
2023-03-21 14:11:40.179473 Epoch 14, Training loss 347.2164825350046
2023-03-21 14:11:44.446170 Epoch 15, Training loss 323.0865153744817
2023-03-21 14:11:48.817673 Epoch 16, Training loss 307.771915666759
2023-03-21 14:11:53.054788 Epoch 17, Training loss 296.91140746325254
2023-03-21 14:11:57.329586 Epoch 18, Training loss 275.51936756260693
2023-03-21 14:12:01.695809 Epoch 19, Training loss 263.96151395421475
2023-03-21 14:12:05.953788 Epoch 20, Training loss 256.7441857084632
2023-03-21 14:12:10.240371 Epoch 21, Training loss 242.82245896197855
2023-03-21 14:12:14.624054 Epoch 22, Training loss 230.8371398979798
2023-03-21 14:12:18.860785 Epoch 23, Training loss 217.19458629656583
2023-03-21 14:12:23.126515 Epoch 24, Training loss 206.65163813810796
2023-03-21 14:12:27.509610 Epoch 25, Training loss 197.8450617659837
2023-03-21 14:12:31.774086 Epoch 26, Training loss 189.97711831890047
2023-03-21 14:12:36.054089 Epoch 27, Training loss 182.47794525744393
2023-03-21 14:12:40.464240 Epoch 28, Training loss 179.13462788495235
2023-03-21 14:12:44.736746 Epoch 29, Training loss 163.64707081811503
2023-03-21 14:12:49.005597 Epoch 30, Training loss 161.92300346447155
2023-03-21 14:12:53.400045 Epoch 31, Training loss 151.0857154338155
2023-03-21 14:12:57.668523 Epoch 32, Training loss 150.98103542320314
2023-03-21 14:13:01.942768 Epoch 33, Training loss 138.0103317243047
2023-03-21 14:13:06.322413 Epoch 34, Training loss 138.37134184670867
2023-03-21 14:13:10.578457 Epoch 35, Training loss 133.02833284426015

Test set: Average loss: 1.9490, Accuracy: 212/324 (65%)
```

We achieve a final accuracy of 212 / 324 (65%)

Evaluation of results:

I will create a confusion matrix to analyse exactly which classes the model gets confused, and inspect examples of these classes

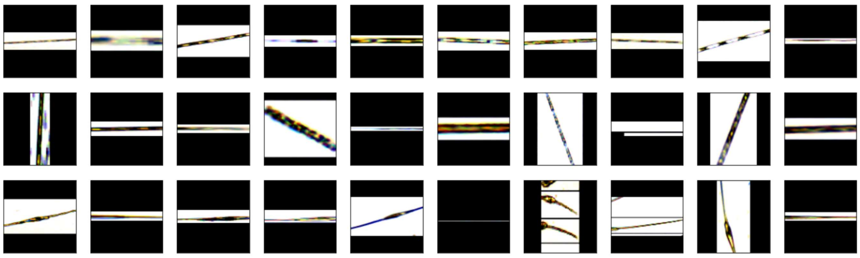
```
# Adapted from https://christianbernecker.medium.com/how-to-create-a-confusion-matrix-in-pytorch-38d06a7f04b7
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd

def get_confusion_matrix(input_model, test_loader):
    input_model = input_model.cpu()
    y_pred = []
    y_true = []

    with torch.no_grad():
        for data, target in test_loader:
            output = input_model(data)
            pred = output.argmax(dim=1, keepdim=True).cpu().numpy()  # get the index of the max log-probability
            y_pred.extend(pred)
            y_true.extend(target.cpu().numpy())

    # constant for classes
    classes = [i for i in range(12)]

    # Build confusion matrix
    cf_matrix = confusion_matrix(y_true, y_pred)
    df_cm = pd.DataFrame(cf_matrix / np.sum(cf_matrix, axis=1)[:, None], index = classes,
                         columns = classes)
    plt.figure(figsize = (12,7))
    plt.ylabel("Actual label")
    plt.xlabel("Predicted label")
    sn.heatmap(df_cm, annot=True)
    plt.savefig('cfm.png')

# print(type(model_conv))

get_confusion_matrix(model_conv, norm_test_loader)
```



The actual labels are plotted on the Y axis and the predicted labels on the X axis. We can see that the model has the most problems with class 10, only correctly classifying 28% of these inputs. It often mistakes them for classes 7 and 8. Let's examine these classes:

```
# we now print some examples from each class for visualisation
fig = plt.figure(figsize=(20,6))
plankton_classes = [10, 7, 8]

n = 10 # number of examples to show per class
test_images, test_labels = norm_test_set[:]
for i, plankton_class in enumerate(plankton_classes):
  idx = test_labels == plankton_class
  imgs = test_images[idx,...]
  for j in range(n):
    ax = plt.subplot(3,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 dimensi
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
```



As we can see, not only are these images very similar, they also often do not contain many non-black pixels, which may make it difficult for the model to understand the difference. This dataset is very challenging, particularly with these three classes, so it is difficult to achieve high accuracy

Using pre-trained models

```
!wget https://github.com/Adamanatite/deep-learning-coursework-models/blob/master/models/cnn.pt?raw=true
os.rename("cnn.pt?raw=true", "cnn.pt")
```

```
--2023-03-21 15:07:33--  https://github.com/Adamanatite/deep-learning-coursework-models/blob/master/models/cnn.pt?raw=true
Resolving github.com (github.com)... 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/Adamanatite/deep-learning-coursework-models/raw/master/models/cnn.pt [following]
--2023-03-21 15:07:33--  https://github.com/Adamanatite/deep-learning-coursework-models/raw/master/models/cnn.pt
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/Adamanatite/deep-learning-coursework-models/master/models/cnn.pt [following]
--2023-03-21 15:07:34--  https://raw.githubusercontent.com/Adamanatite/deep-learning-coursework-models/master/models/cnn.pt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 700343 (684K) [application/octet-stream]
Saving to: 'cnn.pt?raw=true'

cnn.pt?raw=true     100%[===================>] 683.93K  --.-KB/s    in 0.006s

2023-03-21 15:07:34 (120 MB/s) - 'cnn.pt?raw=true' saved [700343/700343]
```

```
loaded_cnn = torch.load("cnn.pt")

test_loop(model = loaded_cnn, device = device, loss_fn = loss_fn, test_loader = norm_test_loader)
```

```
Test set: Average loss: 1.9490, Accuracy: 212/324 (65%)
```

✓ 0s completed at 15:07

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.