coursework_02

February 25, 2021

1 Coursework 2: Image Classification

In this coursework, you are going to develop a neural network model for for image classification.

1.1 What to do?

- Complete and run the code using `jupyter-lab` or `jupyter-notebook` to get the results.
- Export (File | Export Notebook As...) or print (using the print function of your browser) the notebook as a pdf file, which contains your code, results and text answers, and upload the pdf file onto Cate.

1.2 Dependencies:

If you do not have Jupyter-Lab on your laptop, you can find information for installing Jupyter-Lab here.

There may be certain Python packages you may want to use for completing the coursework. We have provided examples below for importing libraries. If some packages are missing, you need to install them. In general, new packages (e.g. imageio etc) can be installed by running

`pip3 install [package_name]`

in the terminal. If you use Anaconda, you can also install new packages by running `conda install [package_name]` or using its graphic user interface.

[1]:
```
# Import libraries (provided)
import numpy as np
import matplotlib.pyplot as plt
import time
import random
from sklearn import metrics
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
```

1.3 1. Load and visualise data. (25 marks)

Throughout this coursework. you will be working with the Fashion-MNIST dataset. If you are interested, you may find information about the dataset in this paper.

[1] Han Xiao, Kashif Rasul, Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747

The dataset is prepared in a similar way to MNIST. It is split into a training set of 60,000 images and a test set of 10,000 images. The images are of size 28x28 pixels.

There are in total 10 label classes, which are:

Label	Description
0	T-shirt/top
1	Trousers
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

[2]:
```
transform = transforms.ToTensor()

# Load data (provided)
# Note that some early versions of torchvision use different names, such as␣
 ↪train_data, test_data instead of data.
# However, after torchvision version 0.4.0, data is used as the variable name.
train_set = torchvision.datasets.FashionMNIST(root='.', download=True,␣
 ↪transform=transform, train=True)
train_image = train_set.data.numpy()
train_label = train_set.targets.numpy()
class_name = train_set.classes

test_set = torchvision.datasets.FashionMNIST(root='.', download=True,␣
 ↪transform=transform, train=False)
test_image = np.array(test_set.data)
test_label = np.array(test_set.targets)
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
images-idx3-ubyte.gz to ./FashionMNIST/raw/train-images-idx3-ubyte.gz
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0),␣
 ↪HTML(value='')))
```

```
Extracting ./FashionMNIST/raw/train-images-idx3-ubyte.gz to ./FashionMNIST/raw
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
labels-idx1-ubyte.gz to ./FashionMNIST/raw/train-labels-idx1-ubyte.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0),␣
 ↪HTML(value='')))

Extracting ./FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./FashionMNIST/raw
Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to
./FashionMNIST/raw/t10k-images-idx3-ubyte.gz


HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0),␣
 ↪HTML(value='')))

Extracting ./FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./FashionMNIST/raw
Downloading http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to
./FashionMNIST/raw/t10k-labels-idx1-ubyte.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0),␣
 ↪HTML(value='')))

Extracting ./FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./FashionMNIST/raw
Processing…
Done!

/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:480:
UserWarning: The given NumPy array is not writeable, and PyTorch does not
support non-writeable tensors. This means you can write to the underlying
(supposedly non-writeable) NumPy array using the tensor. You may want to copy
the array to protect its data or make it writeable before converting it to a
tensor. This type of warning will be suppressed for the rest of this program.
(Triggered internally at  /pytorch/torch/csrc/utils/tensor_numpy.cpp:141.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

1.3.1 1.1 Display the dimension of the training and test sets. (5 marks)

[3]:
```
print("Training set: {}".format(train_image.shape))
print("Test set: {}".format(test_image.shape))
```

```
Training set: (60000, 28, 28)
Test set: (10000, 28, 28)
```

1.3.2 1.2 Visualise sample images for each of the 10 classes. (5 marks)

Please plot 10 rows x 10 columns of images. Each row shows 10 samples for one class. For example, row 1 shows 10 `T-shirt/top` images, row 2 shows 10 `Trousers` images.

[4]:
```
N = 10
fig, axes = plt.subplots(N, N, figsize=(15, 15))
```

```
for row in range(N):
    col = 0
    for img, label in train_set:
        if col == 10: break
        if label == row:
            axes[row, col].imshow(img.squeeze(), cmap="gray")
            axes[row, col].set_xticks([])
            axes[row, col].set_yticks([])
            col += 1
```

1.3.3 1.3 Display the number of training samples for each class. (5 marks)

[5]:
```
labels = np.zeros(10, dtype=int)
for i in train_label:
    labels[i] += 1
for i in range(len(class_name)):
    print("{}: {}".format(class_name[i], labels[i]))
```

```
T-shirt/top: 6000
Trouser: 6000
Pullover: 6000
Dress: 6000
Coat: 6000
Sandal: 6000
Shirt: 6000
Sneaker: 6000
Bag: 6000
Ankle boot: 6000
```

1.3.4 1.4 Discussion. (10 marks)

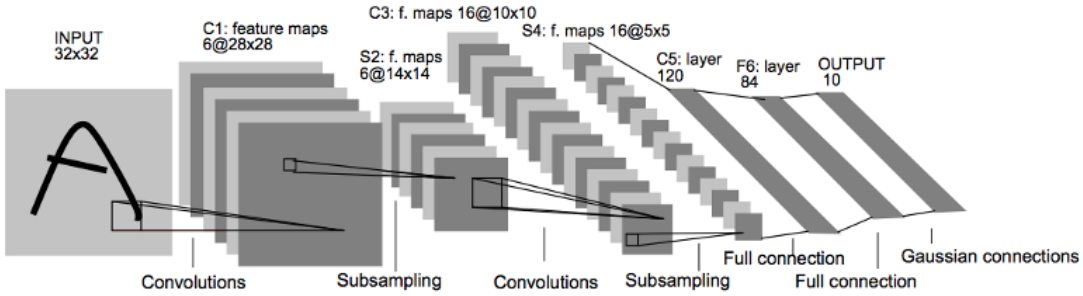
Is the dataset balanced? Would you prefer a balanced or unbalanced dataset? Explain why?

The dataset is balanced. Unbalanced datasets create bias during training, as the model will be accurate in predicting one class, but inaccurate in other classes with fewer of such images in the dataset. Using part of an unbalanced dataset as the test set would also output a false test accuracy (higher than expected), as the dataset is dominated by classes which the model is good at predicting. Hence, balanced datasets ensures that the model is equally trained to predict each class, and is much preferred.

1.4 2. Image classification. (55 marks)

1.4.1 2.1 Build a convolutional neural network using the `PyTorch` library to perform classification on the Fashion-MNIST dataset. (15 marks)

You can design a network architecture similar to LeNet (shown below), which consists a number of convolutional layers and a few fully connected layers at the end.



[6]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5),
            nn.BatchNorm2d(6),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2))
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=5),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2))
        self.fc1 = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256, 120),
            nn.ReLU())
        self.fc2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.ReLU())
        self.out = nn.Linear(84, 10)

    def forward(self, x):
        # Forward propagation
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.fc1(x)
        x = self.fc2(x)
        return self.out(x)

# Since most of you use laptops, you may use CPU for training.
# If you have a good GPU, you can set this to 'gpu'.
device = 'cuda'

# Network
model = Net().to(device)
print(model)
```

```
Net(
  (conv1): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
  )
  (conv2): Sequential(
    (0): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc1): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=256, out_features=120, bias=True)
    (2): ReLU()
  )
  (fc2): Sequential(
    (0): Linear(in_features=120, out_features=84, bias=True)
    (1): ReLU()
  )
  (out): Linear(in_features=84, out_features=10, bias=True)
)
```

1.4.2 2.2 Define the loss function, optimiser and hyper-parameters such as the learning rate, number of iterations, batch size etc. (5 marks)

[7]:
```
max_epochs = 10
batch_size = 128
lr = 0.001

loss_fn = nn.CrossEntropyLoss()
opt = optim.Adam(model.parameters(), lr=lr)
```

1.4.3 2.3 Start model training. (15 marks)

At each iteration, get a random batch of images and labels from train_image and train_label, convert them into torch tensors, feed into the network model and perform gradient descent.

Print out training loss and training time.

[8]:
```
def train(model, train_loader, loss_fn, opt, device):
    loss_data = 0
    model.train()

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)
        loss = loss_fn(outputs, labels)
        loss_data += loss.data
```

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

    return loss_data / len(train_loader)

train_loader = torch.utils.data.DataLoader(dataset=train_set,␣
↪batch_size=batch_size, shuffle=True)
training_start = time.time()

for i in range(max_epochs):
    start = time.time()
    loss = train(model, train_loader, loss_fn, opt, device)
    end = time.time()

    print("Iteration: {}, Time: {:.4f}s, Loss: {:.4f}".format(i+1, end-start,␣
↪loss))

training_end = time.time()
print("Total time: {:.4f}s".format(training_end-training_start))
```

```
Iteration: 1, Time: 5.9185s, Loss: 0.5416
Iteration: 2, Time: 6.0133s, Loss: 0.3583
Iteration: 3, Time: 5.5452s, Loss: 0.3201
Iteration: 4, Time: 5.9443s, Loss: 0.2947
Iteration: 5, Time: 5.5536s, Loss: 0.2767
Iteration: 6, Time: 5.6993s, Loss: 0.2615
Iteration: 7, Time: 5.8841s, Loss: 0.2502
Iteration: 8, Time: 5.9700s, Loss: 0.2392
Iteration: 9, Time: 6.1039s, Loss: 0.2268
Iteration: 10, Time: 5.8948s, Loss: 0.2199
Total time: 58.5321s
```

1.4.4 2.4 Deploy the trained model onto the test set. (10 marks)

Please also evaluate how long it takes for testing.

[9]:
```
def evaluate(test_loader):
    total = 0
    correct = 0

    pred_list = torch.tensor([]).to(device)
    label_list = torch.tensor([]).to(device)

    model.eval()
```

```
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)

        _, pred = torch.max(outputs.data, 1)
        total += len(labels)
        correct += (pred == labels).sum()

        pred_list = torch.cat((pred_list, outputs), dim=0)
        label_list = torch.cat((label_list, labels), dim=0)

    acc = 100 * correct / total

    return pred_list, label_list, acc

test_loader = torch.utils.data.DataLoader(dataset=test_set)

start = time.time()
pred_list, label_list, acc = evaluate(test_loader)
end = time.time()

print("Time: {:.4f}s".format(end-start))
```

```
Time: 11.1978s
```

1.4.5 2.5 Evaluate the classification accuracy on the test set. (5 marks)

[10]:
```
print("Classification accuracy: {:.2f}%".format(acc))
```

```
Classification accuracy: 89.02%
```

1.4.6 2.6 Print out and visualise the confusion matrix. (5 marks)

You can use relevant functions in scikit-learn.

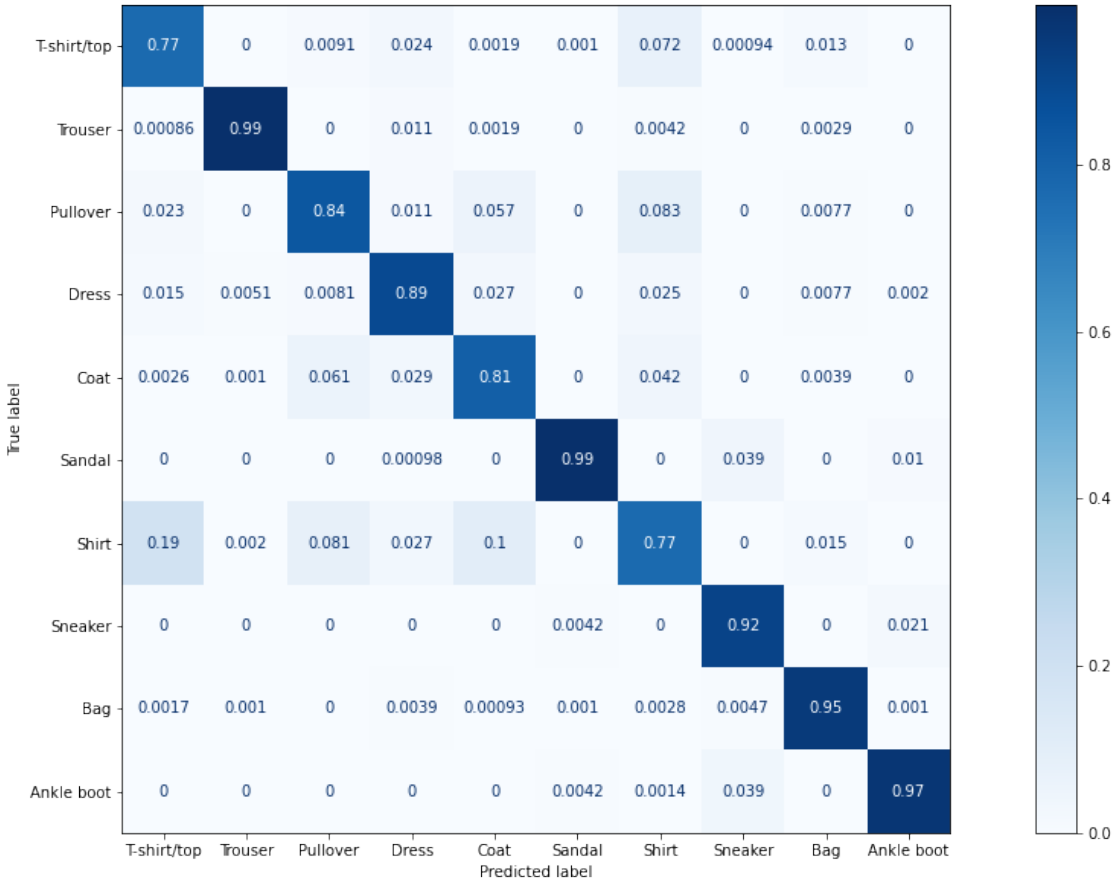
[11]:
```
cm = metrics.confusion_matrix(label_list.cpu(), pred_list.argmax(dim=1).cpu(),␣
↪normalize='pred')

disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm,
                                      display_labels=train_set.classes)

fig, ax1 = plt.subplots()
fig.set_size_inches((20, 10))

disp.plot(ax=ax1, cmap=plt.cm.Blues)
```

[11]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f3350536310>



1.5 3. Deploy in real world. (20 marks)

Take 3 photos that belongs to the 10 classes (e.g. clothes, shoes) in your real life. Use Python or any other software (Photoshop, Gimp etc) to convert the photos into grayscale, negate the intensities so that background becomes black or dark, crop the region of interest and reshape into the size of 28x28. You do not need to show the pre-processing step in this coursework.

1.5.1 3.1 Load and visualise your own images. (5 marks)

[12]:
```
transform = transforms.Compose([transforms.Grayscale(num_output_channels=1),
                                transforms.ToTensor()])

dataset = datasets.ImageFolder('/content', transform=transform)
data_loader = torch.utils.data.DataLoader(dataset=dataset,␣
 ↪batch_size=batch_size)
```

```
fig, axes = plt.subplots(1, 3, figsize=(15, 15))
images, labels = iter(data_loader).next()

for i, ax in enumerate(axes):
    ax.imshow(images[i].squeeze(), cmap='gray')
    ax.set_xticks([])
    ax.set_yticks([])
```

1.5.2 3.2 Test your network on the real images and display the classification results. (5 marks)

[13]:
```
pred_list, _, _ = evaluate(data_loader)

for p in pred_list.argmax(dim=1).cpu():
    print(train_set.classes[p])
```

```
Trouser
Shirt
Sneaker
```

1.5.3 3.3 Comment on the classification results. (10 marks)

Does the model work? Is there anyway to improve the real life performance of the model?

The model is quite accurate and managed to predict all 3 images correctly.However, the model is generally confused between similar classes such as T-shirt/top and Shirt. This can be attributed to the grayscaling of images and small image size, which cause certain subtle features such as design patterns which distinguish these classes to be lost.

Due to time constraints, only one architecture was experimented with. This may not be the best possible model, hence the model can definitely be improved by changing the overall architecture.

It can be noted that the images in the MNIST dataset look quite professionally taken. For the model to work better on real-life images where people normally snap with their phones, it needs to be further trained on a suitably sized dataset containing such images.

Another point to note is that the sneakers and sandals all point in the same direction. The model could not classify a shoe that was pointing in the opposite direction. Hence, the model can be improved either by adding such images to the dataset, so that the detection will not rely heavily on orientation.

1.6 4. Survey: How long does it take you to complete the coursework?

~8 hours