Coursework 2: Image Classification

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

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.

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.

In [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.transforms as transforms
```

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

In [2]:
```
# Load data (provided)
# Note that some early versions of torchvision use different names, such as train_da
# However, after torchvision version 0.4.0, data is used as the variable name.
train_set = torchvision.datasets.FashionMNIST(root='.', download=True, train=True)
train_image = np.array(train_set.data)
train_label = np.array(train_set.targets)
class_name = train_set.classes

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

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

In [3]:
```
print('training set: ', train_image.shape)
print('test set: ', test_image.shape)
```

```
training set:  (60000, 28, 28)
test set:  (10000, 28, 28)
```

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.

In [4]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np

def photoGrid(imgs, r, c):
    assert len(imgs) == r*c
    fig = plt.figure(figsize=(10.,10.))
    grid = ImageGrid(fig, 111, nrows_ncols=(r,c), axes_pad=0.1)
    for ax, img in zip(grid, imgs):
        ax.imshow(img)
        ax.set_axis_off()
    plt.show()
```

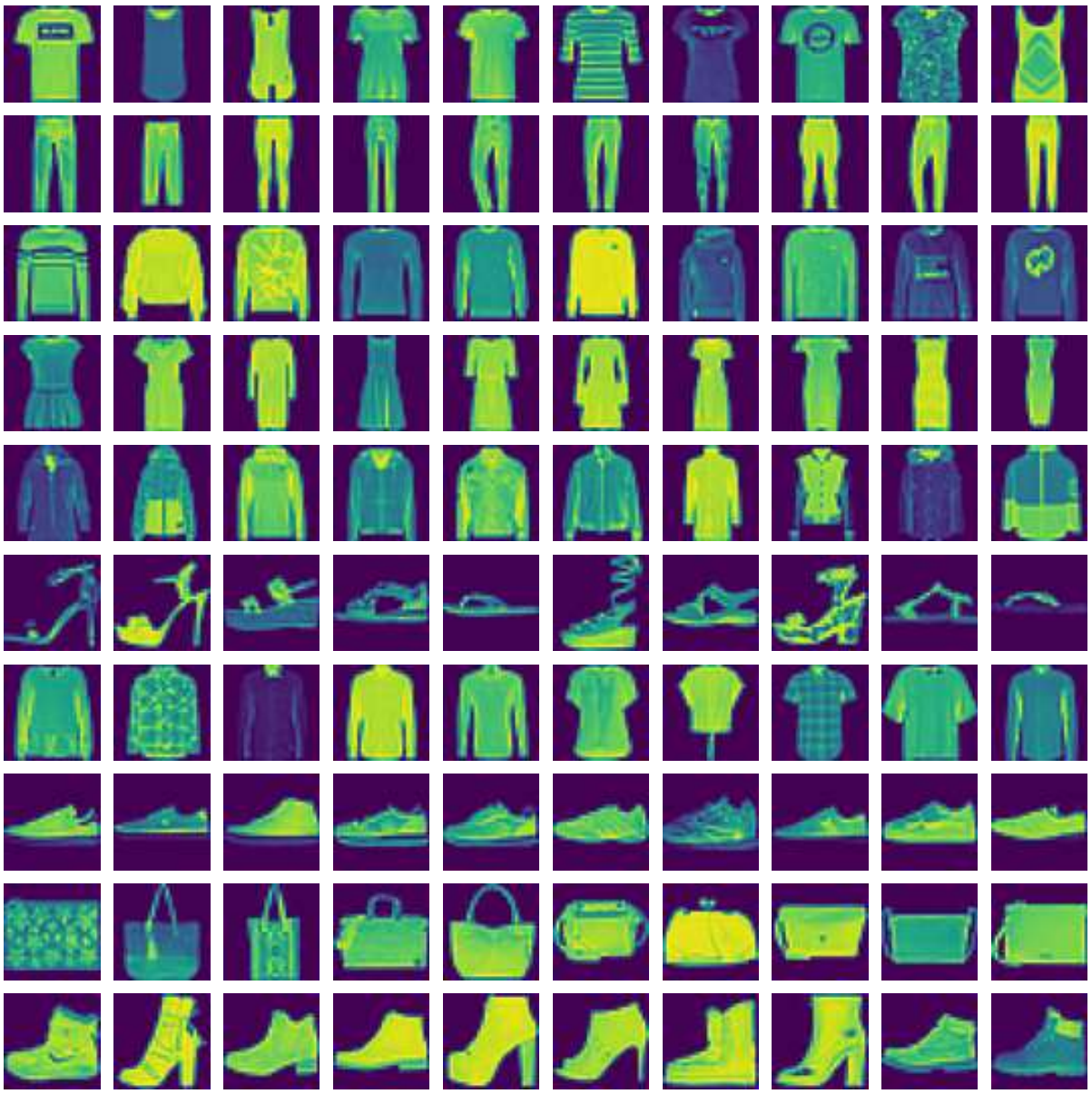
```
def collectImages(imgs, cat, num):
    imgs = np.zeros((0, imgs[0].shape[0], imgs[0].shape[1]))
    for img, label in zip(train_image, train_label):
        if imgs.shape[0] >= num:
            break
        if label == cat:
            imgs = np.append(imgs, [img], axis=0)
    return imgs
```

In [5]:

```
photos = np.zeros((100, 28, 28))

######## GET PHOTOS ##############
for cat in range(0, 10):
    i = cat*10
    photos[i:i+10,:,:] = collectImages(train_image, cat, 10)
##################################

photoGrid(photos, 10, 10)
```



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

In [10]:

```
for cat in range(0, 10):
```

```
    total = 0
    for label in train_label:
        if cat == label:
            total += 1
    print(class_name[cat], ': ', total)
```

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

1.4 Discussion. (10 marks)

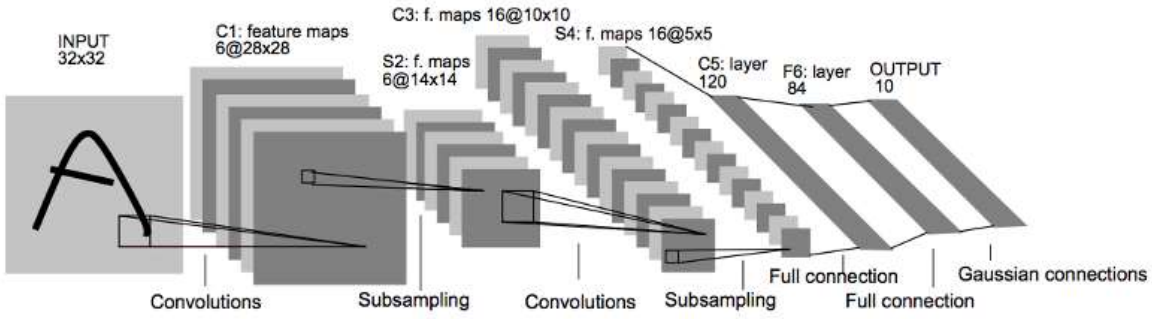
Is the dataset balanced? Would you prefer a balanced or unbalanced dataset? Explain why?

The dataset is perfectly balanced. Each category has exactly the same number of items: 6000. This is better for learning than an unbalanced set, as that would result in a bias towards whichever category has the most samples, as statistically any single sample is more likely to belong to that category.

2. Image classification. (55 marks)

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.



In [60]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 4 ** 2, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, 16 * 4 ** 2)
```

```
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return 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 = 'cpu'

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

```
Net(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=256, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

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

In [61]:

```
loss_fn = nn.CrossEntropyLoss()

lr = 0.01
batch_size = 10
epochs = 10
momentum = 0

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

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.

In [62]:

```
train_tensor_set = torchvision.datasets.FashionMNIST(root='.', download=True, train=
test_tensor_set = torchvision.datasets.FashionMNIST(root='.', download=True, train=F
```

In [63]:

```
train_loader = torch.utils.data.DataLoader(train_tensor_set, batch_size=batch_size,

start_time = time.time()
for epoch in range(epochs):
    for i, data in enumerate(train_loader):
        inputs, labels = data
        optimiser.zero_grad()
        outputs = model(inputs)
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimiser.step()

time_taken = time.time() - start_time
print(f'Training finished in {time_taken}s')
```

```
Training finished in 185.57133269309998s
```

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

Please also evaluate how long it takes for testing.

In [64]:
```
test_loader = torch.utils.data.DataLoader(test_tensor_set, batch_size=batch_size, sh

correct = 0
total = 0

labels_tested = []
predictions = []

start_time = time.time()

with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = model(images)
        _, predicted_labels = torch.max(outputs.data, 1)

        labels_tested += labels
        predictions += predicted_labels

        total += labels.size(0)
        correct += (predicted_labels == labels).sum()

time_taken = time.time() - start_time
print(f'Testing finished in {time_taken}s')
```

```
Testing finished in 2.1402740478515625s
```

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

In [65]:
```
print(f'Accuracy: {correct / total * 100}%')
```

```
Accuracy: 88.63999938964844%
```

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

You can use relevant functions in scikit-learn.

In [66]:
```
print(metrics.confusion_matrix(labels_tested, predictions))
```

```
[[841   1  10  21   5   4 110   0   8   0]
 [  5 974   0  15   2   0   3   0   1   0]
 [ 20   1 773  12  94   0 100   0   0   0]
 [ 26   7   8 903  26   0  25   0   4   1]
 [  2   1  45  33 844   1  68   0   6   0]
 [  0   0   0   0   0 965   0  15   0  20]
 [154   1  49  24  77   0 680   0  15   0]
 [  0   0   0   0   0  20   0 946   0  34]
 [  2   1   3   3   1   2   3   5 980   0]
 [  0   0   0   0   0  10   0  32   0 958]]
```

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.

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

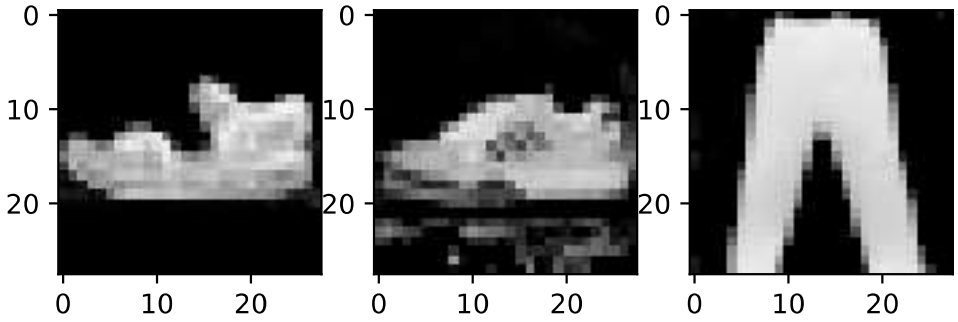
In [89]:
```
import imageio

sandal = imageio.imread('sandal_small.jpg')
plt.subplot(131)
plt.imshow(sandal)

sneaker = imageio.imread('sneaker_small.jpg')
plt.subplot(132)
plt.imshow(sneaker)

trousers = imageio.imread('trousers_small.jpg')
plt.subplot(133)
plt.imshow(trousers)
```

Out[89]:
```
<matplotlib.image.AxesImage at 0x16a321a2910>
```



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

In [91]:
```
def grayscale_norm(img):
    out = []
    for row in img:
        gr = []
        for px in row:
            gr.append(px[0] / 255)
        out.append(gr)
    return out

imgs = [grayscale_norm(x) for x in [sandal, sneaker, trousers]]
inputs = torch.from_numpy(np.expand_dims(imgs, axis=1)).type(torch.float32)
outputs = model(inputs)
_, predictions = torch.max(outputs.data, 1)

np_predictions = predictions.cpu().numpy()
print("\n")
print(f'label: Sandal   - predicted label: {class_name[np_predictions[0]]}')
print(f'label: Sneaker  - predicted label: {class_name[np_predictions[1]]}')
print(f'label: Trousers - predicted label: {class_name[np_predictions[2]]}')
```

```
label: Sandal   - predicted label: Sandal
label: Sneaker  - predicted label: Sneaker
label: Trousers - predicted label: Trouser
```

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 seems quite affective, as it showed an accuracy on the test data over 88% consistently and was able to correctly classify the real world images it was given. The model accuracy could probably be increased. This could be achieved by a more effective hyperparameter search, or increasing the training data available.

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

In [ ]:
```
2 days
```