School of Engineering and Physical Sciences

Electrical Electronic and Computer Engineering

B31SE_2019-2020

ASSIGNMENT 2: Fashion-MNIST DATA CLASSIFICATION

Fashion-MNIST is a dataset of Zalando's article images and is licensed under The MIT License (MIT) Copyright © [2017] Zalando SE, https://tech.zalando.com. The complete fashion-MNIST dataset consists of 60,000 training and 10,000 test examples. In this assignment, we will use a subset of the data to simplify the training process (with a small compromise in the accuracy of the model). Each example is a 28x28 grayscale image, associated with a label from one of the 10 classes:

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

The goal of this assignment is to a) implement a deep artificial neural network, b) train it and c) perform the classification on the testing data (prediction) according to the following guidelines:

1. **DATA** available at: https://www.kaggle.com/zalando-research/fashionmnist/data# .
2. **Convert the DATA to** png format in `MATLAB`. Use scripts `train_data2png.m` and `test_data2png.m` (you can find them in the assignment's folder), which call the functions `loadMNISTImages.m` and `loadMNISTLabels.m`, to create separate folders with the training and testing data, respectively. The created folders 'TRAINING' and 'TESTING' contain 2,000 and 400 examples, respectively. Note that the files `Training_labels.mat` and `Testing_labels.mat` are also generated and contain the data labels in the 'TRAINING' and 'TESTING' folders, respectively.
3. **Manage the DATA**: use `MATLAB`'s `imageDatastore` to define the `imdsTrain` and `imdsTest` data. Provide the labels to the image data store manually as categorical variables.
4. **DATA inspection**: display the 1st sample from the training data and save the 28x28 image as 'train1_yourname.png'). Use `MATLAB`'s `montage` command to save a random collection of 100 training examples and save the figure as 'train_montage_rand100_yourname.png').
5. **Indicative Neural Network architecture**: the first layer (input layer) is the `imageInputLayer`, whereas the last layer (output layers) should be a `fullyConnectedLayer` of dimension equal to the number of classes followed by a `sofmaxLayer` and a `classificationLayer`. For the intermediate layers define two `convolution2dLayer` (convolution layers). Each convolution layer should have 64 filters of 5x5 kernels with the 'Padding' variable set to 'same'. After each `convolution2dLayer` include the following sequence of layers:
   - normalize the data by adding a `batchNormalizationLayer`
   - apply a non-linearity using the `reluLayer`
   - perform down-sampling with a `maxPooling2dLayer` using pool size [2 2] and stride [2 2]

- set input elements to zero at random with probability *p* using `dropoutLayer` (optional)

Next (and before the output layers) define a `fullyConnectedLayer` of dimension 1024 followed by a `reluLayer` and a `dropoutLayer` with probability *p* (optional). Add another one of dimension 512 and include a `reluLayer` and a `dropoutLayer` with probability *p* (optional). Finally, add a third `fullyConnectedLayer` of dimension 128 followed by a `reluLayer`. More details about the syntax of each layer can be found in https://uk.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html.

6. **Split the DATA into Train and Validation**: 80% of the `imdsTrain` data will be used for the training and 20% for the validation. Use `splitEachLabel` to perform this efficiently. Name the validation datastore as `imdsValidation.` Include the `rng('default')` command at the beginning of your script.
7. **Train the network**: use `options` to define the set of training parameters. The following parameters will be fixed to:
   - Shuffle: once
   - MaxEpochs: 15
   - ValidationFrequency: 10
   - ValidationData: `imdsValidation`

   You are free to set the following parameters to optimize the performance of your neural network:

   - MiniBatchSize
   - Optimizer
   - InitialLearnRate

   The rest of the parameters not mentioned here are simply set to their default values. Next, train the network using the `trainNetwork` command, the training data and the parameters defined in `options`. Store the variables of the network (weights and biases) in the variable `net`. You can plot the training progress, which looks like Fig. 1.
8. **Predictions**: make a prediction (`predict`) using the `imdsTest` data and the weights/biases of the trained network from 7.
9. **Results**: plot the confusion matrix, as depicted in Fig. 2, and calculate the accuracy of your model. The accuracy on the testing for our model is **82.5%.**

The training progress as well as the confusion matrix should be included in your report.

Additional Notes:

- You cannot change the training and testing data sets (use the ones generated).
- Data augmentation techniques can be used.
- No other parameters can be modified.
- Use the indicative architecture as a starting point. If you wish, you can modify it by adding or removing layers.

- The ultimate goal is not only to achieve the highest testing accuracy (or lowest loss), but also enable actual learning of the network and not overfitting. Hence, in the proposed model the training accuracy should be similar to the validation/testing accuracy (or loss, respectively).

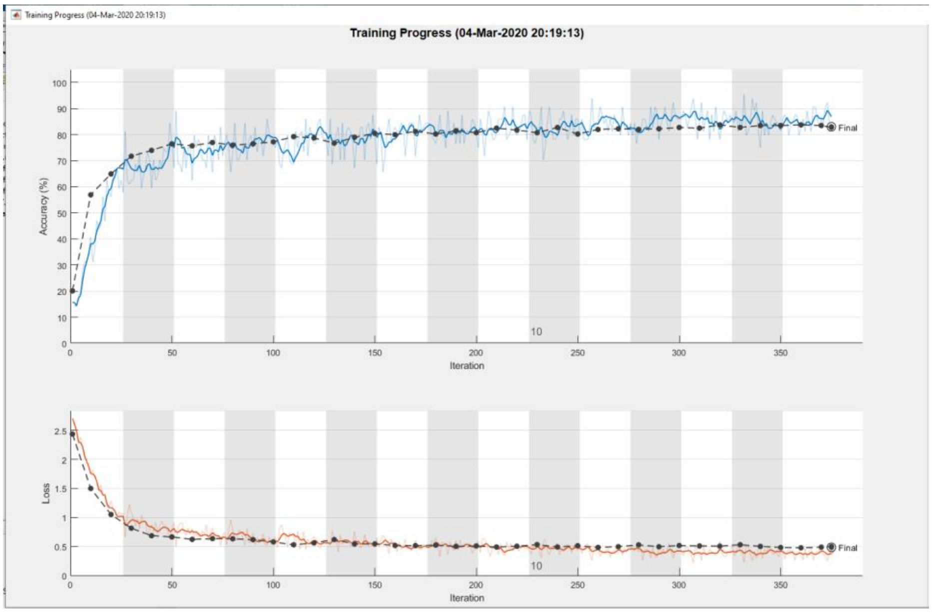

Figure 1*: The training progress. Accuracy (top) and loss (bottom) vs the number of iterations.*

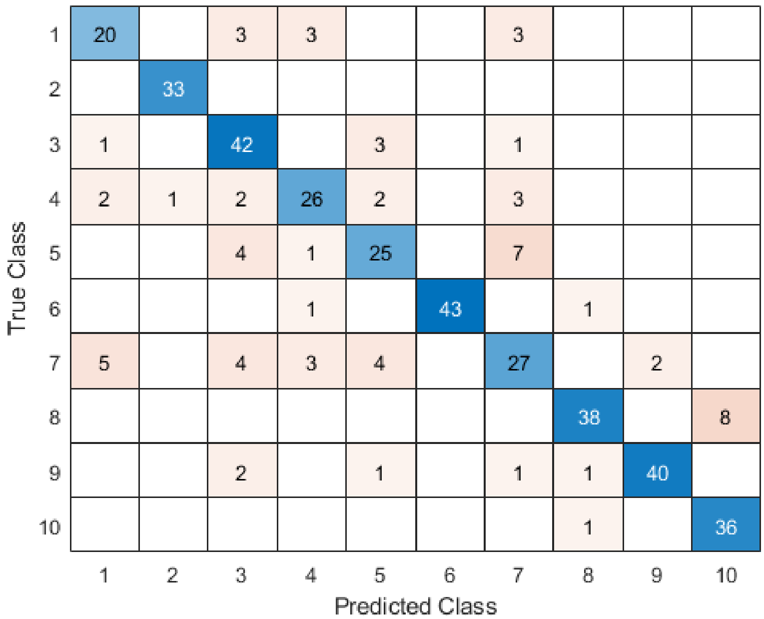

Figure 2*: The confusion matrix in the evaluation of the model in the classification of 400 examples from the fashion-MNIST data.*