Fasion-MNIST classification

In this Assignment you are going to classify a fashion dataset.

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Imad Hanzaz, Yannick Urselmann, Jaylong Verschuren

Initialization

load all needed libraries and functions, check the previos tutorial how to correctly load keras and other modules

Import the needed libraries for this assignment.

```
Current Tensorflow version used is: 2.11.0
```

Functions used/created in this assignment

Create a variable to trigger training of the model or not.

This is done because the whole notebook can be run at once. If a model is trained already, it would be time consuming to create another model.

Load dataset & Plot a subset

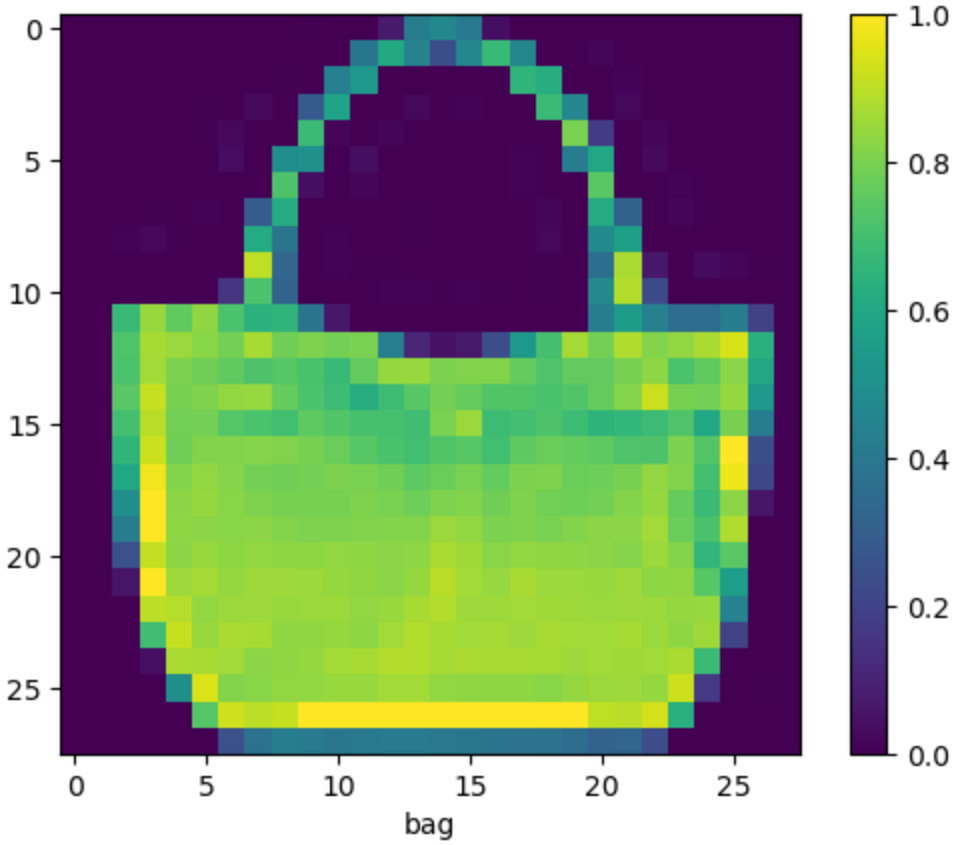
load your dataset and show a plot of the subset of your data

Load the Fashion MNIST dataset and divide it into train and test datasets.

```
(60000, 28, 28)
```

Prepare Data

pre-process your raw input data... rescale... normalize....



Rescale the images by dividing through 255

Plot some of the train images with their classes

Fit the Model

Fitting the model is the time consuming part, this depend on the complexity of the model and the amount of training data. In the fitting process the model is first build up in memory with all the tunable parameters and intercomnnects (with random start values). This is also the limitation of some systems, all these parameters are stored in memory (or when not fitting in a swap file)

TIP: do not start the first time with training a lot of epochs, first see if this and all following steps in your system work and when you are sure that all works train your final model. You can also monitor the Jetson CPU/GPU/Memory performance during this process (see Tips & Tricks)

- Explain what hyperparameters are available and what they do.
- Which hyperparameter result in better training results?

Made a CNN using 3 convolution layers and 2 pooling layers for the feature distraction. In the flatten layer and dense layers, the classification part of the model is made.

Hyperparameter

For hyperparameter there are the layers such as convolution, pooling, flatten and dense which are available.

Convolution

The kernel size is used to specify the width and height of the grid to convolute.

Pooling

The pooling layer is used to downsample with a certain pool size.

Flatten

Flatten is used to create a vector of the input shape to feed to the Neural Network.

Compiled the model

The model is only fitted when the parameter for TrainModel is true to avoid training when not needed.

```
Epoch 1/4
60000/60000 [==============================] - 88s 1ms/step - loss: 0.4480 - accuracy: 0.8380 - va
l_loss: 0.3863 - val_accuracy: 0.8682
Epoch 2/4
60000/60000 [==============================] - 84s 1ms/step - loss: 0.3757 - accuracy: 0.8674 - va
l_loss: 0.4292 - val_accuracy: 0.8697
Epoch 3/4
60000/60000 [==============================] - 89s 1ms/step - loss: 0.3720 - accuracy: 0.8706 - va
l_loss: 0.4088 - val_accuracy: 0.8511
Epoch 4/4
60000/60000 [==============================] - 95s 2ms/step - loss: 0.3715 - accuracy: 0.8717 - va
l_loss: 0.4767 - val_accuracy: 0.8607
```

Evaluate Model

Show the model accuracy after the training process ...

- What is the final accuracy of the trained Network?

```
10000/10000 [==============================] - 8s 826us/step - loss: 0.4767 - accuracy: 0.8607
Validated loss: 0.4766572117805481 , Validated Accuracy: 0.8607000112533569
```

The final validated accuracy is 86%, this is using the own test set.

Save model

Save the model for later usage

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolut
ion_op, _jit_compiled_convolution_op, _update_step_xla while saving (showing 4 of 4). These functi
ons will not be directly callable after loading.
INFO:tensorflow:Assets written to: saved_models/model7\assets
INFO:tensorflow:Assets written to: saved_models/model7\assets
```

Evaluate Final Model

After training and saving the model you can deploy this model on any given input image. You can start a new application in where you import this model and apply it on any given imput images, so you can just load the model and don't need the timeconsuming training anymore.

Loading the model into a test_model

Make Prediction

We can use our saved model to make a prediction on new images that are not trained on... make sure the input images receive the same pre-processing as the images you trained on.

So fetch some images from the internet (similar classes, but not from your dataset), prepare them to fit your network and classify them. Do this for **10 images per class** and show the results!

- How good is the detection on you real dataset? (show some statistics)

```
There are: 100 files
```

The data is loaded a prediction dataset which is used to predict on new data

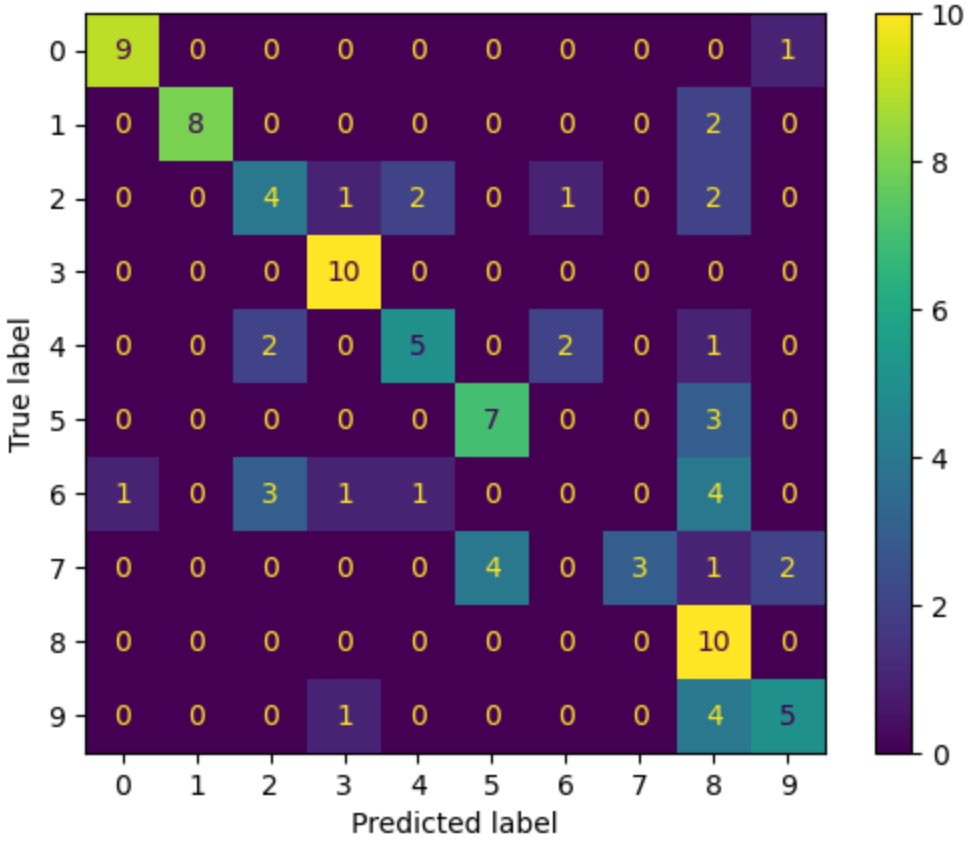
```
Found 100 files belonging to 10 classes.
['tshirt_top', 'trouser', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle
_boot']
```

Iterate through prediction_ds to gather images and labels. After that the data's grayscale is inverted and reshaped. At last, a numpy array for the images and labels are created.

The data is predicted and after that a plot is created where blue is correct predicted and red is false predicted.

Then a confusion matrix is made to see, unfortunately label 6 is not identified.

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19597c00610>
```



Then the accuracy with the predicted set is measured. This is 61% which is not bad for own data which is not as good preprocessed as the dataset.

```
100/100 [==============================] - 0s 819us/step - loss: 2.9123 - accuracy: 0.6100
Validated loss: 2.912320852279663 , Validated Accuracy: 0.6100000143051147
```

Additional Questions

- In which way could the network accuracy be improved further (only explanation, no implementation)?

1. Fine-tune Hyperparameter (Learning Rate, batch size and number of epochs.)
2. A different combination of layers (dropout, conv2D) to the network. This could result in better performance/accuracy.
3. Preprocess the data the same as they did with the CIFAR-10 dataset. They used gaussian processes and other difficult to execute tasks.