ECS659U/P Coursework

Philip Wurzner
210930632

Description of the Script

1.

First all the modules are imported. These modules are the pytorch and my_utils (obtained from d2l and modified)

2.

The script has all the hyperparameters defined at the top. The hyperparameters are:

- batch_size
  - This is the data batch size for the data iterator
- lr
  - this is the learning rate for the optimisation algorithm. Specifically, the size of each update step in the linear regression.
- num_epochs
  - this is the amount of training cycles (iterations through the dataset) that are preformed in training the model.
- input_dim
  - this is resolution of the input data (28 by 28)
- output_dim
  - this is the amount of potential classifications/required output dimensions (10)
- patch_size
  - this is the size of the patch $(K * K)$ taken of the input model, as required by the coursework description
- N_blocks
  - This is the number of blocks N present in the backbone of the model, as required by the coursework
- Hidden_dim
  - This is how many hidden dimensions are present out of the stem of the data pipeline
- Mlp_dim
  - This is the size of the linear layers in the MLPs

3.

The next section defines the stem of the model. Particularly, the modules that will be run at the start of the nn.sequencial are defined here. The stem features a single layer non-overlapping convolutional layer is created as instructed on the coursework specification. This convolution layer interprets the input image in patches (KxK), as defined by the patch_size hyperparameter. To ensure that none of the patches are overlapping, the kernel size and stride are set equal. Additionally, to ensure that no overlapping occurs, the patches must be a factor of the input resolution (28). Once patches have been taken the data is flattened/vectorised for the backbone process and transposed.

4.

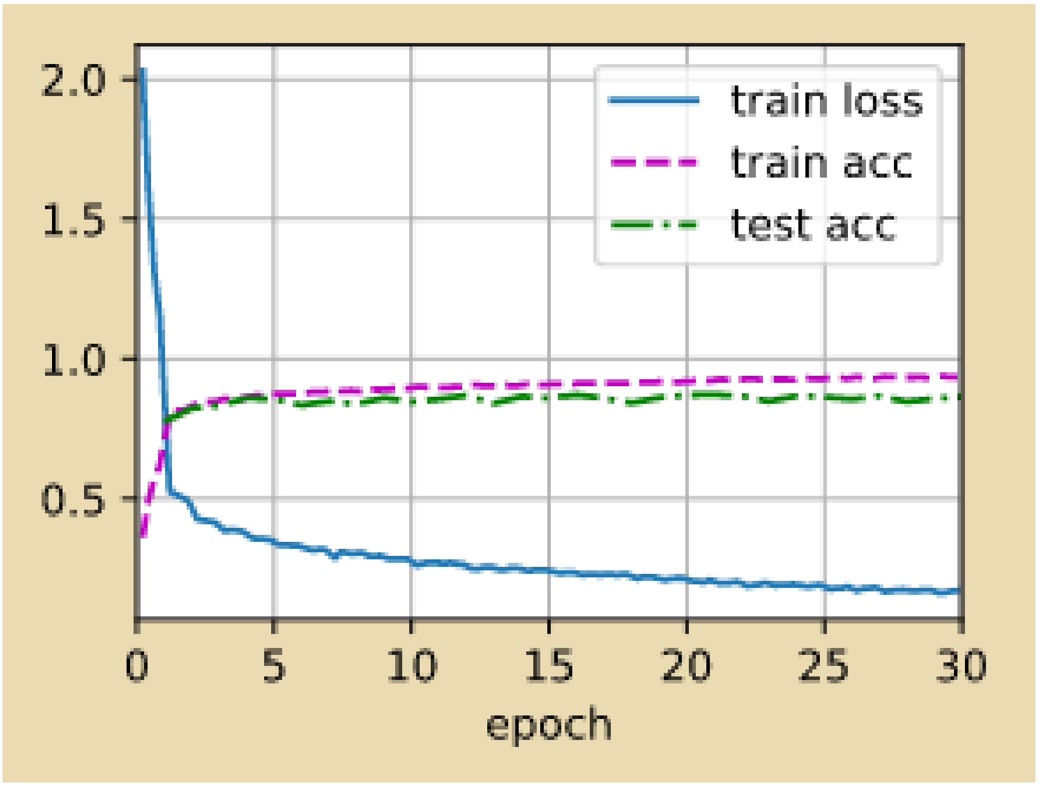
The backbone works by appending the MLP layers into a modules array. For every N_block, the following modules are appended:

- Normalisation layer.
- The data is then transposed and fed into a Linear - activation - Linear MLP.
- The data is transposed again, and the transposed data is fed into another linear - activation - linear MLP
- On the final block the results are averaged/vectorised and appropriately parsed into 10 output variables
  for the classifier

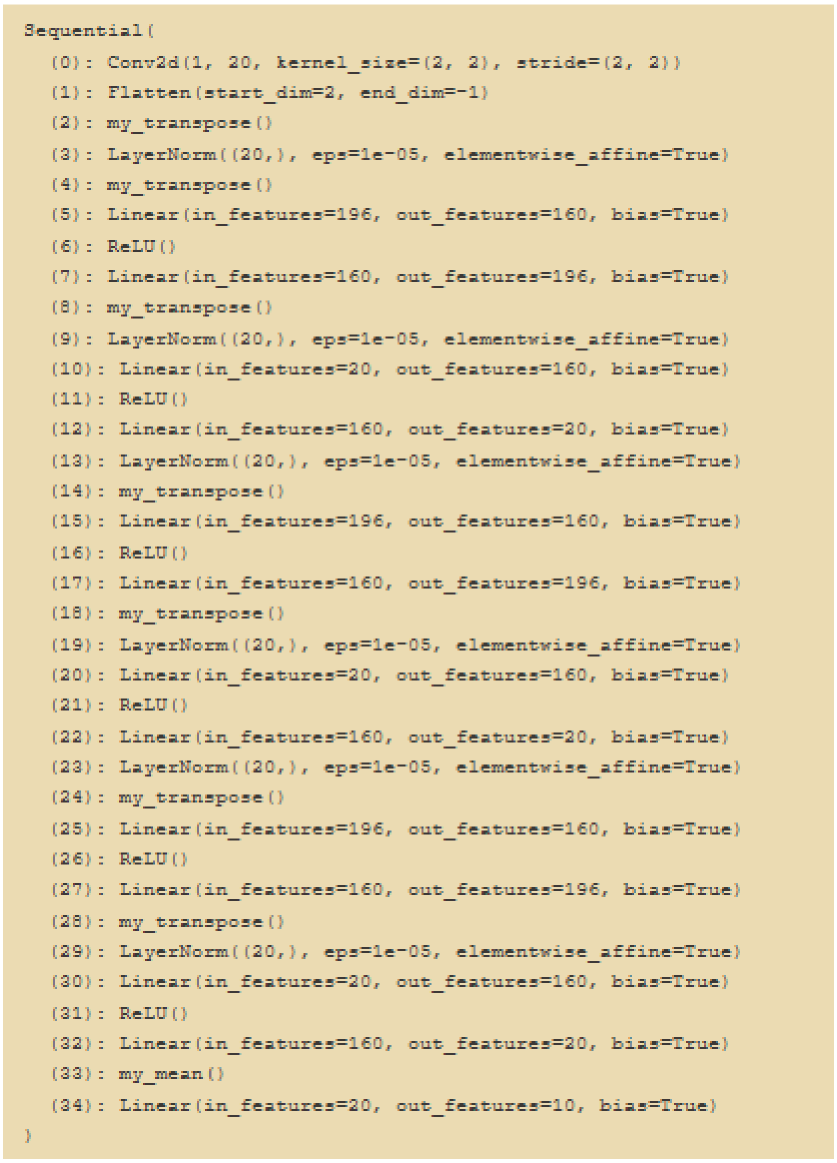
5. The training script from my_utils was used to process the data. This training script uses a stochastic gradient descent optimiser and a cross entropy loss function. The loss and optimiser functions are then used in conjunction with a softmax regression to classify the results. The following hyperparameters were used to obtain the following results.

   Batch size = 256
   Learning rate = 0.05
   Num_epochs = 30
   Patch_size = 4
   N_blocks = 3
   Hidden_dim = 20
   MLP_dim = 160

   Train Loss = 0.173
   Train Accuracy = 0.934
   Test Accuracy = 0.865



The full machine learning sequence is:



```
Sequential(
  (0): Conv2d(1, 20, kernel_size=(2, 2), stride=(2, 2))
  (1): Flatten(start_dim=2, end_dim=-1)
  (2): my_transpose()
  (3): LayerNorm((20,), eps=1e-05, elementwise_affine=True)
  (4): my_transpose()
  (5): Linear(in_features=196, out_features=160, bias=True)
  (6): ReLU()
  (7): Linear(in_features=160, out_features=196, bias=True)
  (8): my_transpose()
  (9): LayerNorm((20,), eps=1e-05, elementwise_affine=True)
  (10): Linear(in_features=20, out_features=160, bias=True)
  (11): ReLU()
  (12): Linear(in_features=160, out_features=20, bias=True)
  (13): LayerNorm((20,), eps=1e-05, elementwise_affine=True)
  (14): my_transpose()
  (15): Linear(in_features=196, out_features=160, bias=True)
  (16): ReLU()
  (17): Linear(in_features=160, out_features=196, bias=True)
  (18): my_transpose()
  (19): LayerNorm((20,), eps=1e-05, elementwise_affine=True)
  (20): Linear(in_features=20, out_features=160, bias=True)
  (21): ReLU()
  (22): Linear(in_features=160, out_features=20, bias=True)
  (23): LayerNorm((20,), eps=1e-05, elementwise_affine=True)
  (24): my_transpose()
  (25): Linear(in_features=196, out_features=160, bias=True)
  (26): ReLU()
  (27): Linear(in_features=160, out_features=196, bias=True)
  (28): my_transpose()
  (29): LayerNorm((20,), eps=1e-05, elementwise_affine=True)
  (30): Linear(in_features=20, out_features=160, bias=True)
  (31): ReLU()
  (32): Linear(in_features=160, out_features=20, bias=True)
  (33): my_mean()
  (34): Linear(in_features=20, out_features=10, bias=True)
)
```