CIFAR-10 Lotus Model

The lotus model was created as part of my Master's Data Science and Artificial Intelligence.
The specification of the model was given to ensure the model was original, so the challenge was to achieve the highest possible accuracy. The following PDF gives an overview of the project and outcomes.

Task 1: Read the dataset and create data loaders

The data loader uses four processes to load the training and validation sets. The training set has multiple transformations applied to it as the data loaders are created. These are:

RandomResizedCrop(size=56, scale=(0.75, 1.0)) , RandomAffine(3) , RandomHorizontalFlip(0.5) , RandomVerticalFlip(0.1) , RandomRotation(45) , RandomPerspective(0.15) , RandomGrayscale(p=0.05) , ColorJitter(brightness=0.25, contrast=0.25, saturation=0.25, hue=0.15,)

Both the training and validation sets have ToTensor() and normalisation applied
(normalisation values are from https://d2l.ai/chapter_computer-vision/kaggle-cifar10.html)
The function also takes an optional resize parameter, which is used in the notebook to resize both sets to 52x52.

Task 2: Create the model

Please see the notebook for a description of the experiments run to develop the final model. **Table 1** (below) gives some developmental milestones which have informed the final model and its accuracy

Brief model description (bold underline shows the main modificiations)	**Accuracy**
3X Blocks of 5X stacked convolutional layers (3x3 kernel) + MLP(3 layers)	75%
3X Blocks of 5X stacked **2X**convolutional layers (3x3 kernel (5x5 receptive field)) + MLP(3 layers)	80%
3X Blocks of 5X stacked **3X**convolutional layers (3x3 kernel (7x7 receptive field)) + MLP(3 layers)	85%
Transformations of the training set + 3X Blocks of 5X stacked 3Xconvolutional layers (3x3 kernel (7x7 receptive field)) + MLP(**6 layers**)	93%
Heavy Transformations of the training set + 3X Blocks of 5X stacked 3Xconvolutional layers (3x3 kernel (7x7 receptive field)) + MLP(**8 layers**)	94.99%
Heavy Transformations of the training set + **4X Blocks** of **10X stacked** 3Xconvolutional layers (3x3 kernel (7x7 receptive field)) + MLP(6 layers)	95.12%

The Lotus model consists of a block repeating 4 times and a multilayer perception with 6 layers.

Channel changes per block
Block 1: 3 channels in 96 channels out
Block 2: 96 channels in 256 channels out
Block 3: 256 channels in 512 channels out
Black 4: 512 channels in 1024 out

Feature map image size 52x52
Block 1: reduces the feature map to 28x28
Block 2: reduces the feature map to 14x14
Block 3: reduces the feature map to 7x7 Block 4: reduces the feature map to 3x3

The repeating block uses the following operations.
Batch **X** is processed by two separate processes:

Process 1: **X** is passed through AdaptiveAvgPool to produce 1 value per channel. This is then passed through a nn.linear model which outputs the same number of features out needed by process 2, nn.ReLu is used for a nonlinear transformation. The output of this process is called **vector A**.

Process 2: **X** is passed into K number of convolutional layers. Each layer is separate and passes X through a 3x3 conv kernel, a ReLU, a second 3x3 conv kernel, a second relu, a third 3x3 conv kernel, a third relu. Process 2 stacks the K number of outputs together into **Stacked K**.

Vector A and Stacked K Linear combination:
Stacked K is then flattened (start_dim=2) and combined with **vector A** using torch.matmul(). The output is processed by nn.batchNorm2D and NN.MaxPool2D
In the final model, the block **implements n=10 stacked convolutional outputs.**

After the 4th block has run, the model uses a 6-layer MLP to classify the images:

- The classifier starts with an AdaptiveAvgPool2d, which summarises the final feature map into a single value for each channel (in line with the specification).
- Each layer in the classifier has 4096 perceptrons with the exception of the input (n=1024) and the output (n=10) layers.
- Each layer has a ReLU transformation and a dropout(p=0.5) layer.
- SoftMax Regression is achieved by using the CrossEntropyLoss function

Task 3: Create the loss and optimiser

The loss function used is CrossEntropyLoss() this allows for a softmax regression to be used.
The optimiser used is torch.optim.SGD() to maximise the final accuracy.

Task 4: Write the training script, including the curves for the evolution of loss, train accuracy, test accuracy, and all the training details, including the hyperparameters used

```
epoch:  349
final train_acc:  0.98434
final test_acc:  0.9517
final loss:  0.05018741971943527
```

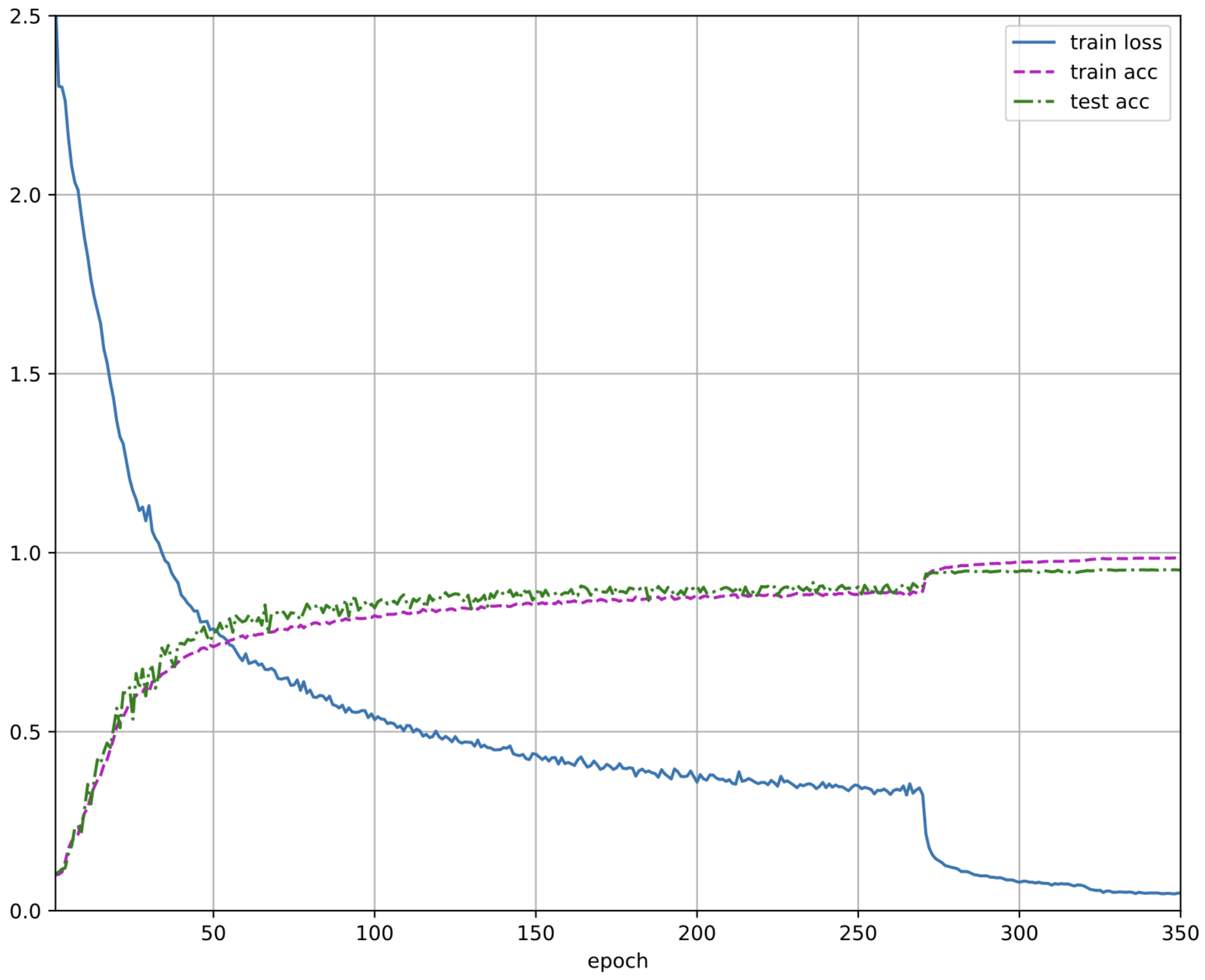


Figure 1: Shows the curves for the evolution of the loss, train accuracy and test accuracy.

The **Lotus model** (described in task 2) is set with the parameters:

- structure = ((10,96), (10, 256), (10,512), (10, 1024))
- batch_size = 192
- weights are initialised with torch.nn.init.xavier_uniform_() for linear and conv2d layers

The **optimiser** is torch.optim.SGD() Stochastic Gradient Decent
The optimisers parameters include:

- learning rate = 0.01 (starting value)
- momentum = 0.95
- weight_decay = 0.001
- Nesterov = False

I have used torch.optim.lr_scheduler.MultiStepLR() **scheduler** to adjust the learning rate during training. This changes the learning rate by a factor (gamma) when certain milestone epochs are reached.
The scheduler's parameters include:

- optimiser (listed above)
- milestones = [270, 320, 340]
- Gamma = 0.1

The training runs for **350 epochs** in total.
This results in a **variable learning rate** for the model training:

0.01	epochs 0-269
0.001	epochs 270-319
0.0001	epochs 320-339
0.00001	epochs 340-350

The function which manages the training of the model using these parameters is called **train_model_on_device()**. This function takes the following parameters:

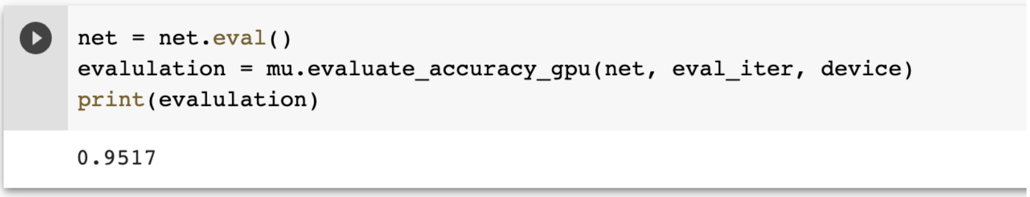
- Model, training_dataloader, validation_dataloader, loss_function, num_epochs, optimiser, scheduler, device)

Task 5: The final model accuracy in the CIFAR-10 Validation set

The final model has an accuracy of 95.17%.

Please see the notebook for the code which:

- Loads the entire validation set in a new dataloader
- Applies the same normalisation as used during training (taken from d2l.ai chapter)
- Resizes the images to 52x52 to match the models requirements
- Sets the model.eval()
- Produces the final model accuracy of 95.17%



```
net = net.eval()
evalulation = mu.evaluate_accuracy_gpu(net, eval_iter, device)
print(evalulation)
```

```
0.9517
```