1 Read dataset and create data loaders

```
train_data = torchvision.datasets.CIFAR10(root='./data',
                                          train=True,
                                          download=True,
                                          transform=train_transform)
train_loader = DataLoader(train_data,
                          batch_size=args.batch_size,
                          shuffle=True)
test_data = torchvision.datasets.CIFAR10(root='./data',
                                         train=False,
                                         download=True,
                                         transform=test_transform)
test_loader = DataLoader(test_data,
                         batch_size=args.batch_size,
                         shuffle=False)
```

The code above reads the training and testing data, shuffles it, divides it into chosen batch sizes, and applies selected transforms. The data is shuffled and the transforms are applied every epoch so the model doesn't just memorise the transformed images, leading to overfitting.

2 Create the Model

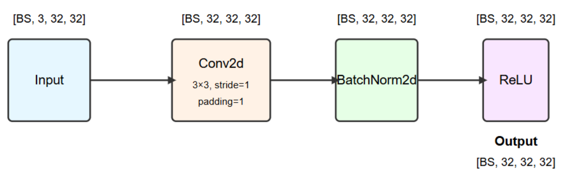


Figure 1: The stem

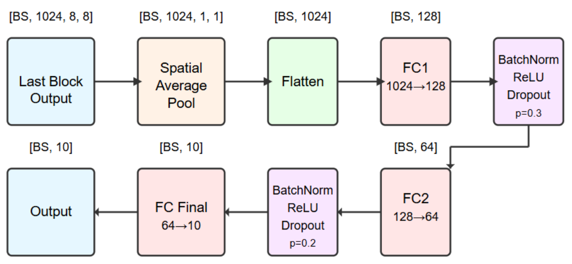


Figure 2: The classifier

For the stem and classifier, the final implementation can be seen in **Figure 1** and **Figure 2**. In terms of the block architecture, it stayed the same as on the slides, but I introduced skip connections and added a bottleneck layer before the convolutions (using its output as input to the K convolutions) as shown by (He et al. 2016). Additionally, I added batch norm and ReLU after each block. Each block increased the number of channels, here are the outputs of each [64, 128, 256, 512, 768, 1024].

3 Create the loss and optimiser

In the final version, for loss, I used Focal Loss (Lin et al. 2017) and I manually adjusted higher loss for more "difficult classes" by a certain factor (**specified in Figure 3**. For optimisation, while starting with ADAM, I ended up using SGD with Nesterov Momentum (converges better) and applied the OneCycleLR learning rate scheduler.

4 Write the training script to train the model

Table 1: Training Hyperparameters

Parameter	**Value**	**Parameter**	**Value**
Training Configuration		**Data Augmentation**	
epochs	200	color_jitter	0.15 (0.15/2 for hue)
batch_size	128	random_horizontal_flip	0.3
optimizer	SGD	random_rotation	10
patience_number	20	cutmix_alpha	1.0
		mixup_alpha	0.2
Optimizer Settings		num_blocks	6
learning_rate	0.01		
weight_decay	0.0005	**Loss Function**	
momentum	0.9	focal_loss_gamma	2.0
dropout	0.3	label_smoothing	0.1
OneCycle LR Schedule		**Class Additional Loss Factor**	
max_lr	0.05	birds	1.1
pct_start	0.15	cats	1.2
div_factor	25	deers	1.05
final_div_factor	10000	dogs	1.1
		other classes	1.0
		Model Architecture	
		num_k	4
		reduction_factor	4
		num_blocks	6

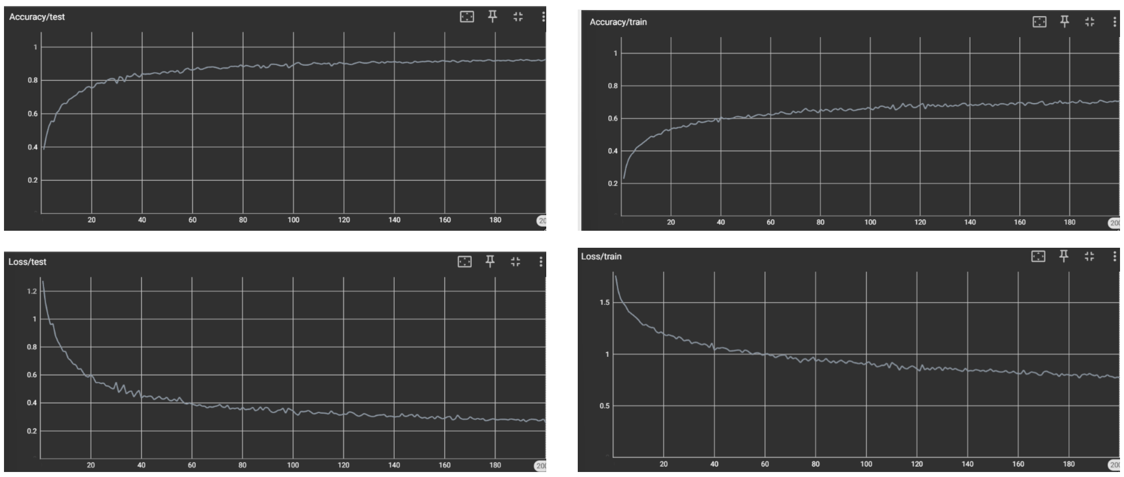
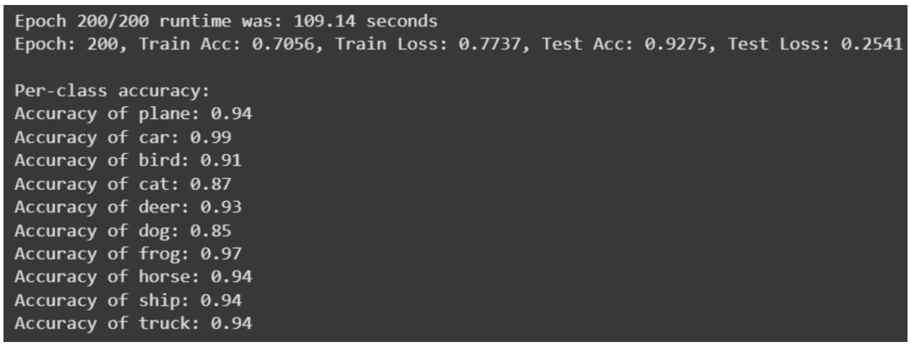


Figure 3: Testing accuracy (top-left), Training accuracy (top-right), Testing loss (bottom-left), Training loss (bottom-right)

5 Final model accuracy on CIFAR10 validation set



```
Epoch 200/200 runtime was: 109.14 seconds
Epoch: 200, Train Acc: 0.7056, Train Loss: 0.7737, Test Acc: 0.9275, Test Loss: 0.2541

Per-class accuracy:
Accuracy of plane: 0.94
Accuracy of car: 0.99
Accuracy of bird: 0.91
Accuracy of cat: 0.87
Accuracy of deer: 0.93
Accuracy of dog: 0.85
Accuracy of frog: 0.97
Accuracy of horse: 0.94
Accuracy of ship: 0.94
Accuracy of truck: 0.94
```

Figure 4: Final accuracy of 92.75% on seed 0 for reproducibility

6 references

References

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)', pp. 770–778.

Lin, T.-Y., Goyal, P., Girshick, R., He, K. & Dollár, P. (2017), Focal loss for dense object detection, *in* 'Proceedings of the IEEE International Conference on Computer Vision (ICCV)', pp. 2980–2988.