CIFAR-10 Classification Coursework

ECS6P9U - Neural Networks and Deep Learning

Christian Juresh - 210517307

1 Dataset and Dataloaders: 5%

Two sets of transformations are defined: one for dataset augmentation (used for the training dataset) and one for standard processing (used for the validation and testing datasets). To prevent overfitting, the dataset augmentation transformation includes several steps to artificially increase the diversity of the training data:

- **RandomHorizontalFlip:** Mirrors images horizontally at random.
- **RandomCrop:** Randomly crops the images and adds padding.
- **ToTensor:** Converts the images to PyTorch tensors.
- **Cutout:** Uses my custom implementation of a random masking technique that "cuts out" one or more rectangular patches from the input image.
- **Normalize:** Standardises the colors of the images.

The testing dataset only includes the ToTensor and Normalize transformations. This is so that the model is tested on the original images for accurate results. DataLoaders are set up for both training and testing datasets with a batch size of 64. This gives a balance of speed and accuracy.

2 Model: 40%

2.1 Block

The `Block` class represents one instance of many blocks, which are responsible for a sequence of operations that map an input tensor to a transformed output. It uses an `AdaptiveAvgPool2d` layer to perform spatial averaging, reducing each channel to a vector of d. This vector is then transformed by a `Linear` layer, which outputs a vector a of length K, the number of convolutional layers, where each element is a weight. The convolutional layers consist of a `Conv2d` layer, batch normalization, and a non-linear `ReLU` activation function. In the forward pass, a weight vector a is calculated from the linear layer's output, softmax is applied to obtain a distrubtion of weights, and the outputs of the convolutional layers are combined based on these weights to produce a single tensor O.

2.2 Backbone

The `Backbone` class is responsible for constructing N Blocks and their convolutional layers. It does this by sequentially stacking multiple different Block instances in two for loops,

2.3 Classifier

The `Classifier` class creates a classifier that takes the output from the final Block in the Backbone and computes a mean feature vector by applying a SpatialAveragePool. This mean feature vector is then passed to a multi-layer perceptron (MLP) classifier, consisting of one fully connected layer, `ReLU` activation, dropout for regularization, and then another fully connected layer.

2.4 Model

The `Model` class combines the Backbone and Classifier to create a Convolutional Neural Network (CNN). The forward pass puts tensor X through the Backbone, recieves the O_N output from the N_{th} Block, and then puts it through the Classifier.

3 Loss and Optimizer: 5%

`CrossEntropyLoss` is used as the loss function. It applies a softmax function to the output of the network to compute probabilities, and then calculates the loss between the predicted probabilities and the true class labels. `optim.Adam` is used as the optimizer. It changes the learning rate and decay of the weights to reduce losses. By calculating the gradient of the loss function with respect to each weight in the model (backpropagation), the weights can be adjusted in the opposite direction of the gradient to minimize the loss. Adams optimiser is used specifically as it is fast and accurate.

4 Training Script: 30%

The majority of the code in this section is for collecting metrics. The trianing script itself is quite simple.

4.1 Training Loop

The training loop initializes training variables and iterates over the number of epochs to train the model. `num_epochs` defines how many times the entire dataset will be passed through the model. For each epoch, the `train_epoch` function is called with the *model, training data loader, optimizer, loss criterion,* and *device*. After each epoch, `lr_scheduler.step` is called to adjust the learning rate based on the number of epochs.

4.1.1 Function `train_epoch`

The `train_epoch` function begins by setting the model to training mode. It iterates through the *loader*, which provides batches of *inputs* and their *labels*. The data is then moved to the correct device.
`optimizer.zero_grad` clears any old gradients from the last step so that they dont accumulate. Then the model computers the predicition for the input data. `criterion` calculates the difference between the models predicitions and the actual labels, and produces a scalar value to represent the model's performance. `loss.backward` computes the gradient of the loss with respect to the model's parameters, which are then used to adjust and update the weights with `optimizer.step`

4.2 Hyperparameters

Dataset Augmentation Transformation

- **RandomHorizontalFlip**
  - Probability: 0.5 (default)
- **RandomCrop**
  - Size: 32
  - Padding: 4
- **Cutout**
  - Number of holes (`n_holes`): 1
  - Length of each hole (`length`): 16
- **Normalize**
  - Mean & SD: $(0.5, 0.5, 0.5)$

DataLoader

- Batch Size: 64

Block

- № of Convolutional Layers (`num_convs`): Variable
- Kernel Size: Variable (default is 3)
- Padding: `kernel_size // 2`

Backbone

- Number of Blocks (`num_blocks`): Array of the number of Blocks in each segment of the Backbone

Classifier

- Hidden Features: `in_features * 2`
- Dropout Rate: 0.5

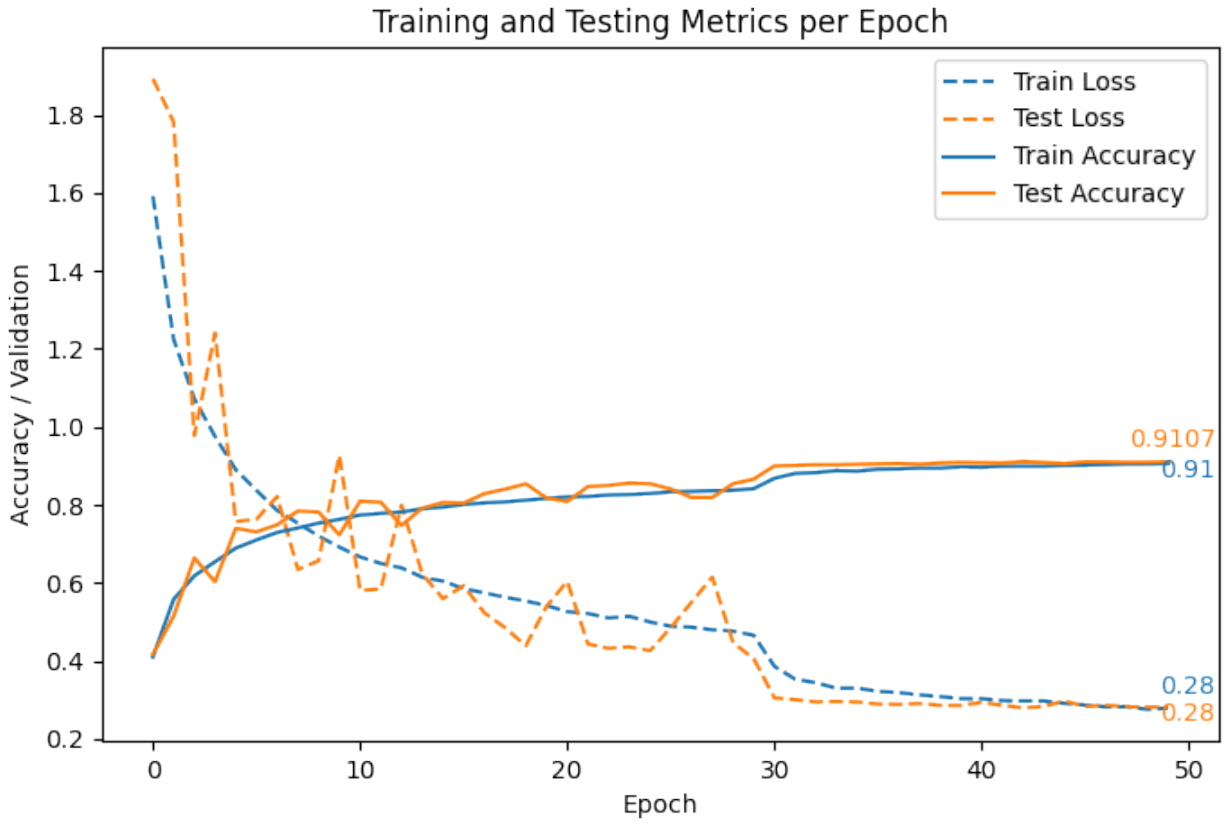
Training Configuration

- Number of Epochs: 50
- Learning Rate (LR): 0.001
- Weight Decay: 1×10^{-4}
- Step Size for LR Scheduler: 30
- Gamma for LR Scheduler: 0.1

Model Configuration

- Number of Classes (`num_classes`): 20
- In Channels: 3 (RGB)

4.3 Plot



5 Final Model Accuracy: 20%

Consistently 90% to 91% accuracy on the test set. See plot above with 91.0700% Test Accuracy