Report ec22262 ECS7026P

4.

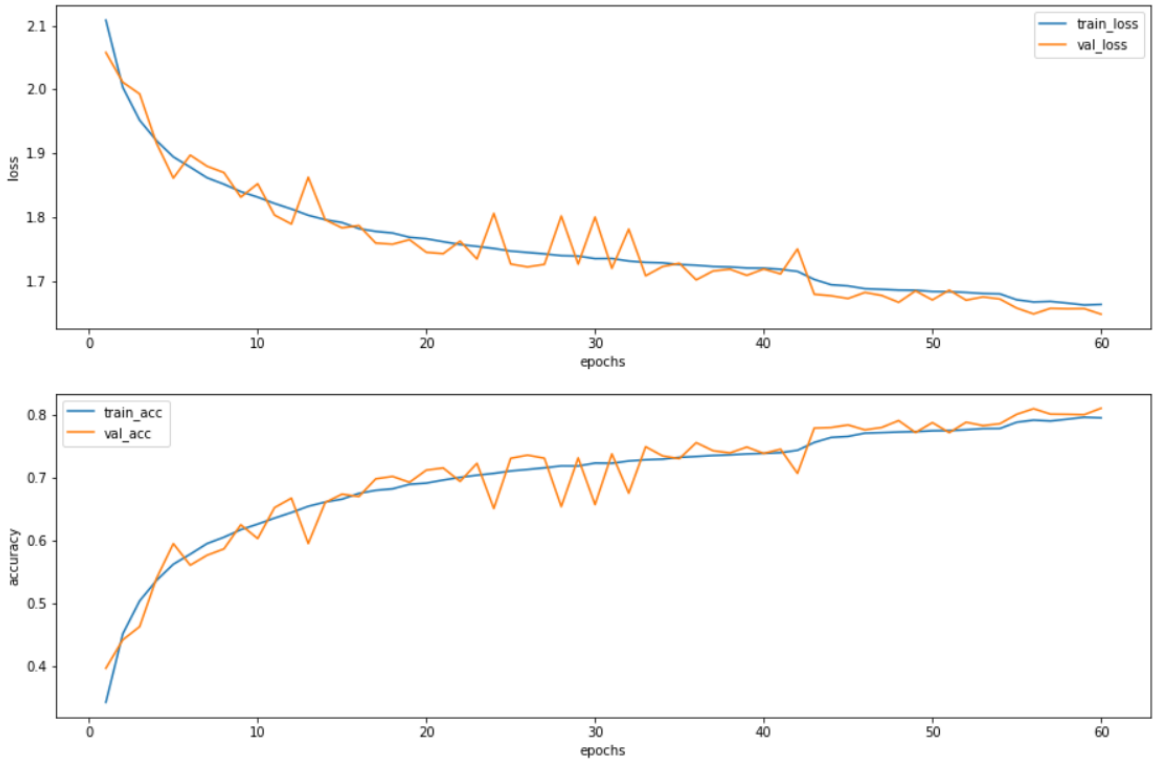

Figure 1:
(top) shows the training and validation loss as the number of epochs increases.
(bottom) shows the training and validation accuracy as the number of epochs increases.

In Figure 1(top), we clearly see the training loss decrease quickly for the first 5 epochs and then the rate of decrease decreases. The validation loss shows a similar pattern but is a lot more erratic. The final validation loss is less than the final training loss suggesting that the model generalizes quite well and doesn't overfit the training data. The validation loss plateaus and the figure suggests that it won't decrease much more. The final validation loss is 1.64946

In Figure 1(bottom), we see the training accuracy rapidly increases for the first 5 epochs and then the rate of increase decreases. The validation accuracy starts increasing quite erratically but then starts to converge in a more stable fashion as the number of epochs increases (likely due to the learning rate decrease). It reaches a final validation accuracy of 81.130%, which is higher than the training accuracy. This suggests that the model generalizes well and doesn't overfit the training data.

Hyperparameters and other details

	Name	Details

Criterion	Cross Entropy Loss	
optimizer	Adam	Default lr of 0.001, weight decay factor of 0.0001.
scheduler	ReduceLROnPlateau	Lr halves in values when validation loss hasn't decreased 5 epochs after the smallest to date.
epochs	60	
Batch Size	256	A good trade-off between speed and performance

Hyperparameters in Class Layers:

In Convolution class: (Used to create the Block class)

- In and out channels, kernel_size, stride, padding of Conv2d (chosen later)
- Initialization of Conv2d (I have used xavier_normal, weights are small to begin with which helps mitigate exploding gradient problem)
- BatchNormalization (I have kept the out channels the same as in channels)
- Activation is ReLU - chosen to avoid vanishing/exploding gradient problem
- Dropout rate (chosen later)

In Block class:

- Can add or create a residual connection
- Kernel_sizes determines the output size of dense layer
- Activation is ReLU
- Number of parallel convolutions is defined in the Backbone class, which uses Block. The kernel_sizes and strides can be uniquely given for each convolution.

In Classifier class:

- Performs spatial avg pooling
- Then dense layer with softmax for the output layer.

In Backbone class:

- Composed of 8 Block classes.
- The blocks have the following in-out channel pairs (3, 16), (16, 32), (32, 64), (64, 32), (32, 48), (48, 64), (64, 64), (64, 64).
- Blocks 2 and 5 reduce the height and width of the image by half. (32x32 – block 2 –> 16x16 – block 5 –> 8x8).
- Block 4 has a residual connection to block 7. This allows for the error to get propagated backward without a signal loss.
- Each block has a dropout_rate of 0.1
- The residual connection goes through a Convolution block to reduce and change its height and width: (16x16 → 8x8).

In Model:

- Simply uses the Backbone and Classifier. Only the hyperparameter depends on the final channel size of the Backbone, which is then passed to the classifier (determine the number of in-features to linear layer).

Preprocessing

I did some data augmentation to increase the size of the training set. I augmented the training set twice to increase the training set size 3-fold, for a final size of 150000 images.

There is a lot of options for data augmentation but I decided to keep it simple. I have 2 augmentation transforms:

Transform 1:

- Random rotation, up to 30 degrees either way
- Then a randomized crop with a scale of up to 0.2 points either way w.r.t original
- Then normalize values between 0 and 1.

Transform 2:

- Apply a random linear translation of up to 0.3 points either way
- Apply a saturation change with a probability of 0.5
- Then a random resized crop as before
- Then normalize values between 0 and 1.

5.

The loss on the test set is: 1.6494630260467529

The accuracy on the test set is: 0.8113.