Assignment Set 6

Pegah Khazaei

11th November 2020

L2 & L1 regularization

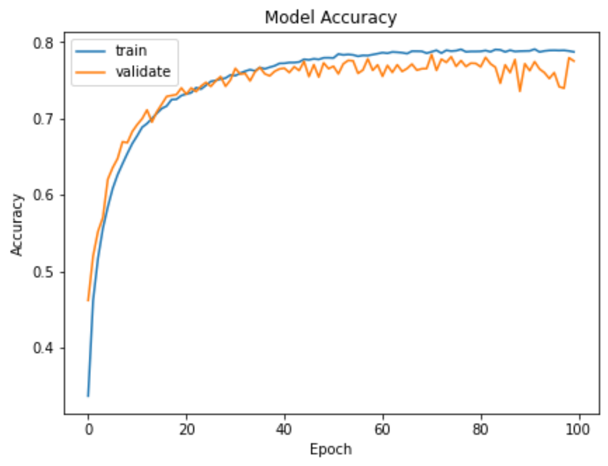
L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.
Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.

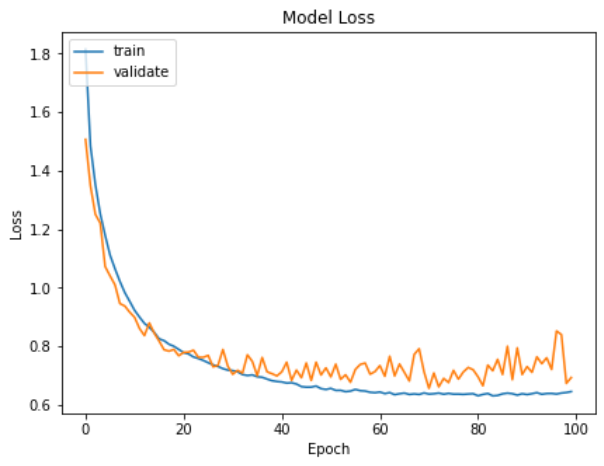
Data Augmentation

The simplest way to reduce overfitting is to increase the size of the training data. In machine learning, we were not able to increase the size of training data as the labeled data was too costly. data augmentation usually provides a big leap in improving the accuracy of the model. It can be considered as a mandatory trick in order to improve our predictions. In keras, we can perform all of these transformations using ImageDataGenerator. It has a big list of arguments which we can use to pre-process our training data.

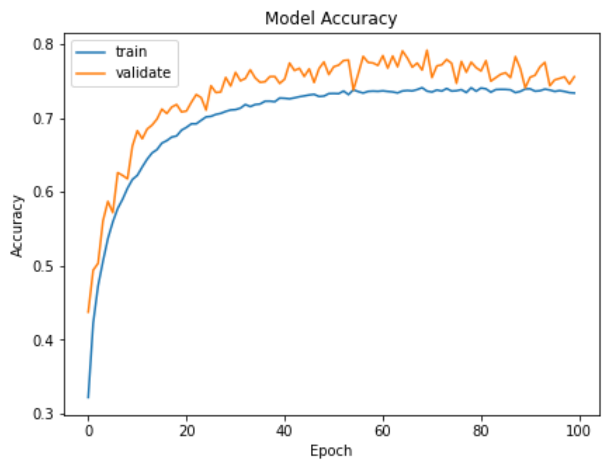
Cifar-10 For First Model

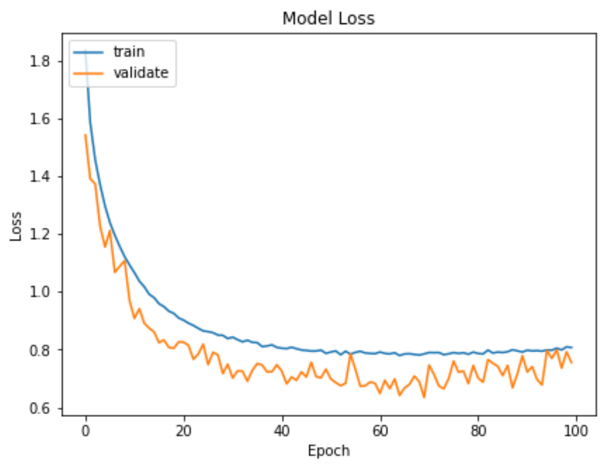
CIFAR-10 without data augmentation





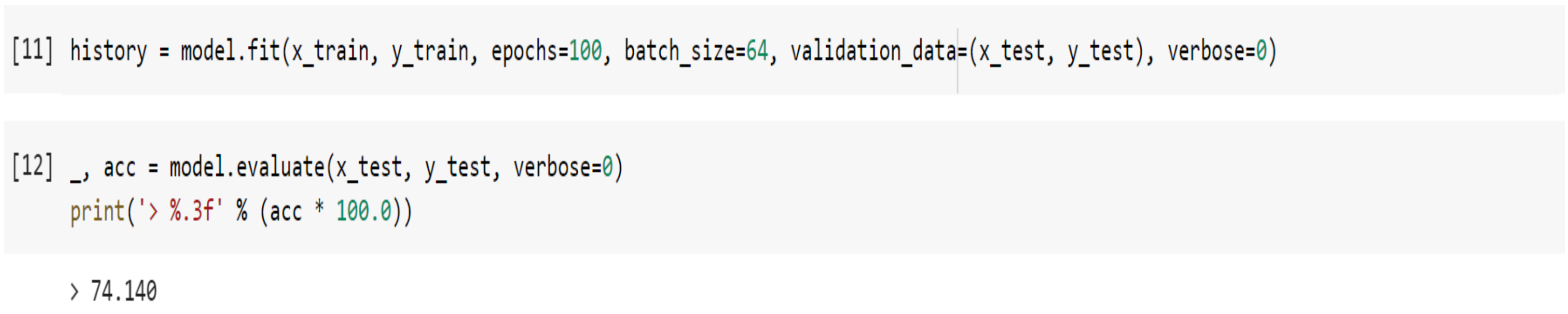
Cifar-10 with Data Augmentation





Cifar-10 For Second Model

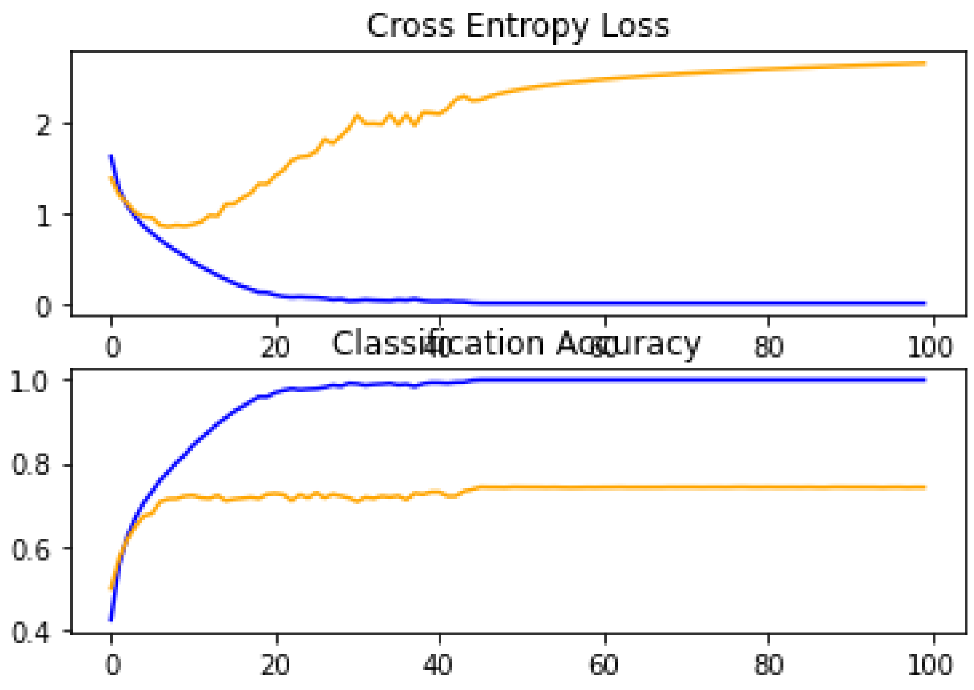
Baseline Model…



```
[11] history = model.fit(x_train, y_train, epochs=100, batch_size=64, validation_data=(x_test, y_test), verbose=0)
```

```
[12] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

     > 74.140
```



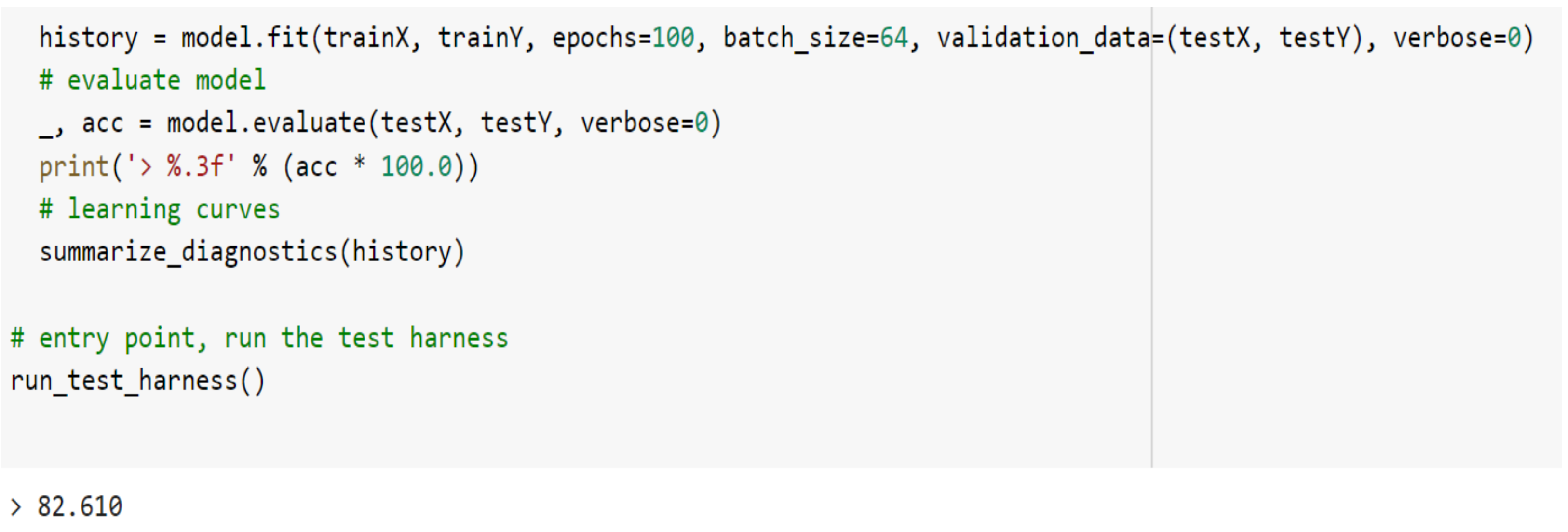
Regularization Techniques

Dropout Regularization

Dropout can be added to the model by adding new Dropout layers, where the amount of nodes removed is specified as a parameter. There are many patterns for adding Dropout to a model, in terms of where in the model to add the layers and how much dropout to use.

baseline model with dropout on the cifar10 dataset

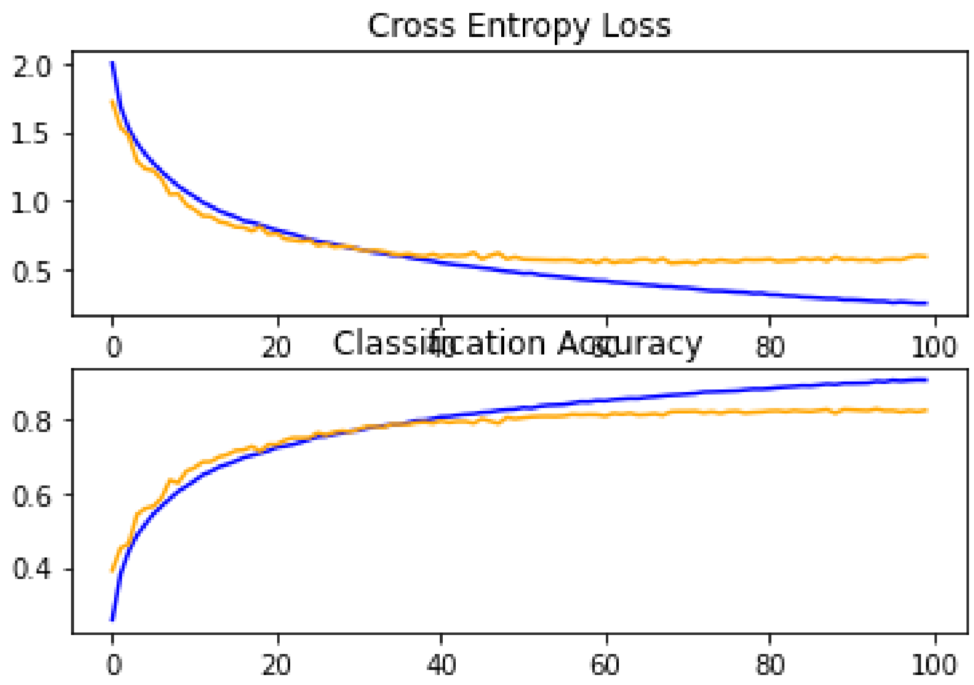
In this case, we will add Dropout layers after each max pooling layer and after the fully connected layer, and use a fixed dropout rate of 20% (e.g. retain 80% of the nodes).



```
    history = model.fit(trainX, trainY, epochs=100, batch_size=64, validation_data=(testX, testY), verbose=0)
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()
```

```
> 82.610
```



In this case, we can see a jump in classification accuracy by about 8% from about 74.15% without dropout to about 82.610% with dropout.

Reviewing the learning curve for the model, we can see that overfitting has been addressed. The model converges well for about 40 or 50 epochs, at which point there is no further improvement on the test dataset.

This is a great result. We could elaborate upon this model and add early stopping with a patience of about 10 epochs to save a well-performing model on the test set during training at around the point that no further improvements are observed.

We could also try exploring a learning rate schedule that drops the learning rate after improvements on the test set stall.

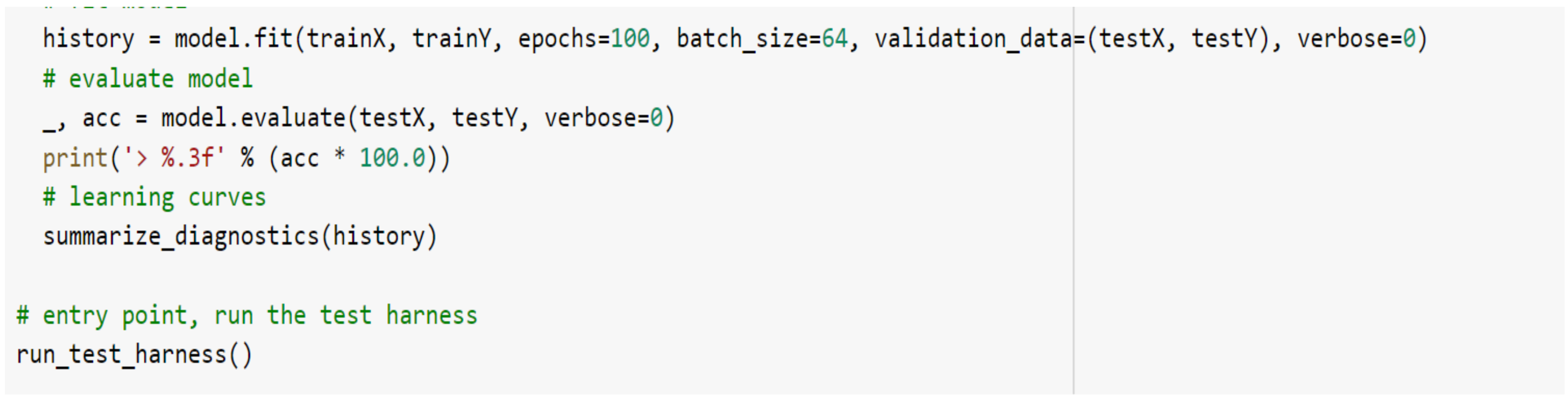
Dropout has performed well, and we do not know that the chosen rate of 20% is the best. We could explore other dropout rates, as well as differing positioning of the dropout layers in the model architecture.

Weight Decay

Weight regularization or weight decay involves updating the loss function to penalize the model in proportion to the size of the model weights.

This has a regularizing effect, as larger weights result in a more complex and less stable model, whereas smaller weights are often more stable and more general.

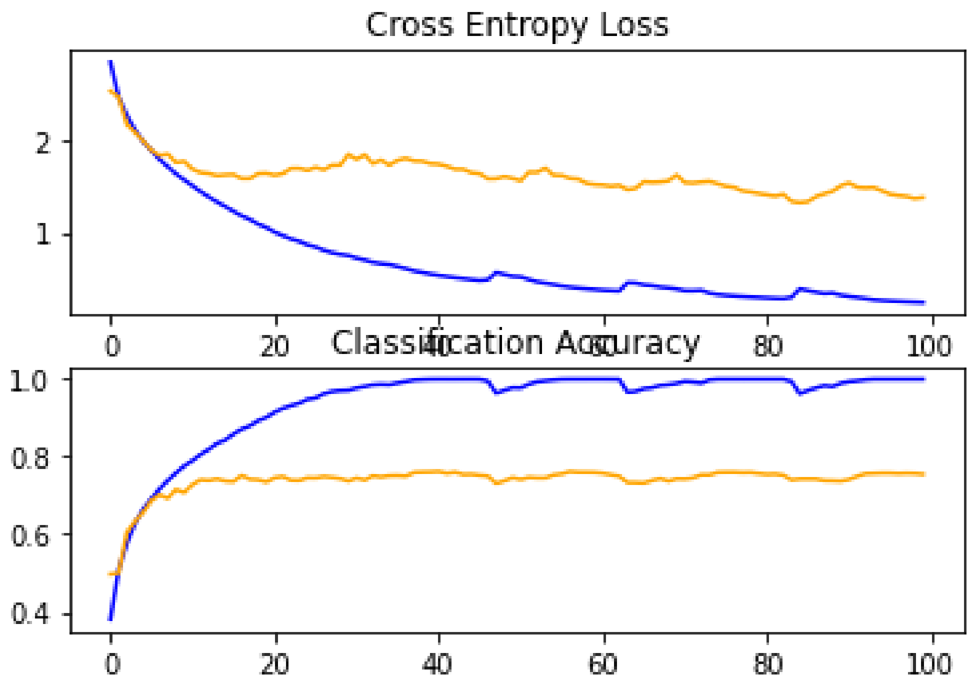
baseline model with weight decay on the cifar10 dataset



```
    history = model.fit(trainX, trainY, epochs=100, batch_size=64, validation_data=(testX, testY), verbose=0)
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()
```

```
> 75.510
```



In this case, we see no improvement in the model performance on the test set; in fact, we see a small drop in performance from about 74% to about 76% classification accuracy.

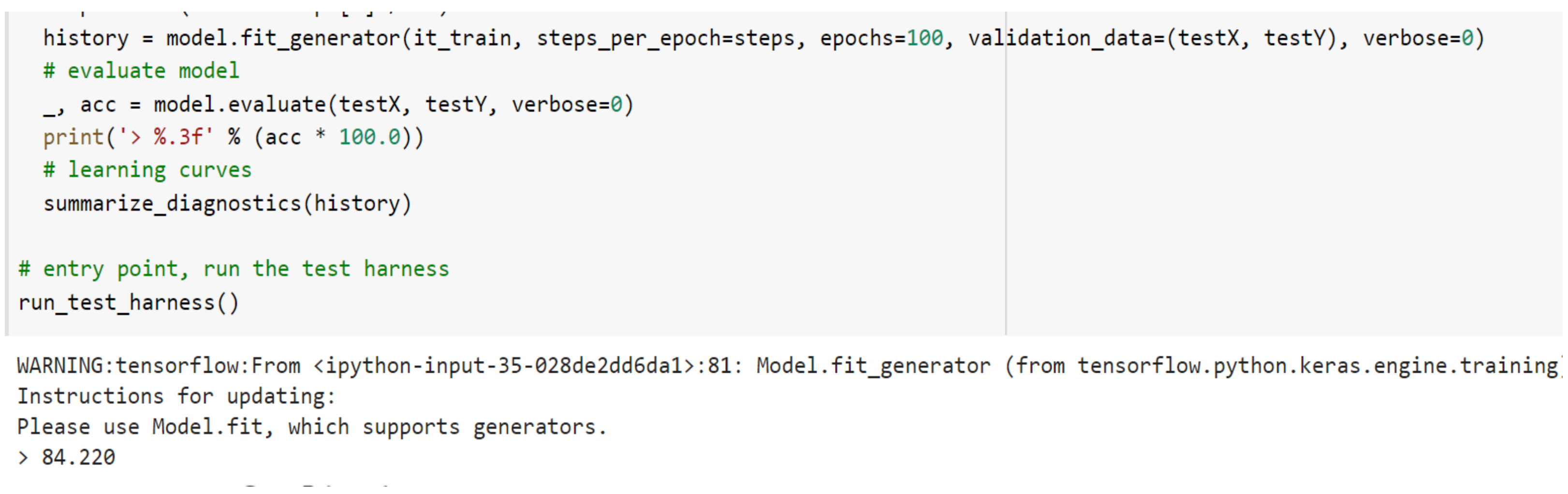
Reviewing the learning curves, we do see a small reduction in the overfitting, but the impact is not as effective as dropout.

We might be able to improve the effect of weight decay by perhaps using a larger weighting, such as 0.01 or even 0.1.

Data Augmentation

We will investigate the effect of simple augmentation on the baseline image, specifically horizontal flips and 10% shifts in the height and width of the image.

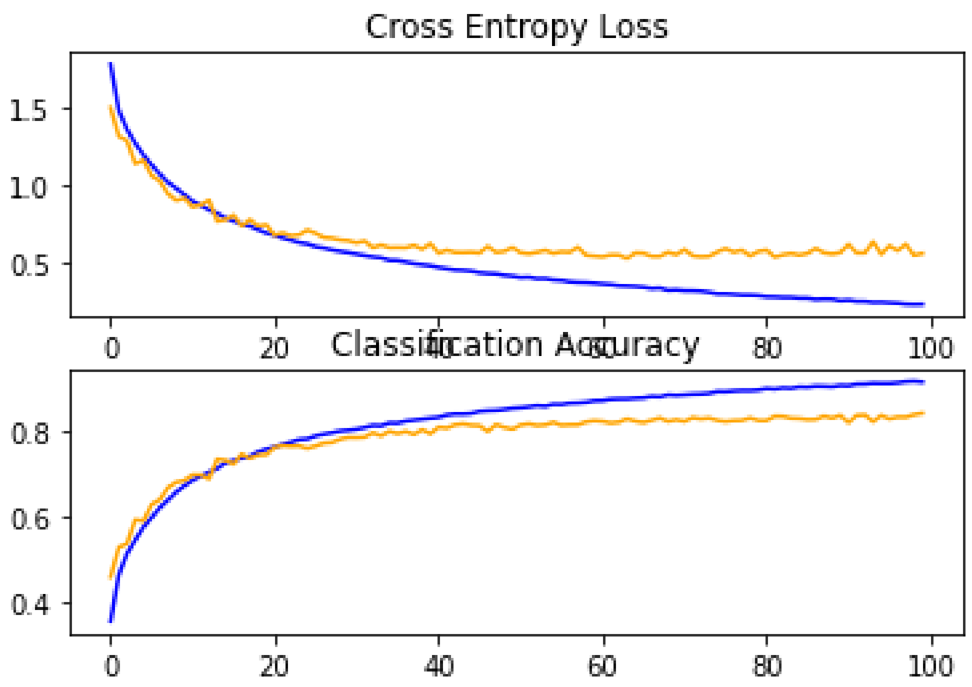
baseline model with data augmentation on the cifar10 dataset



```
  history = model.fit_generator(it_train, steps_per_epoch=steps, epochs=100, validation_data=(testX, testY), verbose=0)
  # evaluate model
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))
  # learning curves
  summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()
```

```
WARNING:tensorflow:From <ipython-input-35-028de2dd6da1>:81: Model.fit_generator (from tensorflow.python.keras.engine.training
Instructions for updating:
Please use Model.fit, which supports generators.
> 84.220
```



We will investigate the effect of simple augmentation on the baseline image, specifically horizontal flips and 10% shifts in the height and width of the image.

Reviewing the learning curves, we see a similar improvement in model performances as we do with dropout, although the plot of loss suggests that model performance on the test set may have stalled slightly sooner than it did with dropout.

The results suggest that perhaps a configuration that used both dropout and data augmentation might be effective.

Discussion

In this section, we explored three approaches designed to slow down the convergence of the model.

A summary of the results is provided below:

- **Baseline + Dropout**: 82.610%
- **Baseline + Weight Decay**: 75.510%
- **Baseline + Data Augmentation**: 84.220%