	Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych **Projekt BIAI**			
Rok akademicki	**Rodzaj studiów*: SSI/NSI/NSM**	**Numer ćwiczenia:**	**Grupa**	**Sekcja**
2019/2020	**SSI**	**-**	**-**	**-**
Data i godzina planowana ćwiczenia: dd/mm/rrrr - gg:mm	-	**Prowadzący**: OA/JP/KT/GD/BSz/GB	**GB**	
Data i godzina wykonania ćwiczenia: dd/mm/rrrr - gg:mm	-			

Sprawozdanie

Temat Projektu: Analysis of impact of Convolutional Neural Network structure on the performance

Skład sekcji:	1. Adrian Skutela 2. Dawid Musialik

1. Introduction

The goal of our project was to analyze the impact of different structures of convolutional neural networks on image classification performance. We used the CIFAR-10 dataset for its relative simplicity. To assess the performance, we decided to measure accuracy and cross entropy loss of the neural networks.

2. Analysis

Initially we have considered the following possibilities:

- create the (non-convolutional) neural networks ourself from scratch using C++. The advantage of this approach would be that we would have learned a lot about inner workings of neural networks. The disadvantage however was that complexity and difficulty of building a neural network framework, even if it would be very basic when compared to top solutions in the industry, would make it very hard to compete the project in time, or even at all. Another disadvantage would be that non-convolutional neural networks would perform very poorly when classifying images.
- Use existing frameworks such us TensorFlow with Keras, and Python to develop convolutional neural networks. The advantage of this approach is that it is much easier to implement a functioning convolutional neural network in an already existing framework. Another advantage is that the neural network implemented with this approach, would perform much better, than one written from scratch by us. After TensorFlow has been around for years now, and has been developed by team of specialist, so we could not even hope to come close to it's performance. The disadvantage however, was that the neural networks we developed would be „black boxes" that is we would not have the opportunity to get to know their inner workings.

After considering the advantages and disadvantages of the above approaches, and after consulting the issue with the teacher, we have decided to use the second approach i.e. use convolutional neural networks developed TensorFlow with Keras and python. When it came to the networks' structure, the options we considered were to use VGG convolutional neural networks (referred further as CNNs). Our research in the topic has revealed that those kinds of networks are one of the best for image classification. The alternative would be to develop the CNNs structure ourselves. We have decided to combine both approaches and experiment ourselves with different VGG combinations and other techniques to improve the networks' performance.

We have chosen the CIFAR-10 for its simplicity - the images are small, and there are only 10 classes, ease-of-use – Keras already comes with CIFAR-10 dataset support, ready to install and use, with minimal preprocessing required, and availability – it is free to use.

3. Program Specification

3.1 Internal structure

We have divided our program into the following Python scripts:

- trainer.py - contains classes and methods to train multiple networks, to report the results to files, and to save the models
- myModels.py – contains classes that generate many different models of neural networks
- classify.py – has a console interface, that allows to classify to classify selected image, with a model in a .h5 file.

Below is the detailed description of the most important parts of our program:

File trainer.py:

class Trainer:

- def __init__(self) – constructor that initializes the Trainer object with default values.
- def assignDataset(self, dataset) – assigns the specified dataset to the Trainer object. All models trained with this trainer will use this dataset.
- def test(self, model): - trains and tests the model passed as an argument, and saves graph with training performance to the file.

class Dataset:

- def __init__(self, trainIn=None, trainOut=None, testIn=None, testOut=None) – constructor that initializes the Dataset object.
  Arguments:
  - trainIn – input for training the NN
  - trainOut – expected output for training the NN
  - testIn – input for validating the NN
  - testOut – expected output for validating the NN
- def prepare(self) – transforms the image set into the form expected by the NN. It normalizes the pixel values into the range <0; 1> and converts the outputs from an integer indicating the image class to a vector that contains „1" at the appropriate position, and „0" on others – for example „3" would be converted to [0, 0, 0, 1, 0, 0, 0, 0, 0, 0] (classes start from 0 – thus 3 has „1" at fourth position)

- def fileReport(history, accuracy, filename) – function that saves training history to a file as a graph

- def main() - when the script is executed, the function gets all models from classes in myModels.py, and then trains them on the CIFAR-10 dataset. It prints the results to the console, saves them as graphs, and saves the trained models as .h5 files.

File mymodels.py:

class BaseModels(ModelInterface):

- def make1VGGModel() - creates a basic model with 1 VGG layer
- def make2VGGModel() - creates a basic model with 2 VGG layers
- def make3VGGModel() - creates a basic model with 3 VGG layers

class Dropout_Models(ModelInterface):

- def make1VGGModel() - creates a model with 1 VGG layer and a dropout layer after each VGG layer
- def make2VGGModel() - creates a model with 2 VGGs layers and a dropout layer after each VGG layer
- def make3VGGModel() - creates a model with 3 VGGs layers and a dropout layer after each VGG layer

class WeightDecay_Models(ModelInterface):

- def make1VGGModel() - creates a model with 1 VGG layer with weight decay
- def make2VGGModel() - creates a model with 2 VGG layers with weight decay
- def make3VGGModel() - creates a model with 3 VGG layers with weight decay

class WeighrDecay_Dropout_Models(ModelInterface):

- def make1VGGModel() - creates a model with 1 VGG layer with weight decay and a dropout layer after each VGG layer
- def make2VGGModel() - creates a model with 2 VGG layers with weight decay and a dropout layer after each VGG layer
- def make3VGGModel() - creates a model with 3 VGG layers with weight decay and a dropout layer after each VGG layer

3.2 User manual

Two scripts are intended to be used by the program user:

- trainer.py requries no user interaction, and no command line parameters, when launched it will train the CNNs, save the results and models to files.
- Cassify.py requires no user interaction, but it requires command line parameters it can be launched as following:
  python classify.py -i automobile5.png -m models/2020-09-20_17-24_base_1VGG.h5
  where -i option specifies the image to classify, and -m option specifies the model to use for classification. It can be also launched without any parameters, or with -h option to display help.

4. Experiments

We have trained multiple models of CNN on a CIFAR-10 dataset, and then noted the accuracy, cross entropy loss, and epoch needed to achieve the best performance. Each model has been trained once.

We have recorded the final performance of each model, as well as training history – presented as a grapgh.

The training of the model has been stopped, once accuracy stopped improving.

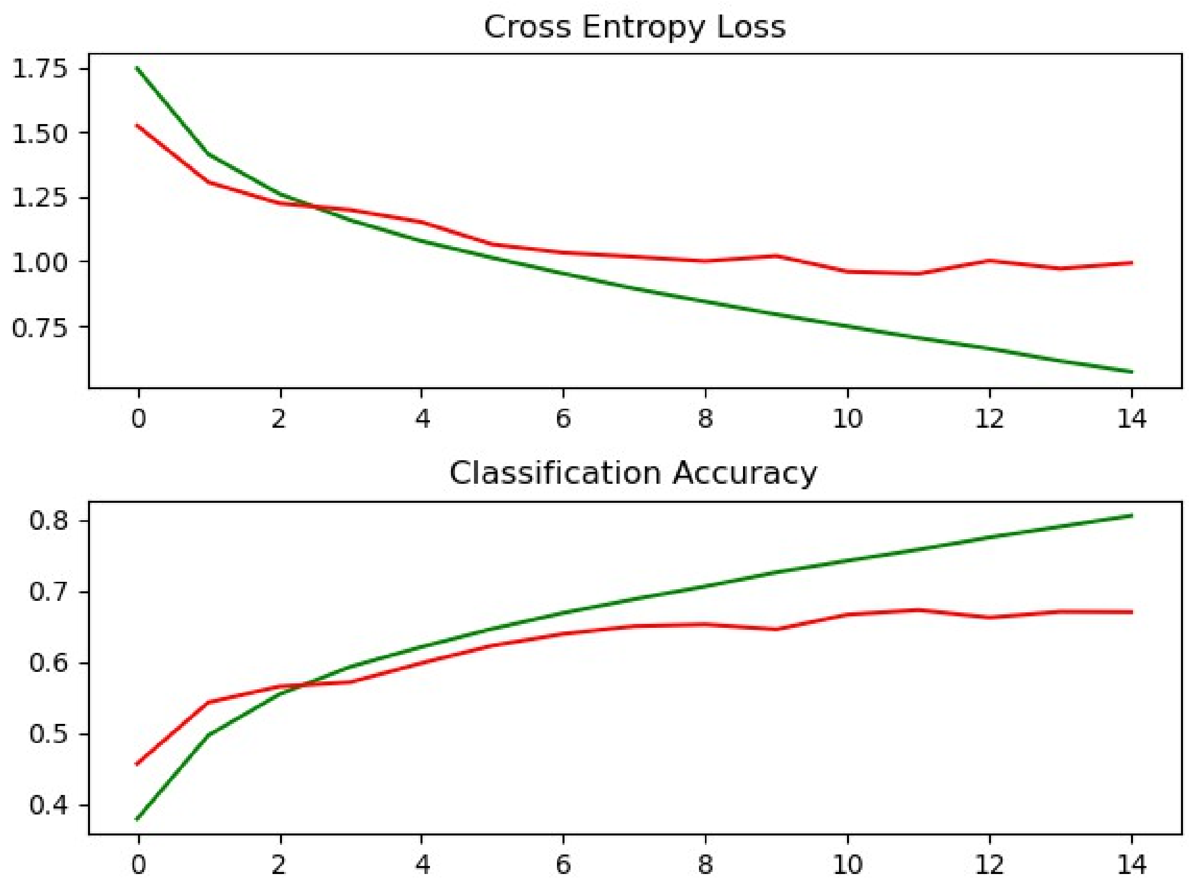
All graphs and results are included in this report.

When reading graph: On the x axis is the number of epochs, and y axis in the upper graph is a value of a loss function, and in the bottom graph is the fraction of images that the model classified correctly. Green line represents training data, while red line represents test data.

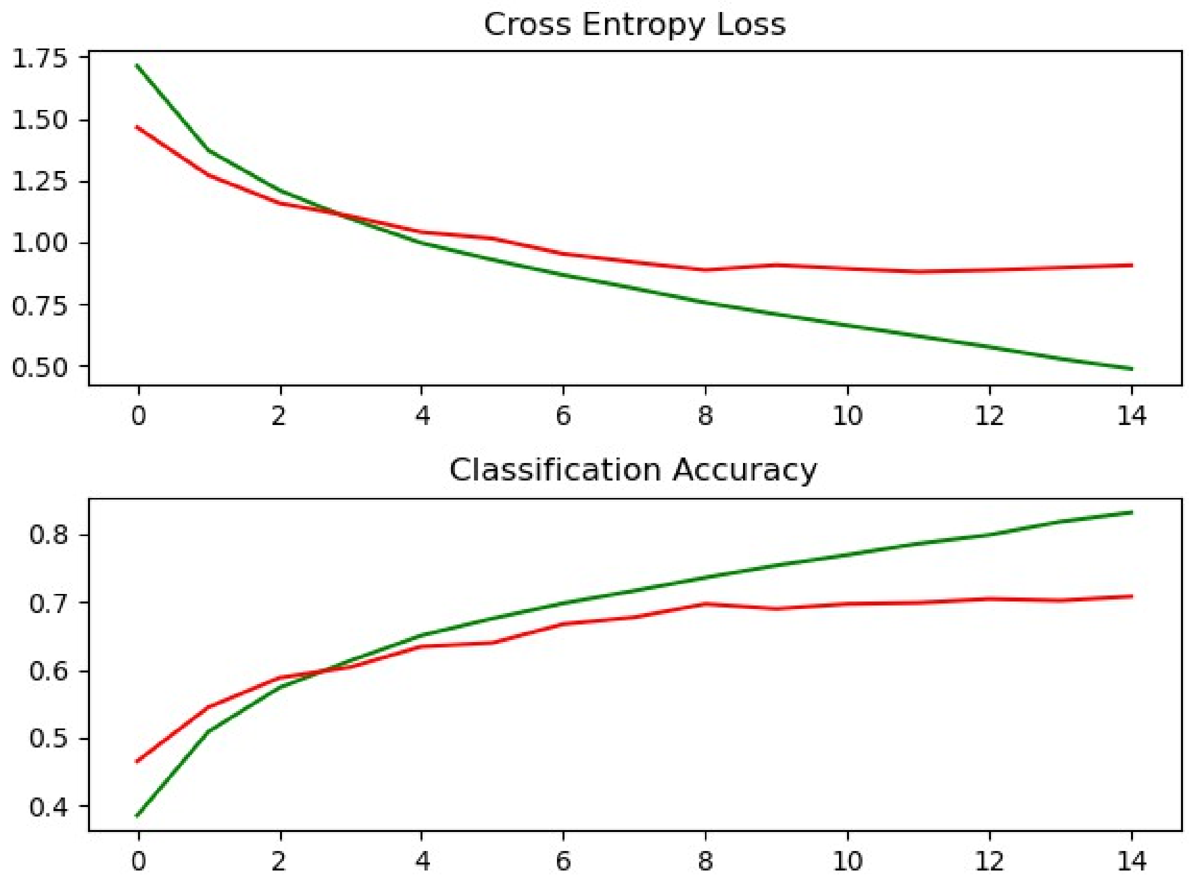
a) VGG layers

Model	accuracy
6 layer	0.6679
9 layer	0.7059
12 layer	0.7327

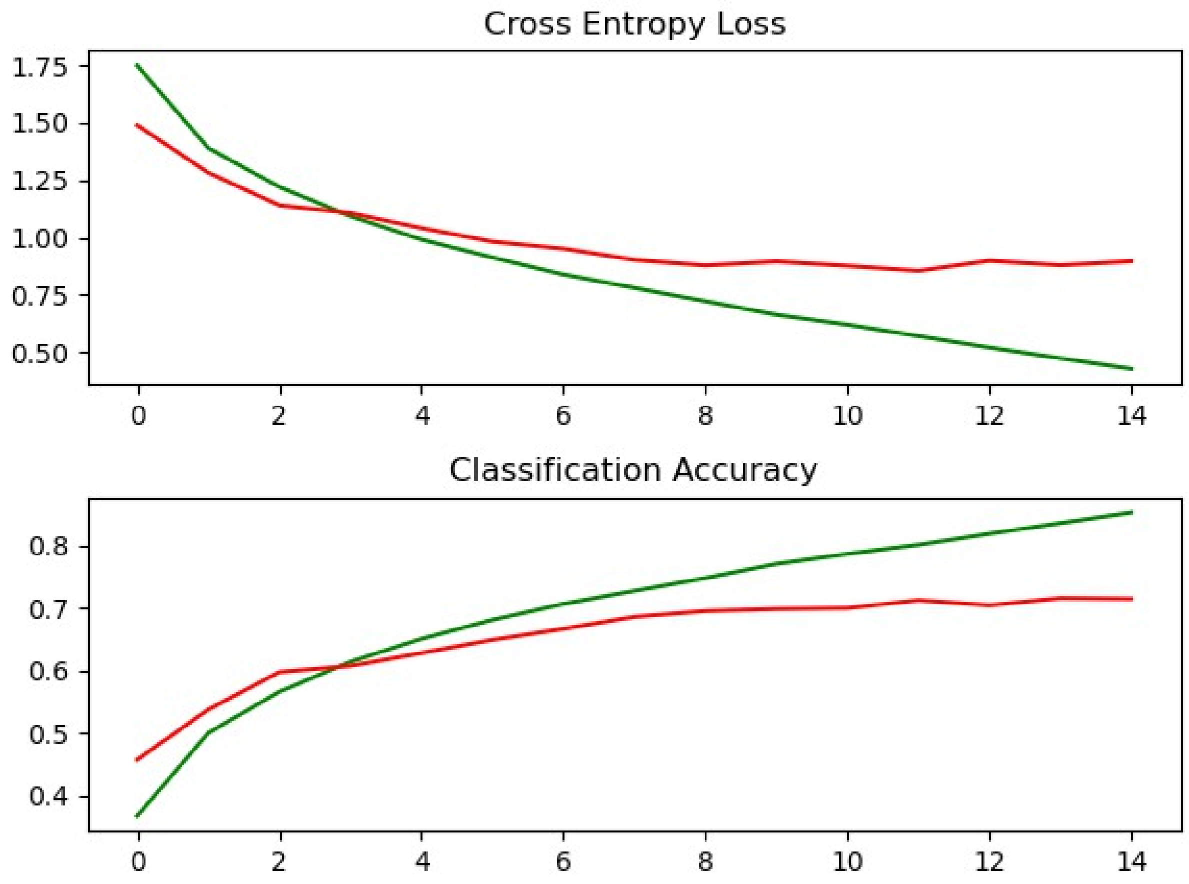
6 layers:



9 layers:



12 layers:



Observations and conclusions:

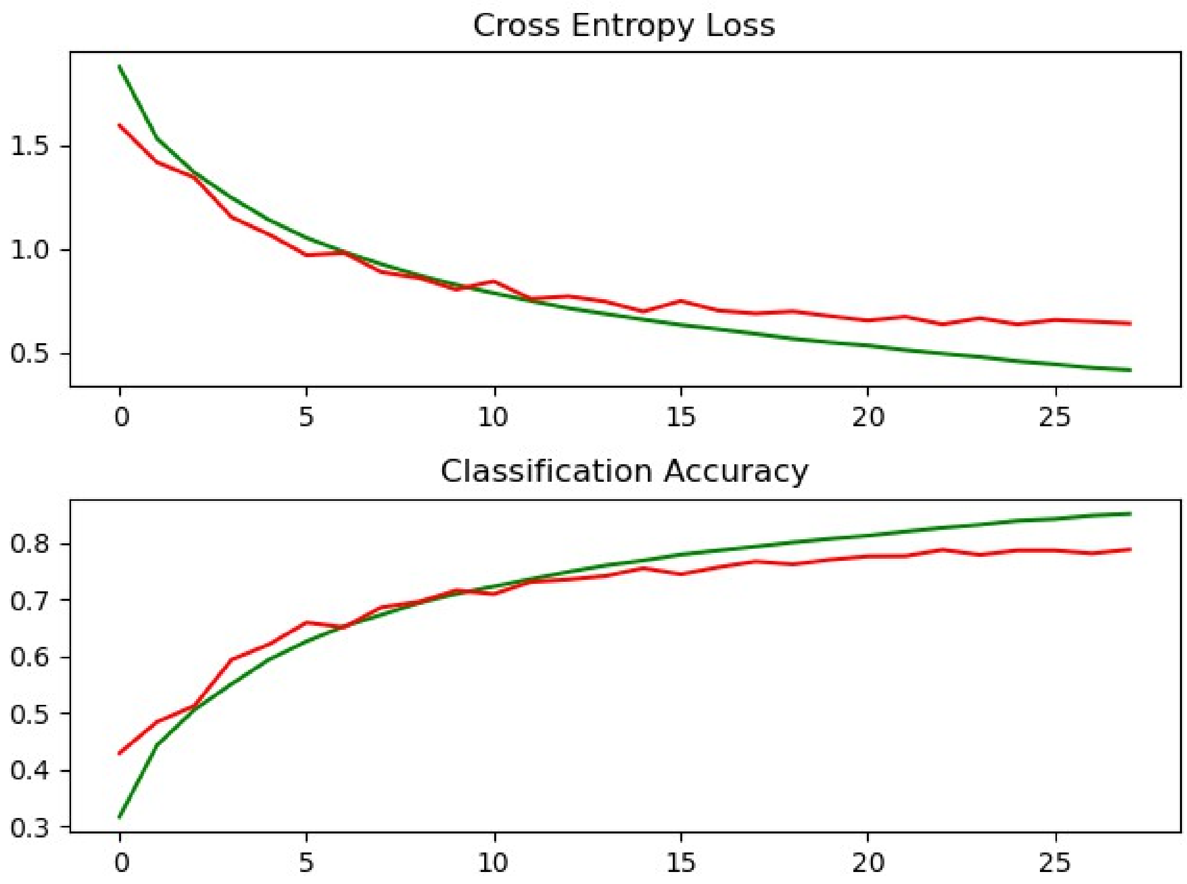
- adding more VGG layers to the network increases the accuracy, but increases the number of epochs needed to achieve the best performance
- the best results have been achieved with 12 layers, so in next experiments we're using models only with 12 layers

b) same dropout on each layer

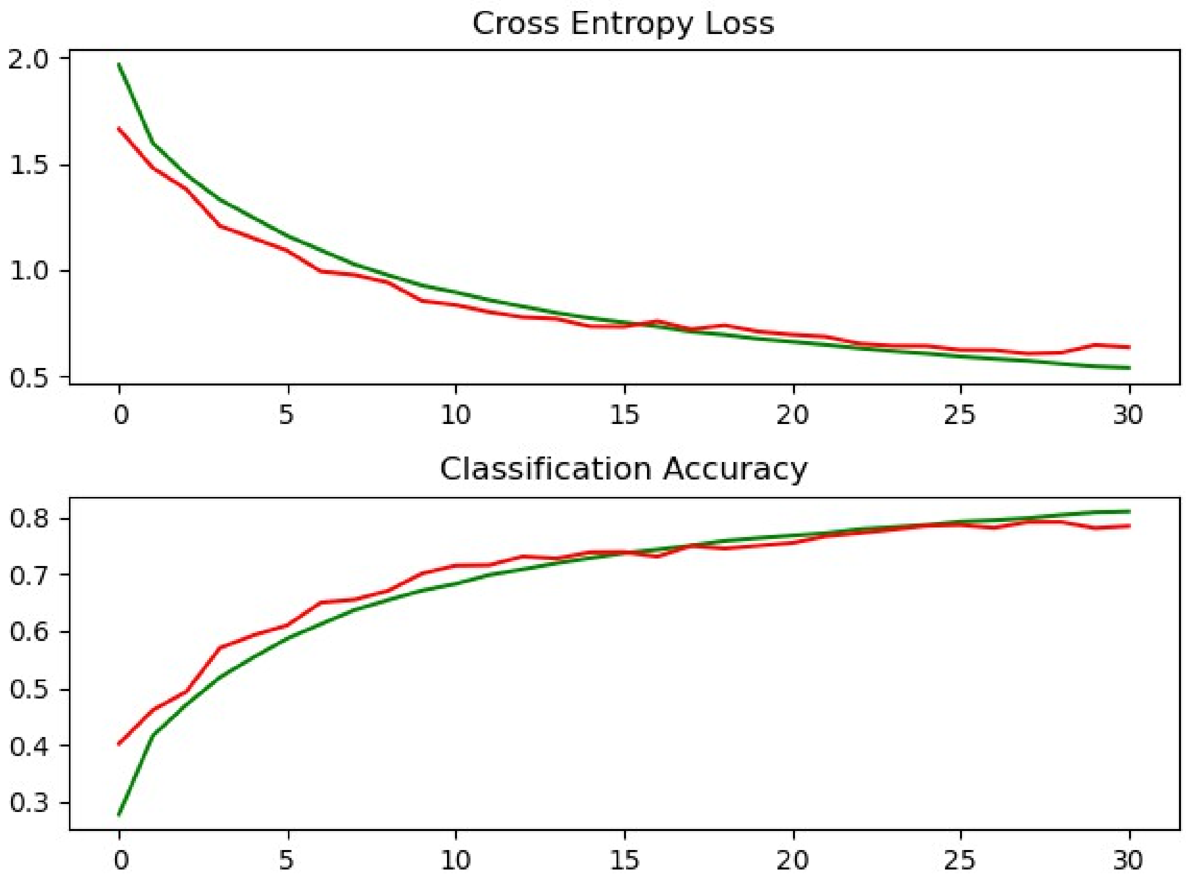
Results:

Variation	accuracy
10% dropout	**0.7885**
20% dropout	0.7839
30% dropout	0.7810
40% dropout	0.7417
50% dropout	0.5811
60% dropout	0.2144

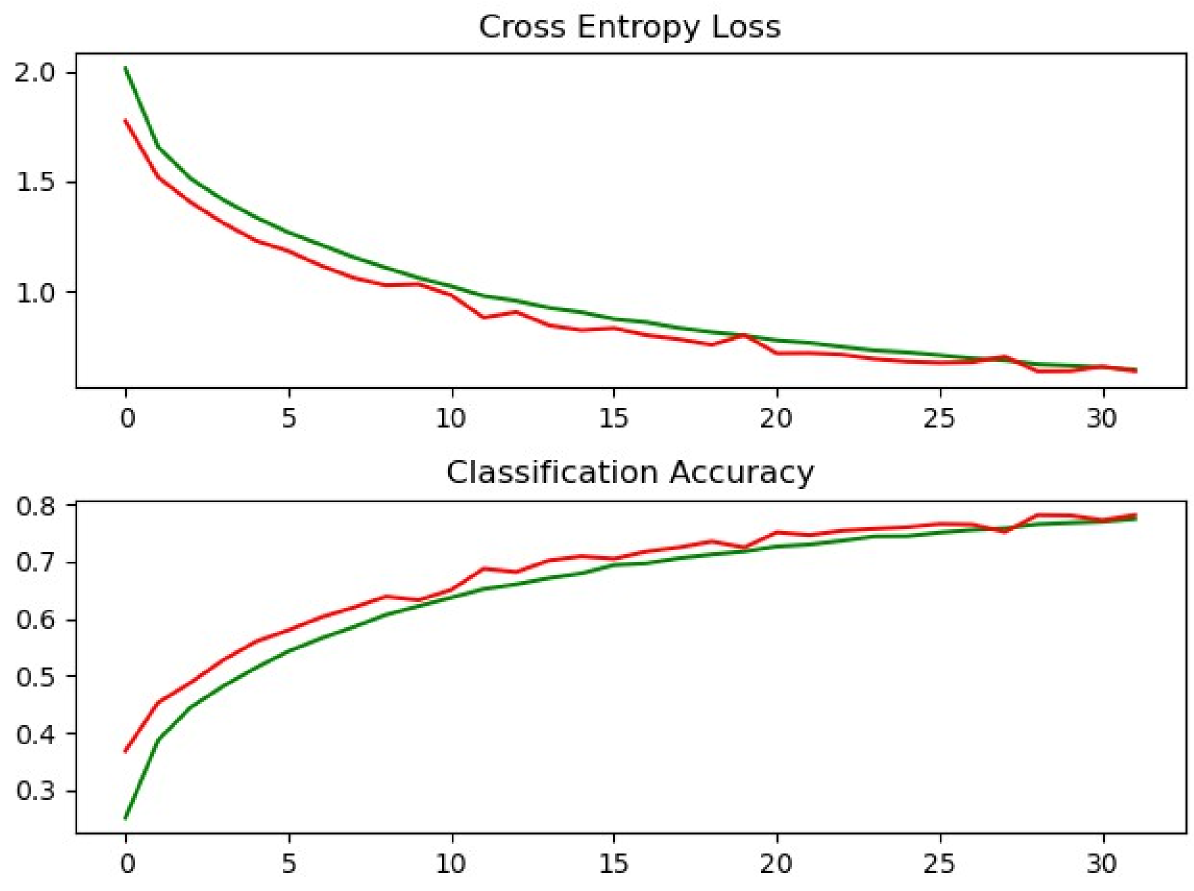
10% dropout:



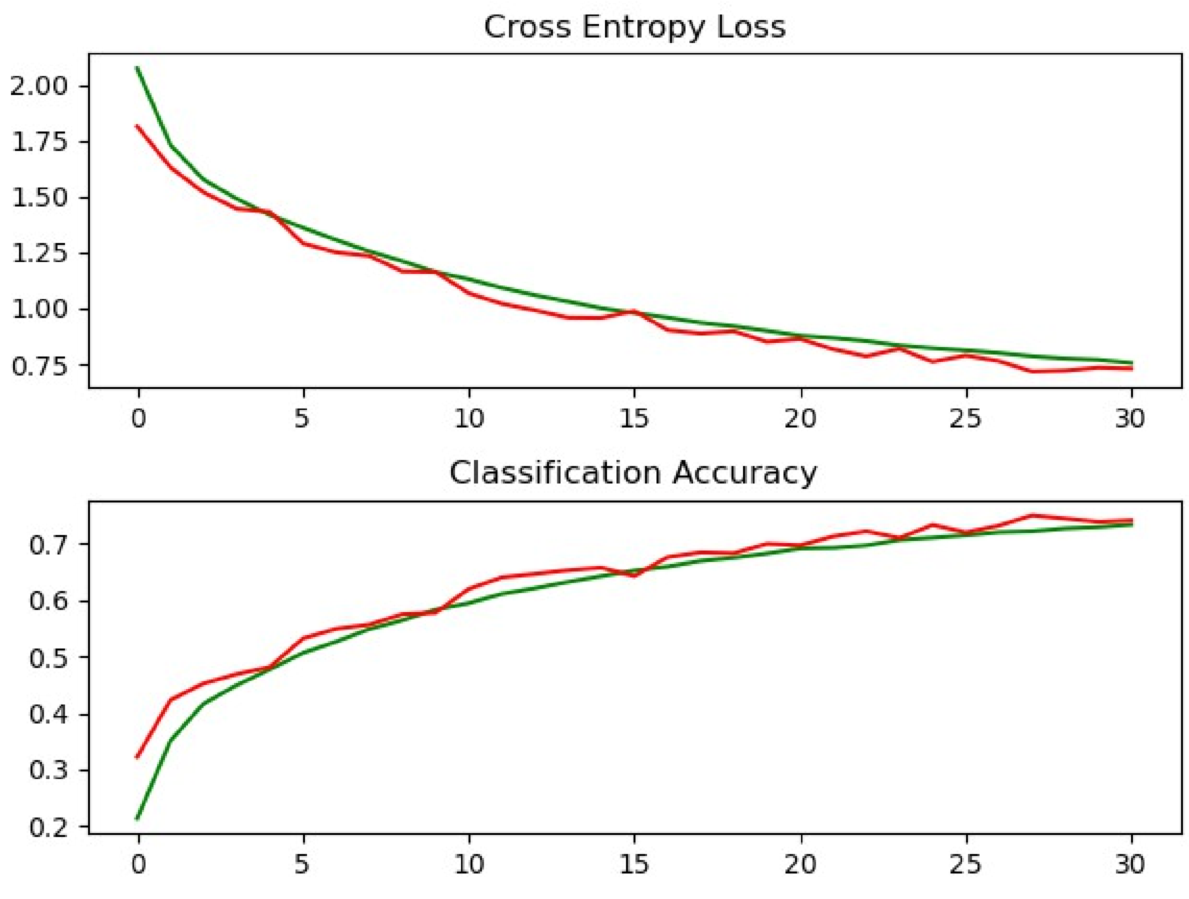
20% dropout:



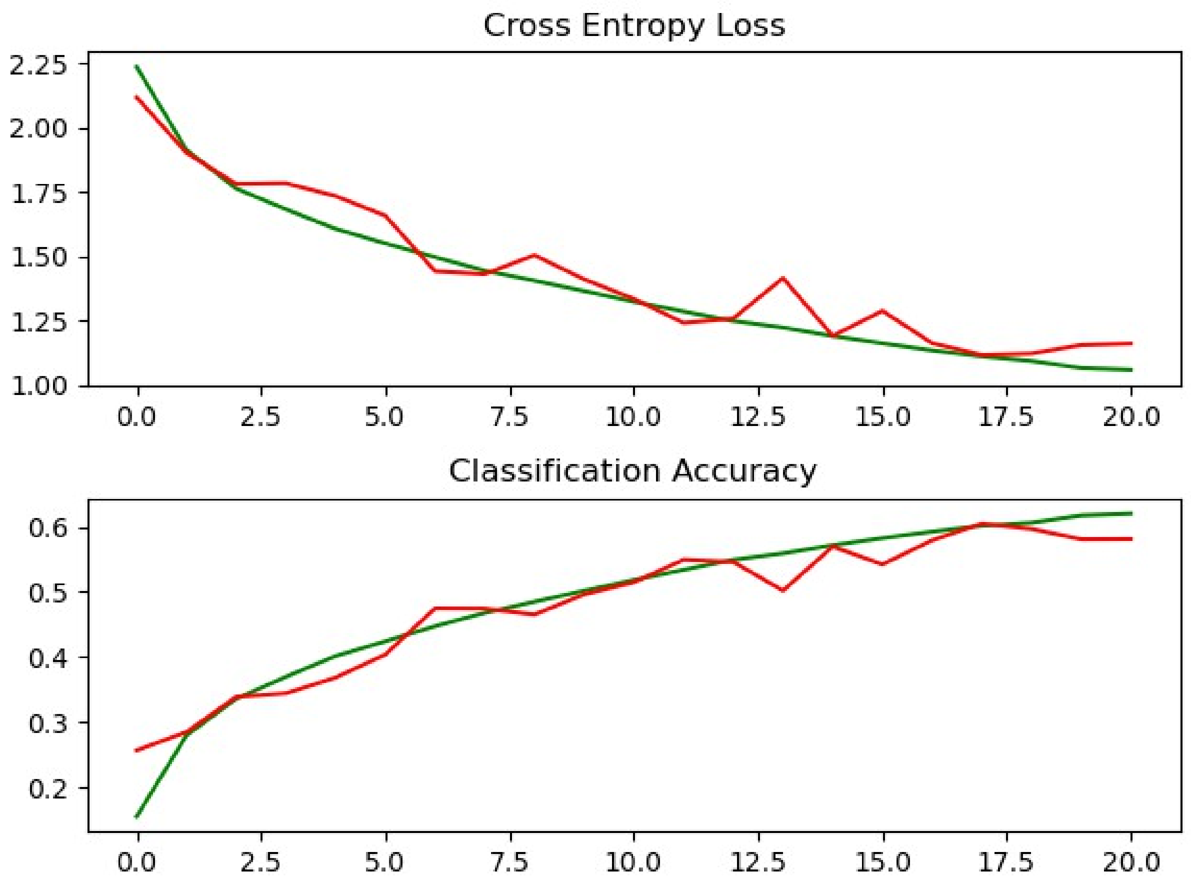
30% dropout:



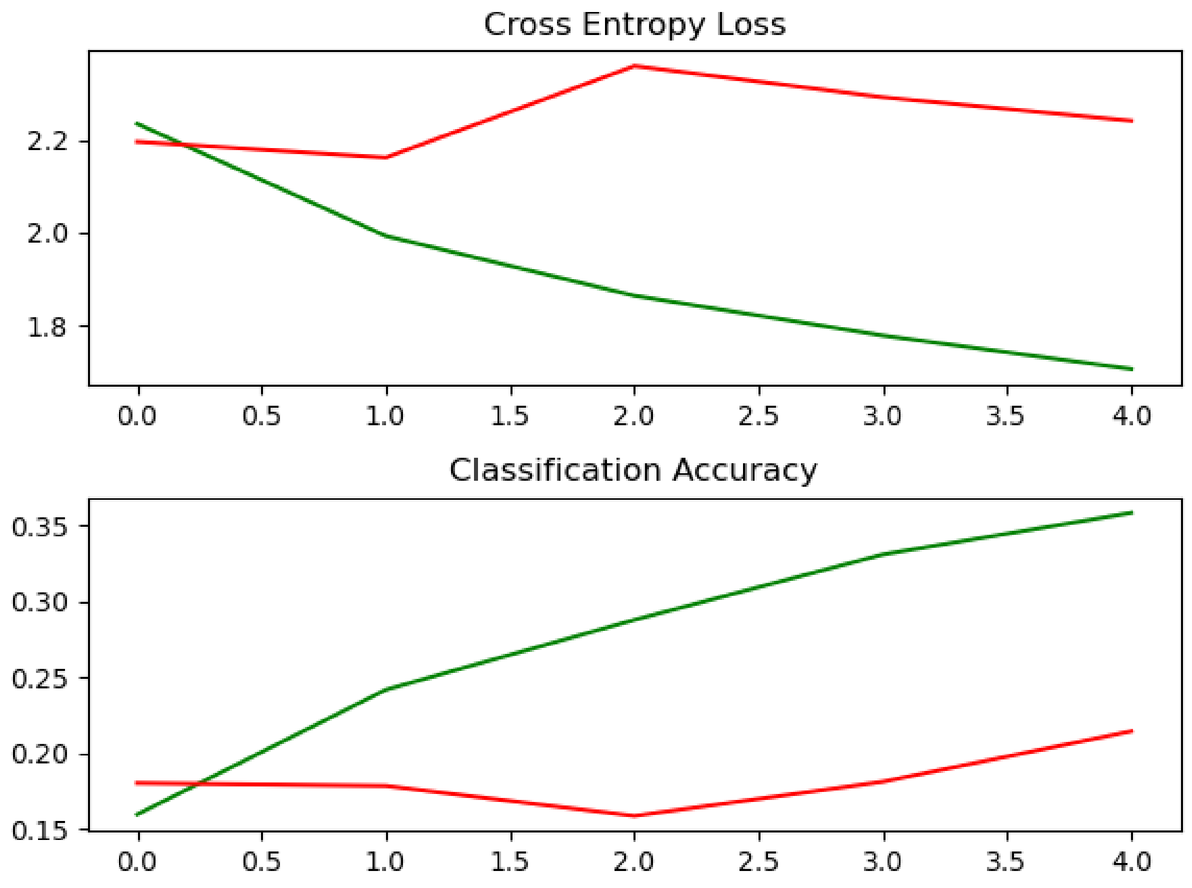
40% dropout:



50% dropout:



60% dropout:



Observations and conclusions:

- adding dropout layers increases the accuracy, but also increases number of epochs needed to achieve the best performance
- 50% of dropout and over makes accuracy significantly worse, so in following experiments we we're considering dropout only from 10 to 40%

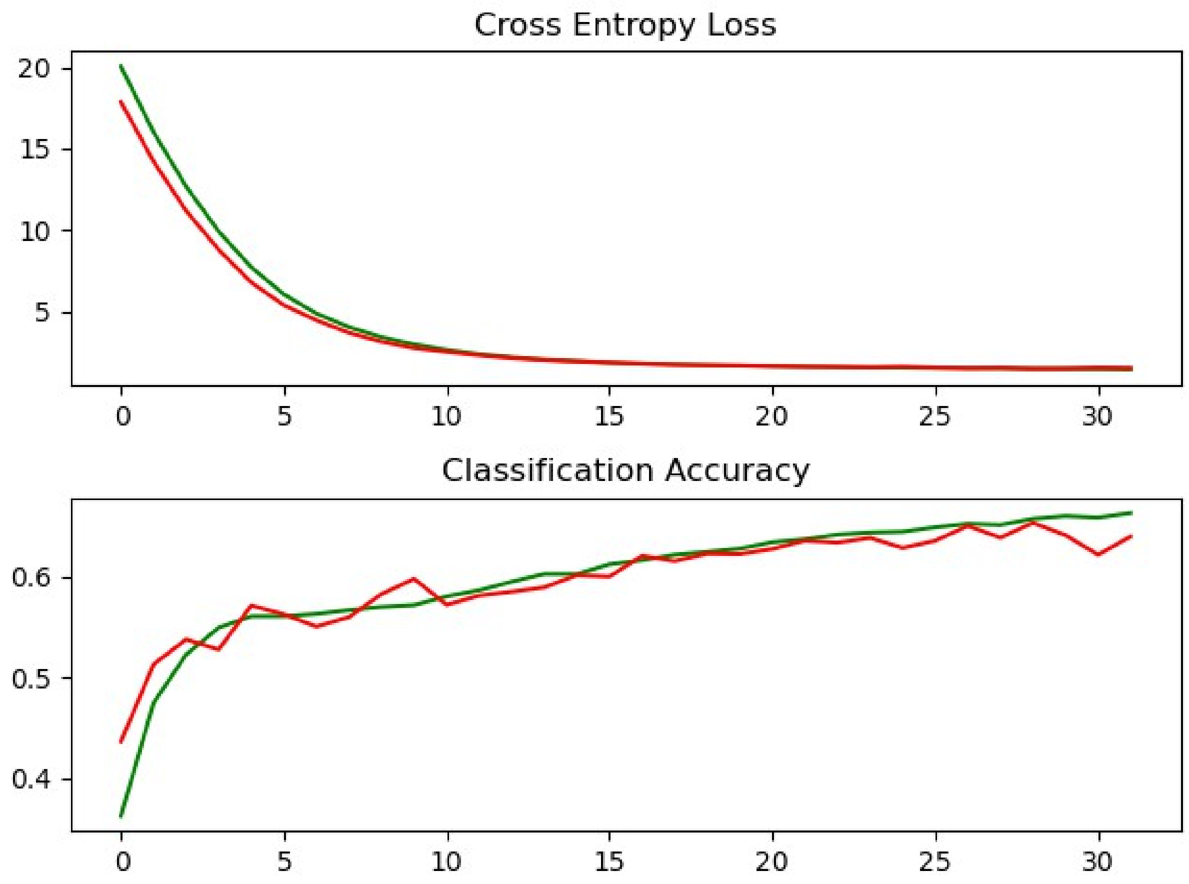
c) weight decay

Using weight decay we can use different kernel_regularizers: l1, l2, l1_l2, and we can use different values.

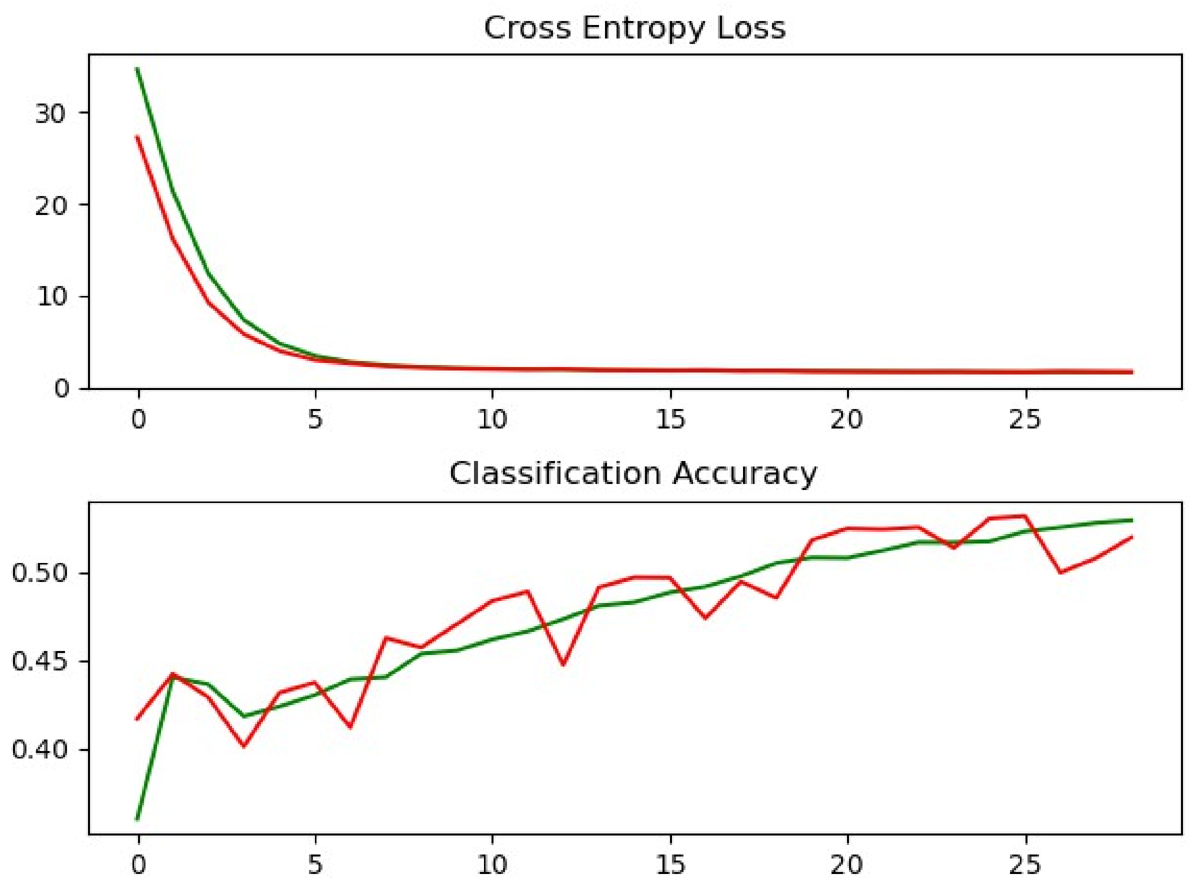
Results for regularizer l1:

WeightDecay Value	Accuracy
0.006	**35.550**
0.007	33.990
0.008	32.580
0.009	33.460
0.01	30.890
0.02	10

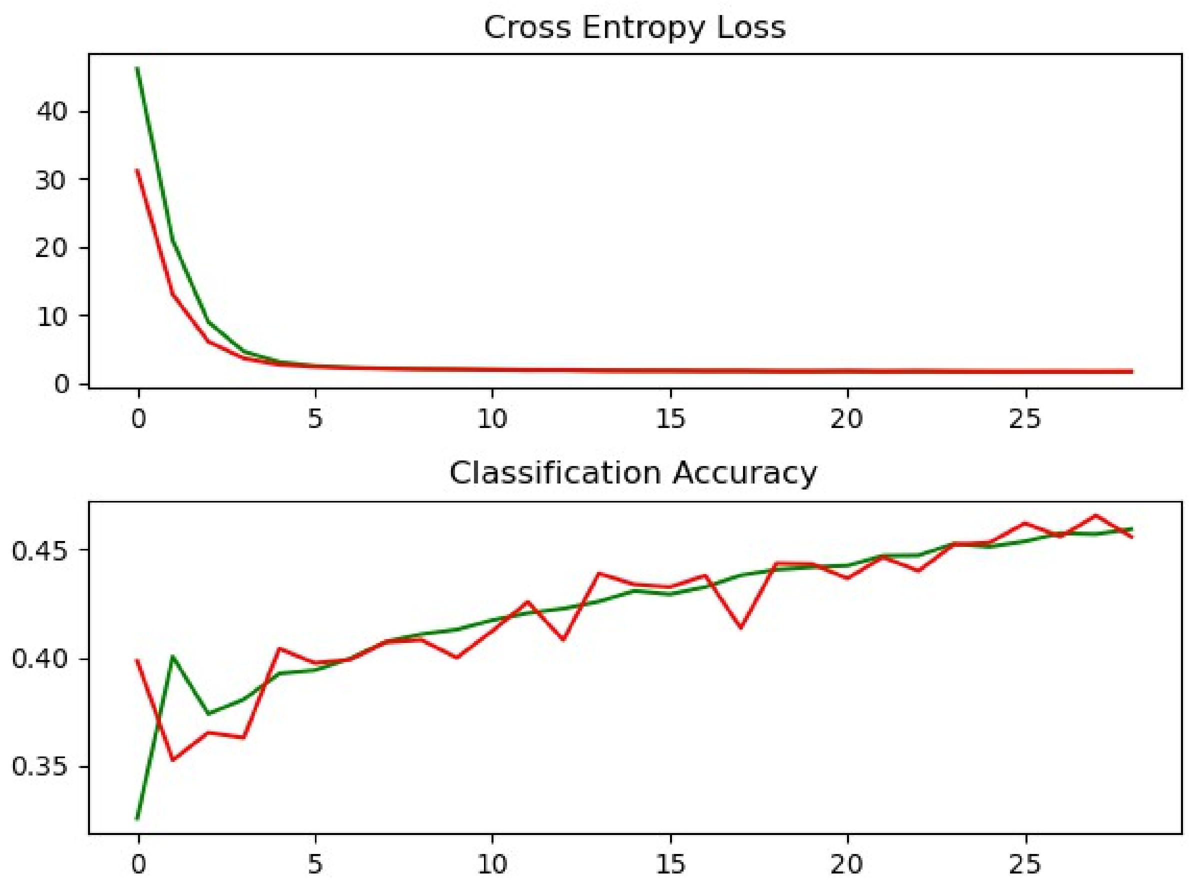
Weight decay 0.001:



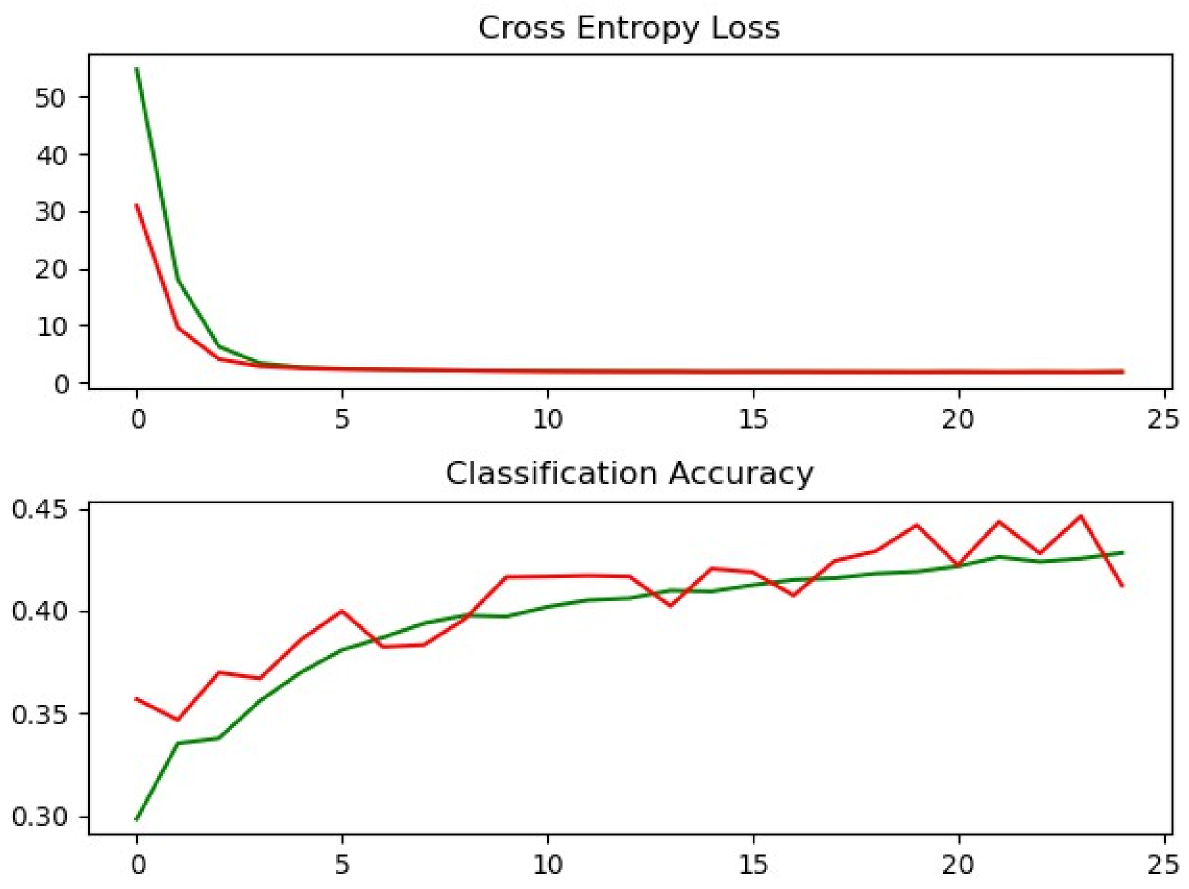
Weight decay 0.002:



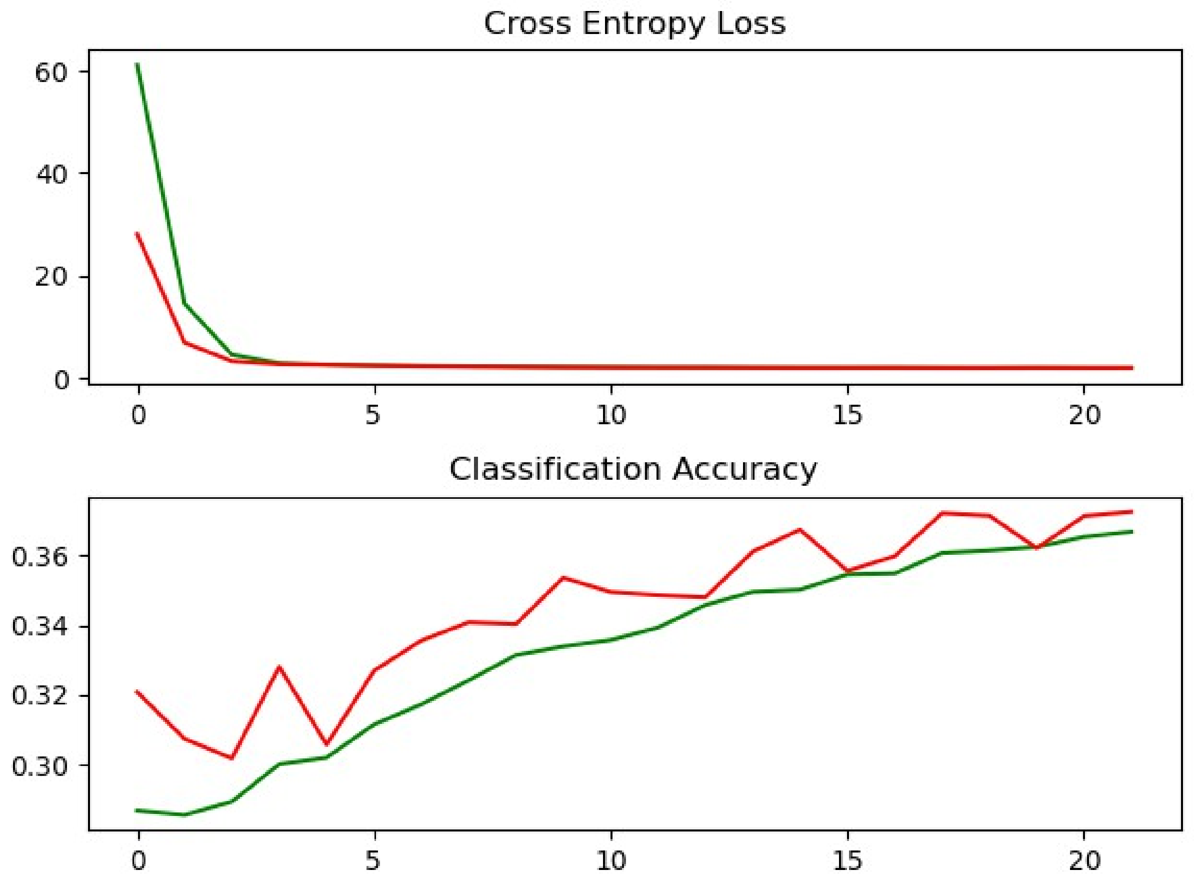
Weight decay 0.003:



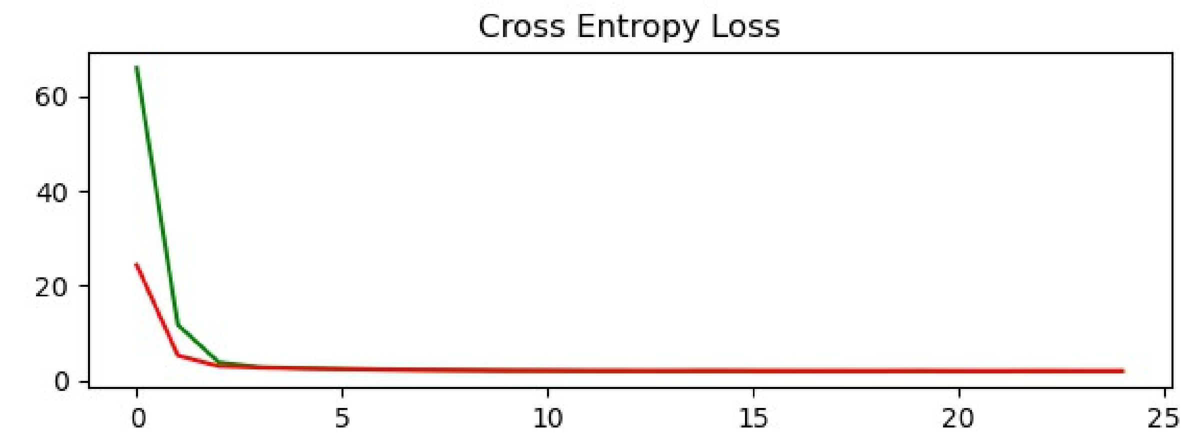
Weight decay 0.004:

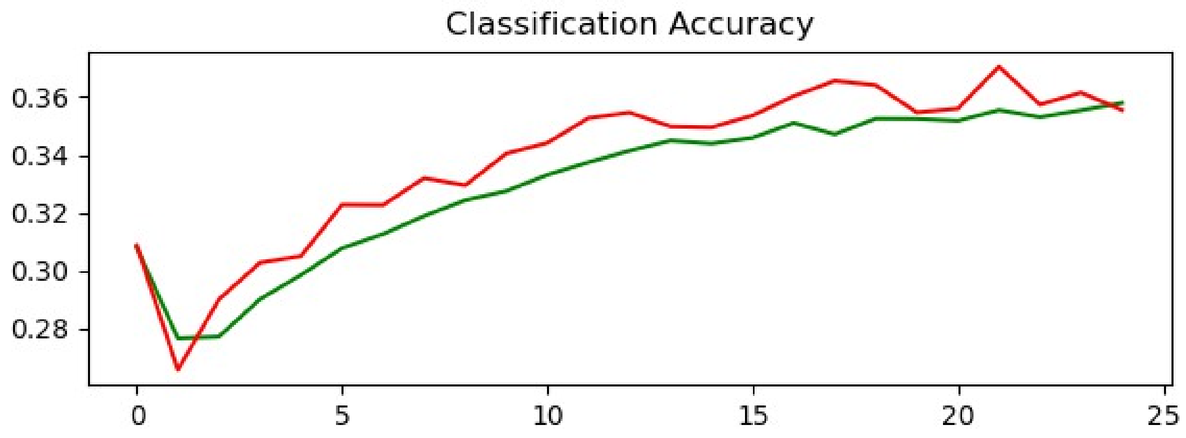


Weight decay 0.005:

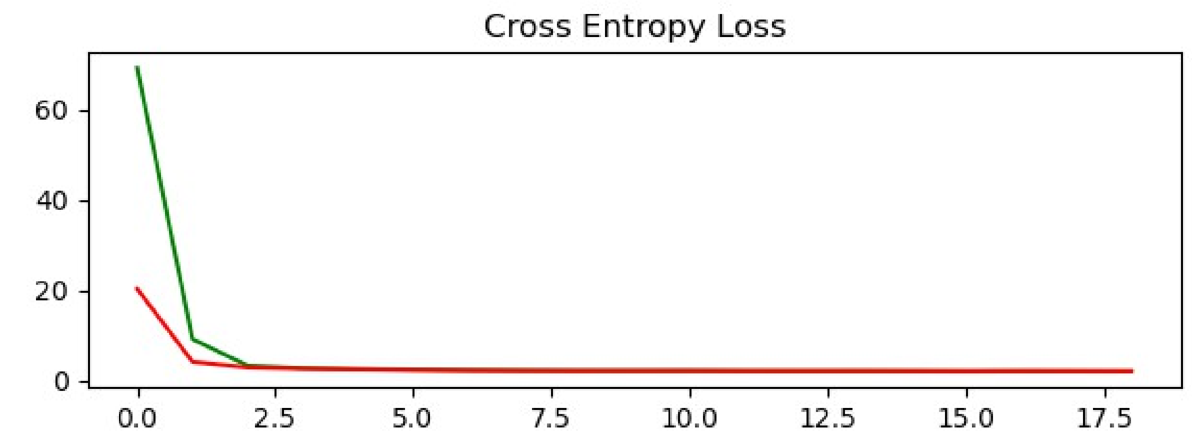


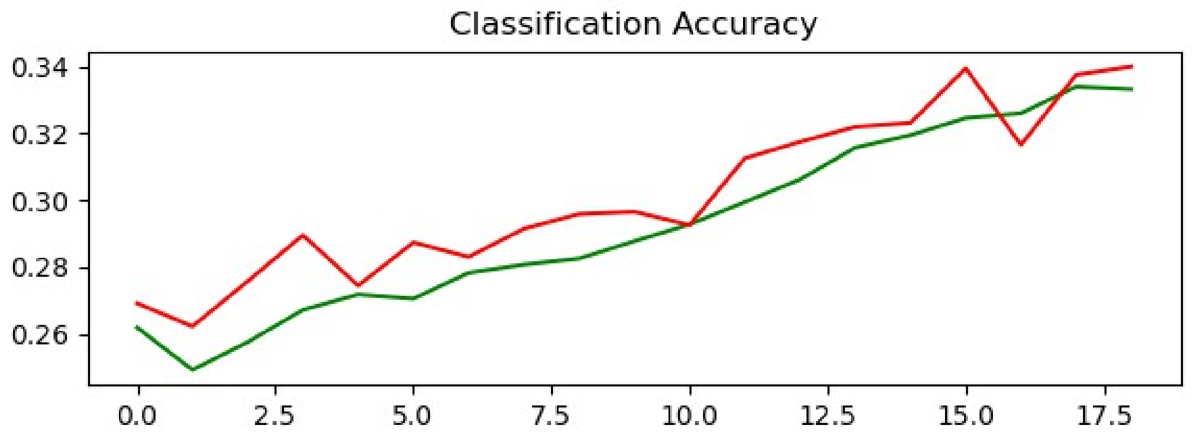
Weight decay 0.006:



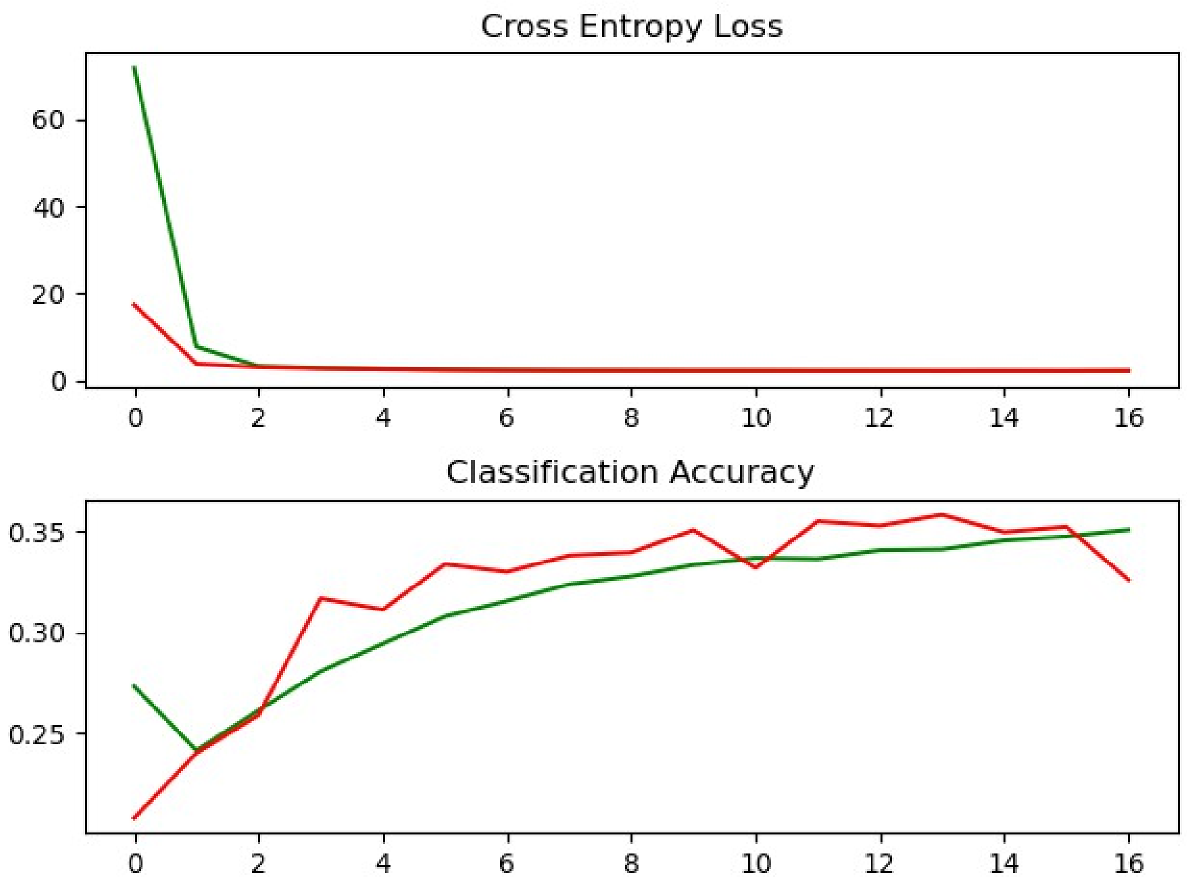


Weight decay 0.007:

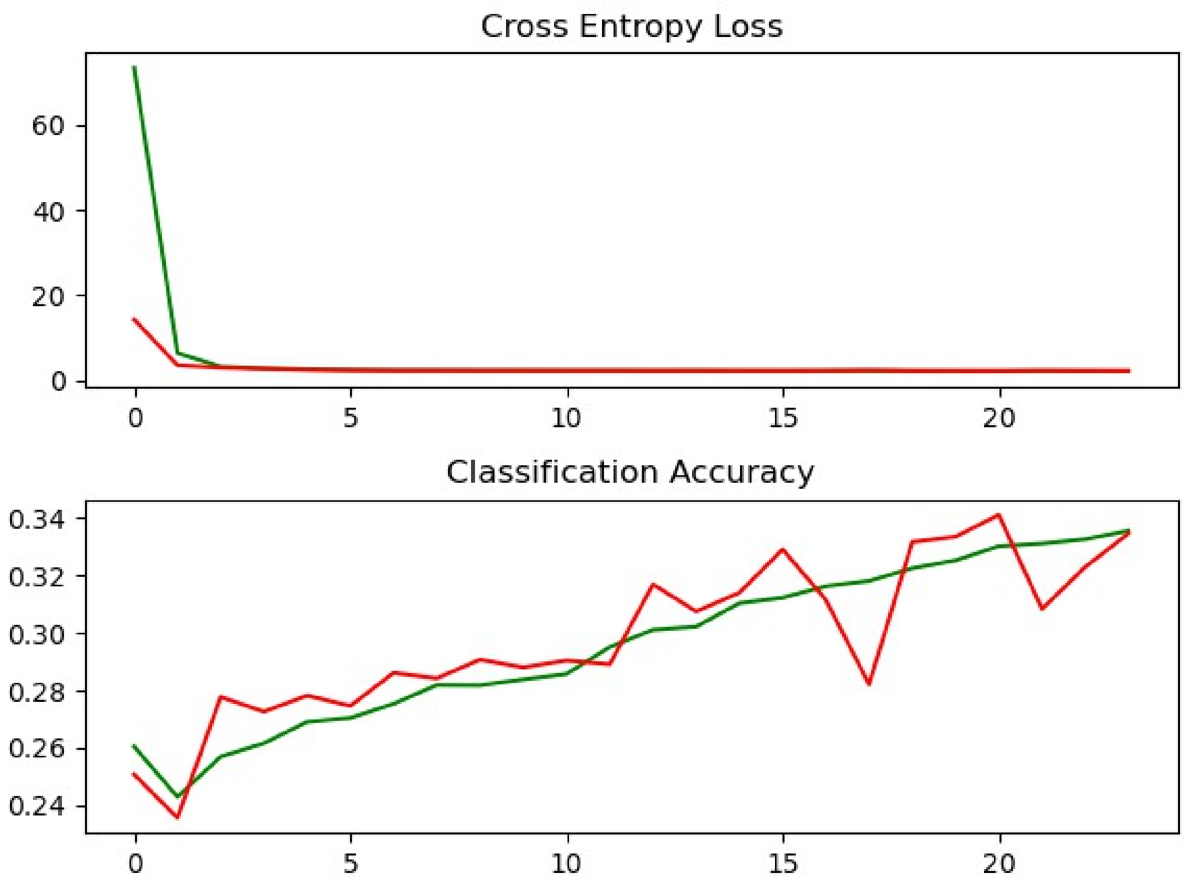




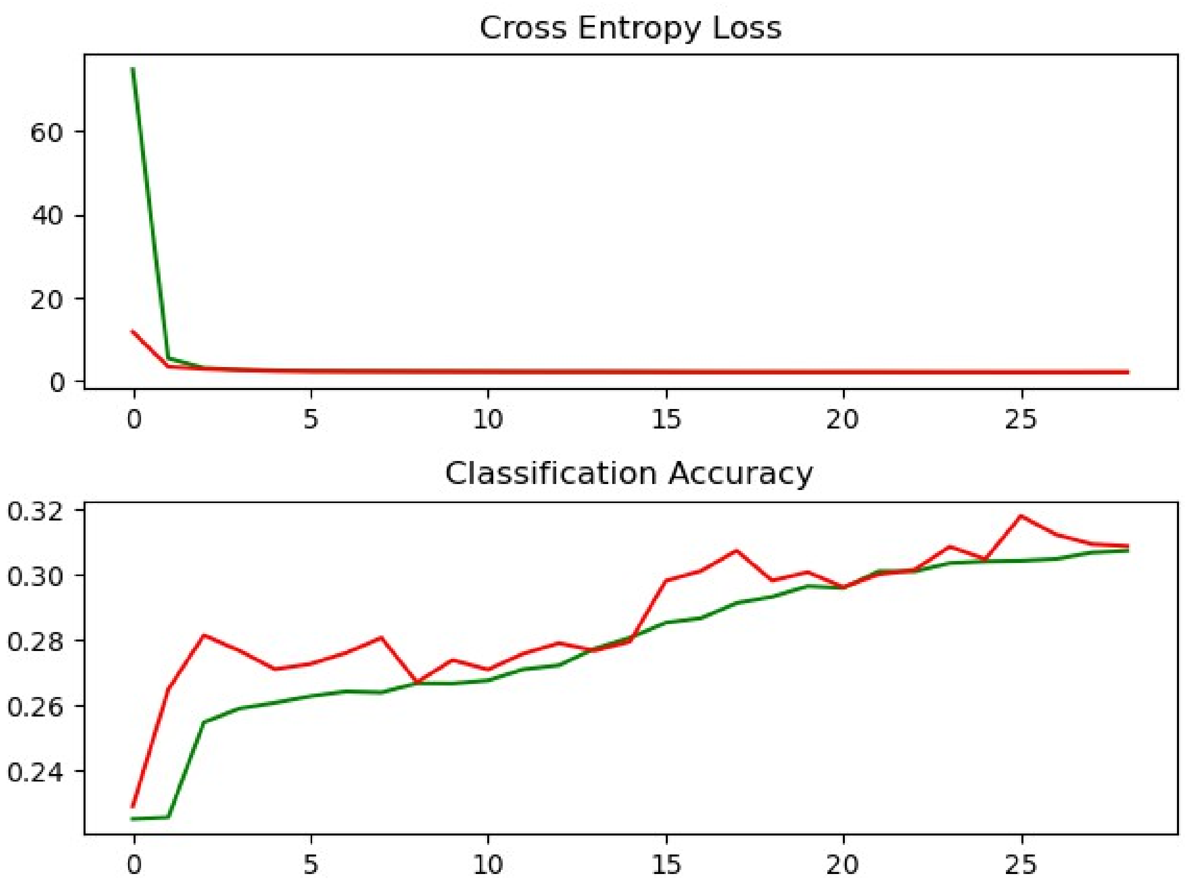
Weight decay 0.008:



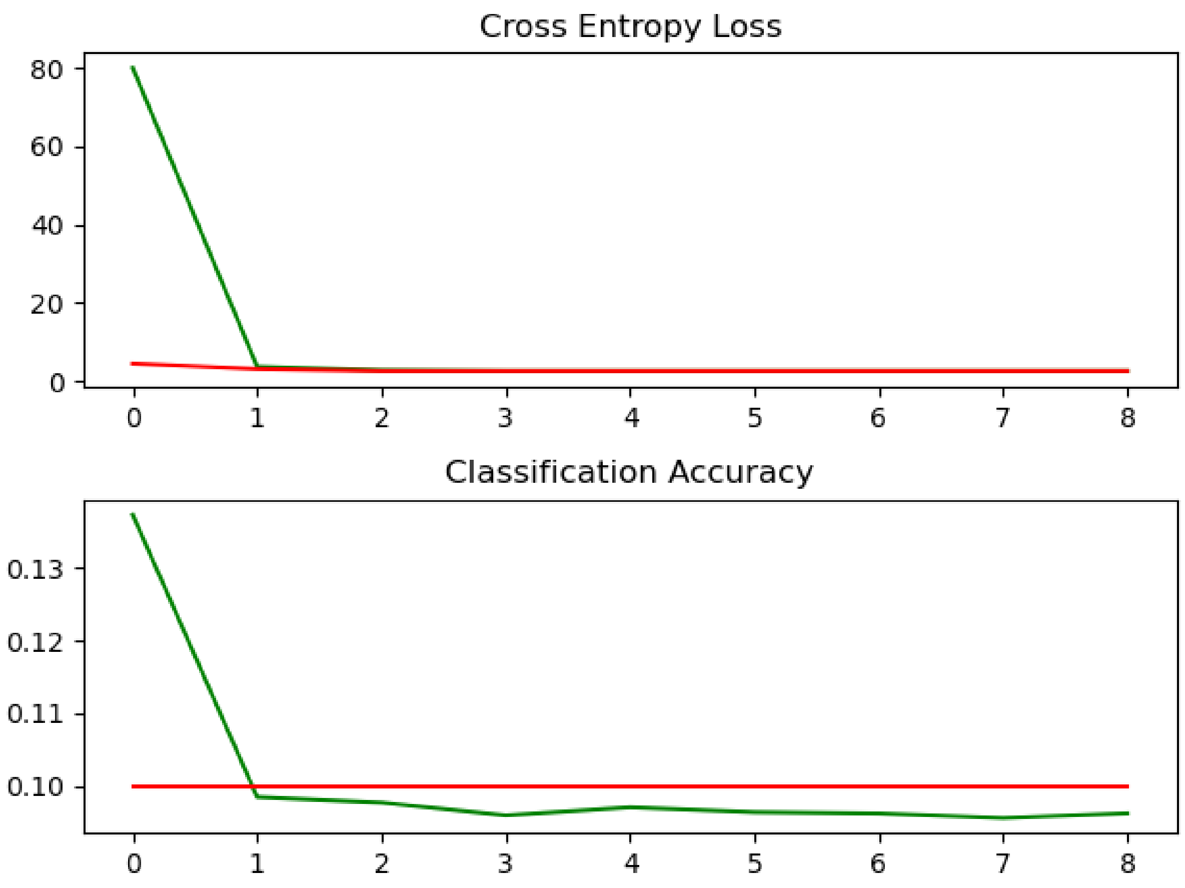
Weight decay 0.009:



Weight decay 0.01:



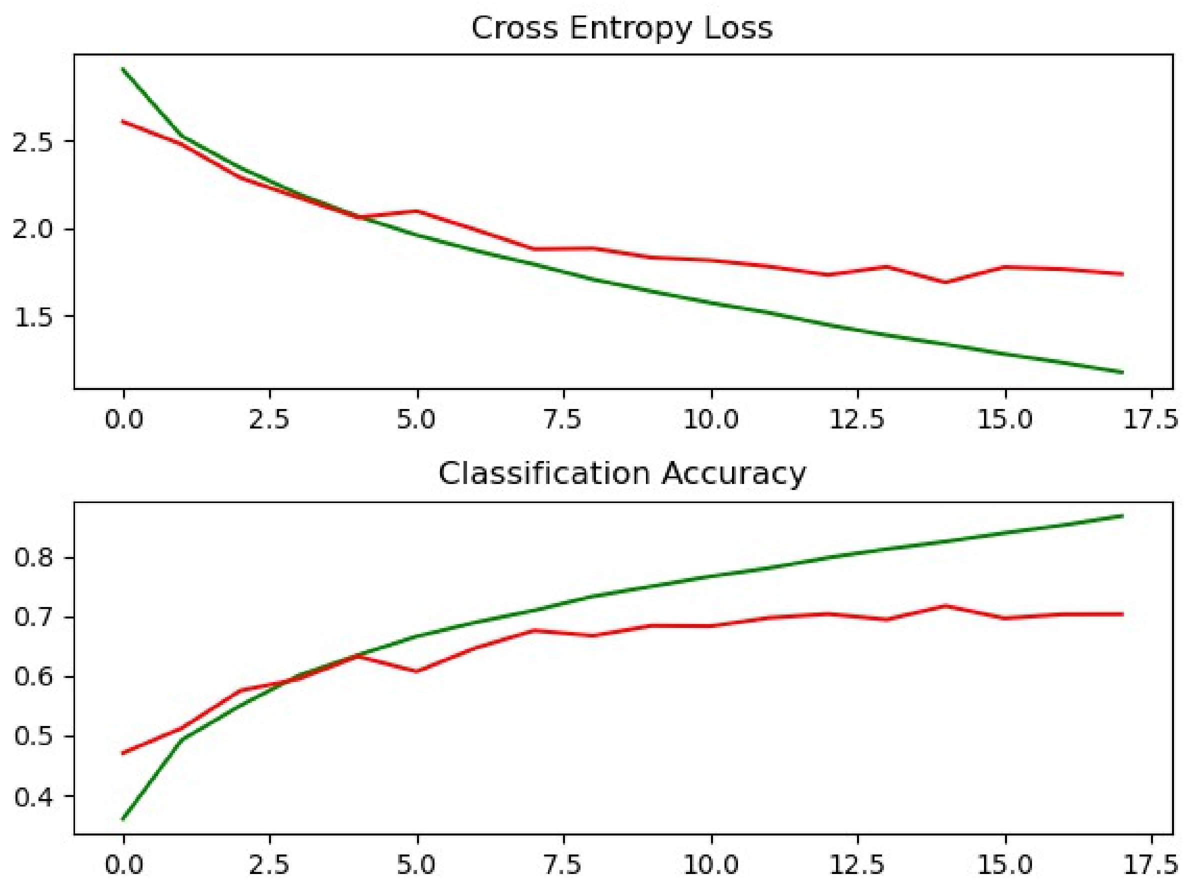
Weight decay 0.02:



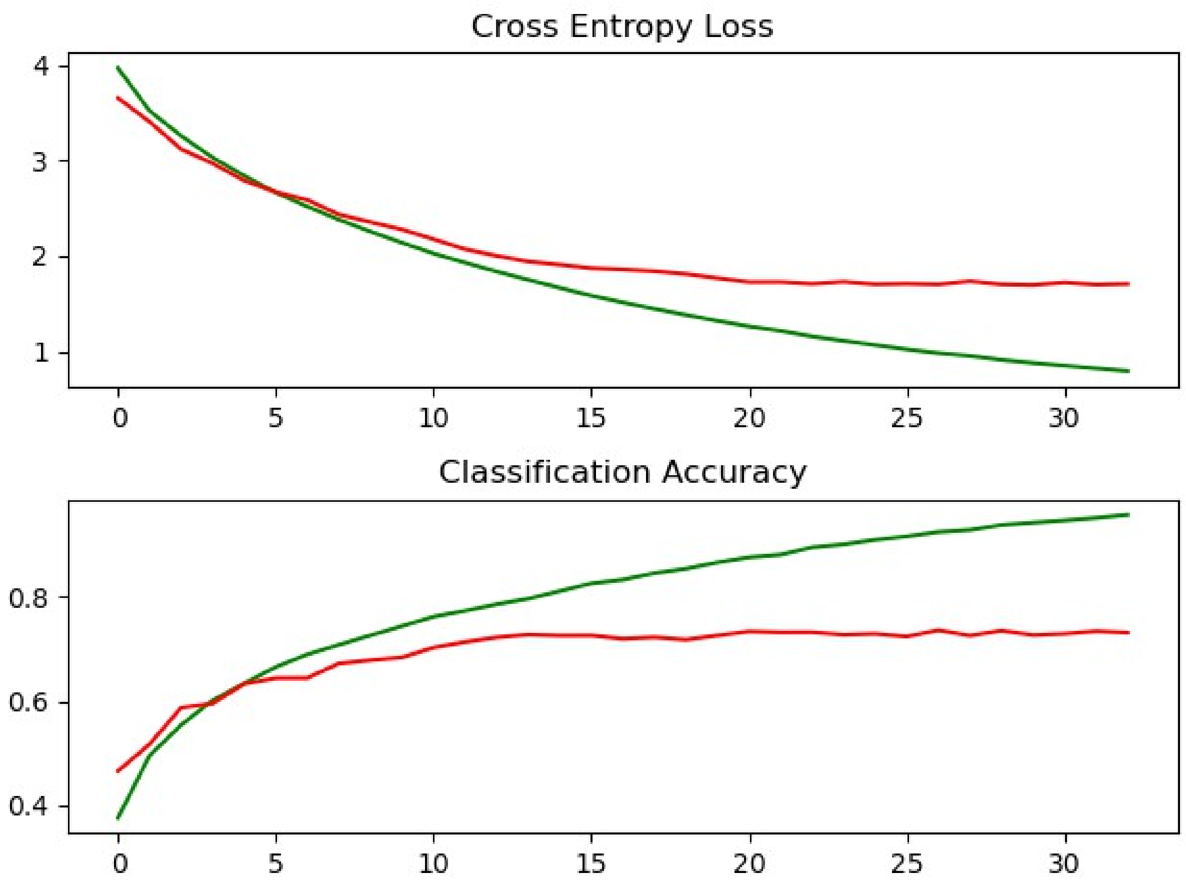
Results for regularizer l2:

WeightDecay Value	Accuracy
0.006	**74.370**
0.007	72.170
0.008	70.540
0.009	70.450
0.01	71.430
0.02	54.430

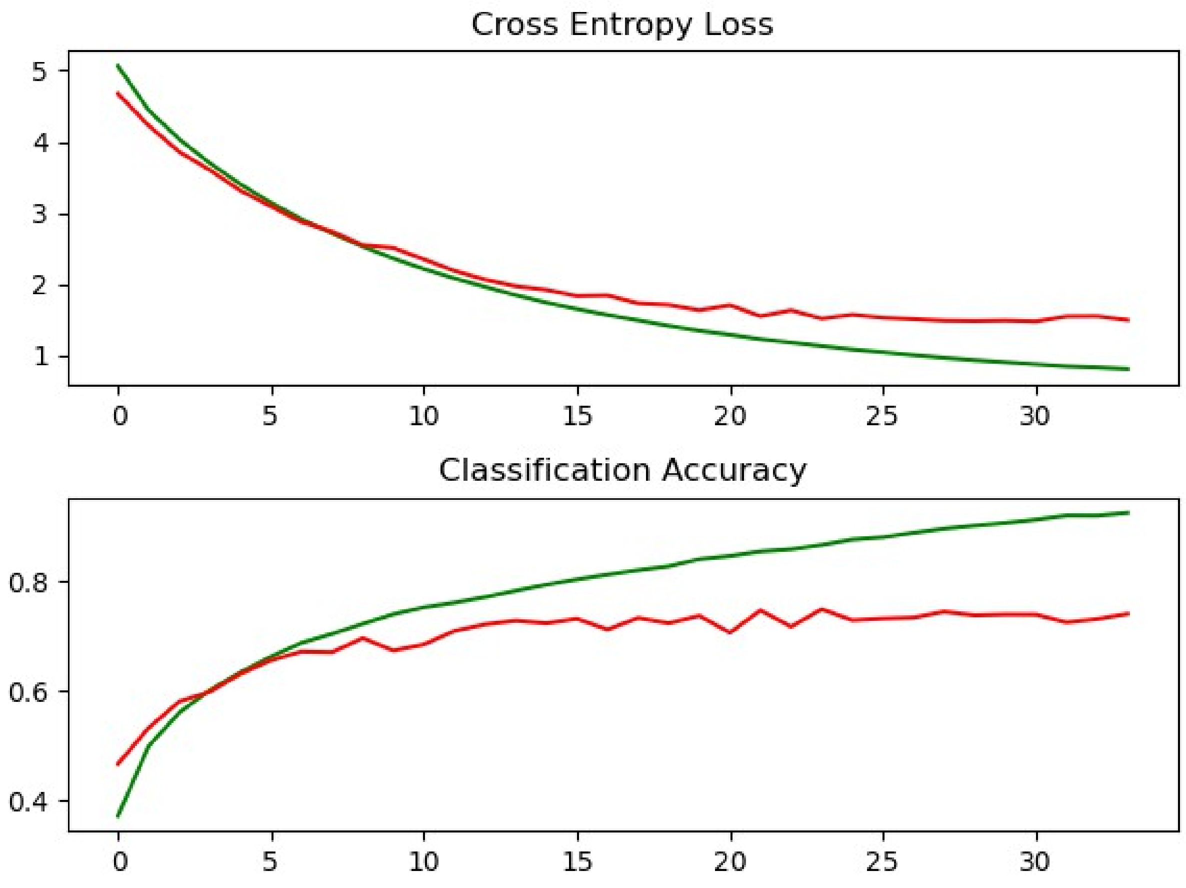
Weight decay 0.001:



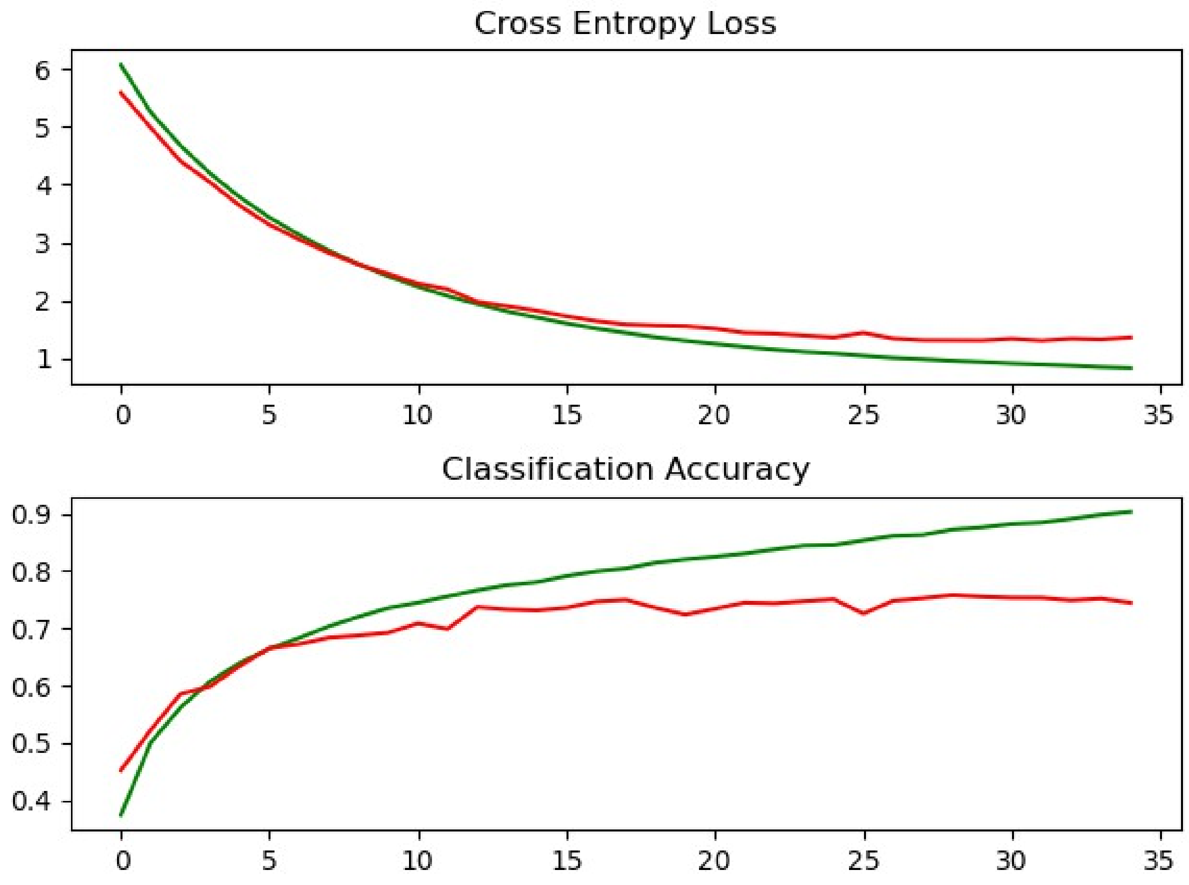
Weight decay 0.002:



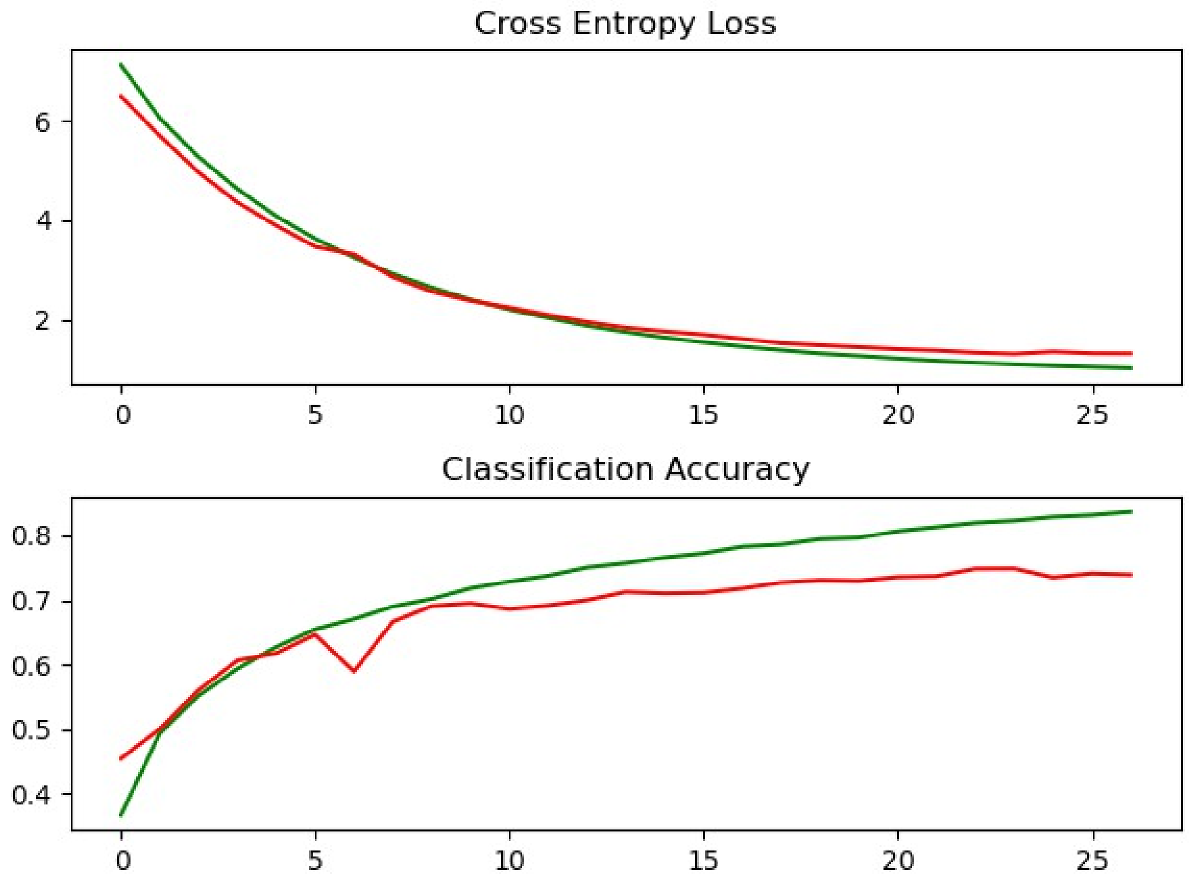
Weight decay 0.003:



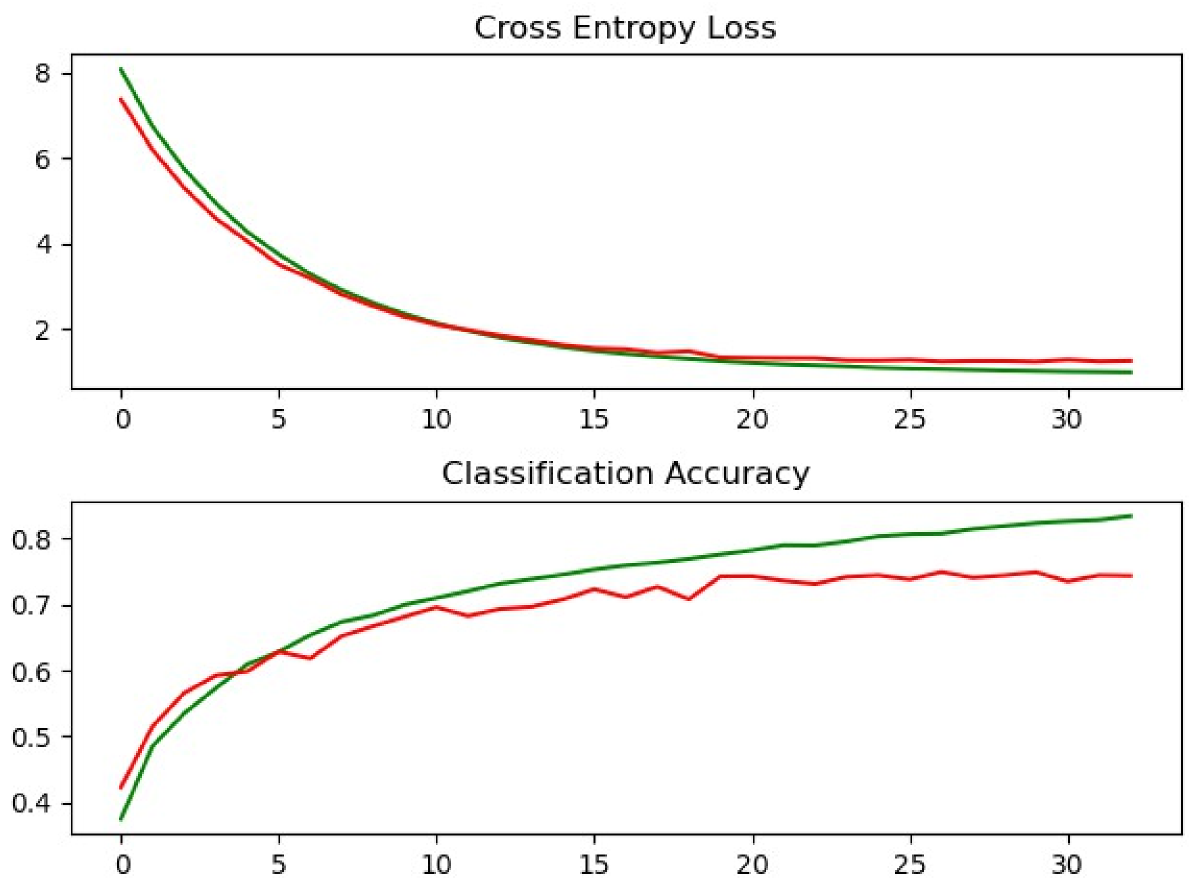
Weight decay 0.004:



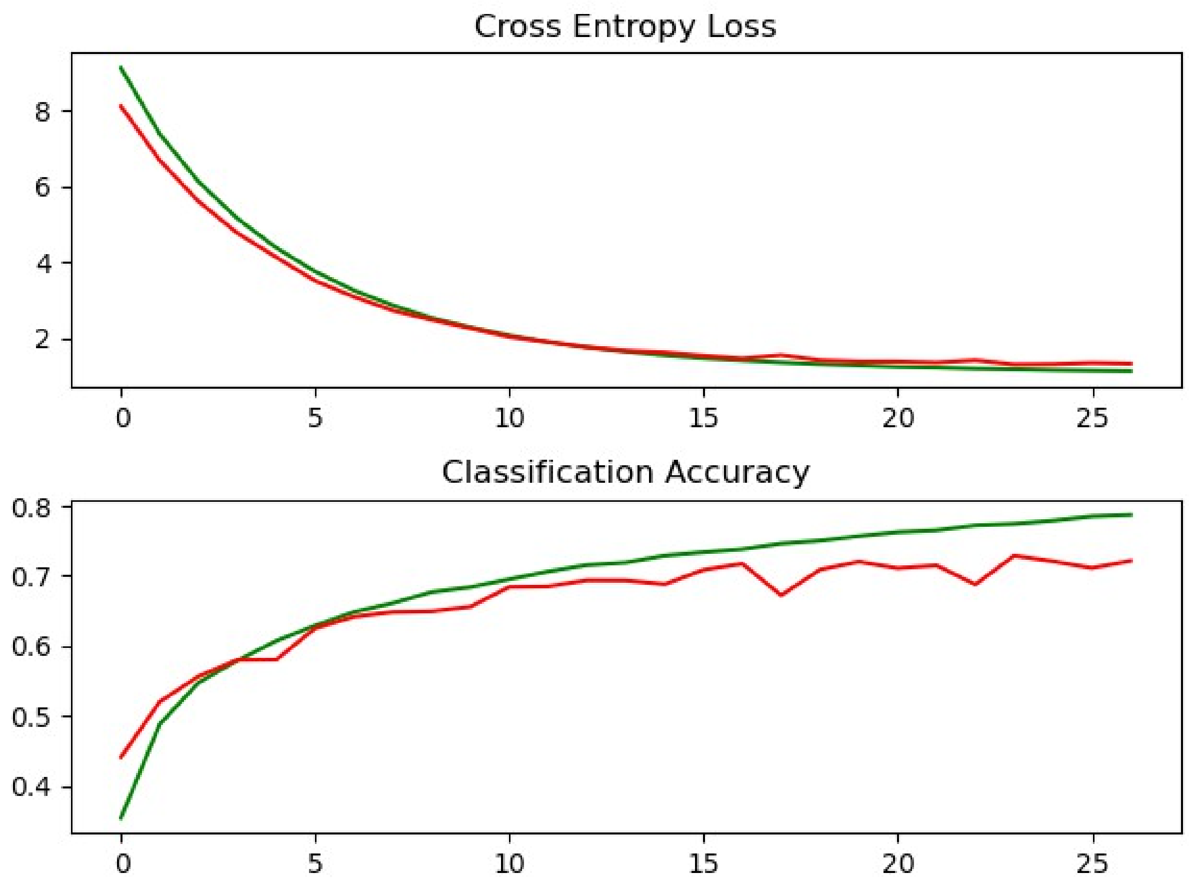
Weight decay 0.005:



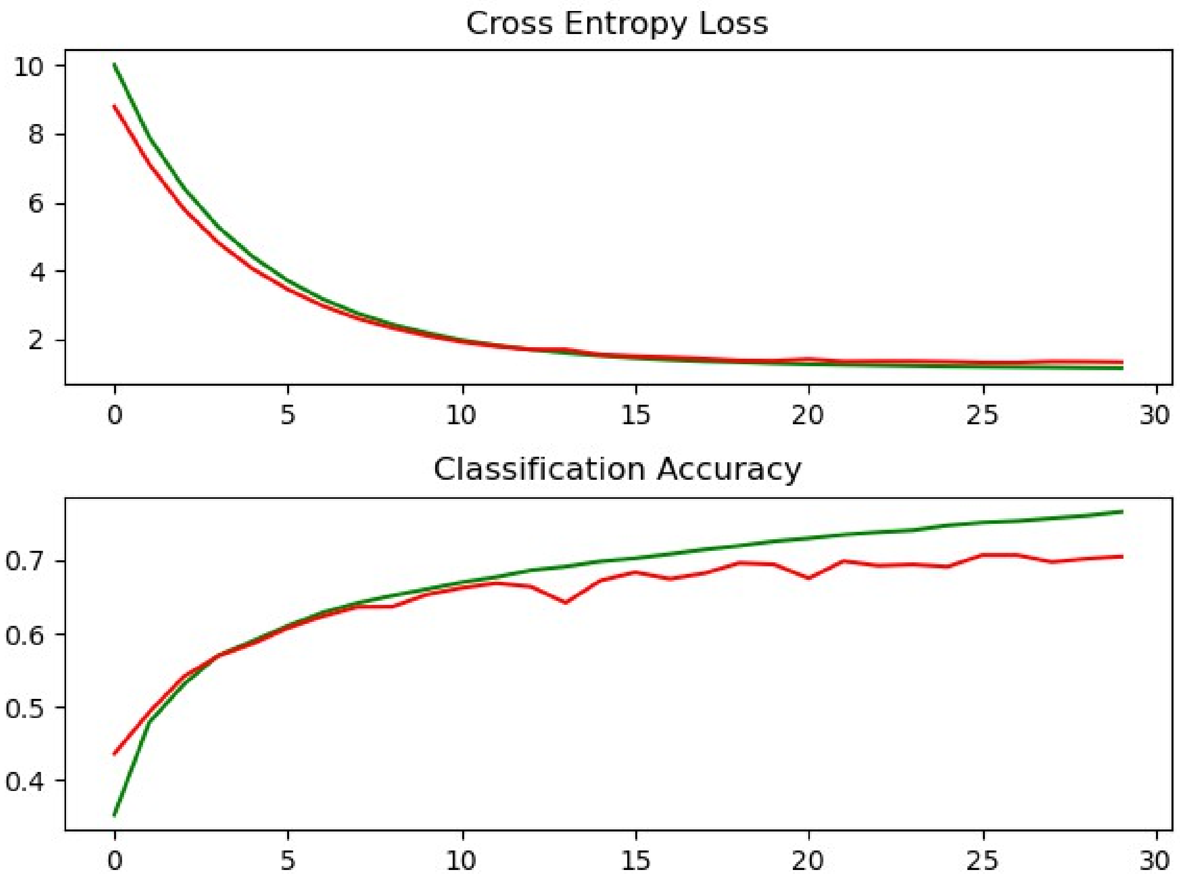
Weight decay 0.006:



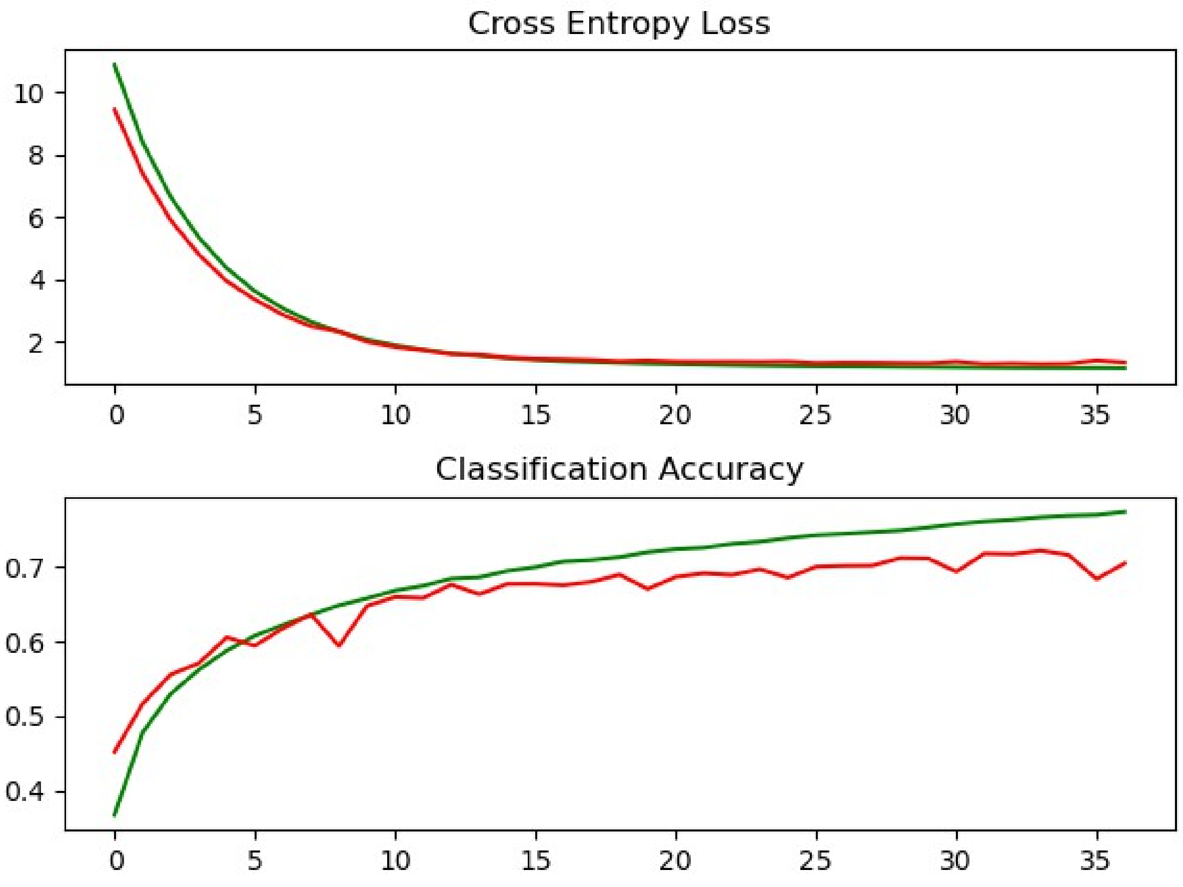
Weight decay 0.007:



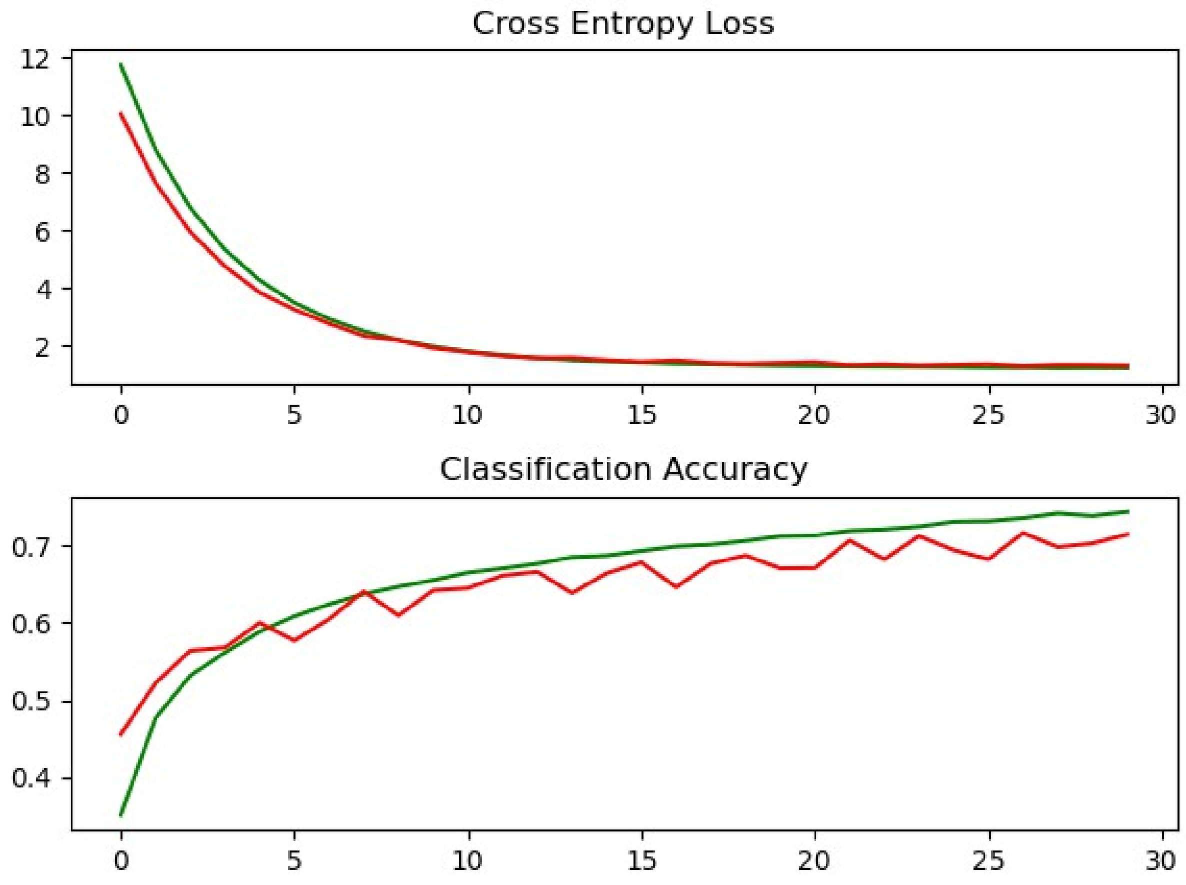
Weight decay 0.008:



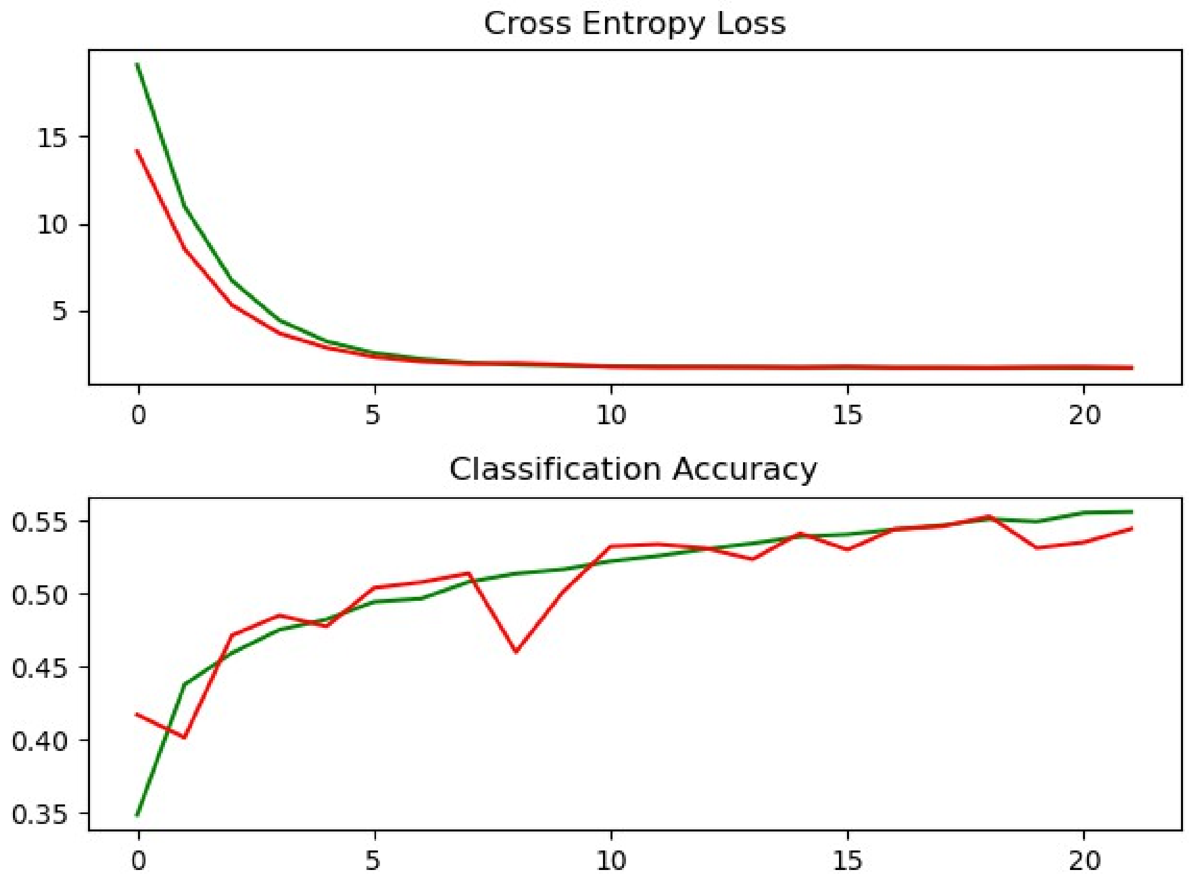
Weight decay 0.009:



Weight decay 0.01:



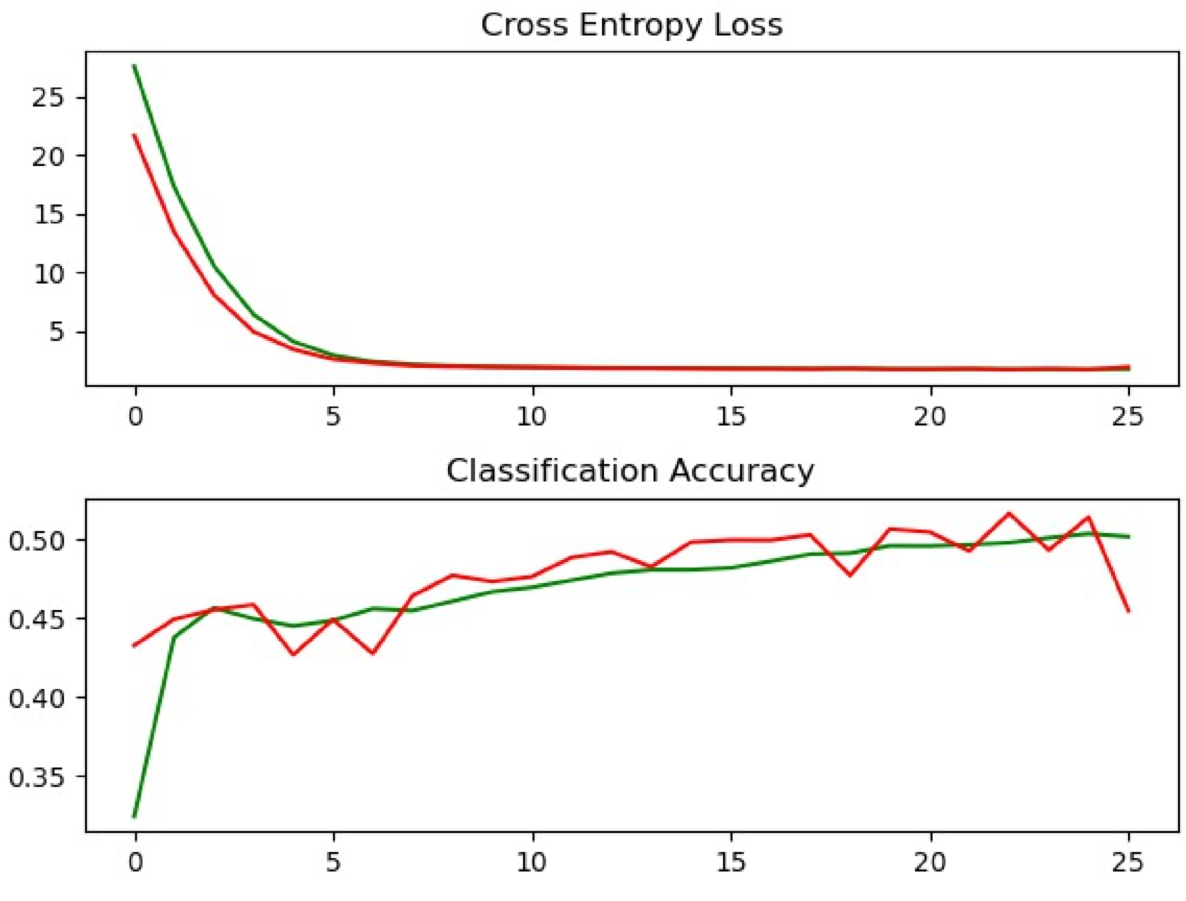
Weight decay 0.02:



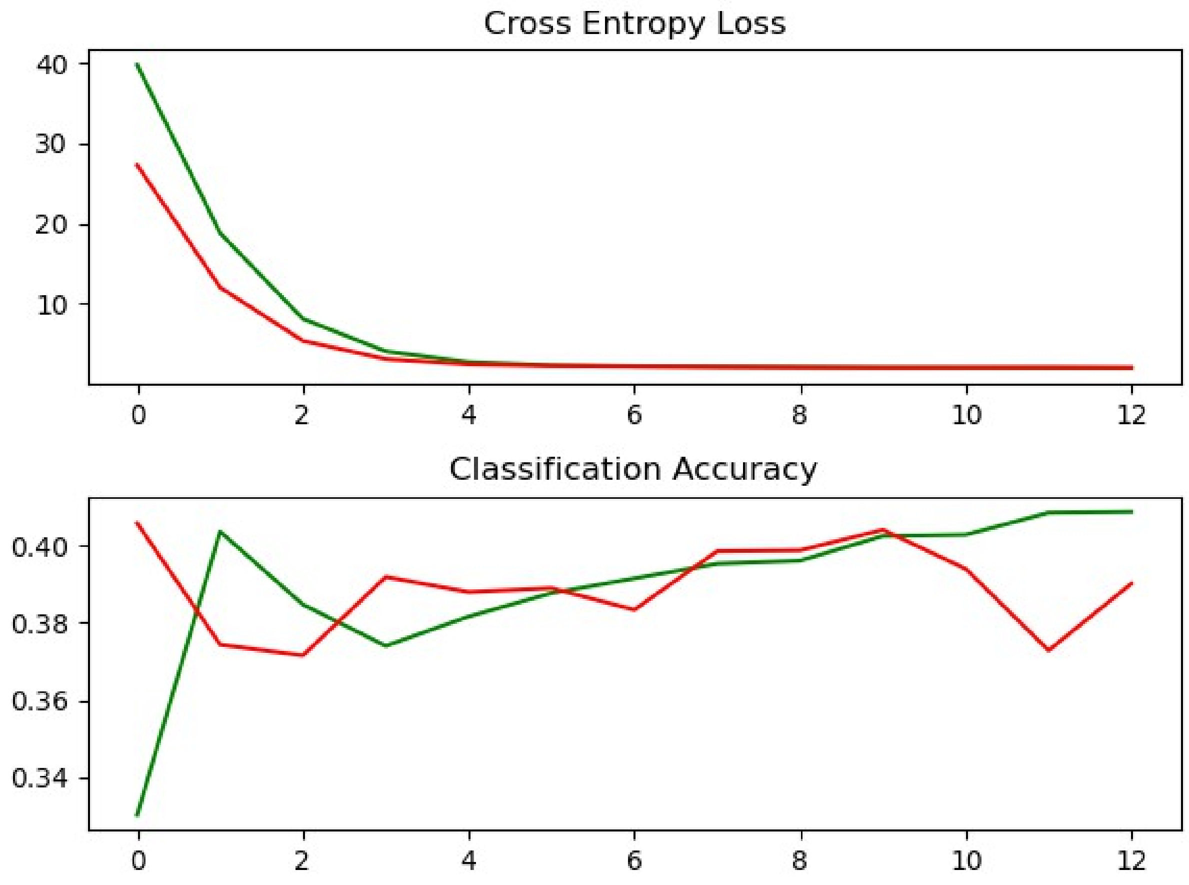
Results for regularizer l1_l2:

WeightDecay Value	Accuracy
0.006	**31.420**
0.007	30.030
0.008	29.350
0.009	10
0.01	10
0.02	10

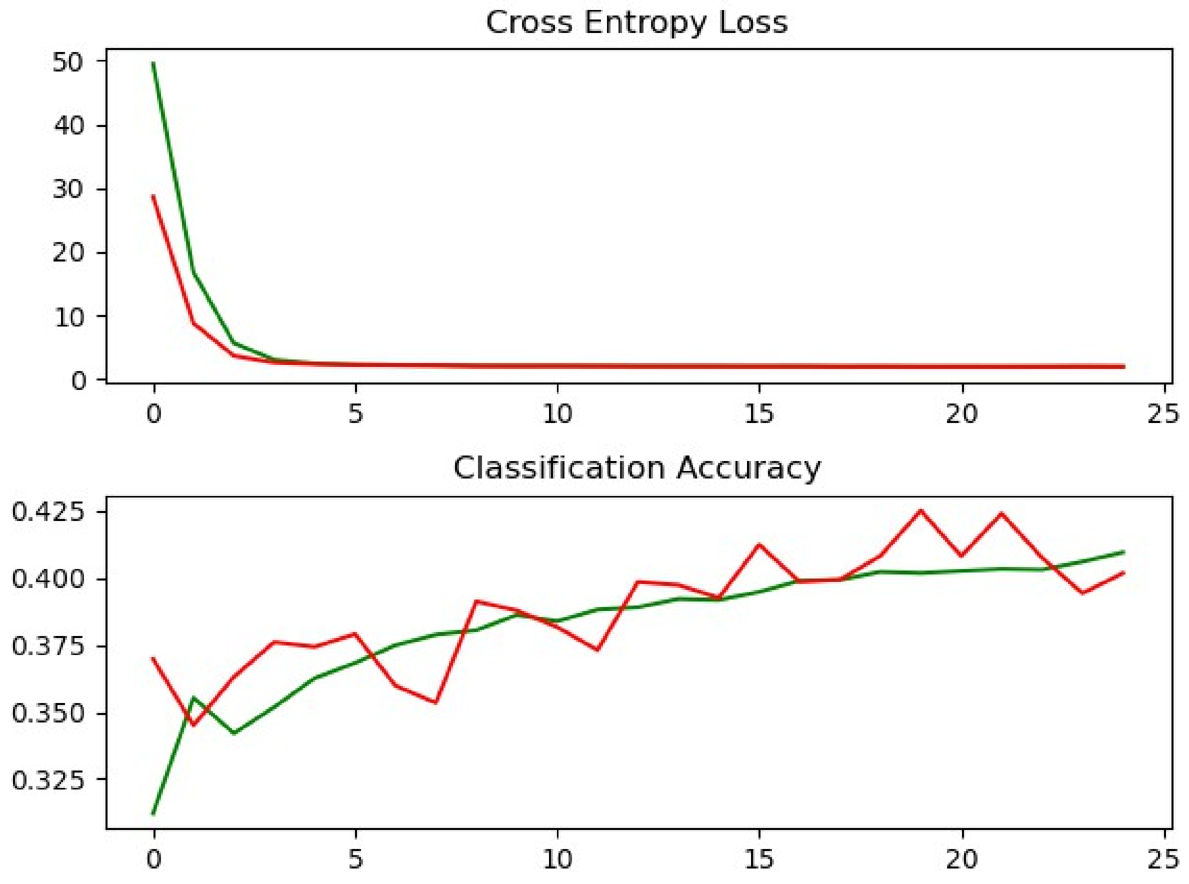
Weight decay 0.001:



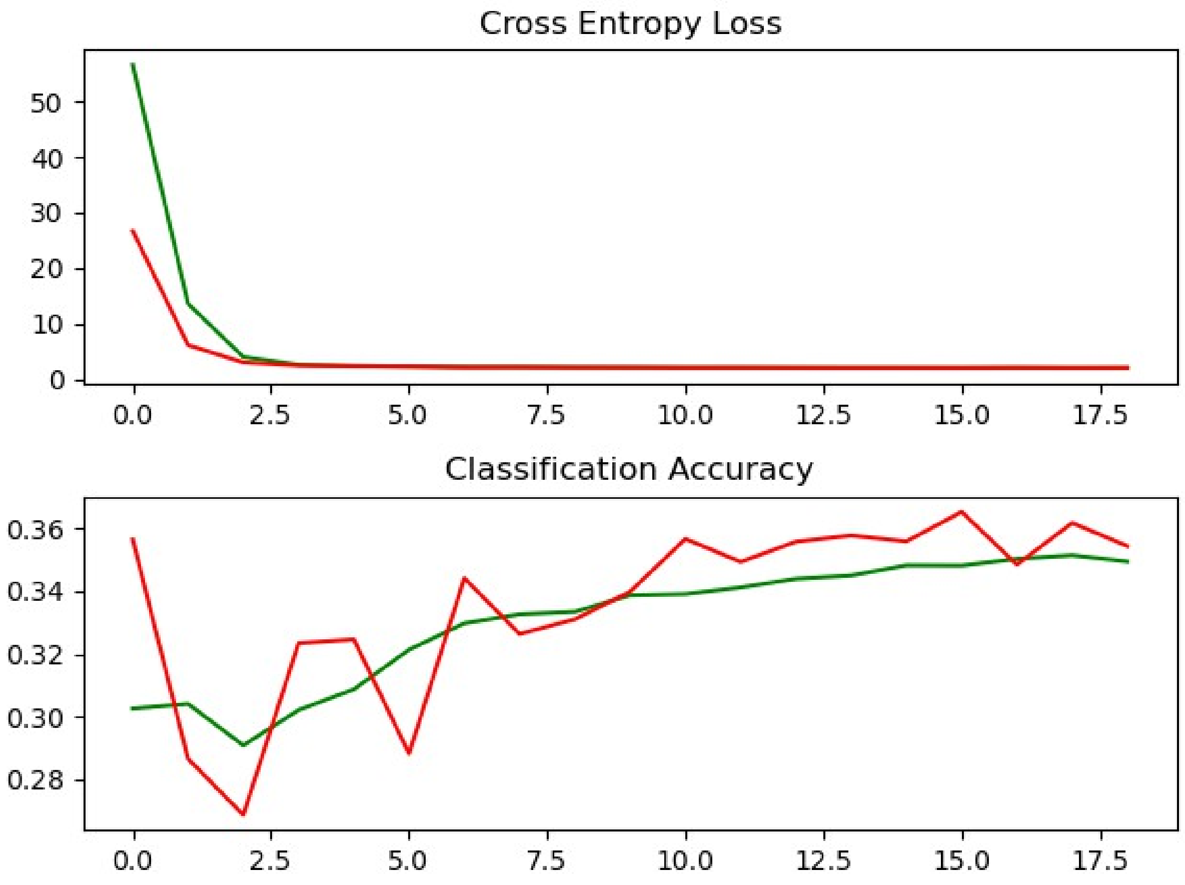
Weight decay 0.002:



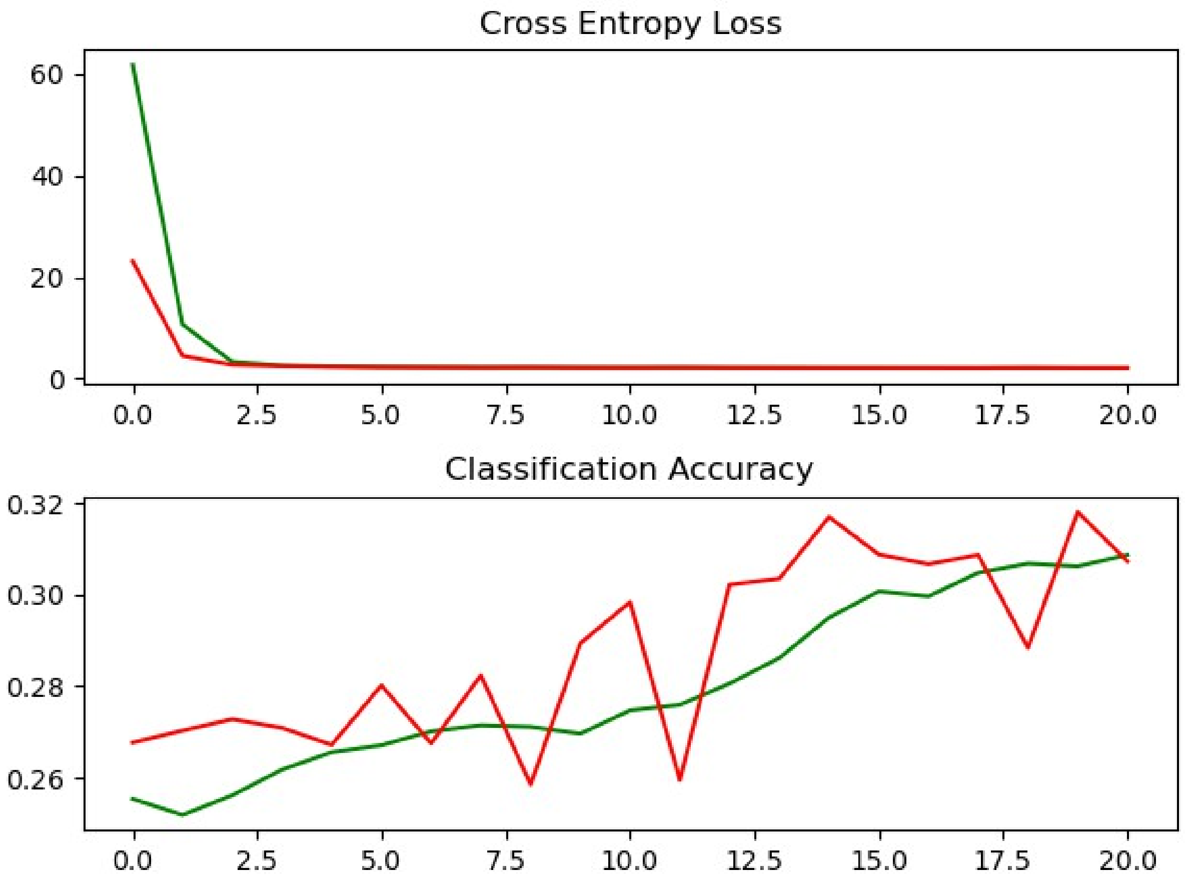
Weight decay 0.003:



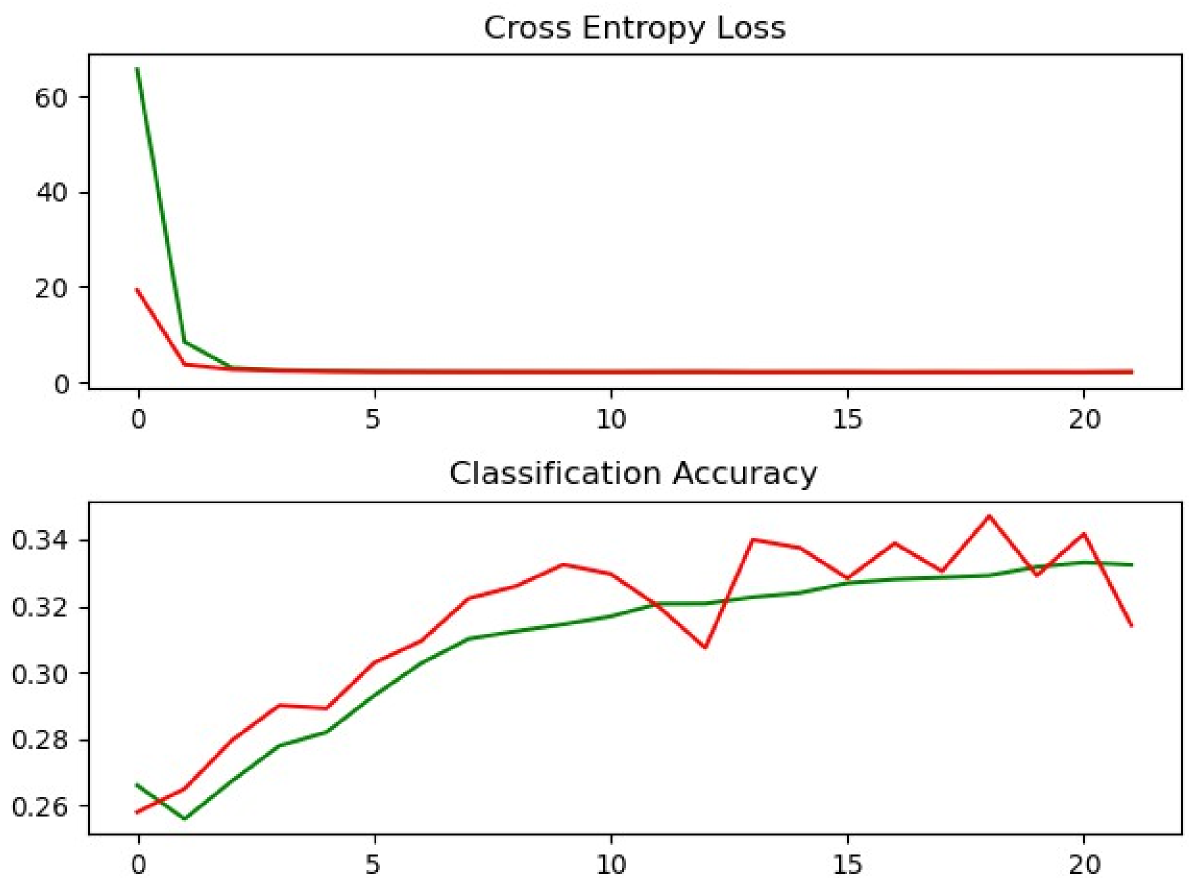
Weight decay 0.004:



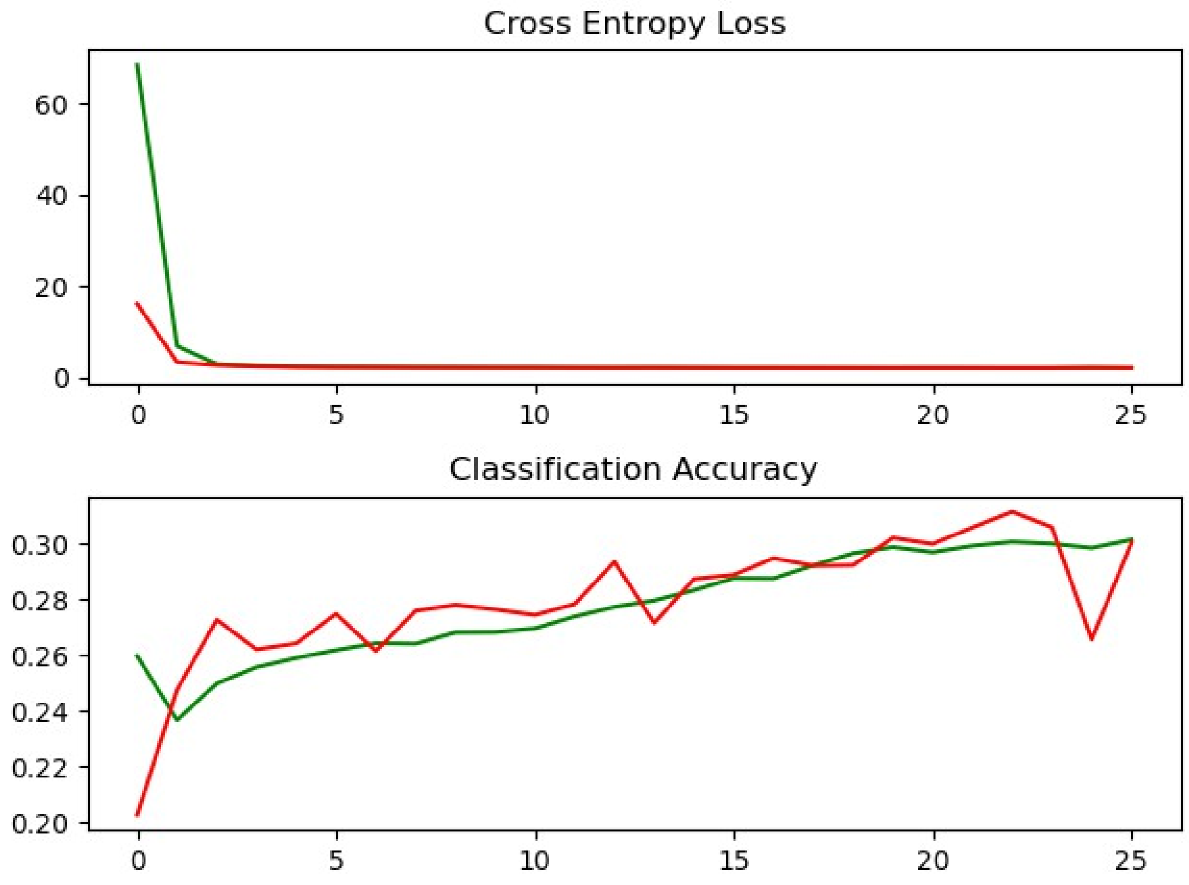
Weight decay 0.005:



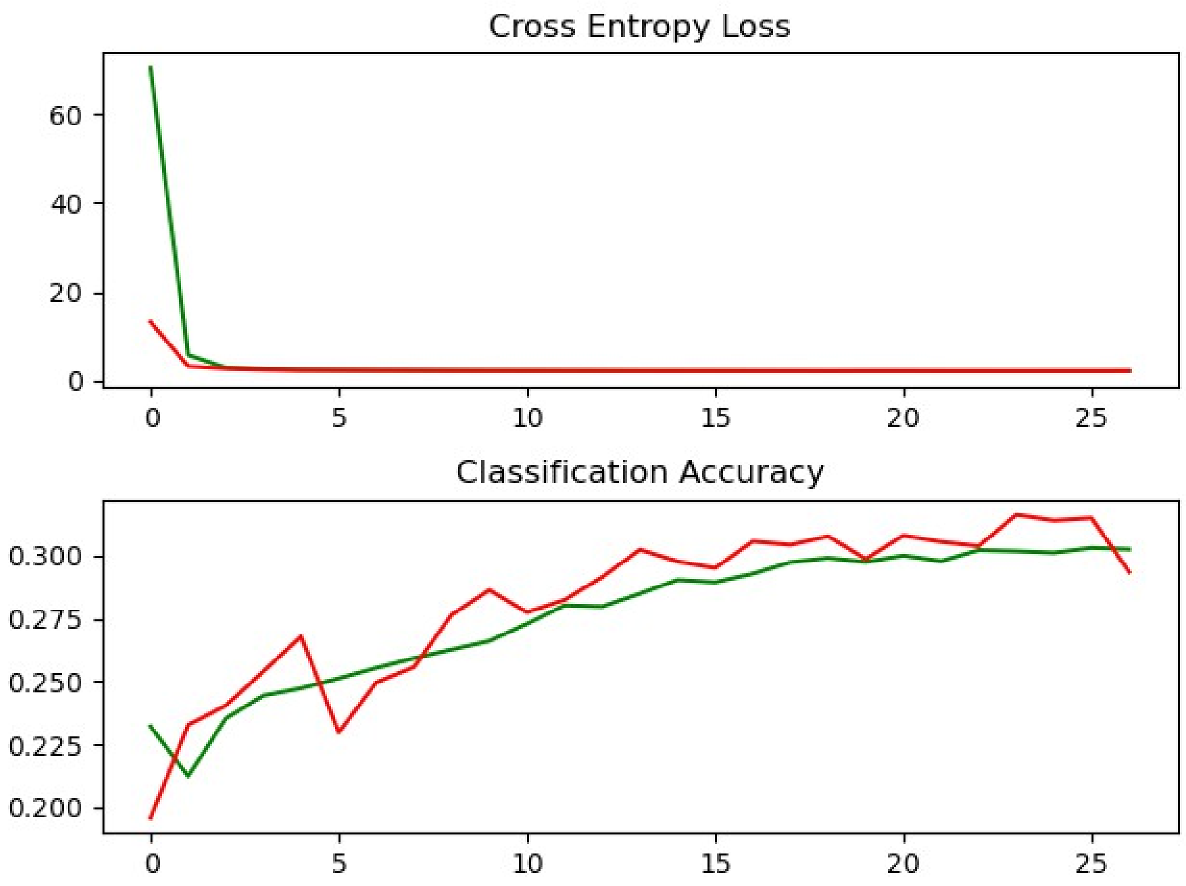
Weight decay 0.006:



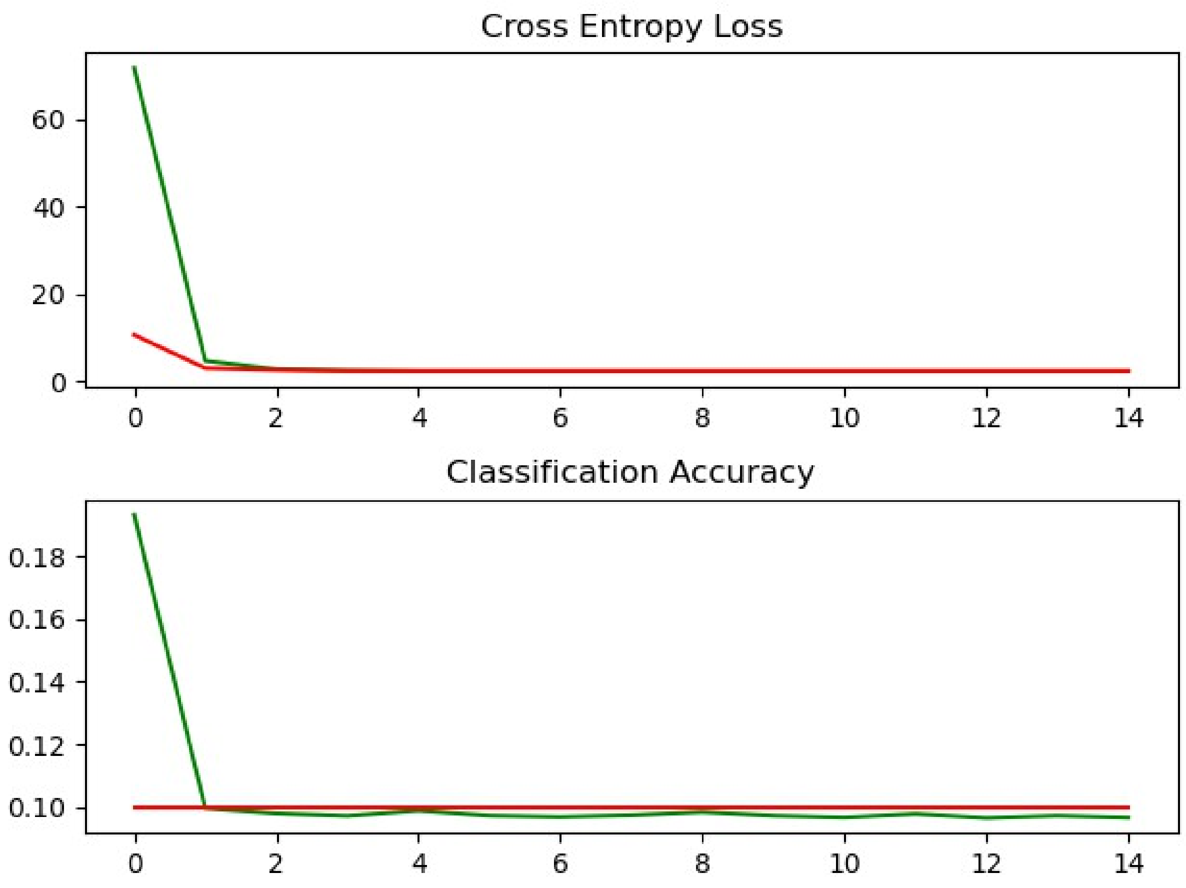
Weight decay 0.007:



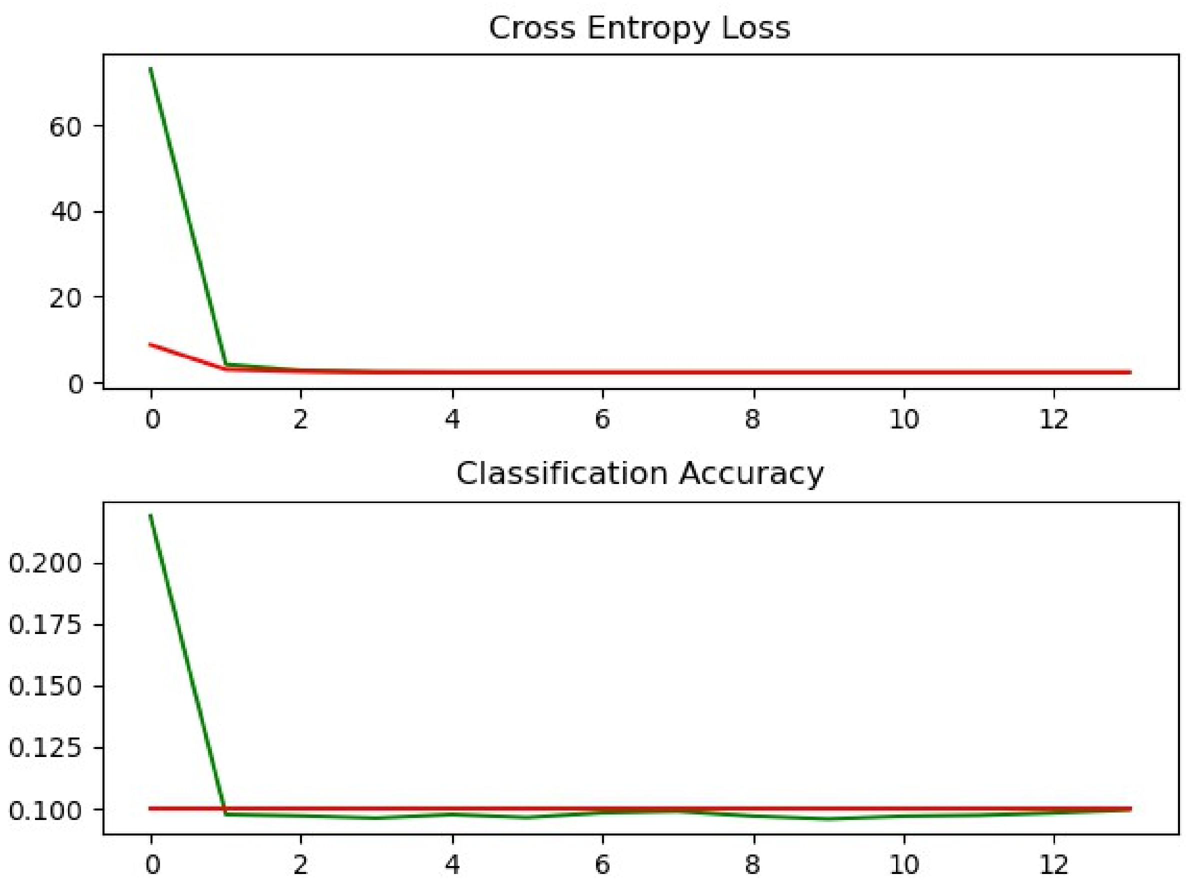
Weight decay 0.008:



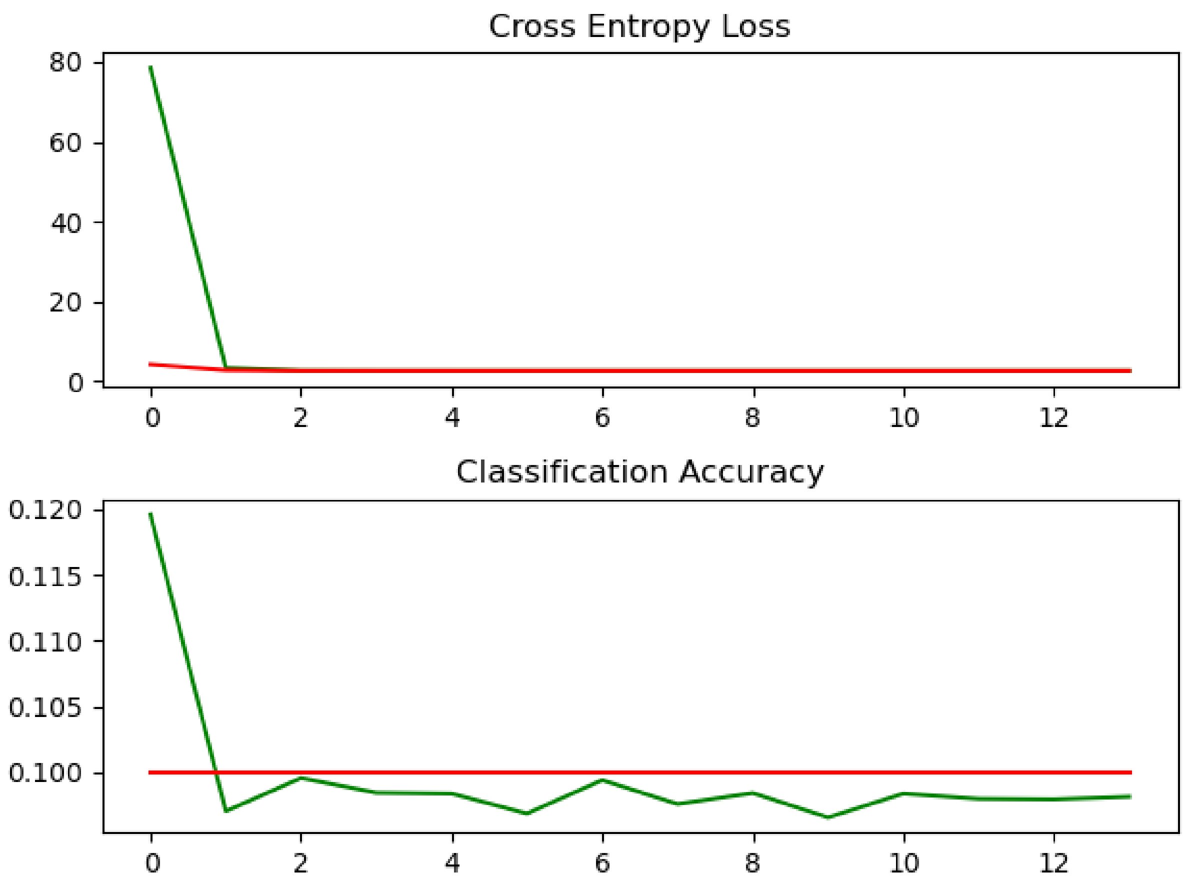
Weight decay 0.009:



Weight decay 0.01:



Weight decay 0.02:



Observations and conclusions:

- Using weight in a proper configuration decay may increase our accuracy
- After this experiment we can crearly see that the best regularizer for this case is l2 so it is regularizer we we're using in following tests.
- Weight decay values form 0.001 to 0.007 are the most promising so we're using them in next experiments.

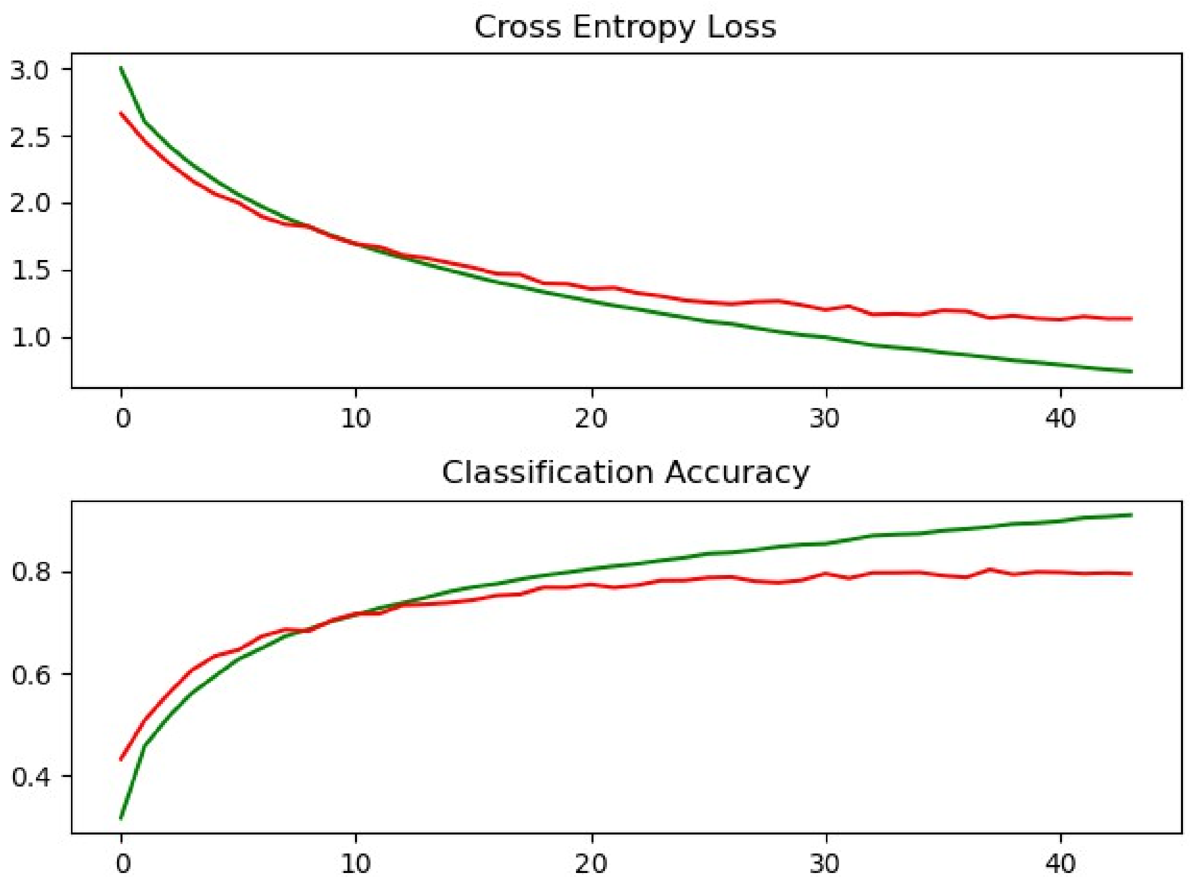
d) dropout with weight decay

In this experiment we are using dropout from 10-40% and l2 regularizer with value from 0.001 to 0.007 as we figured out that its most promising configuration.

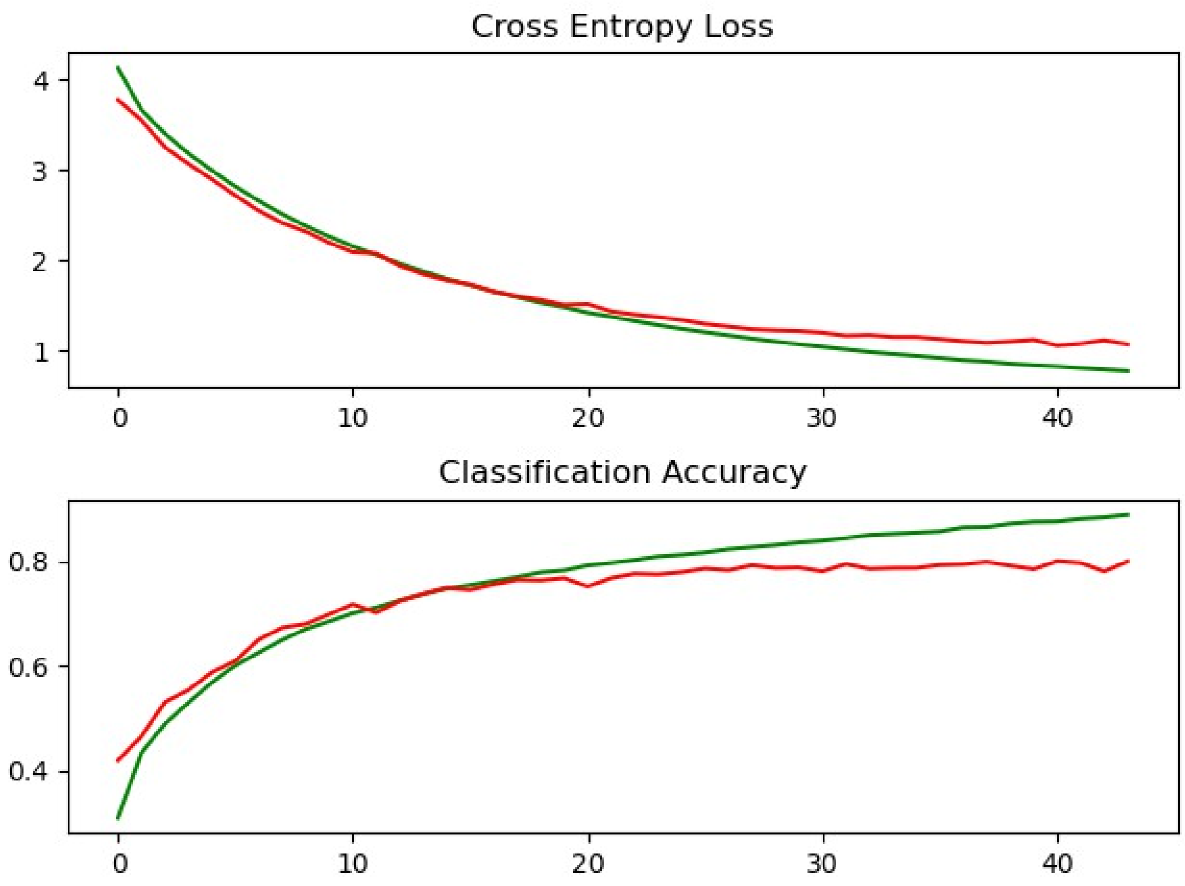
1. 10% dropout:

Weight decay	accuracy
0.001	79.500
0.002	**79.870**
0.003	78.680
0.004	76.600
0.005	76.790
0.006	73.460
0.007	73.160

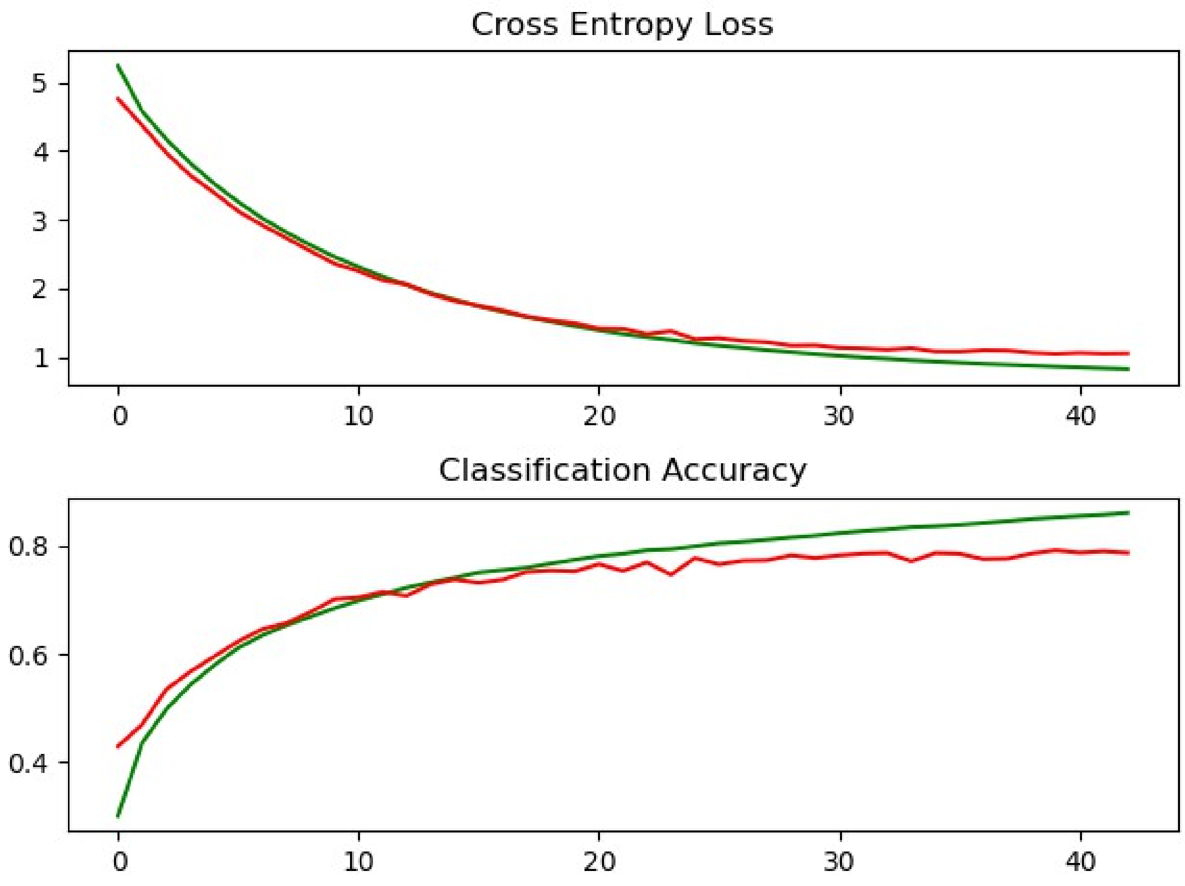
Weight decay: 0.001



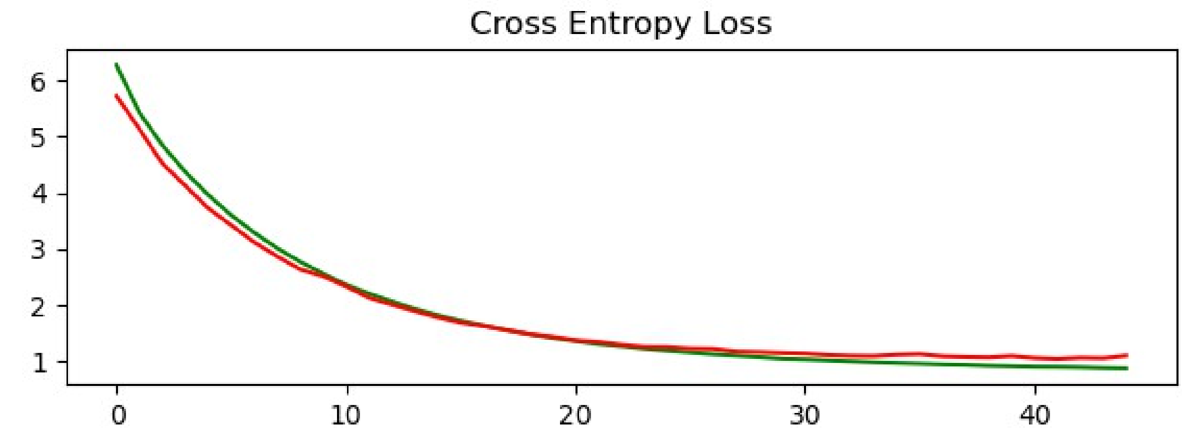
Weight decay: 0.002

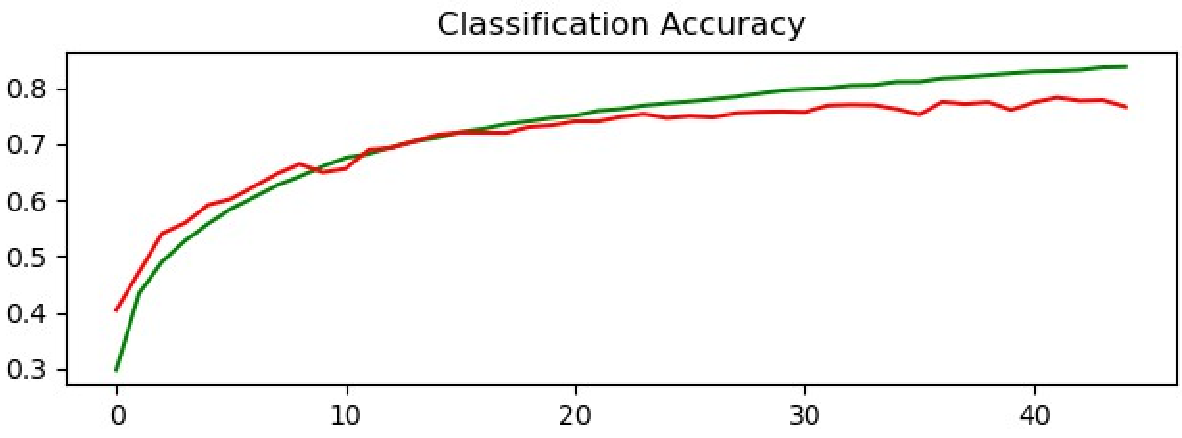


Weight decay: 0.003

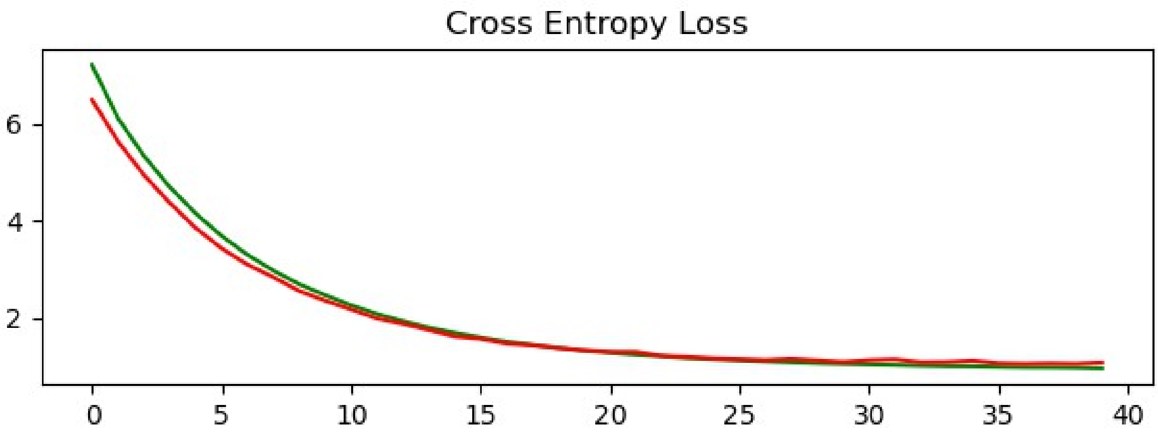


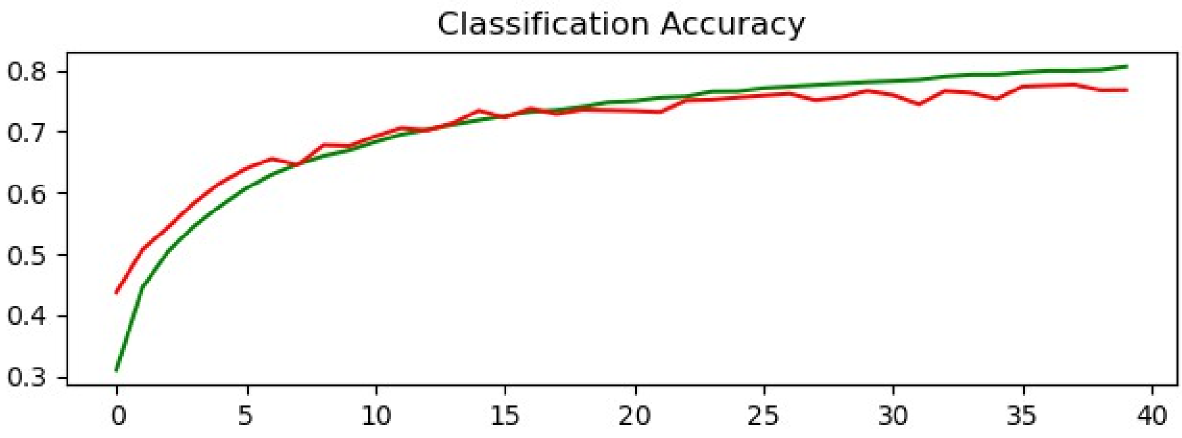
Weight decay: 0.004



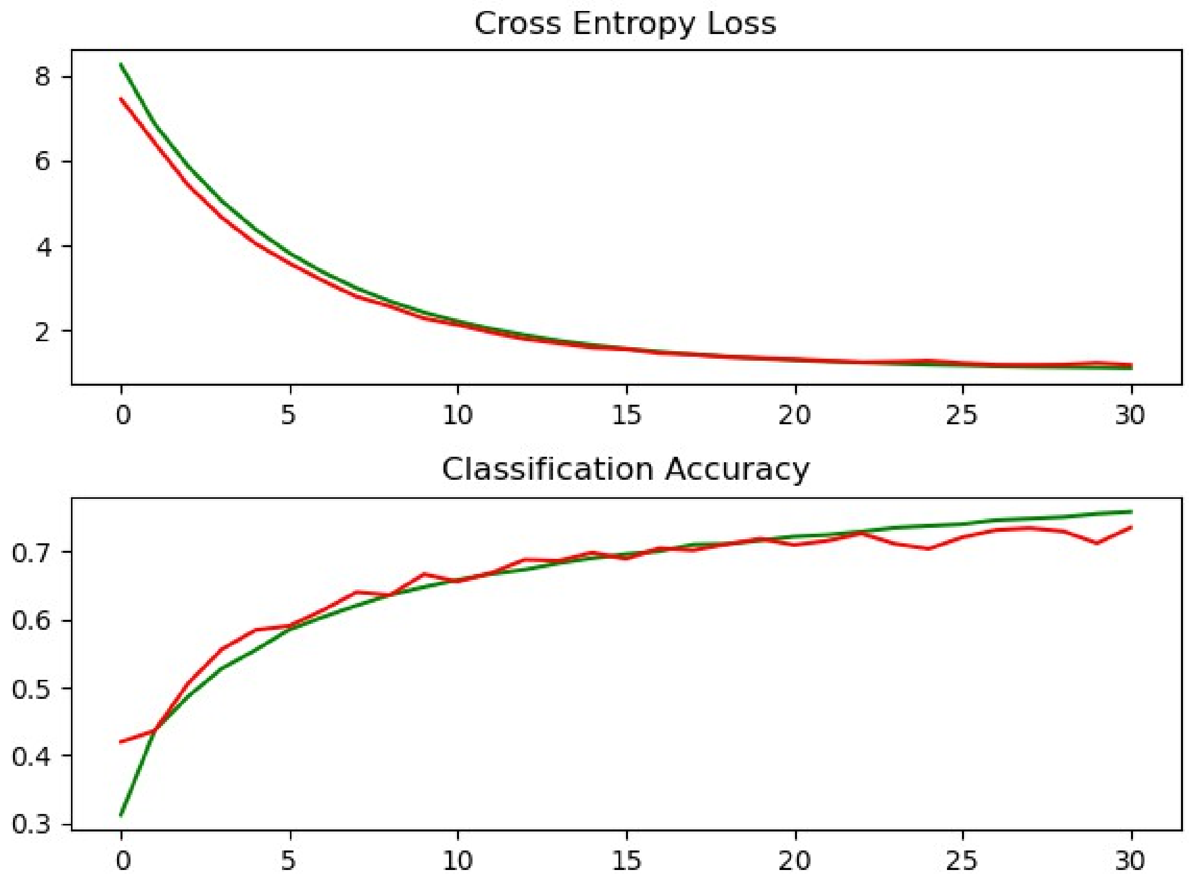


Weight decay: 0.005

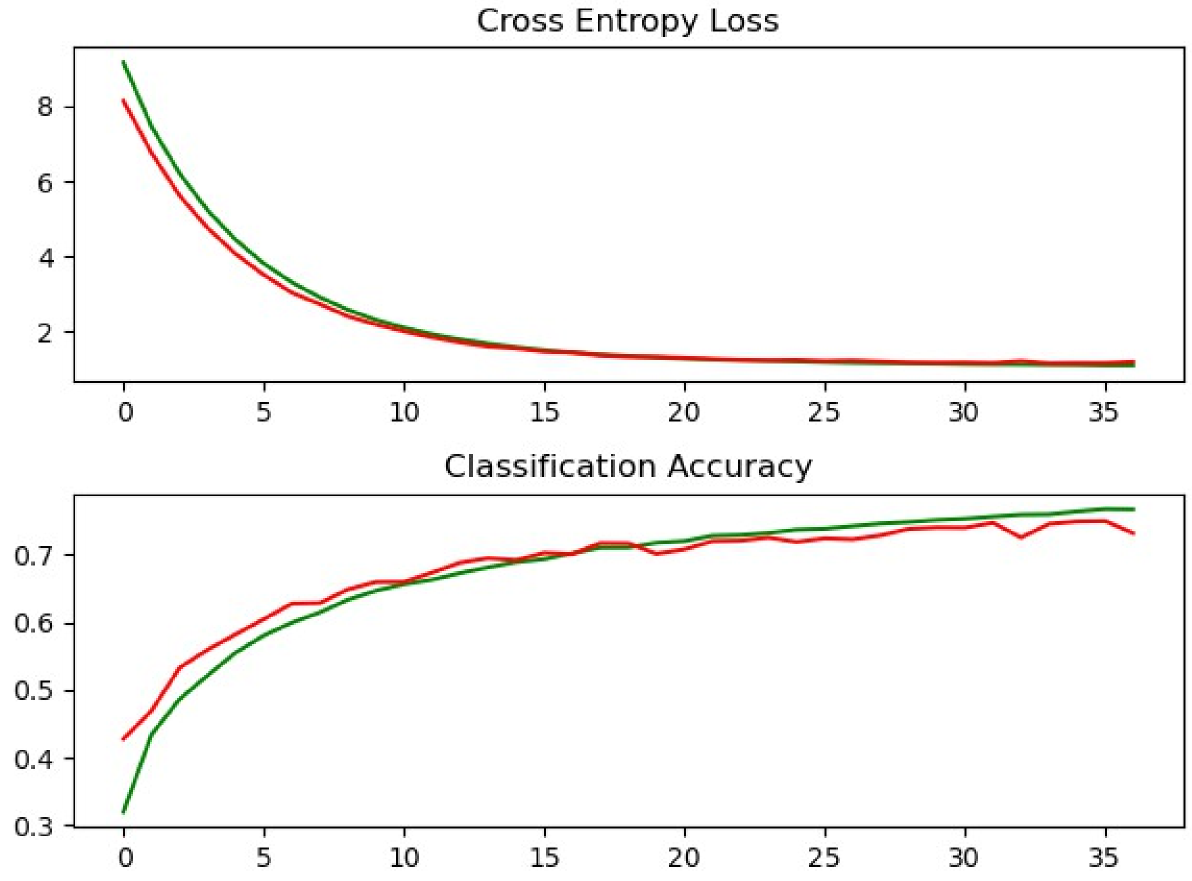




Weight decay: 0.006



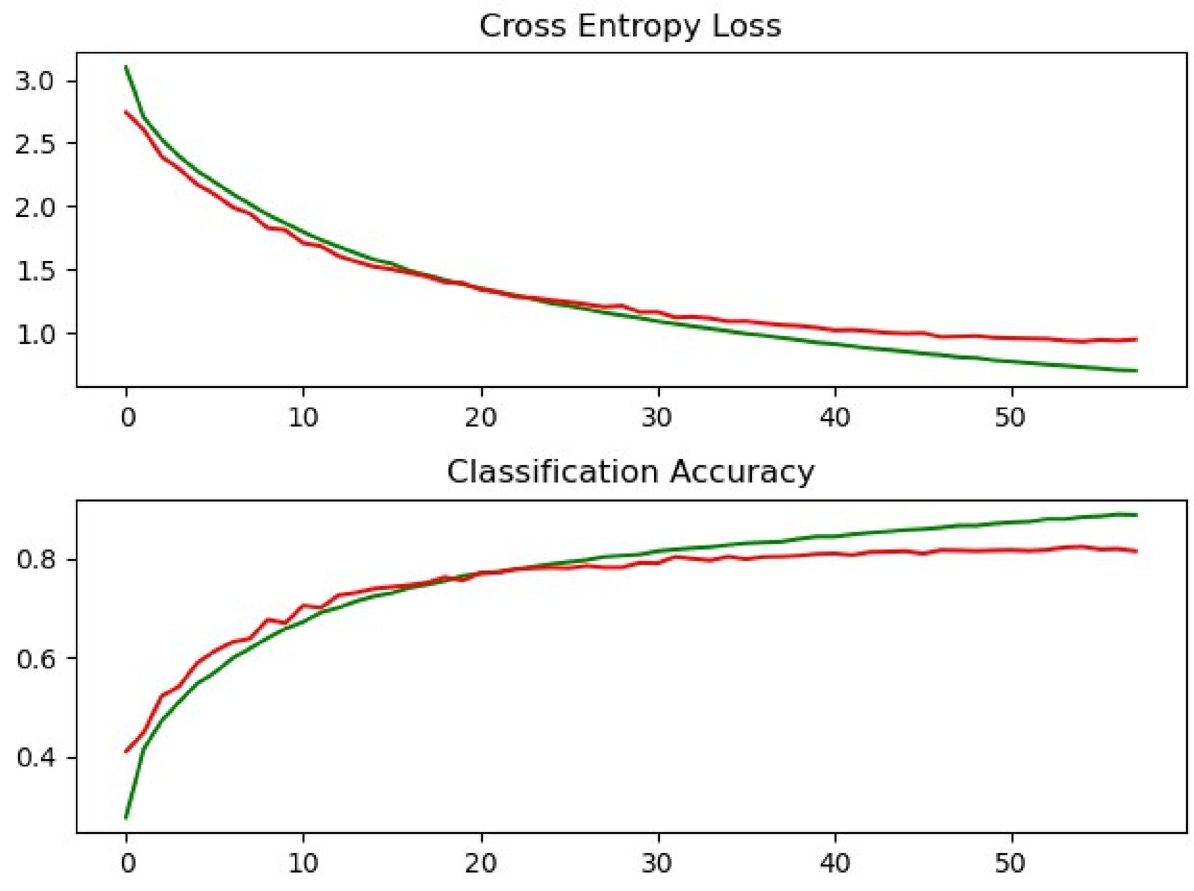
Weight decay: 0.007



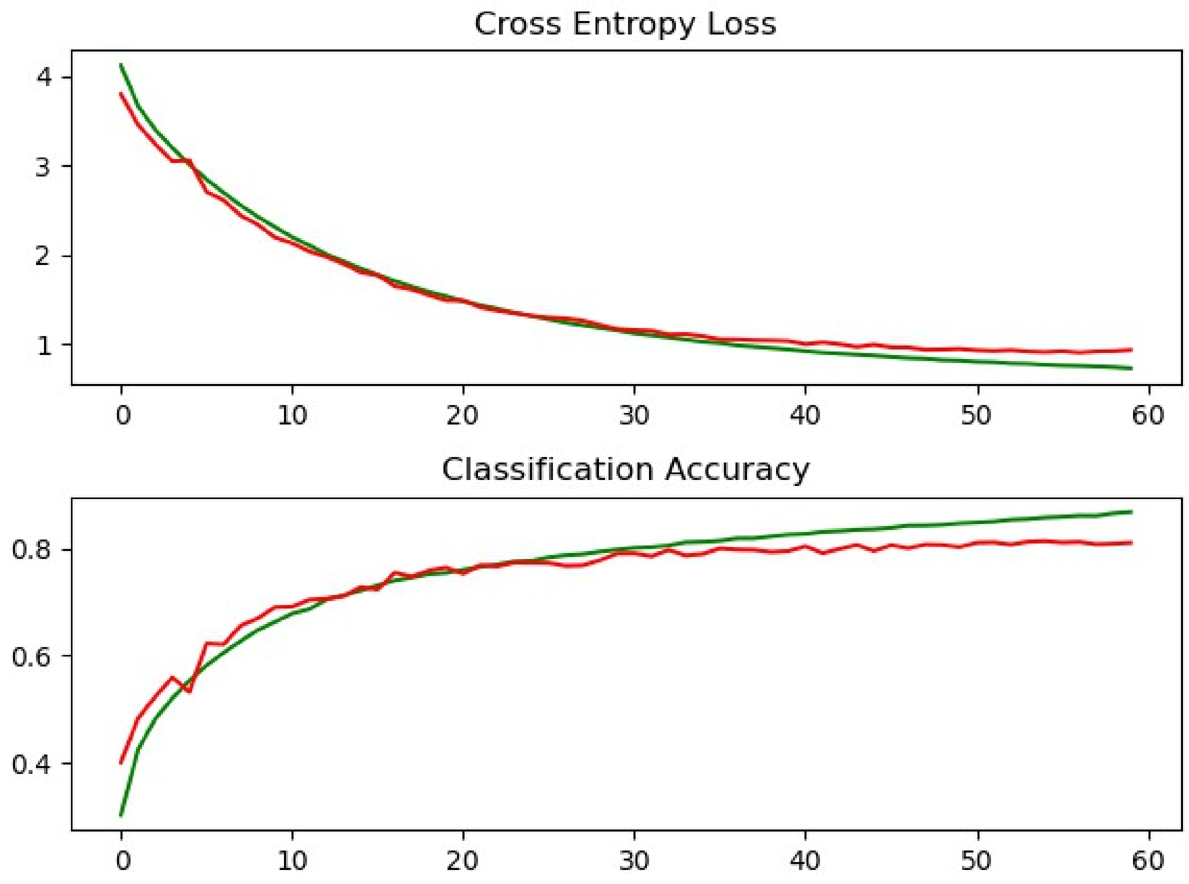
2. 20% dropout:

Weight decay	accuracy
0.001	**81.510**
0.002	81.090
0.003	79.110
0.004	76.720
0.005	72.410
0.006	75.030
0.007	67.370

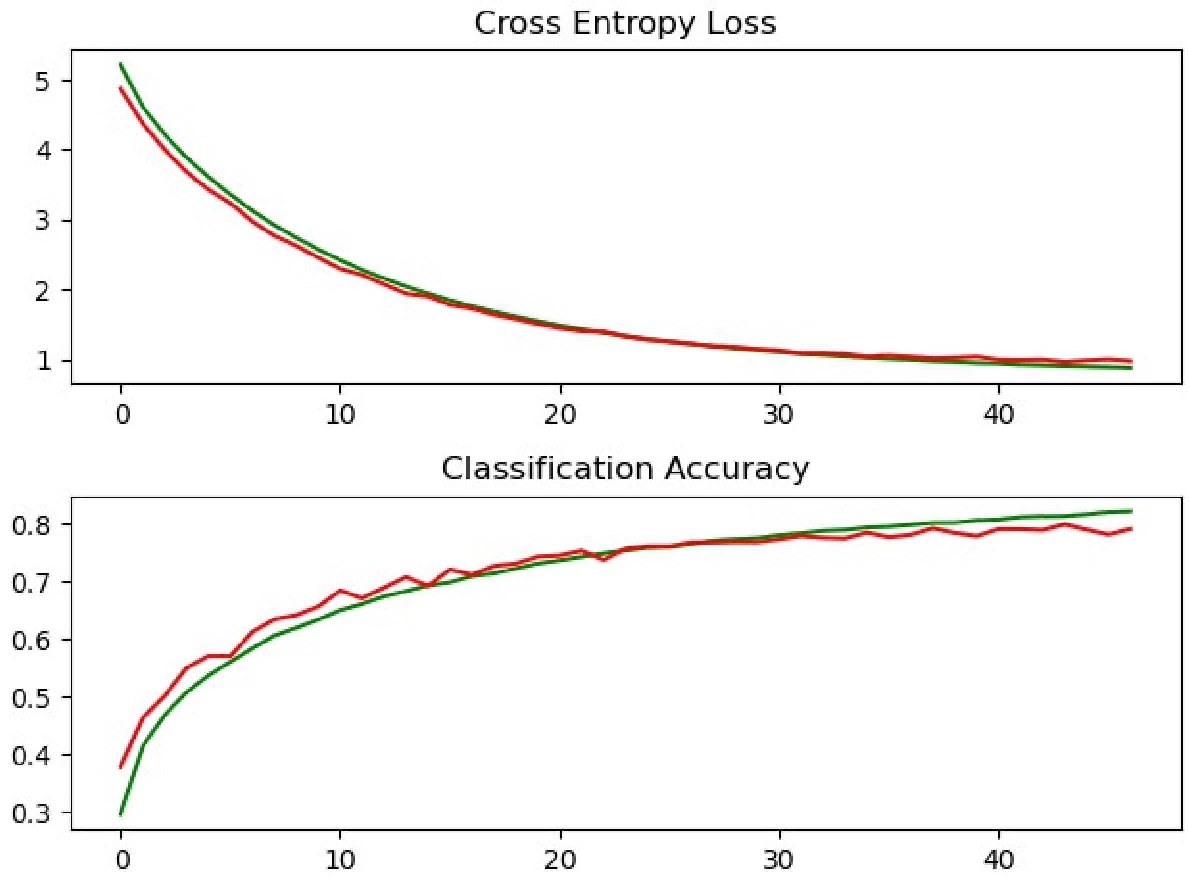
Weight decay: 0.001



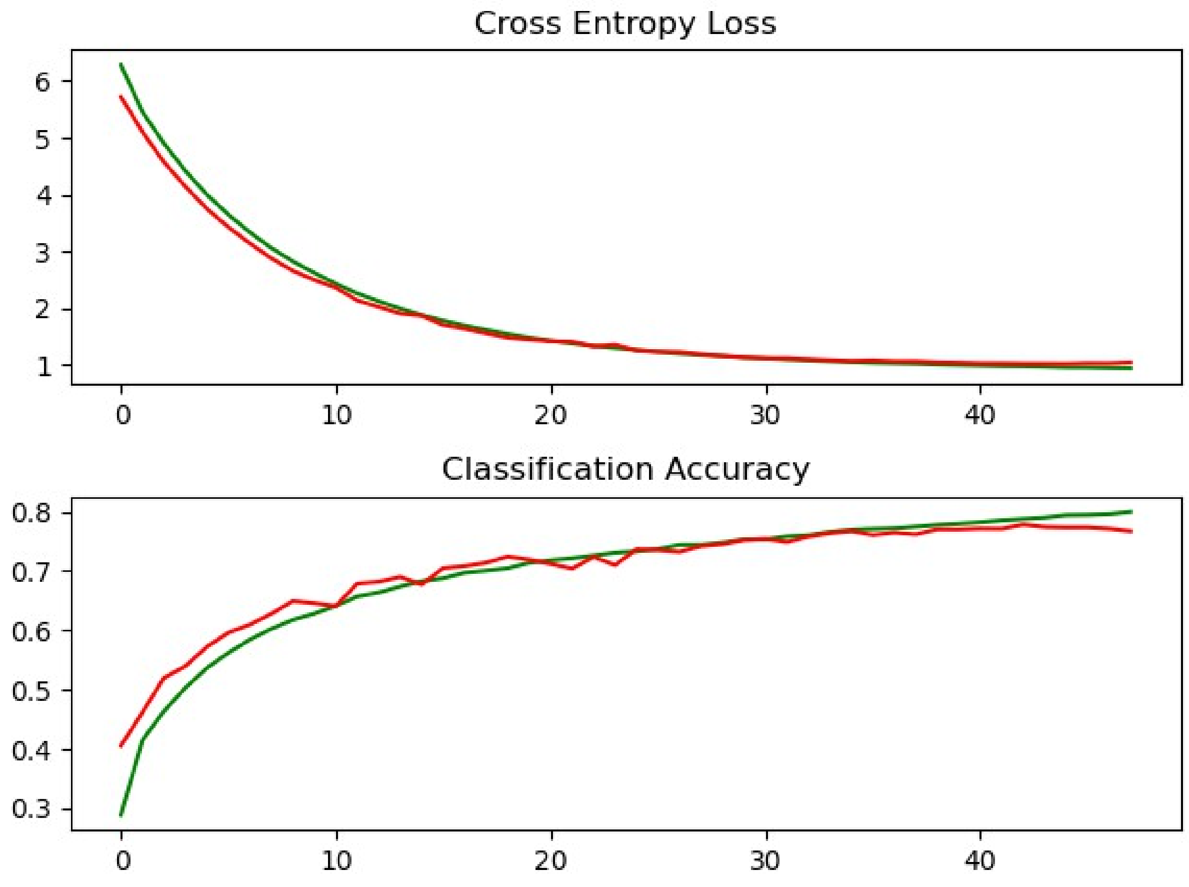
Weight decay: 0.002



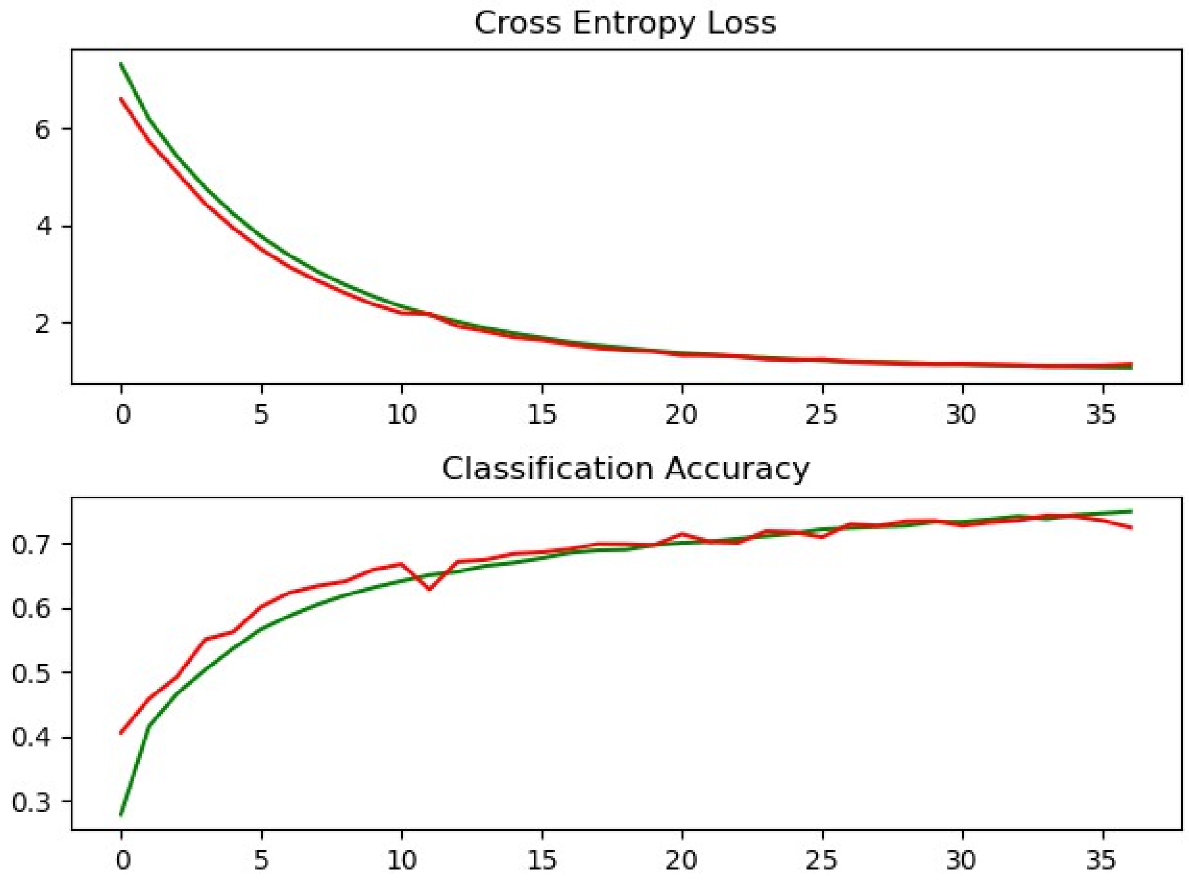
Weight decay: 0.003



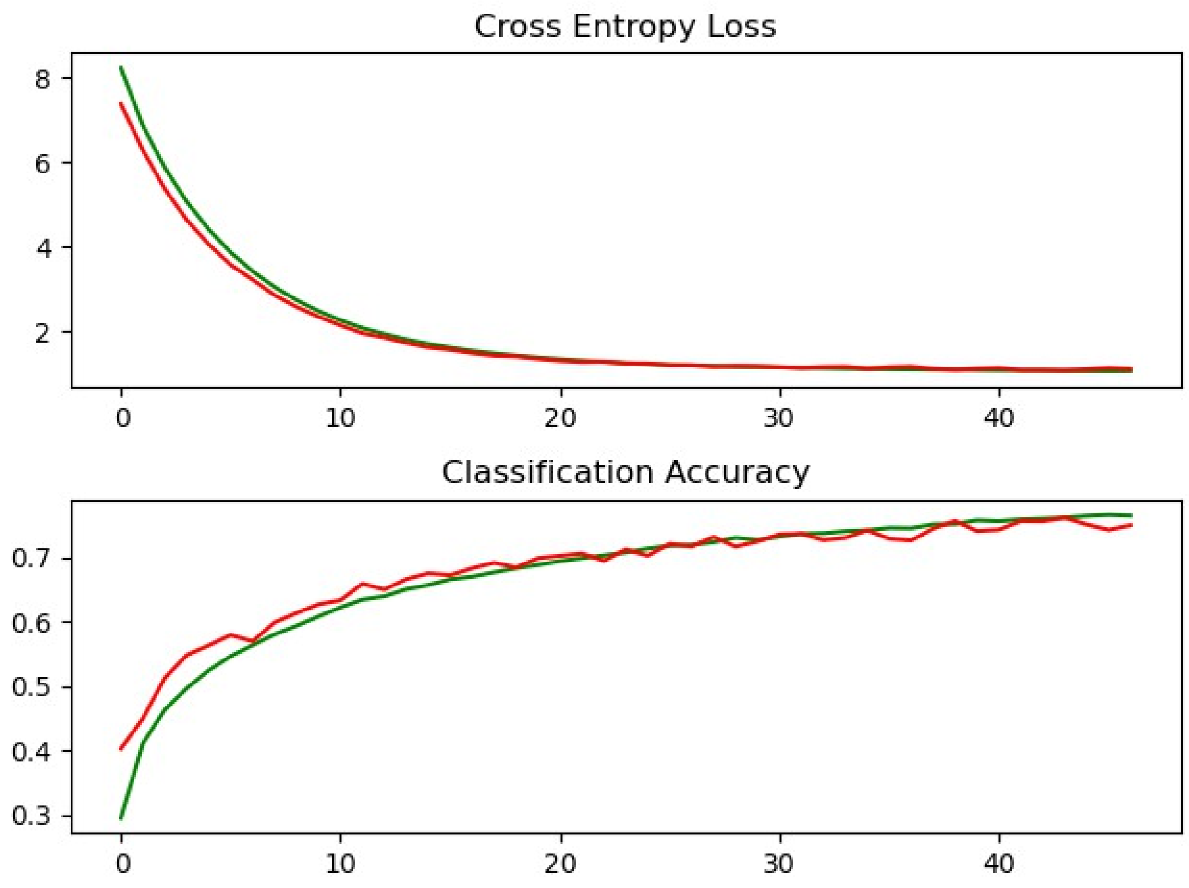
Weight decay: 0.004



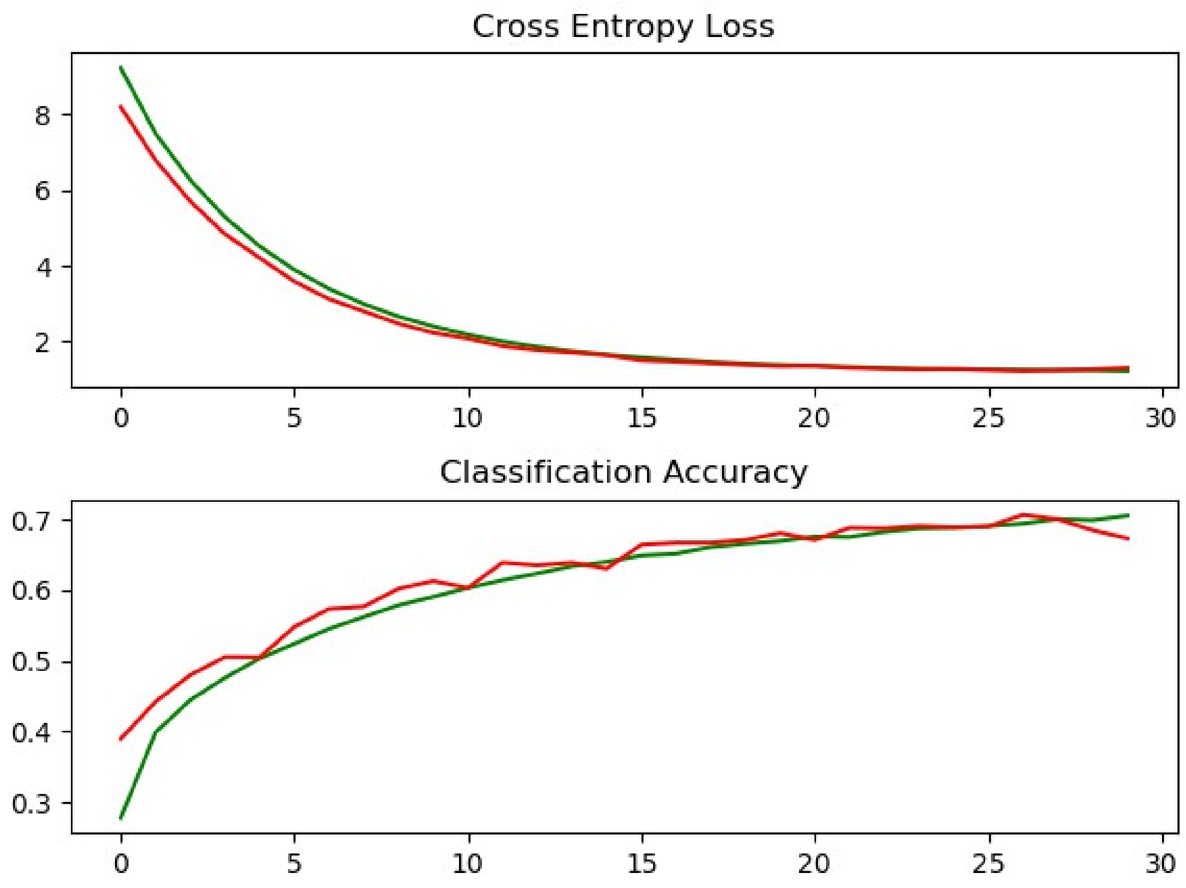
Weight decay: 0.005



Weight decay: 0.006



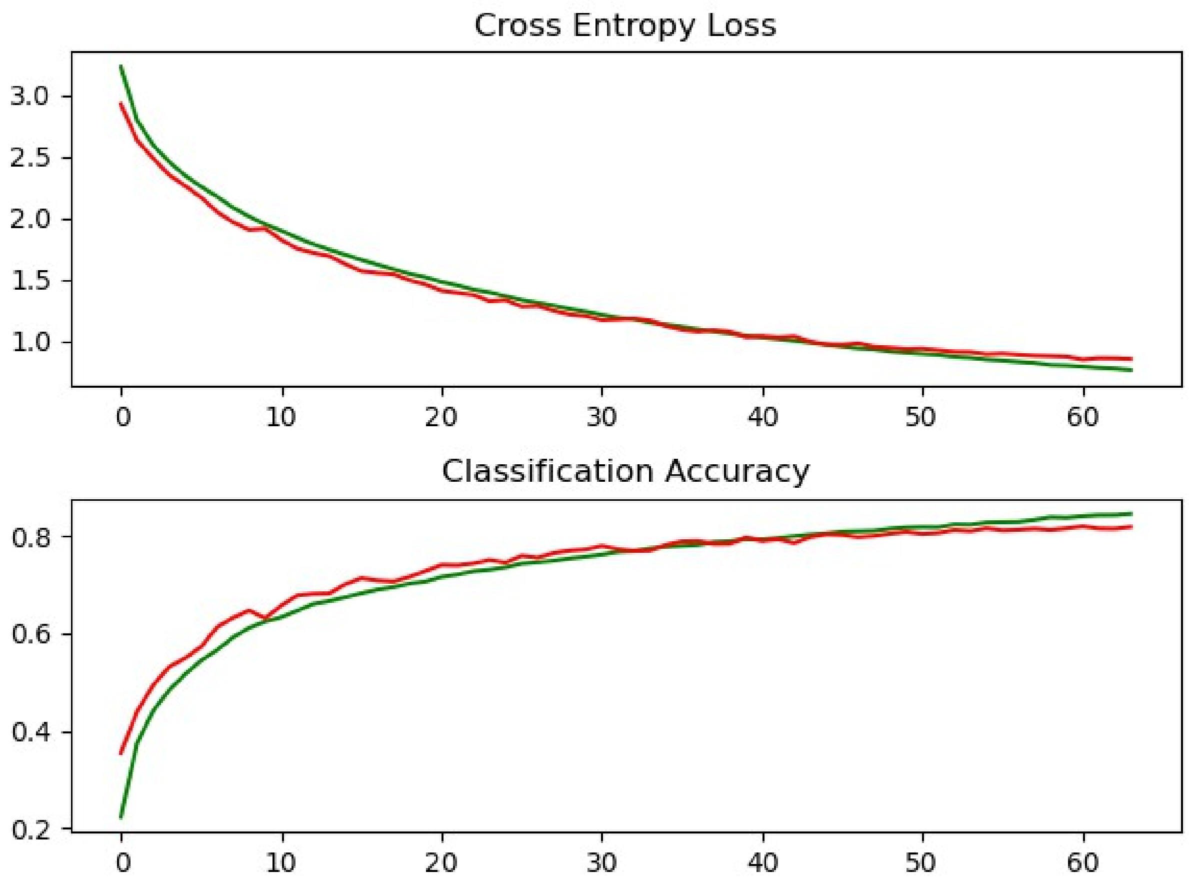
Weight decay: 0.007



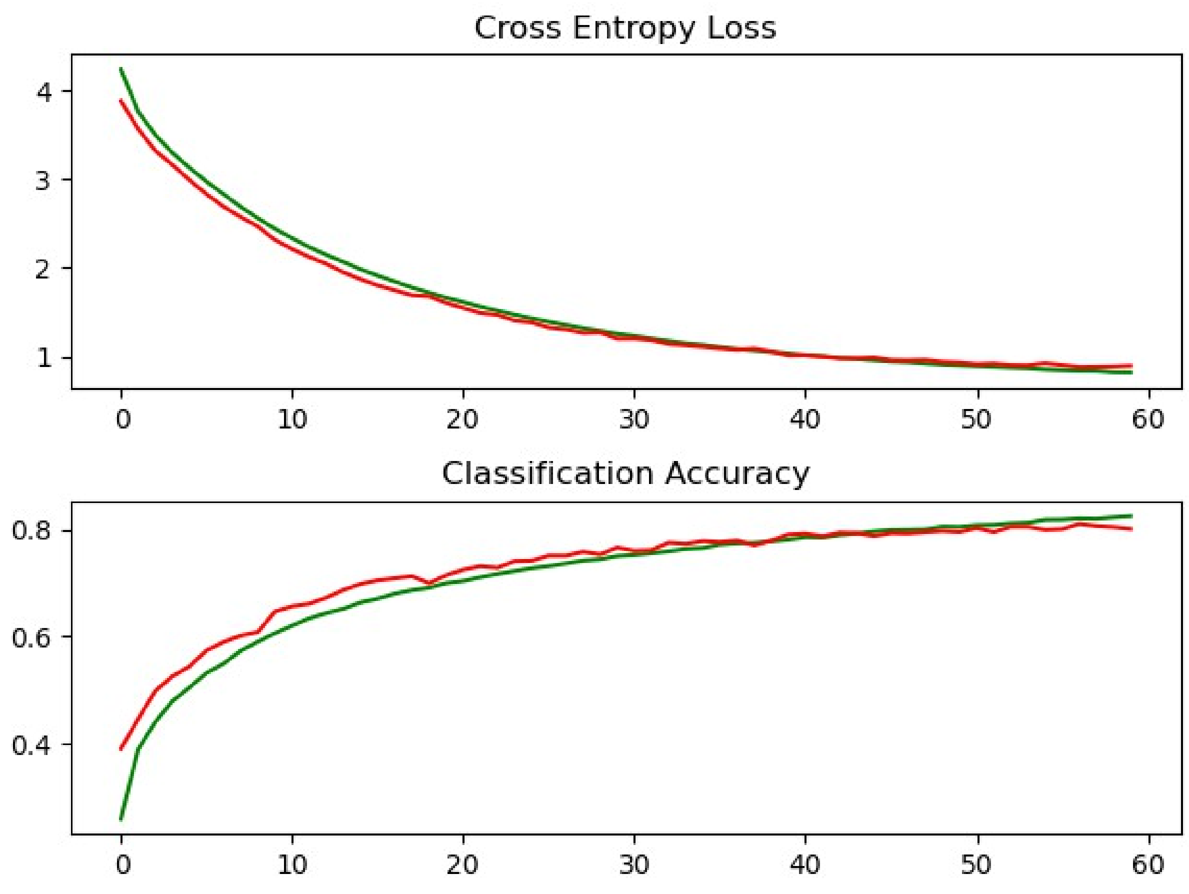
3. 30% dropout:

Weight decay	accuracy
0.001	**81.870**
0.002	80.060
0.003	76.880
0.004	75.580
0.005	73.940
0.006	72.420
0.00767.070	67.070

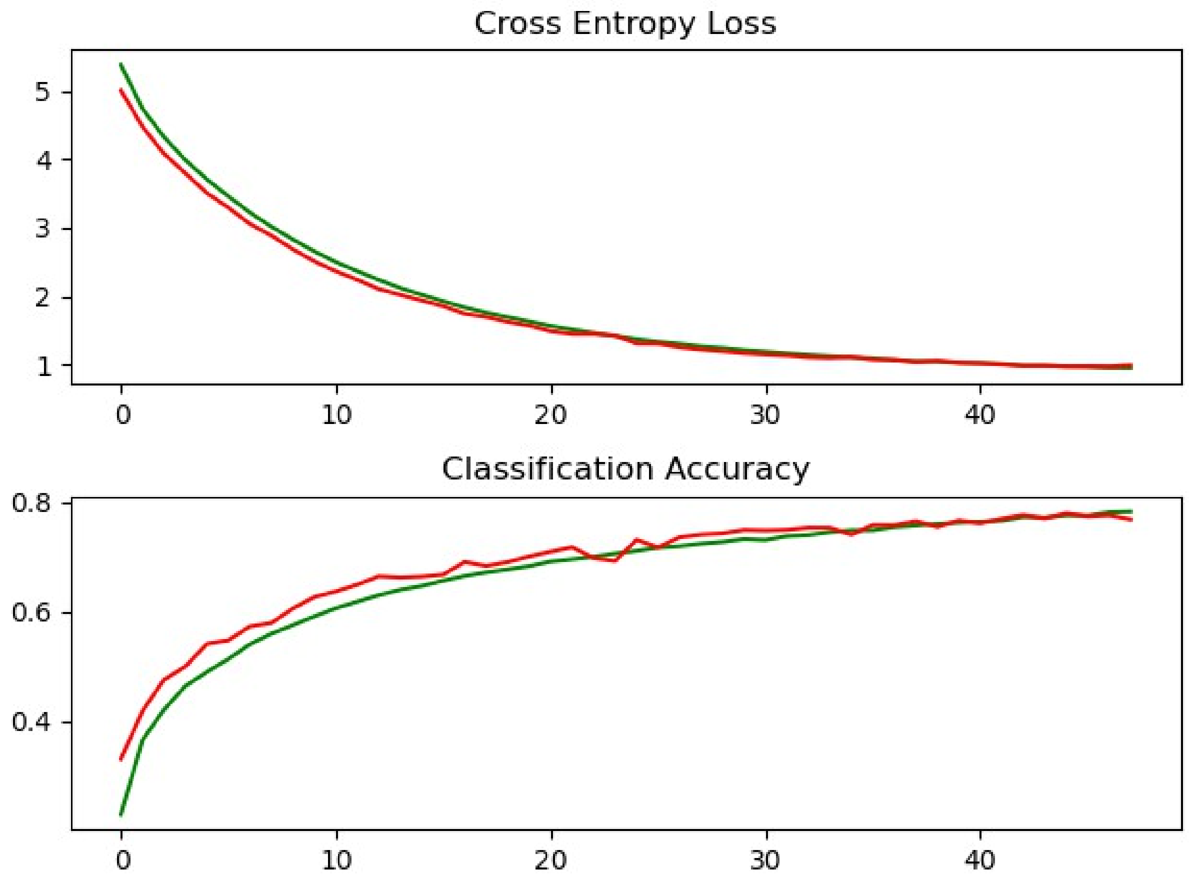
Weight decay: 0.001



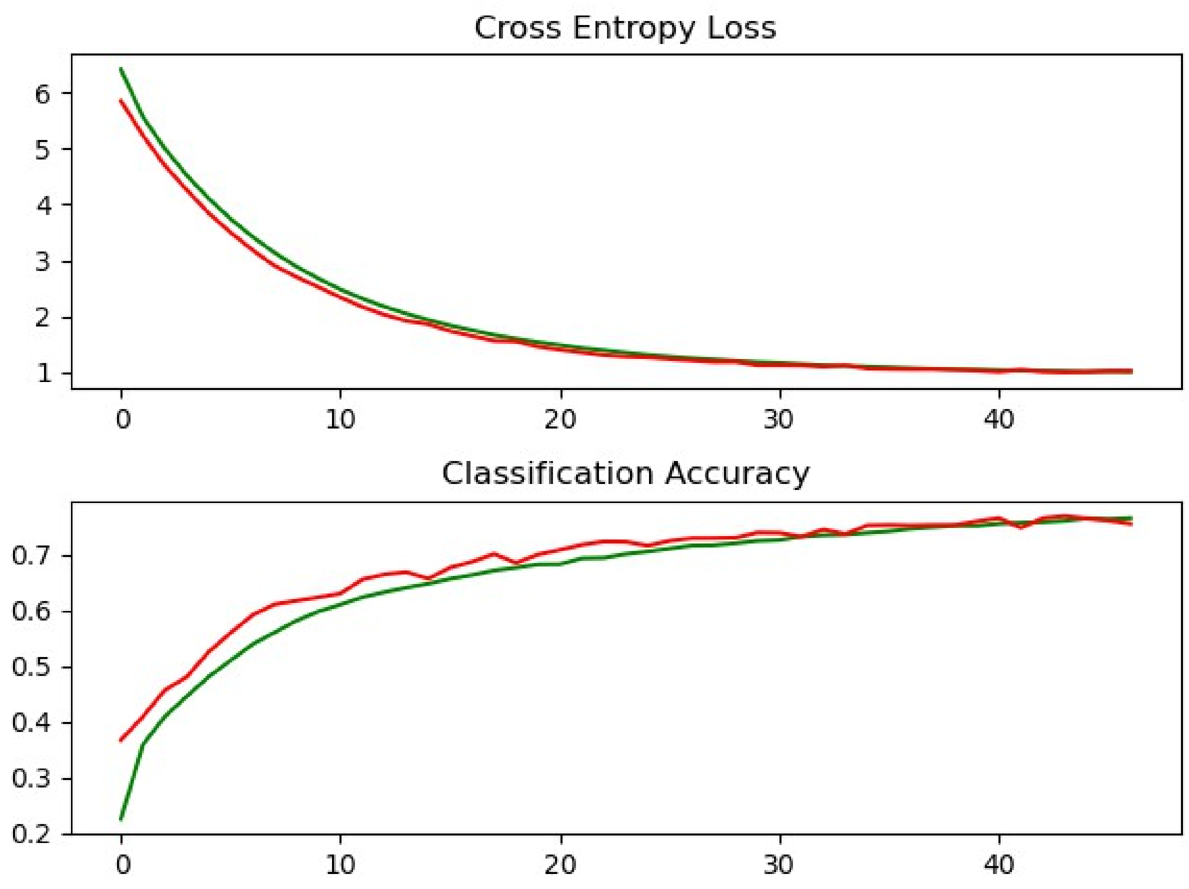
Weight decay: 0.002



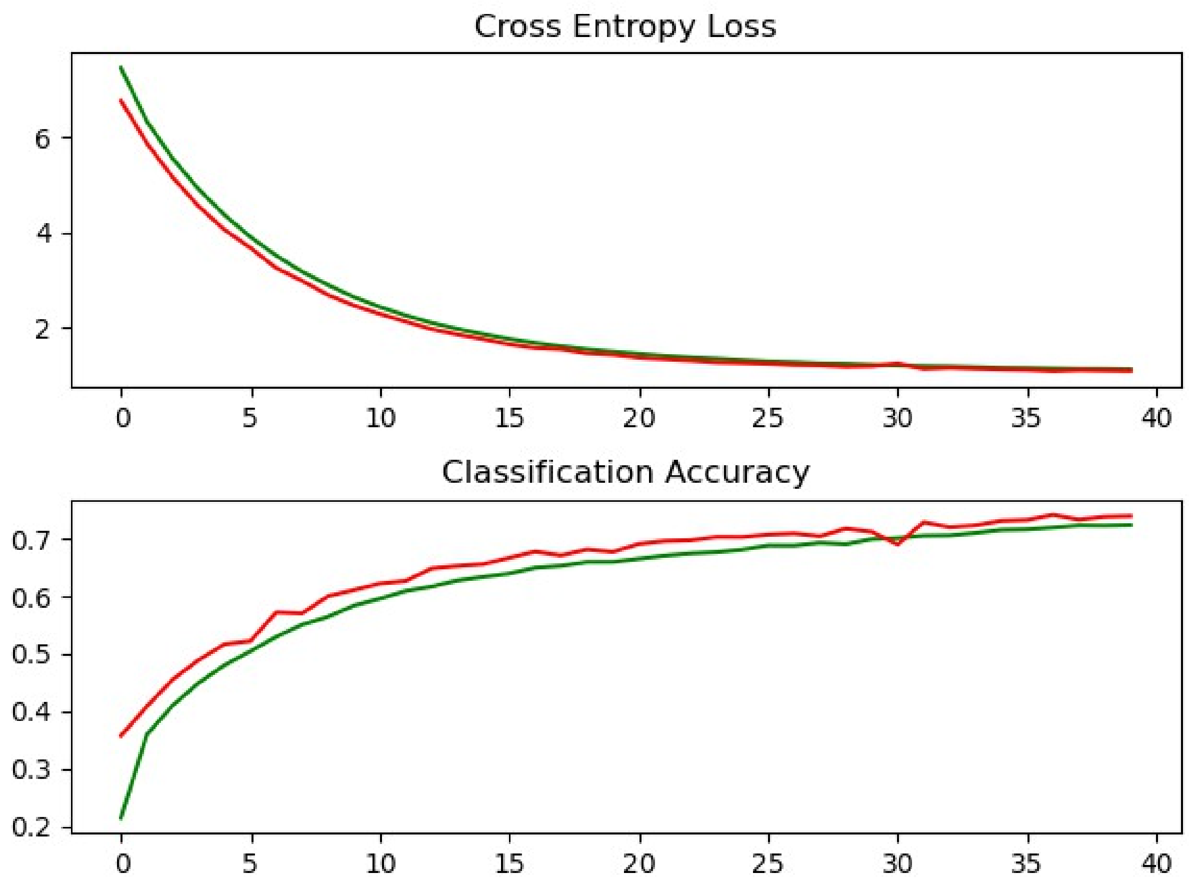
Weight decay: 0.003



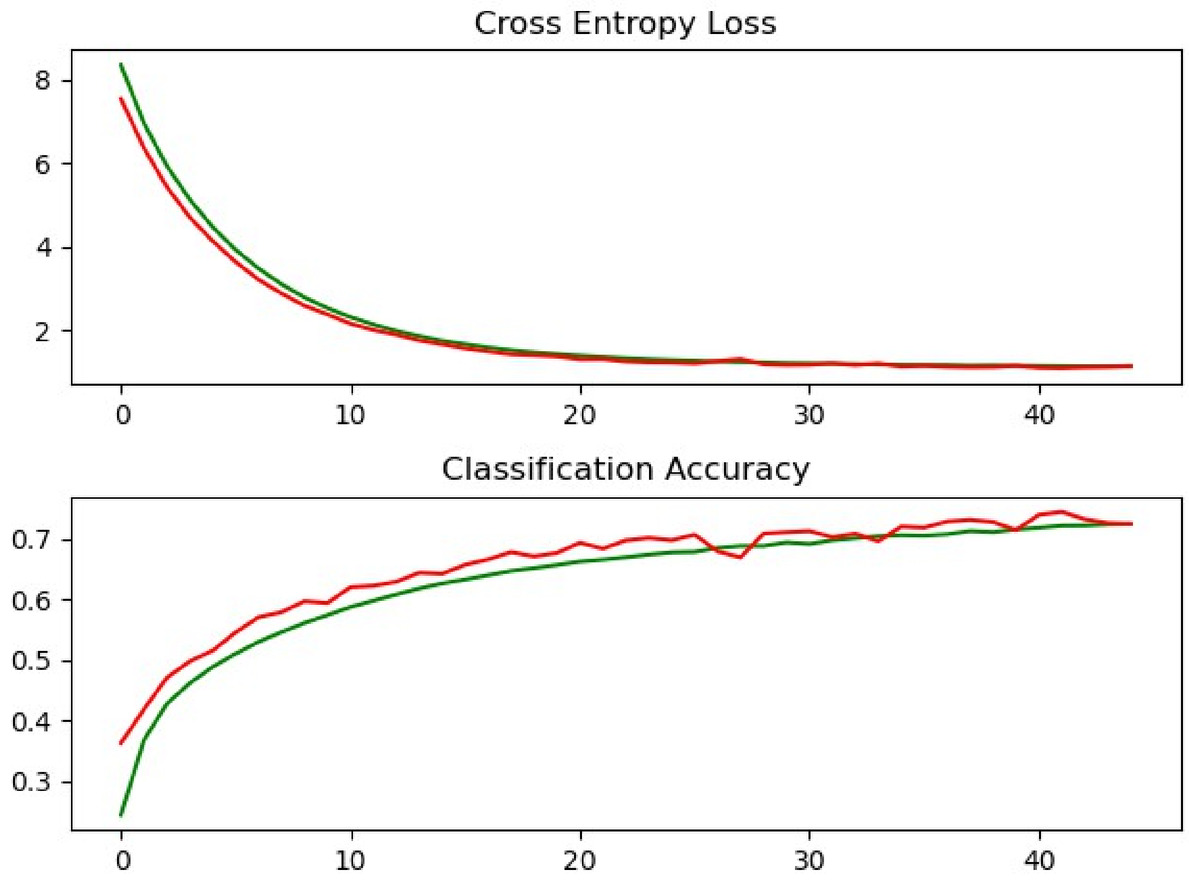
Weight decay: 0.004



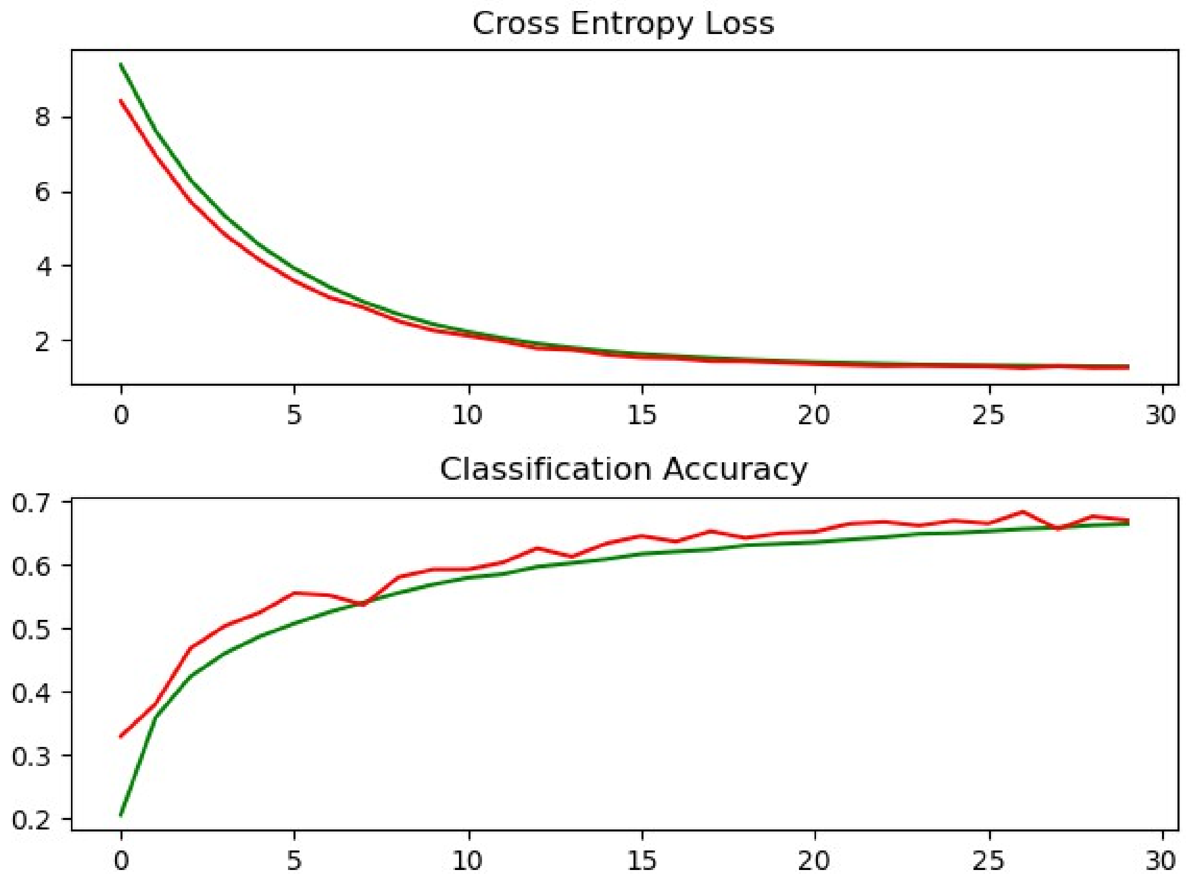
Weight decay: 0.005



Weight decay: 0.006



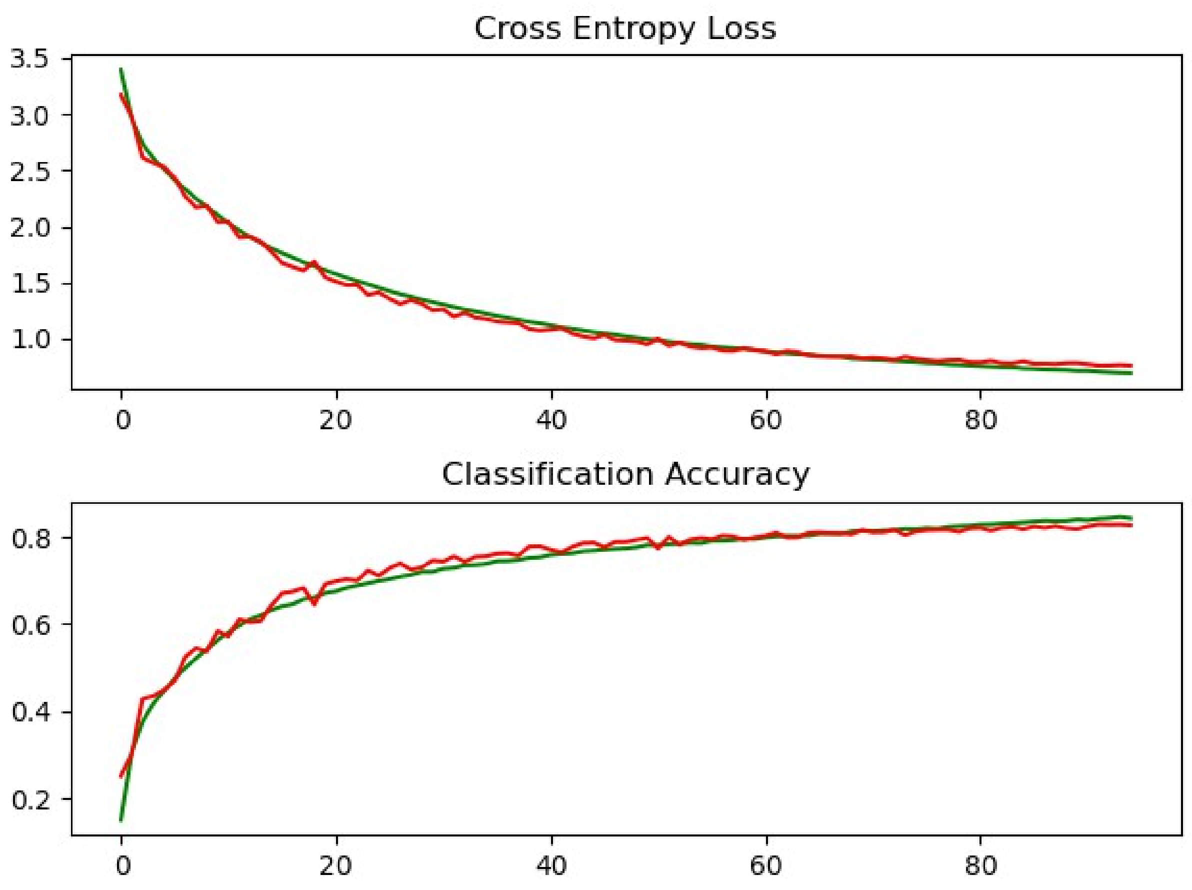
Weight decay: 0.007



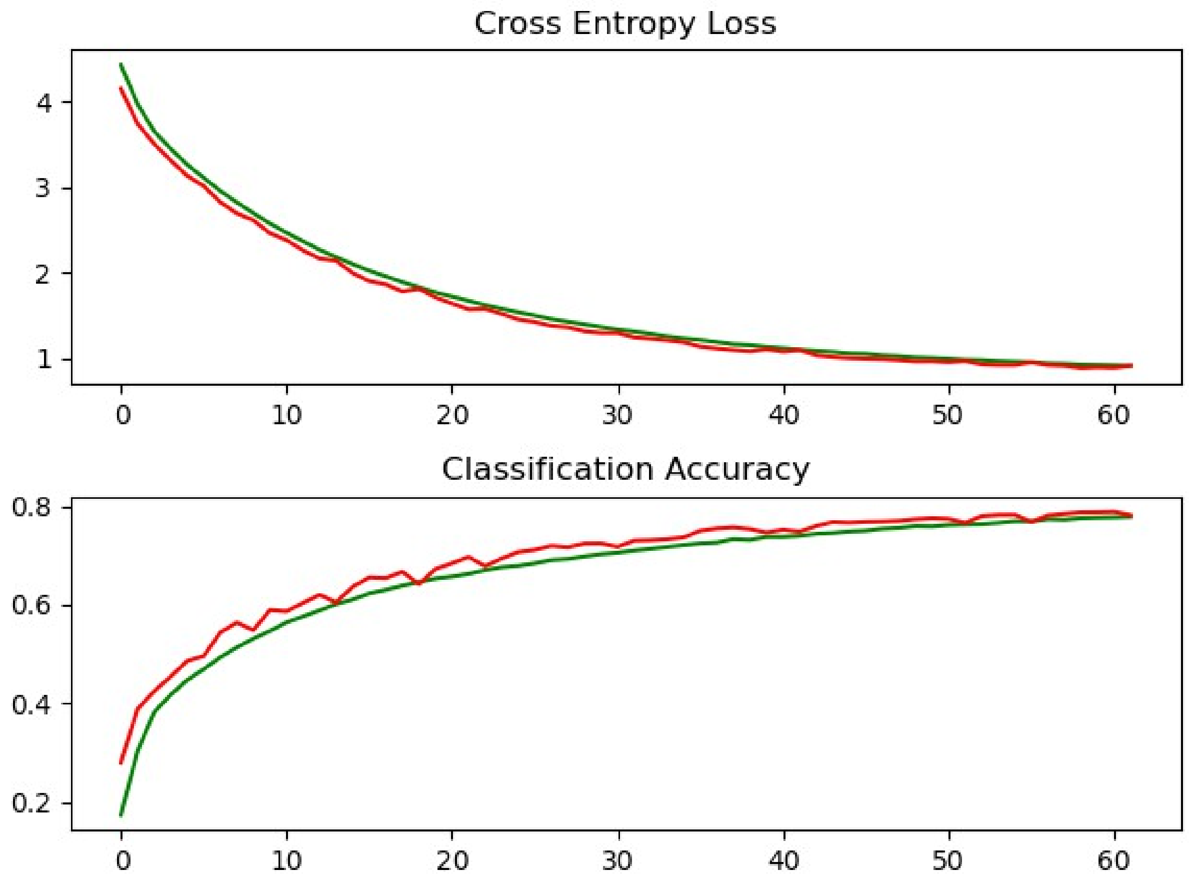
4. 40% dropout:

Weight decay	accuracy
0.001	**82.710**
0.002	78.130
0.003	76.810
0.004	76.350
0.005	68.240
0.006	67.820
0.007	66.150

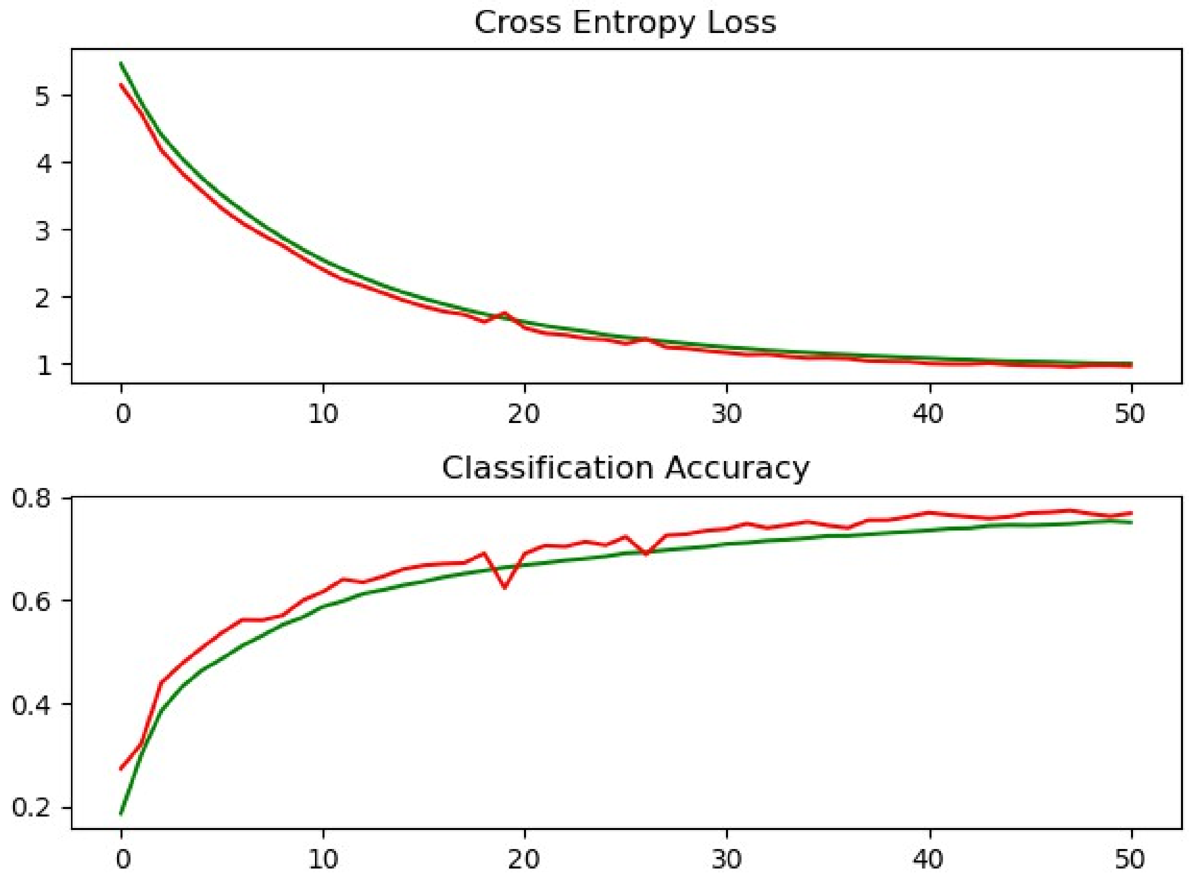
Weight decay: 0.001



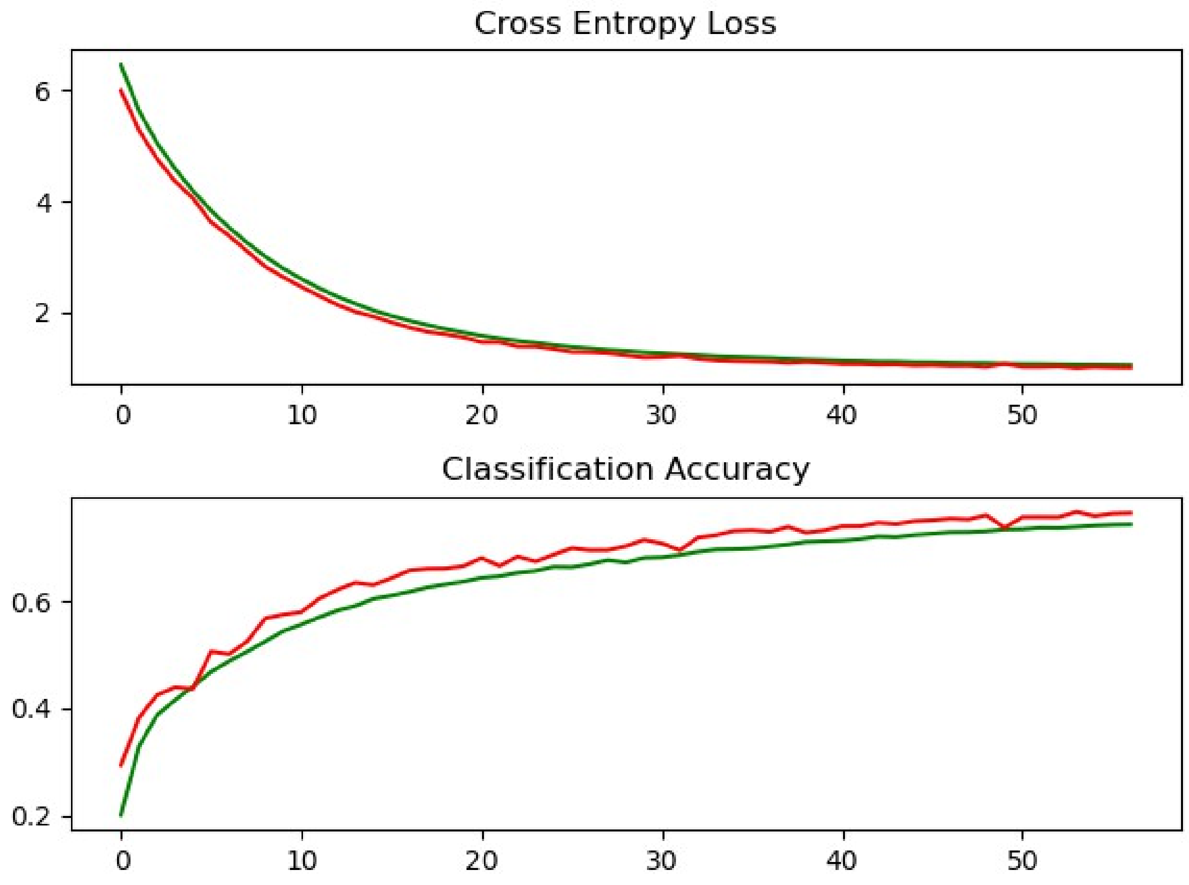
Weight decay: 0.002



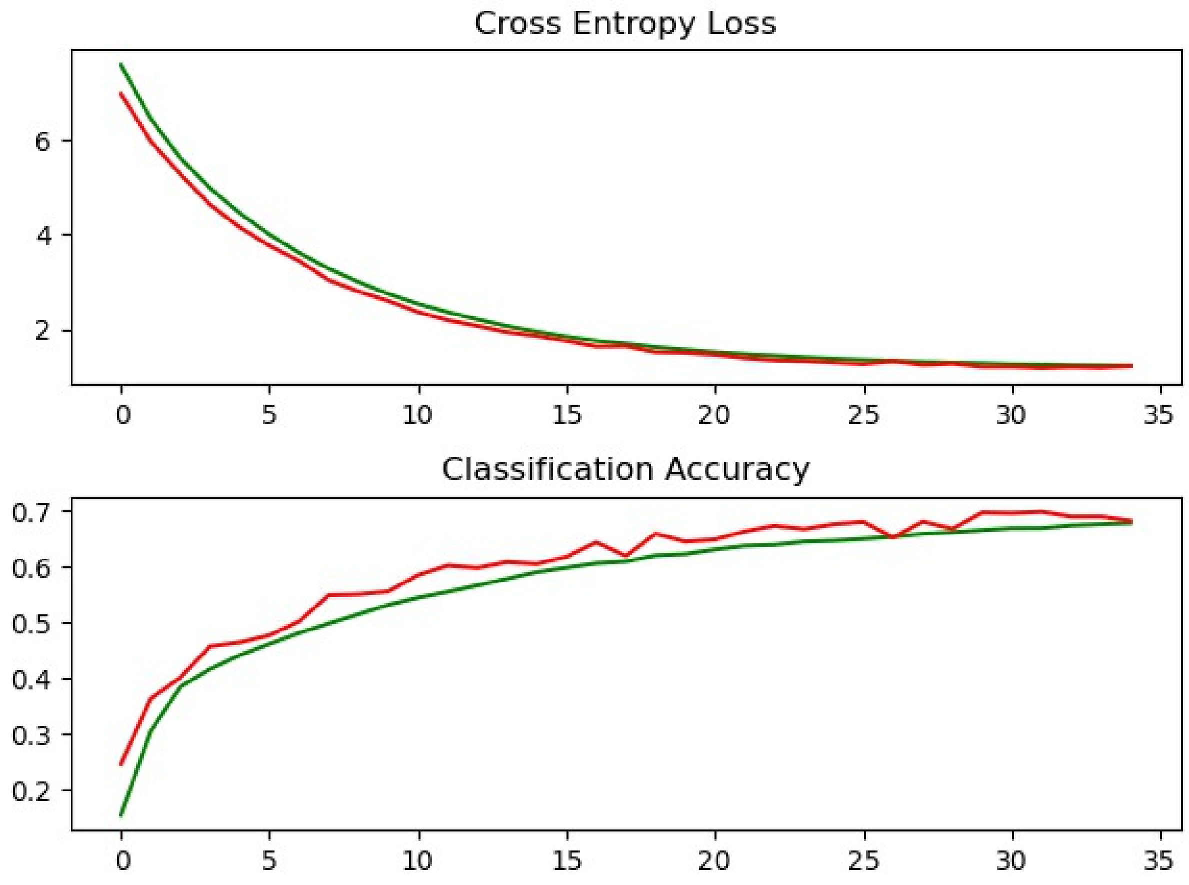
Weight decay: 0.003



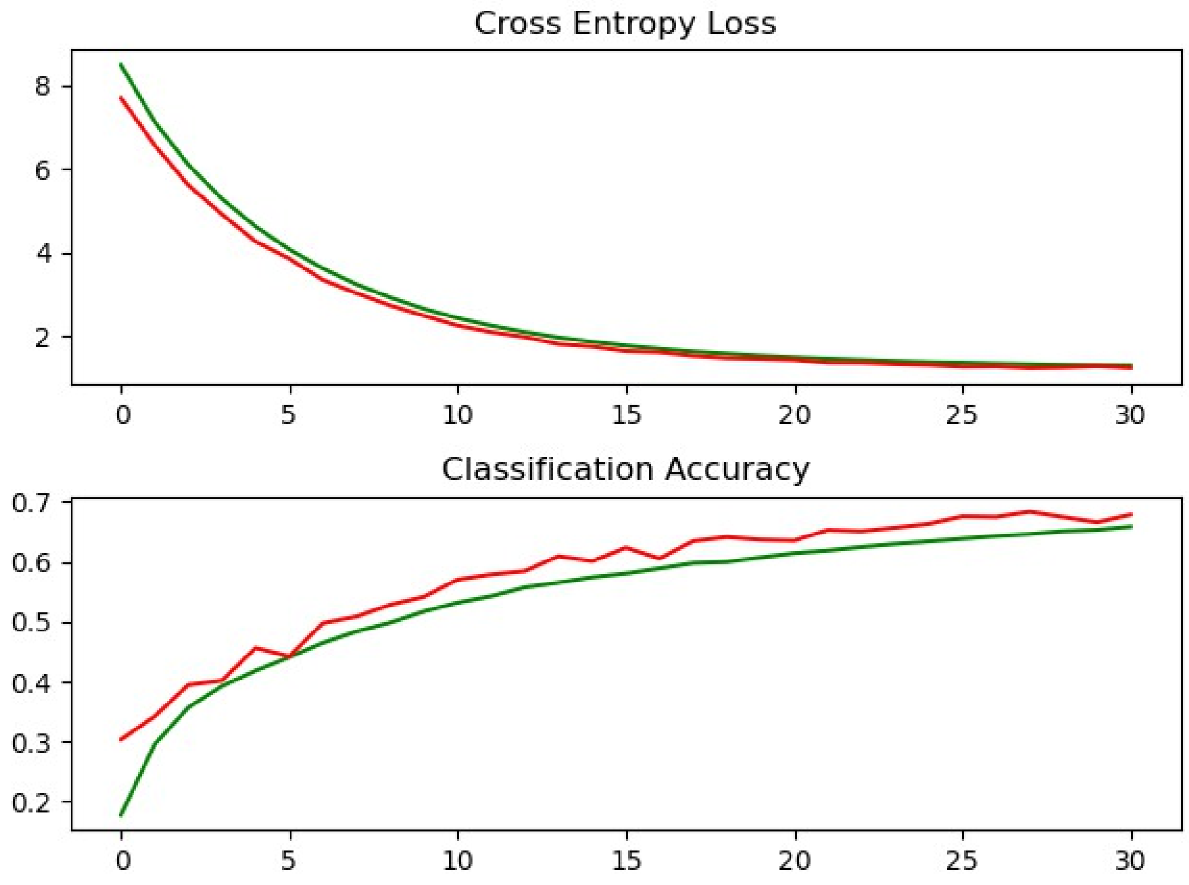
Weight decay: 0.004



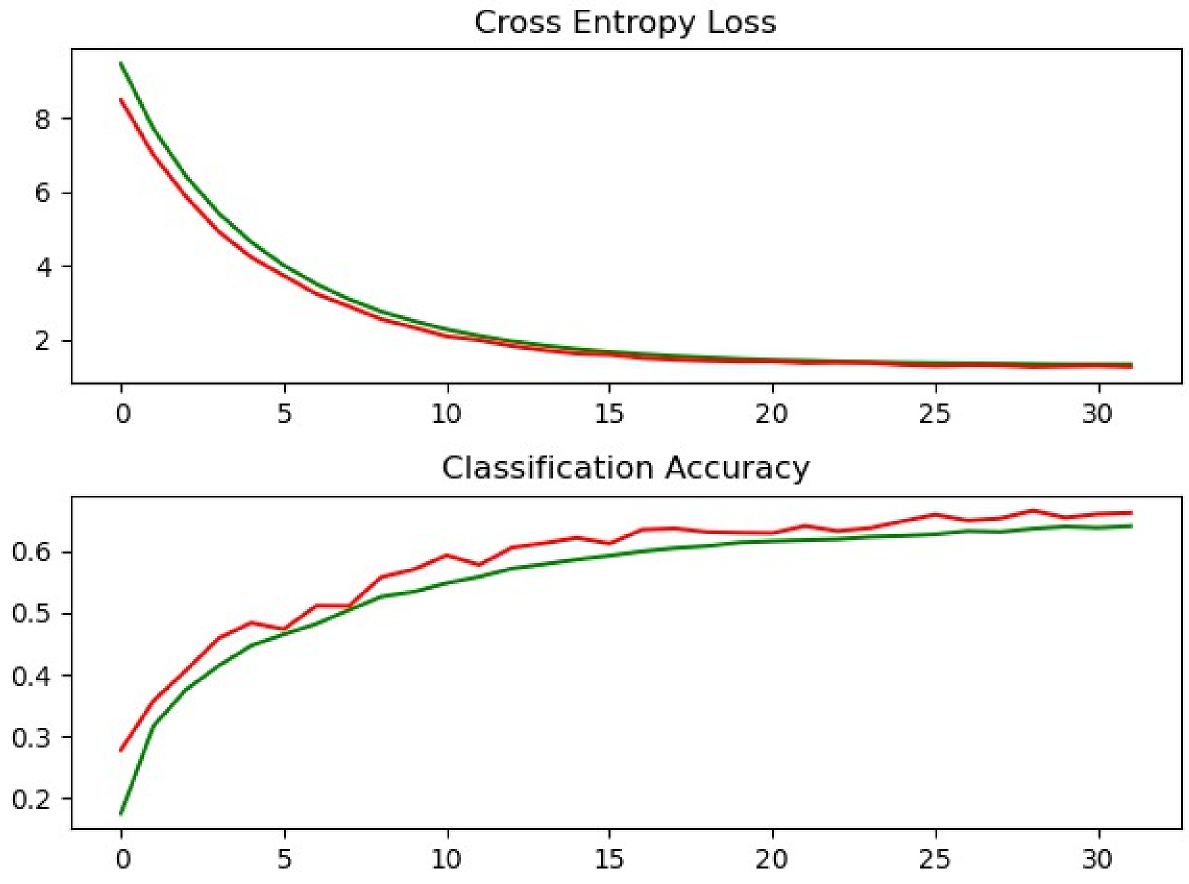
Weight decay: 0.005



Weight decay: 0.006



Weight decay: 0.007



Observations and conclusions:

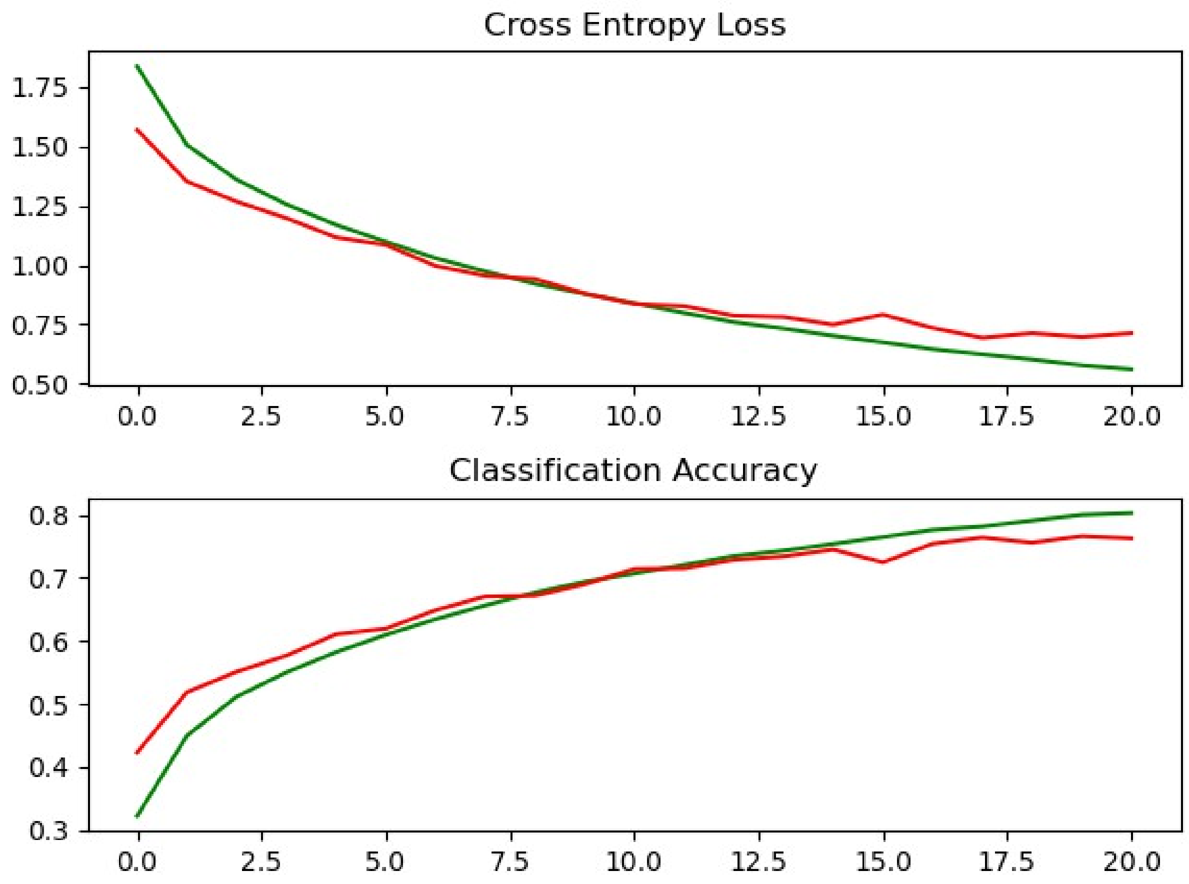
- combining weight decay and dropout could bring better results
- best accuracy, using dropout and weight decay together, has been achieved with values other then in separate experiments(best dropout: 10%, best weight decay: 0.006, tohether best: 40% and 0,001)

e) different dropout after each VGG

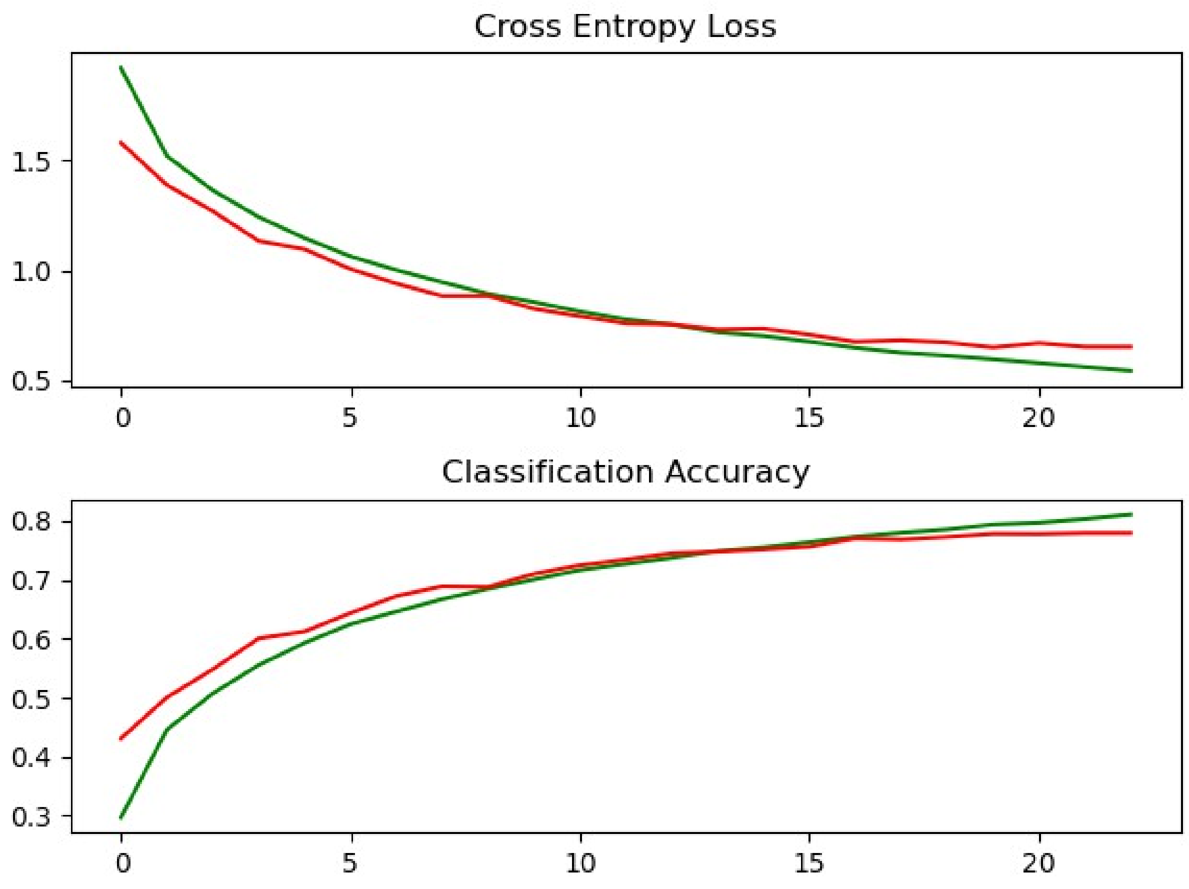
In this experiment we're trying various dropout after each VGG. Earlier we've decided that we consider only dropout from 10% to 40%.. This experiment is still in progress, cause there is a lot of possibilities. Precisely- 4 x 4 x 4 = 64. Its a lot even for GPU with CUDA. At this moment we have results from 26 combinations:

1st dropout	2nd dropout	3rd dropout	accuracy
10,00%	10,00%	10,00%	76.290
10,00%	10,00%	20,00%	77.880
10,00%	10,00%	30,00%	79.850
10,00%	10,00%	40,00%	80.190
10,00%	20,00%	10,00%	77.960
10,00%	20,00%	20,00%	80.790
10,00%	20,00%	30,00%	80.390
10,00%	20,00%	40,00%	79.740
10,00%	30,00%	10,00%	78.200
10,00%	30,00%	20,00%	80.180
10,00%	30,00%	30,00%	80.070
10,00%	**30,00%**	**40,00%**	**81.120**
10,00%	40,00%	10,00%	79.280
10,00%	40,00%	20,00%	80.650
10,00%	40,00%	30,00%	78.870
10,00%	40,00%	40,00%	80.220
20,00%	10,00%	10,00%	78.140
20,00%	10,00%	20,00%	77.940
20,00%	10,00%	30,00%	78.800
20,00%	10,00%	40,00%	77.990
20,00%	20,00%	10,00%	79.820
20,00%	20,00%	20,00%	76.030
20,00%	20,00%	30,00%	79.250
20,00%	20,00%	40,00%	80.040
20,00%	30,00%	10,00%	77.740
20,00%	30,00%	20,00%	79.330

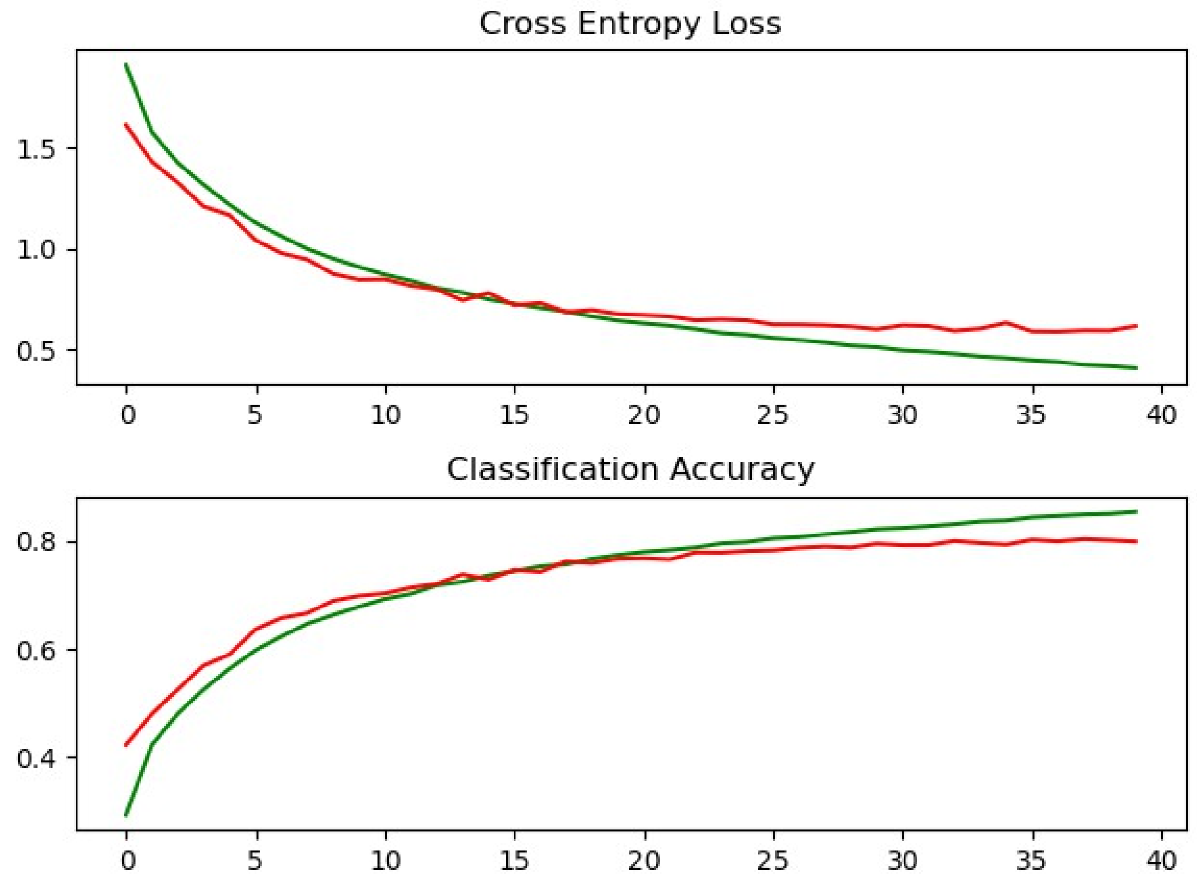
10%, 10%, 10%:



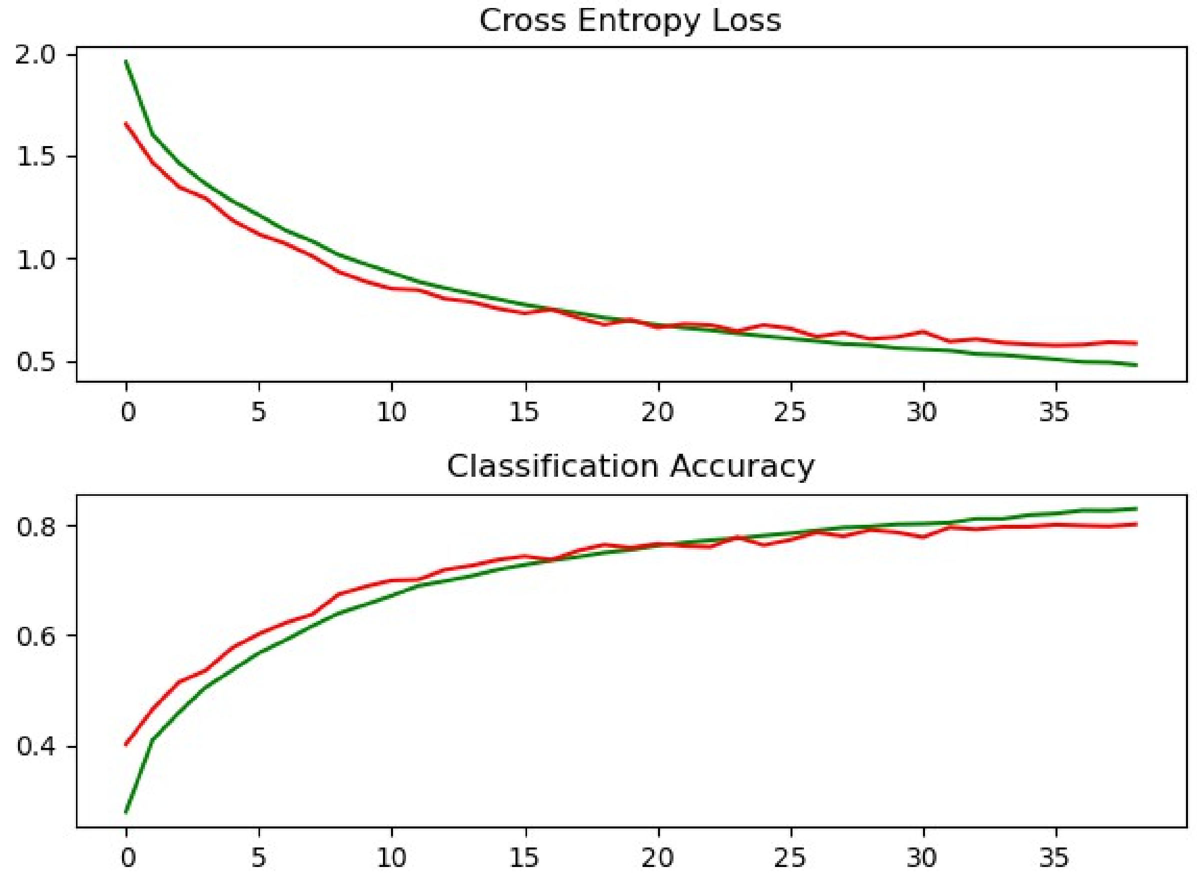
10%, 10%, 20%:



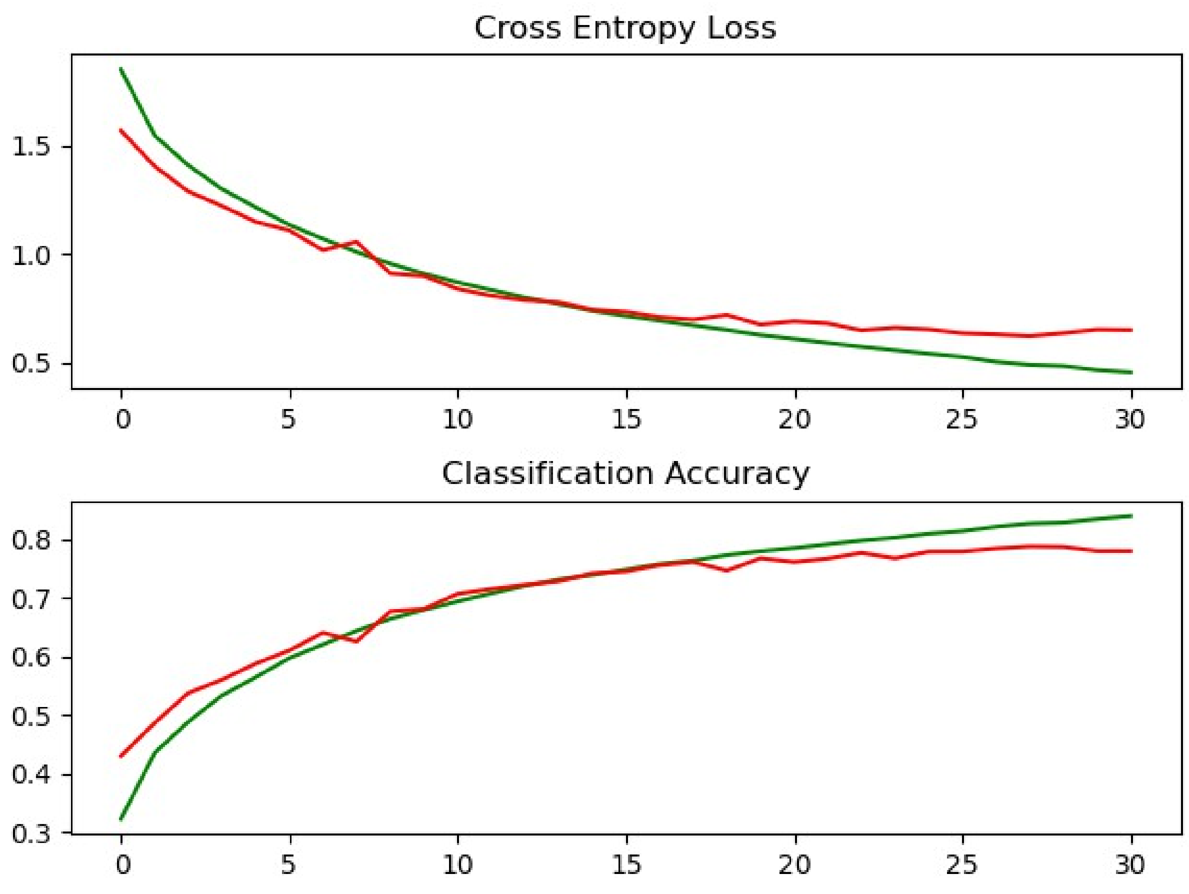
10%, 10%, 30%:



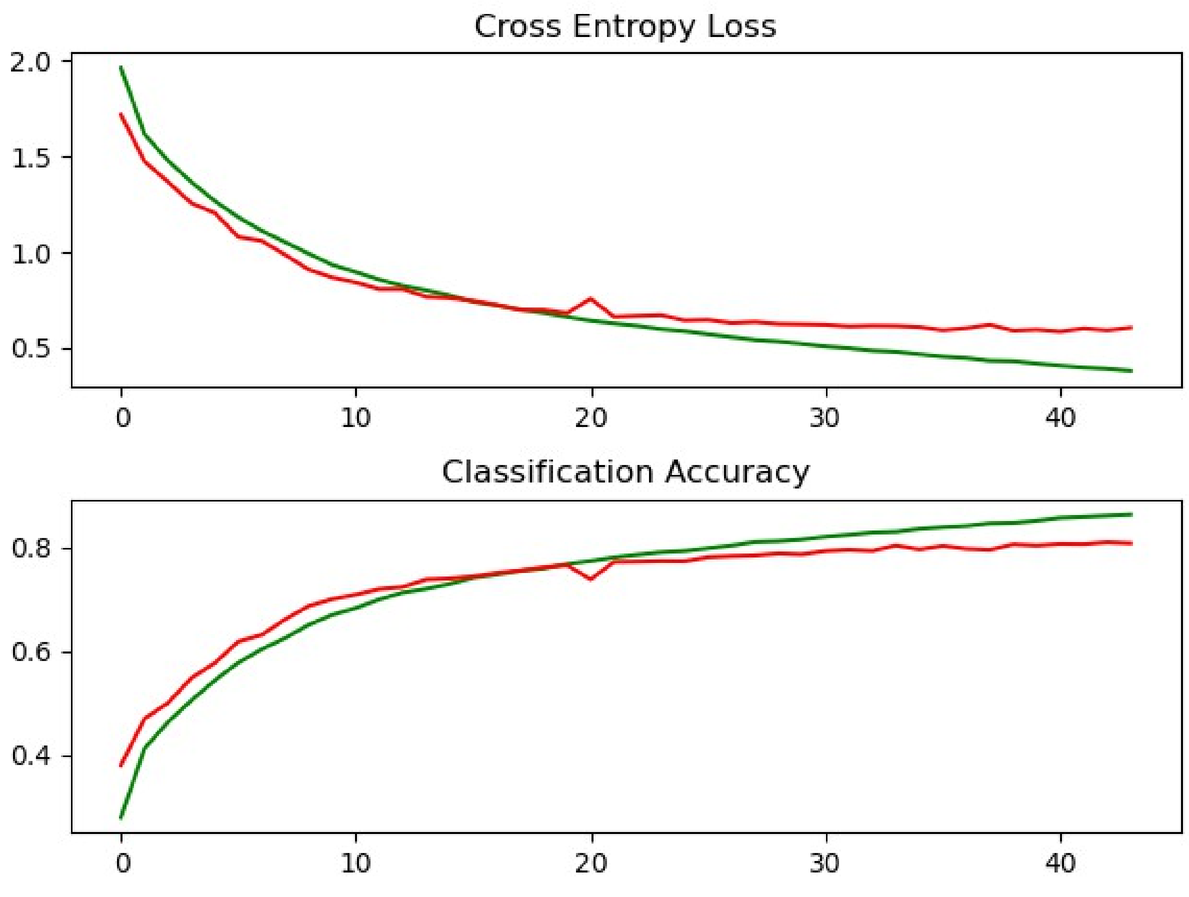
10%, 10%, 40%:



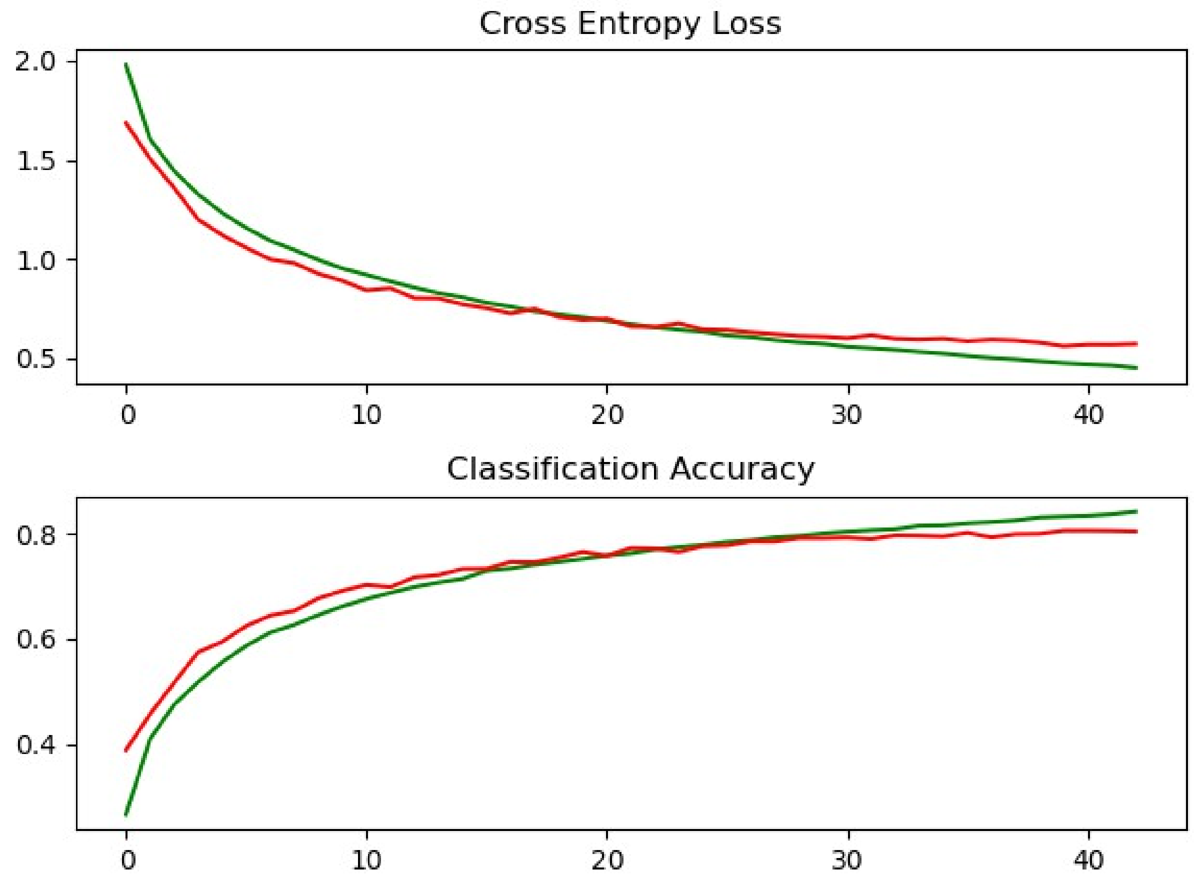
10%, 20%, 10%:



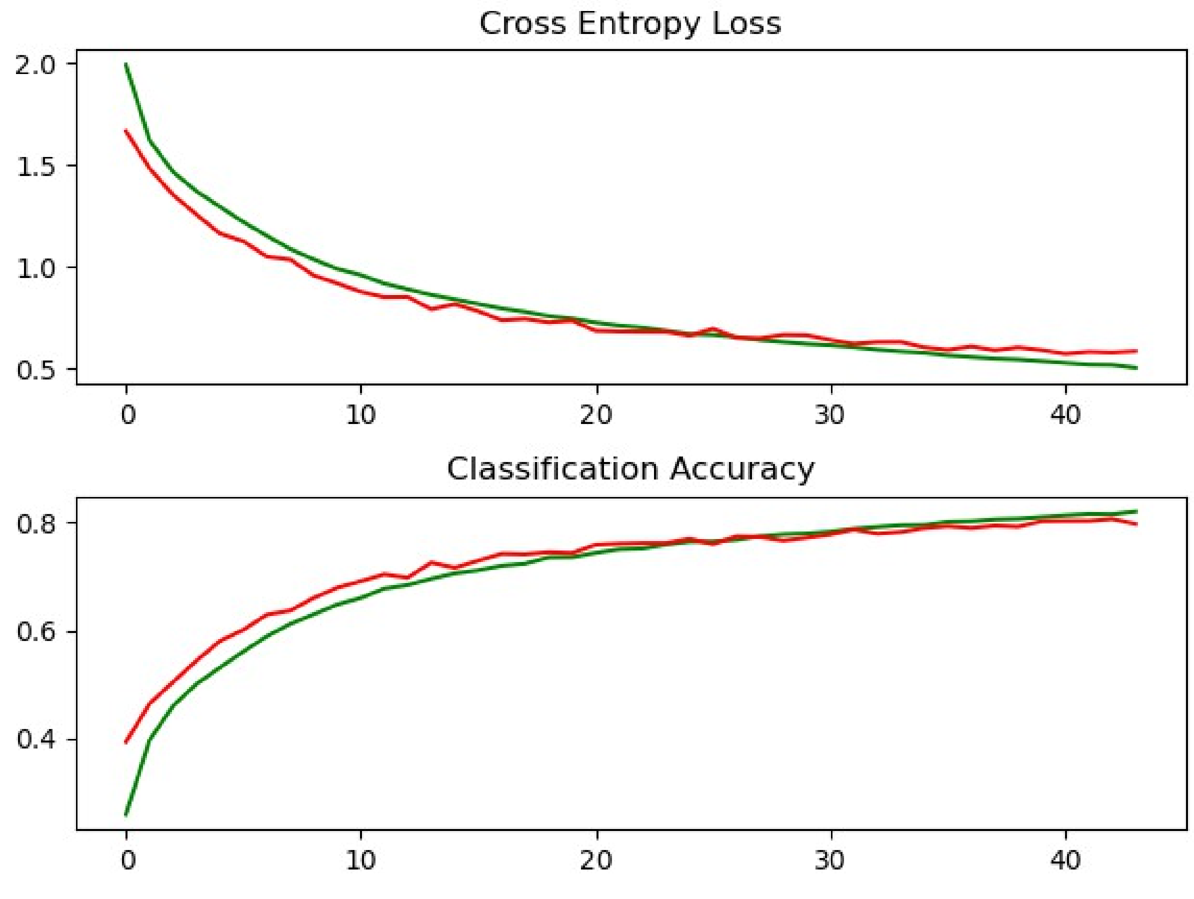
10%, 20%, 20%:



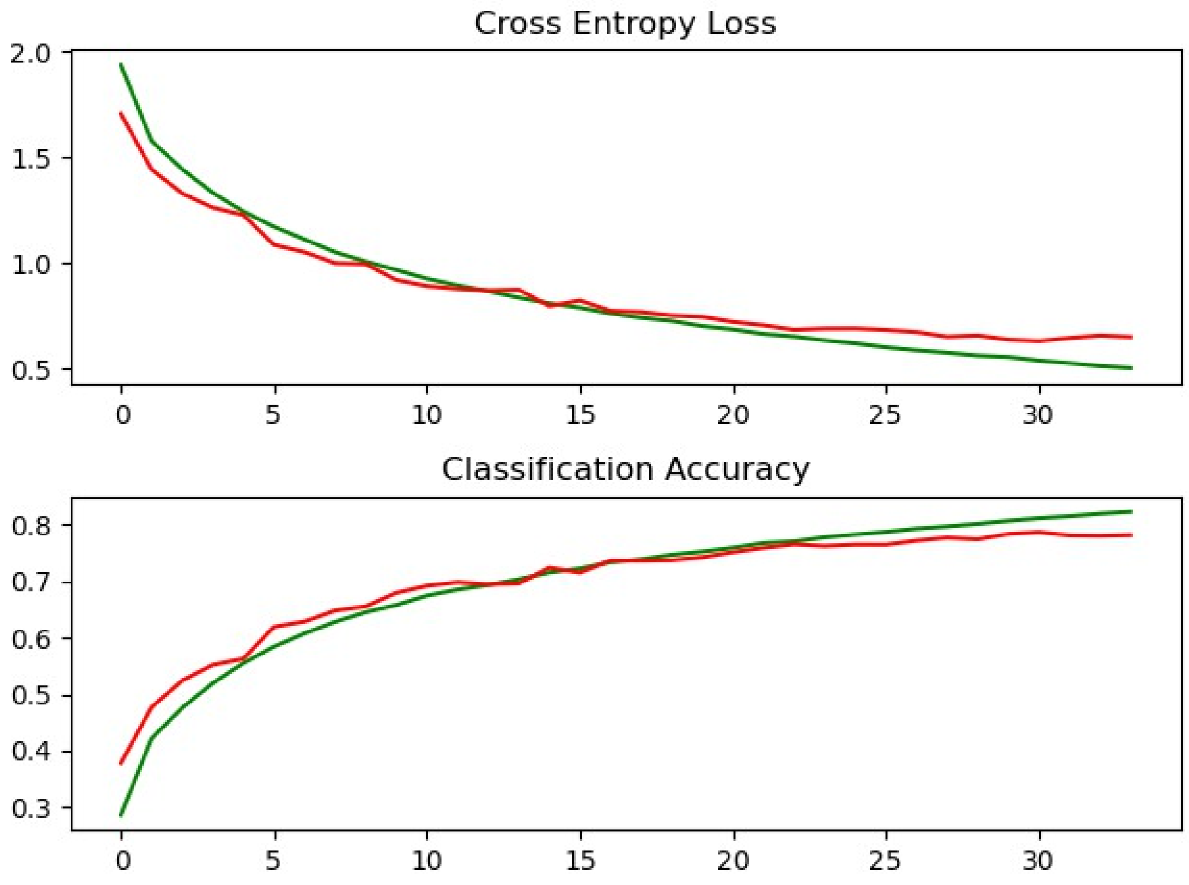
10%, 20%, 30%:



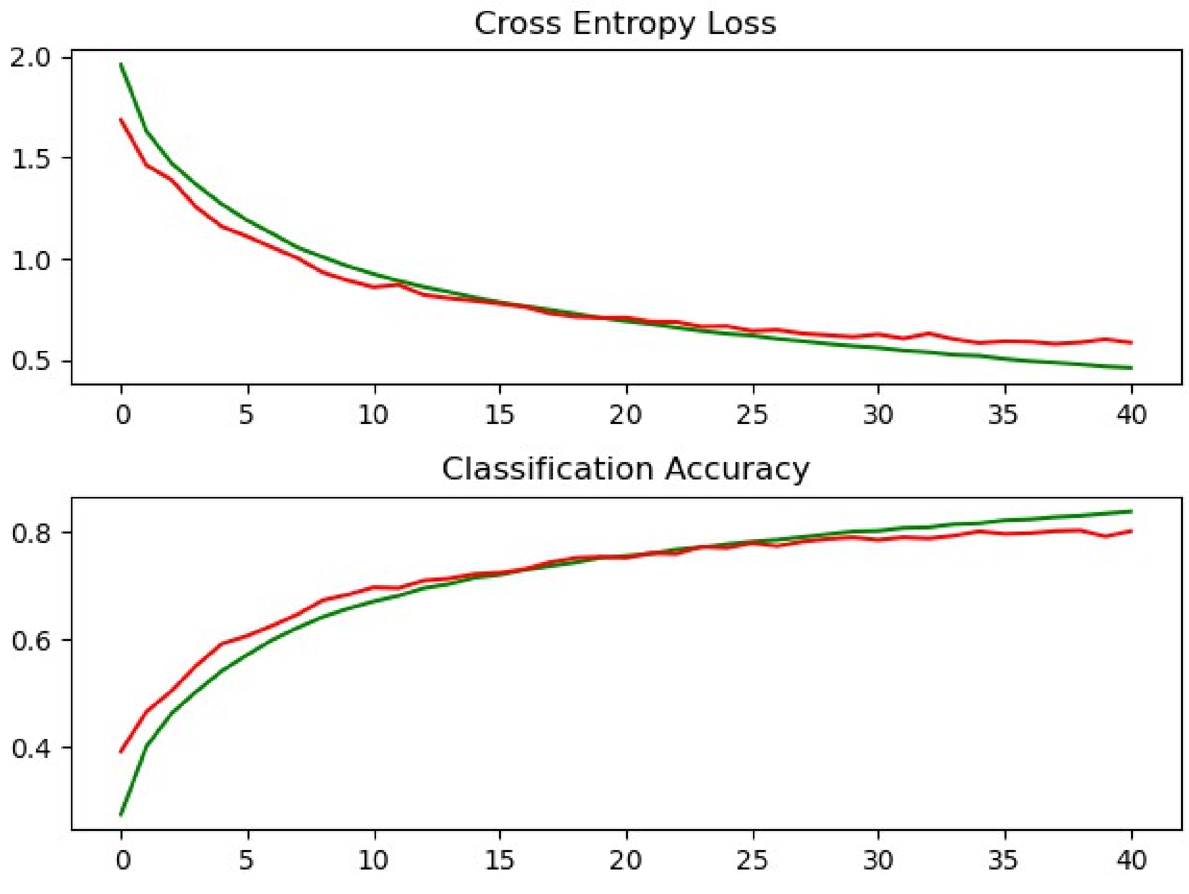
10%, 20%, 40%:



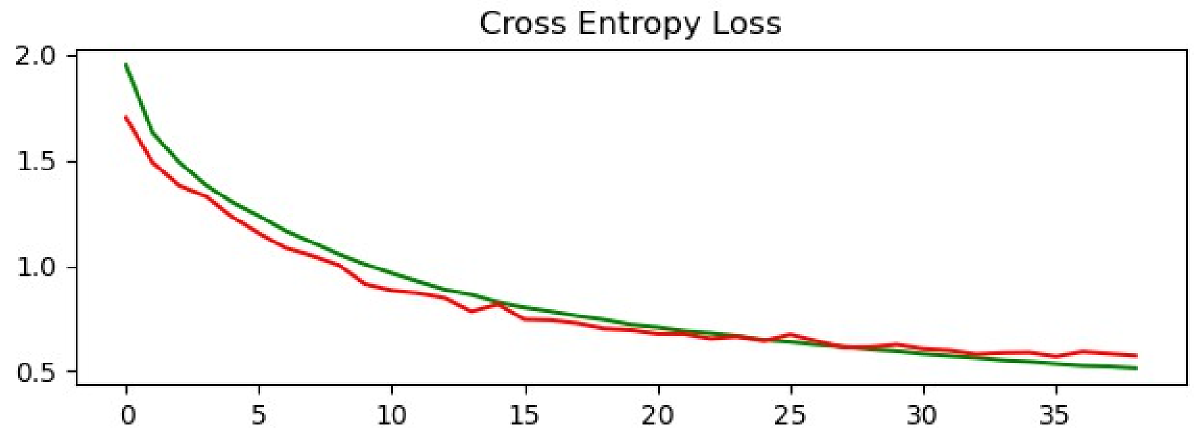
10%, 30%, 10%:

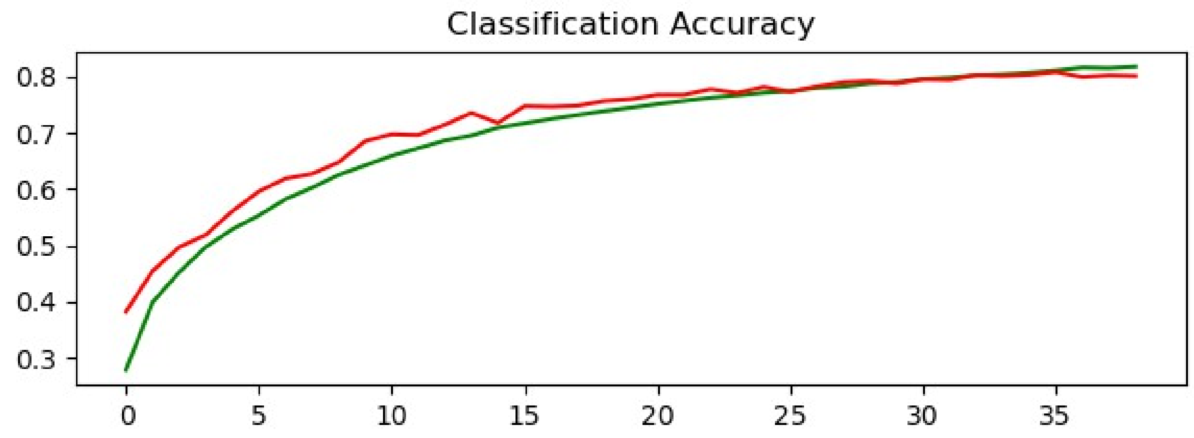


10%, 30%, 20%:

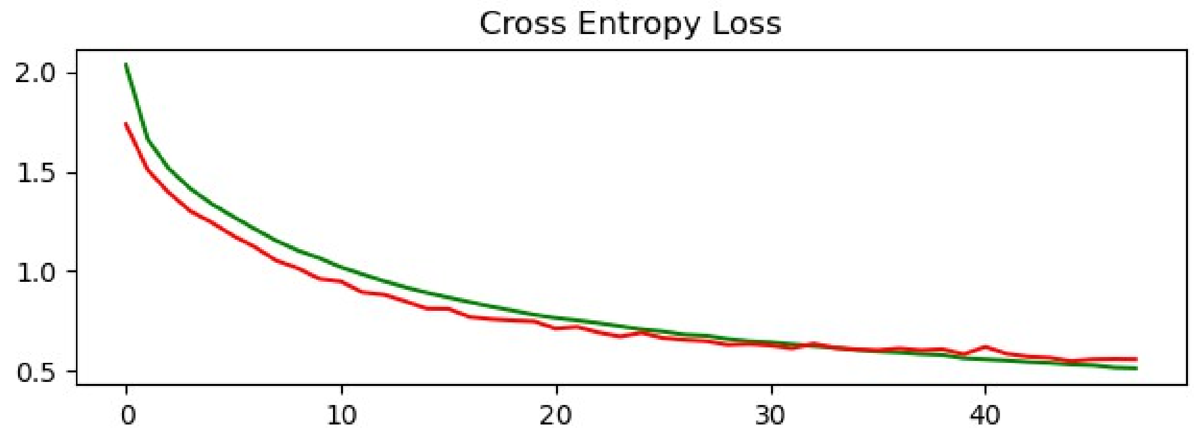


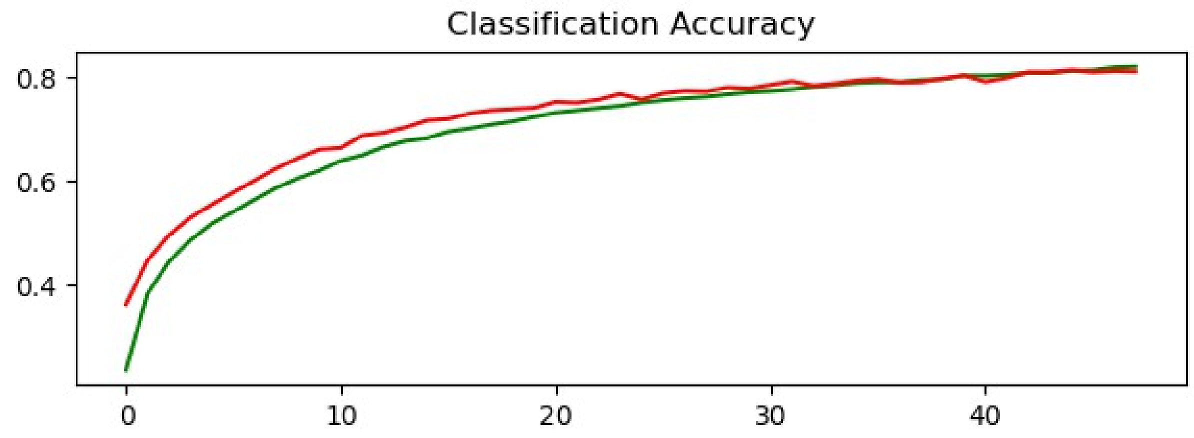
10%, 30%, 30%:



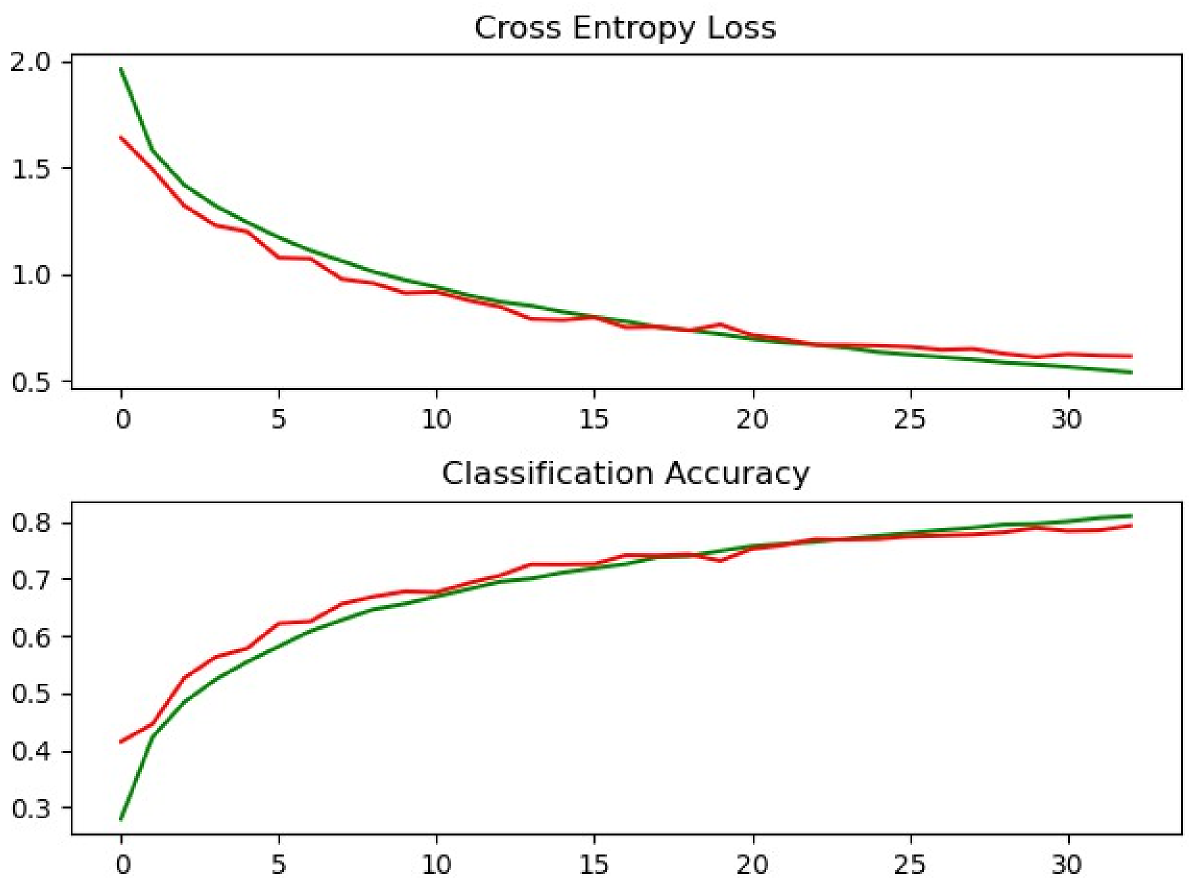


10%, 30%, 40%:

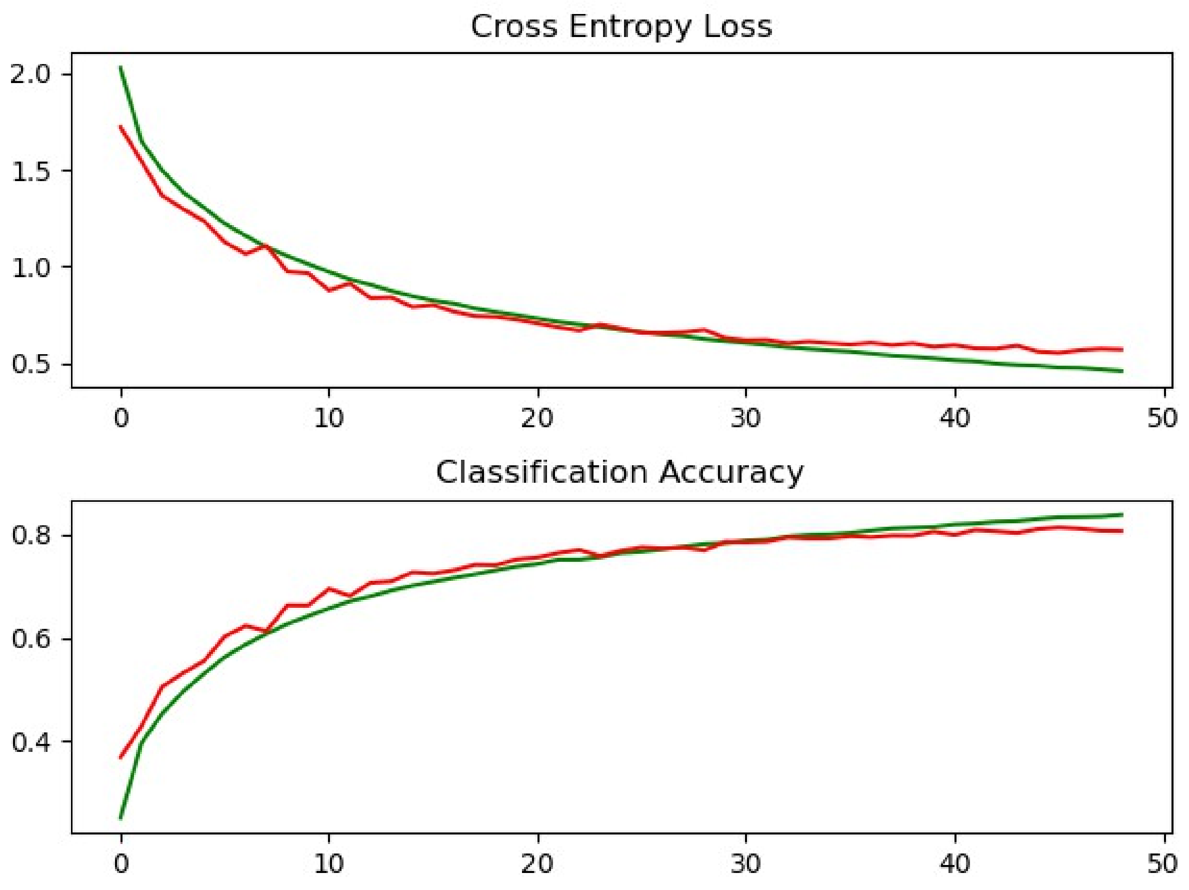




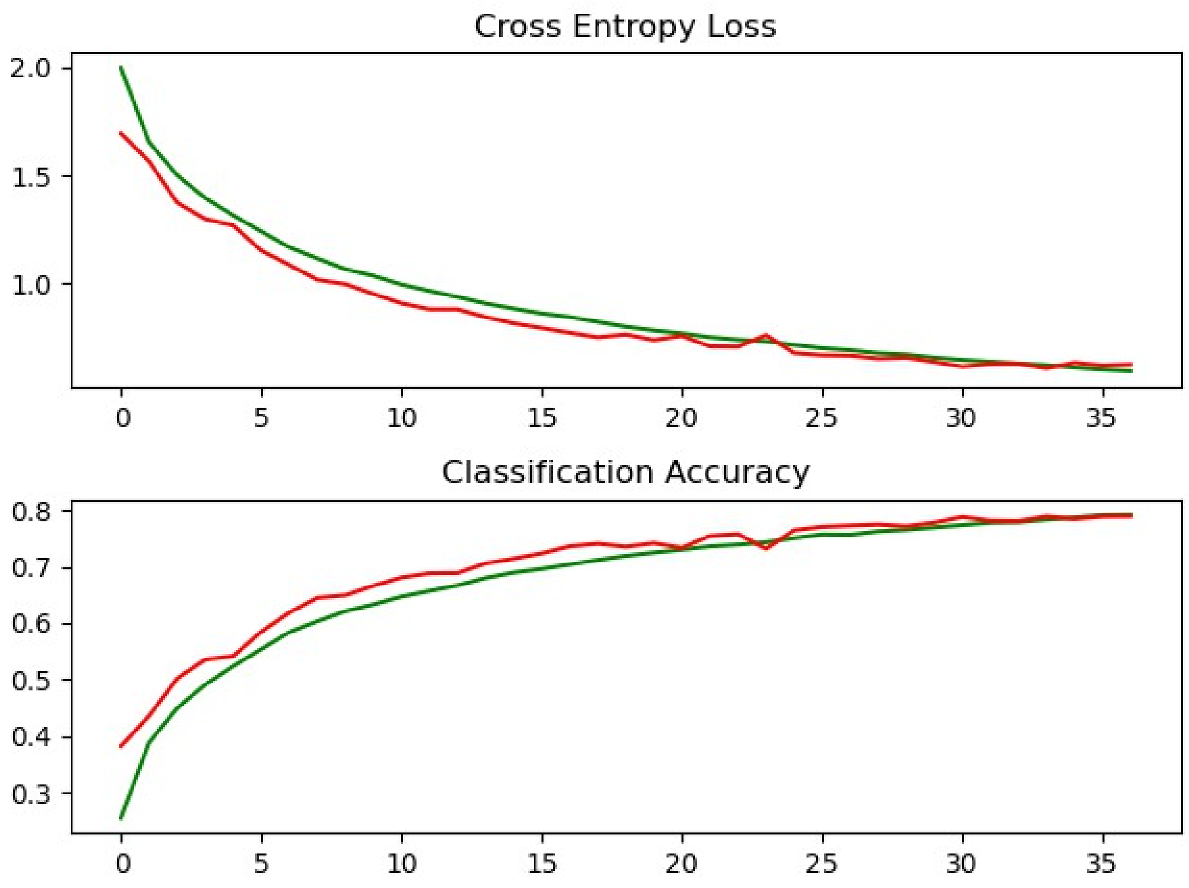
10%, 40%, 10%:



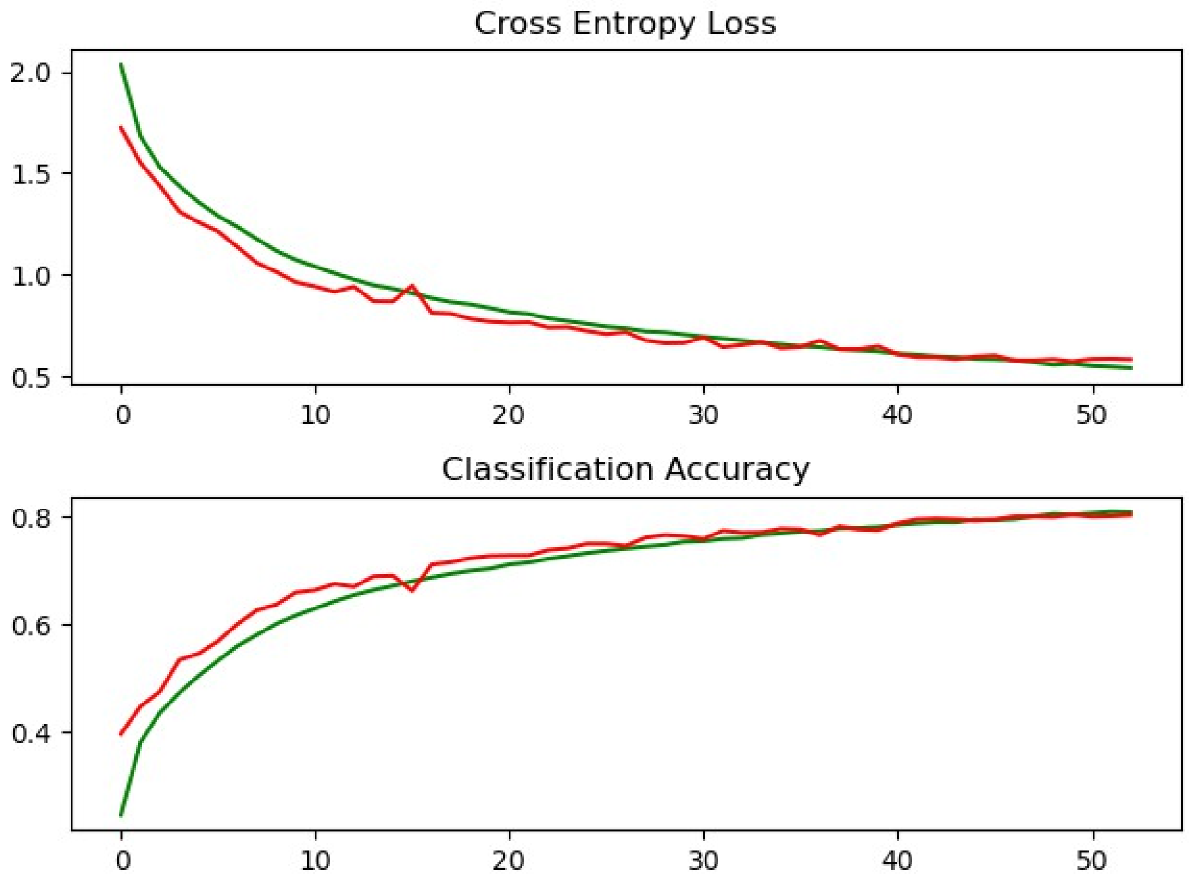
10%, 40%, 20%:



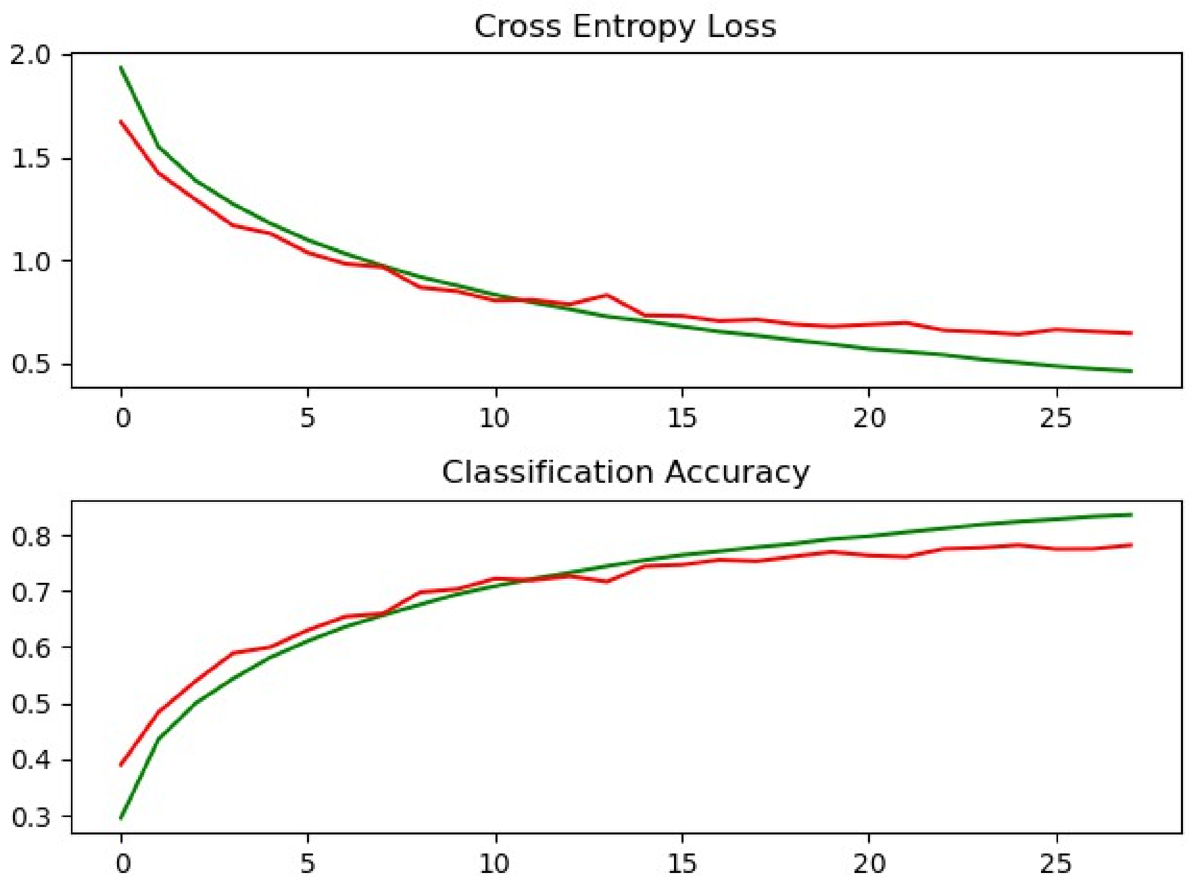
10%, 40%, 30%:



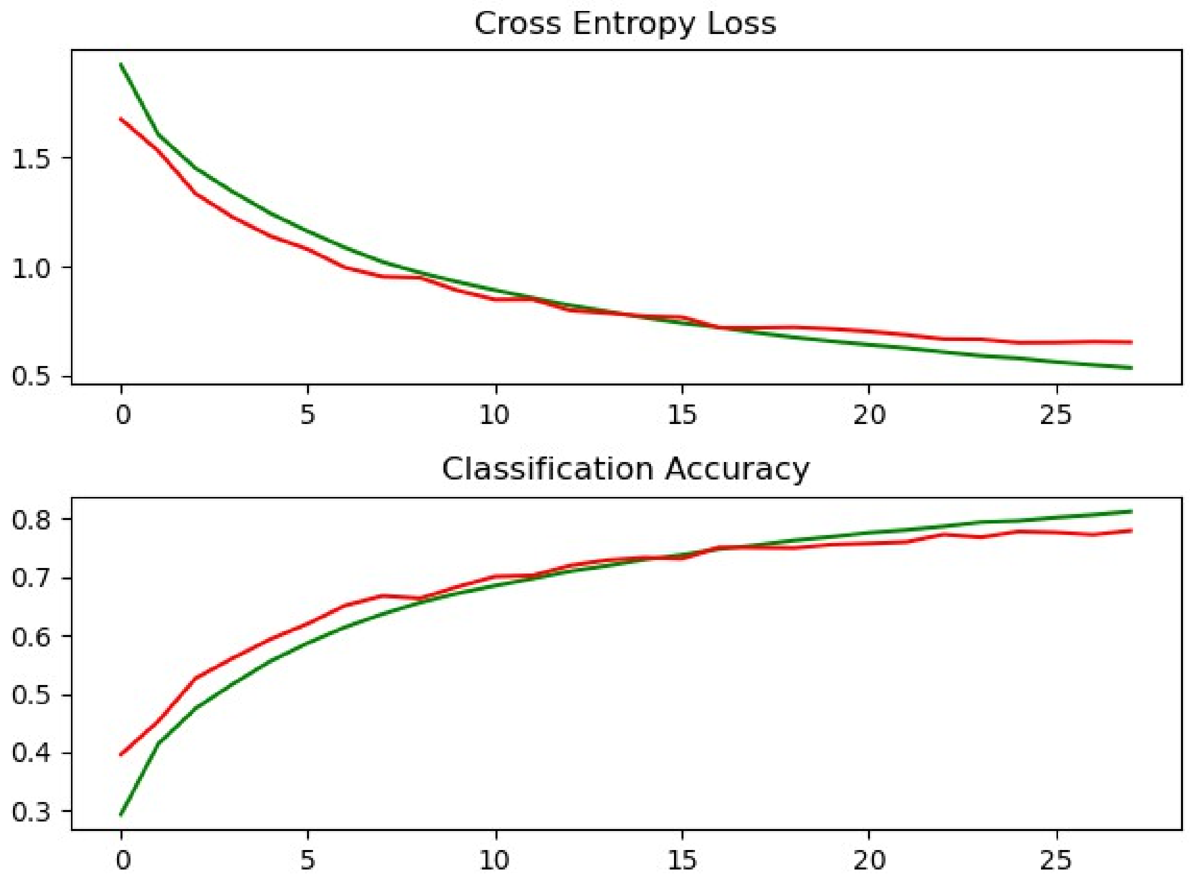
10%, 40%, 40%:



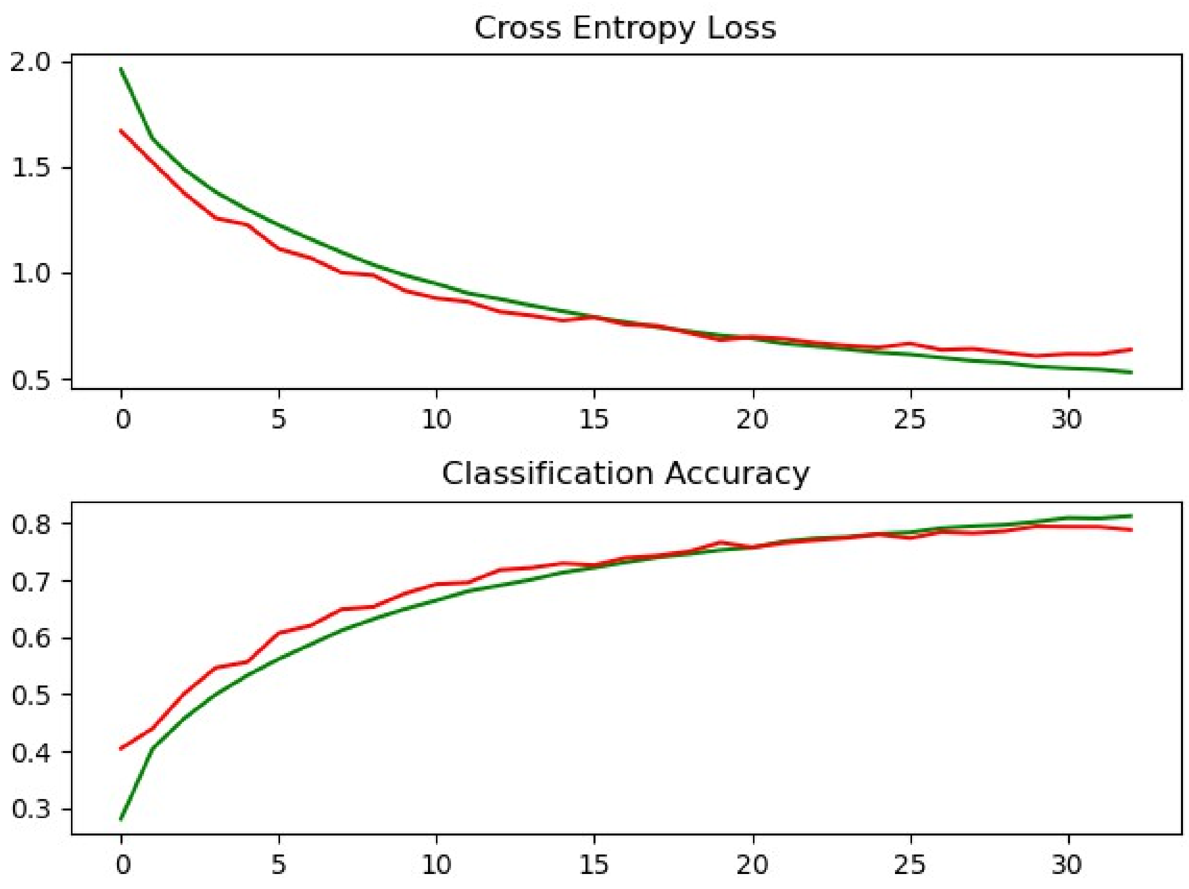
20%, 10%, 10%:



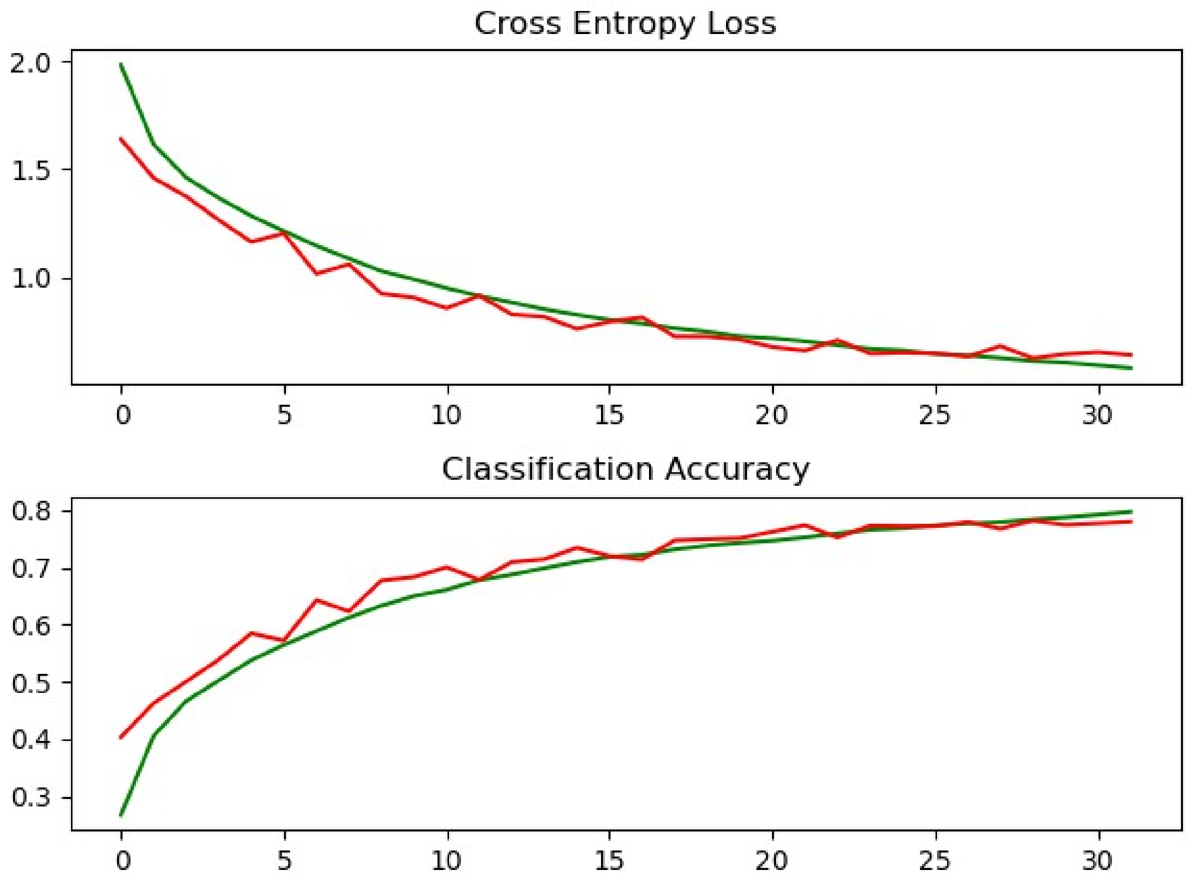
20%, 10%, 20%:



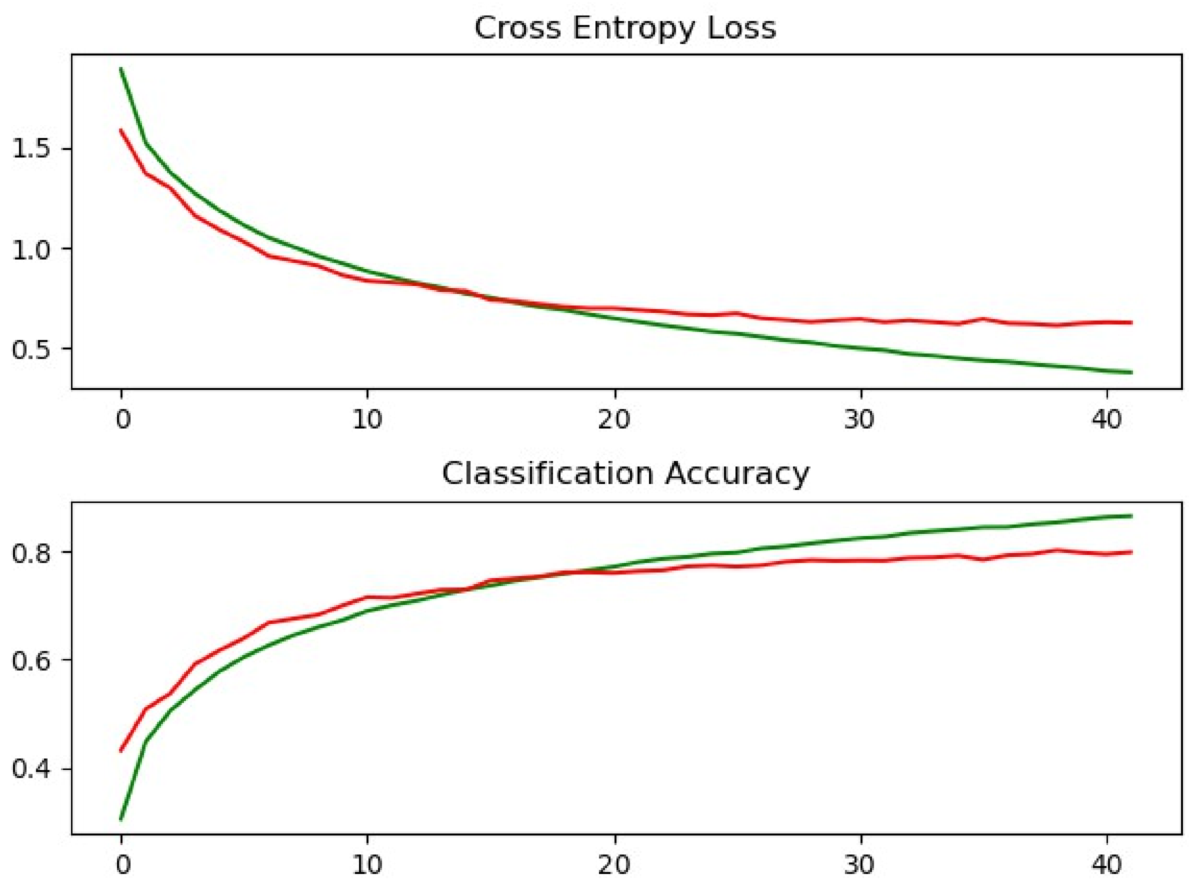
20%, 10%, 30%:



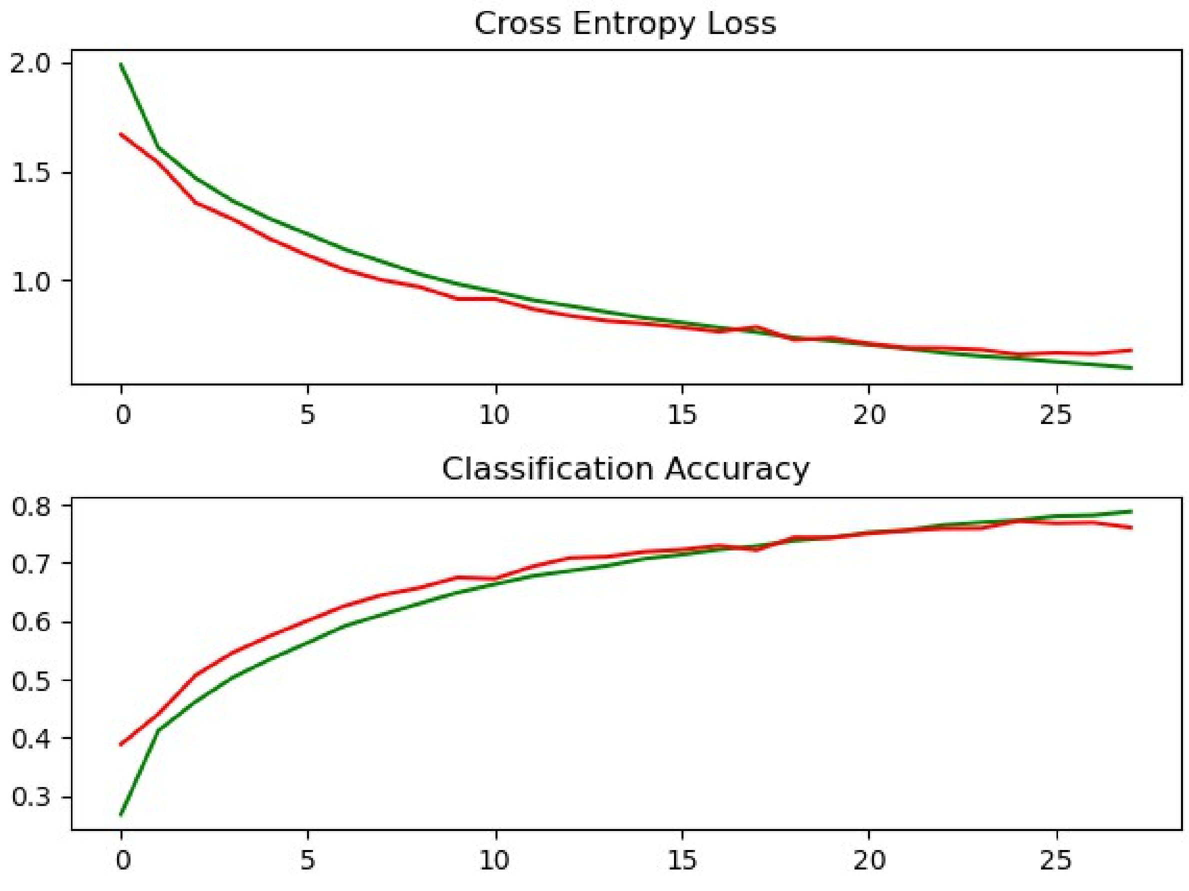
20%, 10%, 40%:



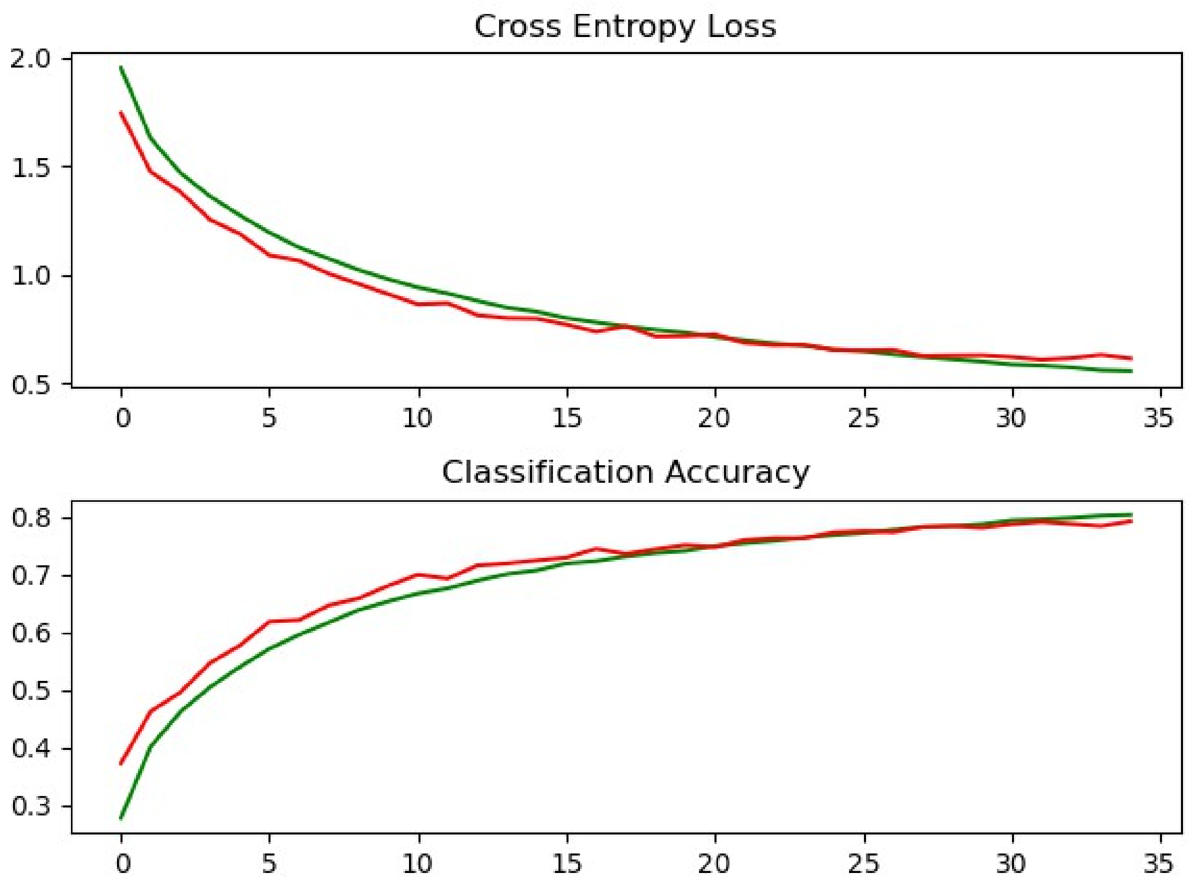
20%, 20%, 10%:



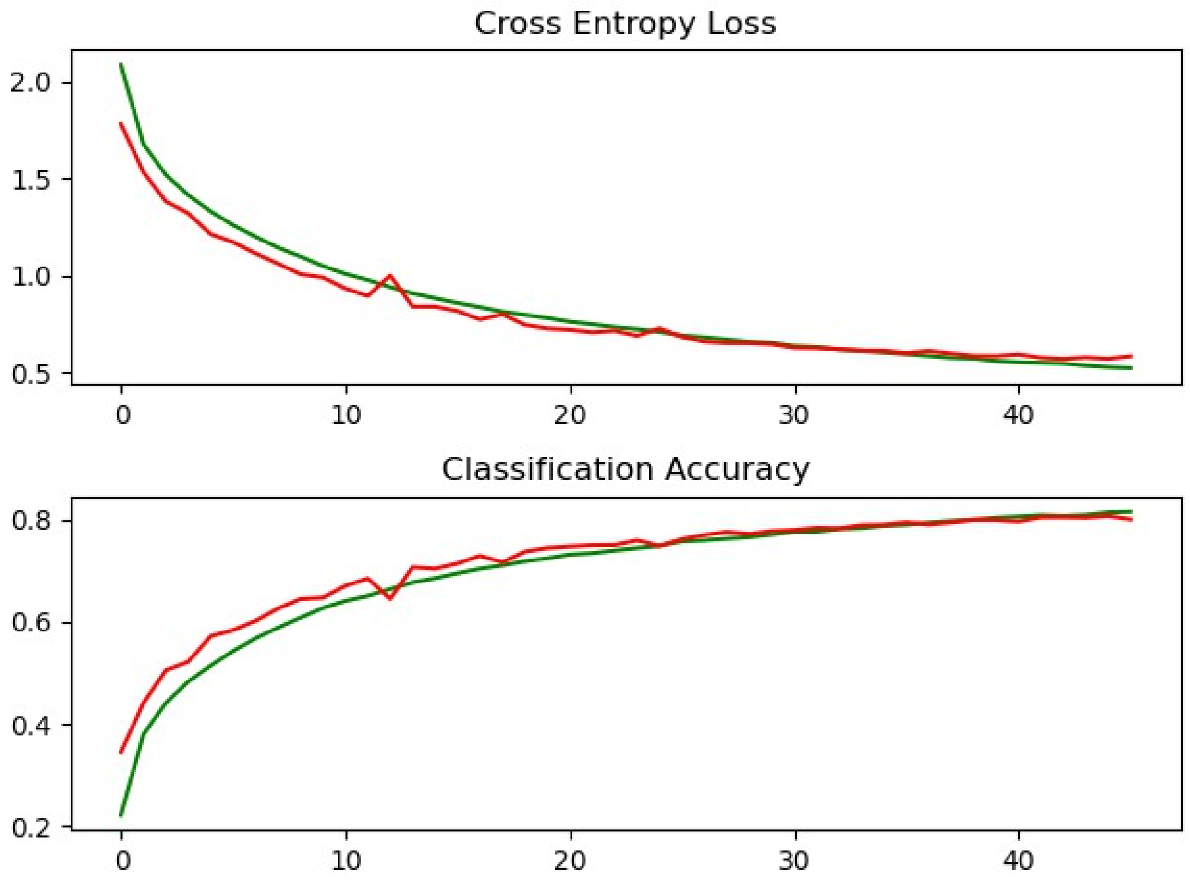
20%, 20%, 20%:



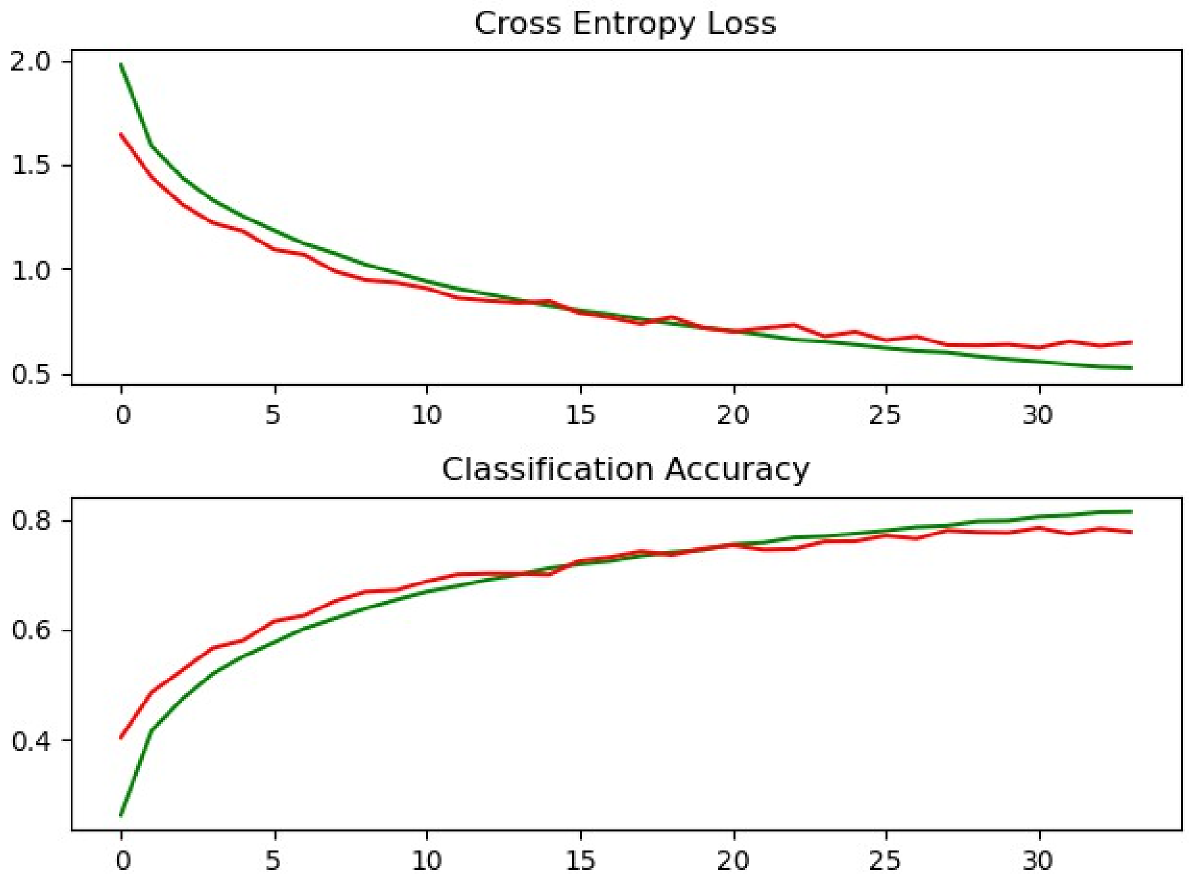
20%, 20%, 30%:



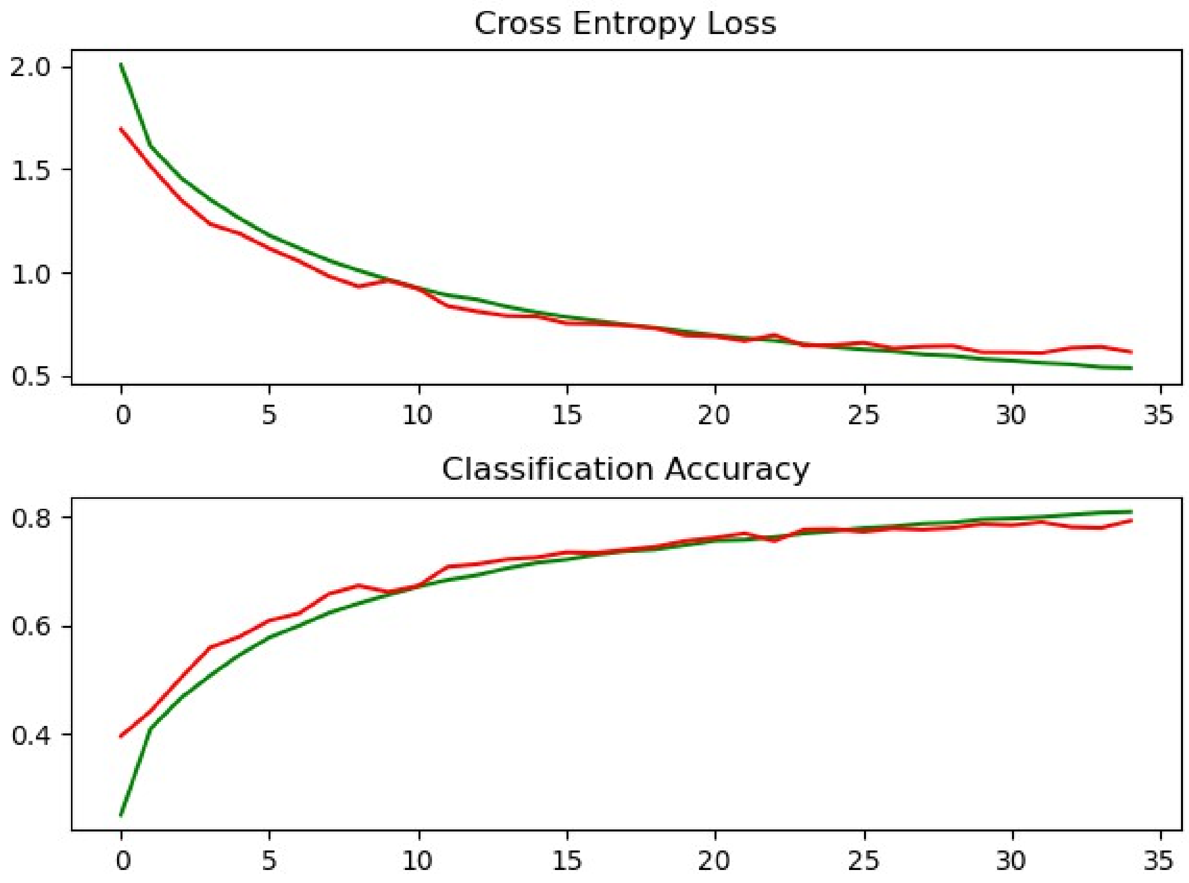
20%, 20%, 40%:



20%, 30%, 10%:



20%, 30%, 20%:



Observations and conclusions:

- using different dropout after each VGG increases accuracy
- the worst accuracy is achieved when dropouts are the same
- the best accuracy(>80) is achieved usually when dropout is rising

5. Summary

After performing the experiments we have came to the following conclusions

- Deeper networks have better accuracy, but take longer to train
- Deeper networks give diminishing returns, each layer added increases the accuracy by smaller amount
- Effect of weight decay is rather small
- Effect of dropout layers is noticeable
- Combination of dropout layers and weight decay offers significant improvement over basic models

Convolutional neural networks are an interesting topic, and in the future, we would like to experiment some more with data augmentation techniques, and do dome more research in this field. We would also like to experiment more with the values of dropout and check batch normalization influence on the net.

Those tests are in progress but they takes a lot of time. They might appear in report v2 if it will be possible,

6. References

- official documentation of TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/
- articles by Jason Brownlee: https://machinelearningmastery.com/how-to-load-convert-and-save-images-with-the-keras-api/#:~:text=Keras%20provides%20the%20load_img(),details%20about%20the%20loaded%20image.&text=Running%20the%20example%20loads%20the,details%20about%20the%20loaded%20image, https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
- article by Sumit Saha: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- dataset: https://www.cs.toronto.edu/~kriz/cifar.html

7. Files

https://github.com/dawidmusialik898/BIAI-project/tree/presentation