Classification on the CIFAR10 Dataset

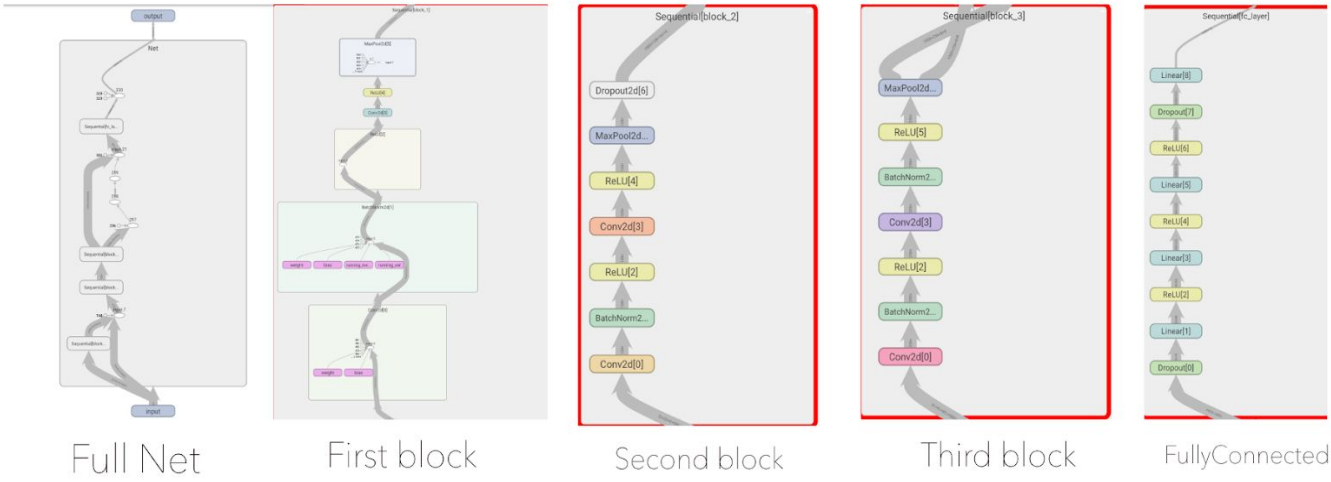
Introduction

In the machine learning domain, classification is a task where the learner learns how to classify a given image into its corresponding label; in this dataset, the learner assigns an RGB image to one of the ten classes namely: Plane, Car, Bird, Cat, Deer, Dog, Frog, Horse, Boat, and Truck. The reason behind classification algorithms' popularity is the fact that it is a form of data analysis that can be used to extract models describing important data classes -- as such, it is a data mining technique used to predict group membership for data instances[1].

The CIFAR-10 dataset is published by the Canadian Institute for Advanced Research, and is widely used as a dataset to introduce the task of classification. Given the novice status of this classification problem, it is far from straightforward in order to achieve substantial classification accuracy. This report delves into the common and uncommon enhancements obtained through literature that are implemented in order to achieve a substantial result of 87% accuracy. State of the art classification models will easily surpass this classification accuracy, but this model was made for the purposes of this project. However, enhancements such as those used in ResNet, AlexNet, and DenseNet have been implemented as shown in the Methods section of this report.

Methods:

Architecture:



[1] https://arxiv.org/pdf/1011.0628.pdf

This architecture has enhancements seen in literature such as Dropout[2], Skip Connection[3], and BatchNormalization[4]. Dropout can be visualized in the Fully Connected block in the architecture figure, skip connection can be visualized by observing that the out-degree of the input node in Full Net is 2, and thus that same input is combined with the output of block 1 to feed into the second block, and batch normalization is observed after almost every convolution layer. On producing the exact model as above, train the model and test it by adjusting hyperparameters based on your preferences, you could also perform grid search or other hyper-parameter tuning methods to find the optimal-ish values for the hyperparameters but do note that it will take a lot of time. Furthermore, do note that adding dropout and batch normalization helps reduce the risk of overfitting a model as explained in the papers cited. In addition to this, l2 regularizer is used with a weight-decay of 5e-3 to aid the prevention of overfitting the model. Skip connection is beneficial for model convergence as it provides an alternative path for the gradients to be computed. It can aid in prevention of the vanishing gradient problem.

This process was run on Google Colab using Pytorch and GPU as a hardware accelerator. As you will find in the notebook, there is support for both: running on CUDA and CPU, pick your choice by adjusting the "device" variable. All you need to do is to press run in the notebook and it will start training a model, additionally you can mount your google drive to save the project as you please.

Training process -- Based on the current hyper-parameters at the time of writing this report, the optimizer algorithm used is Stochastic Gradient Descent which outperforms a few other commonly used algorithms, for example: Adam. After every training epoch, validation is performed on the validation set, as is evident from the test function being called on the validation set. The validation accuracy is then printed, it is also stored to be plotted (if needed, but not recommended as the images will overcrowd the output cell) after all the epochs are done.

Testing process -- Based on the number of epochs and the hyper-parameters, this process can yield different testing accuracies. In this case no labels are provided as the learner is tested to produce its own labels. These predicted labels are then tested with the correct labels *to compute the accuracy.*

Tensorboard metrics -- These metrics are added to enhance understanding of the model and the way it solves the problem. For instance, the architecture image above can be interacted with and understood better in the tensorboard cell of the notebook. Additionally, the probabilities per class are categorically graphed in an interactive tensorboard window called PR Curves. They are adjusted after each training iteration so it would be wise to refresh the tensorboard (unless you turn on automatic refresh) to view the current graphs.

[2] https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
[3] https://arxiv.org/pdf/1512.03385.pdf
[4] https://arxiv.org/pdf/1502.03167.pdf

Evaluation:

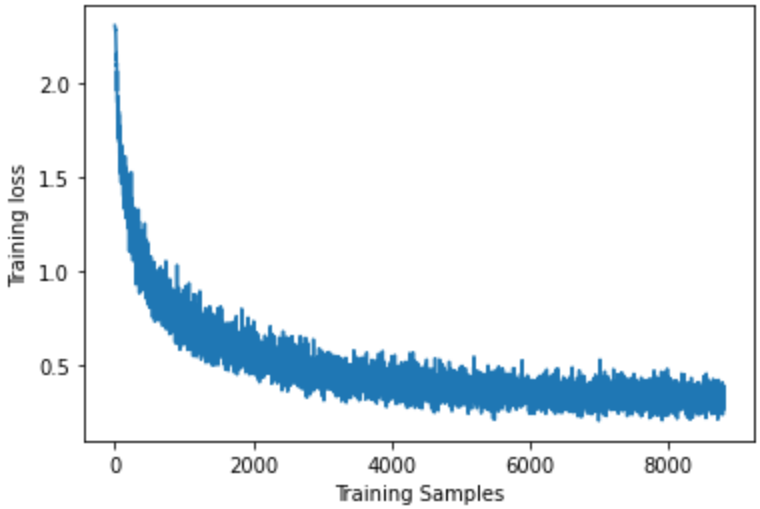
Result: This image contains the accuracy after running the model for 50 epochs and the prediction probabilities of some samples along with their corresponding labels. All the text boxes highlighted in green refer to rightly classified images as opposed to red ones. In this example however, there are no wrongly classified images as the accuracy is pretty high and this is a very small subset of the test images.

```
Result:
 OrderedDict([('correct', 8745), ('accuracy', 87.44999694824219)])
```

Plots:

The graphs below refer to validation accuracy and loss, plotted after each epoch and batch-iteration respectively. Note that the loss could easily be averaged to show a single smooth line but this fluctuation was shown to indicate the intuitiveness in the fluctuation of the CrossEntropyLoss function with each sample of the respective batch.

Training loss:



Validation accuracy is extremely close if not slightly lower than the testing accuracy which is a good sign because that shows that there was no under/over fitting of the model. The accuracy after every epoch is plotted below, the validation accuracy for the final epoch was around 85.1% and the testing accuracy after this entire training process was 87.45%.