# Micronet - Cifar100

This assumes that the system is a linux machine with basic utility tools like tar installed. It also assumes that Anaconda is installed in the machine.

**Setup**
To reproduce the results we will need to create an environment and set up the data which can be done using *setup.sh*.
Now we have the correct environment set up and data is now in the format which the training notebooks expect. We can move forward with the notebook name *13-cifar100-WRN-final.ipynb.*

**Train**
The model is trained using fastai which is a wrapper over PyTorch.
Simply run all the cells of the notebook to train the network, if ran in this specific order it will reach the accuracy > 80% in 600 epochs.

**Explanation**
In my approach I tried using a very small model, to reach an accuracy greater than 80%. This will reduce the efforts required for quantization hence making my focus concentrated on making the model and training regime better. Otherwise, I would have to train a large model and then focus on quantization. All of the development is done on fastai library.

1) **Data Augmentation**
    a) Common data augmentation
    Augmentations like Cropping with reflection padding, left right flip, symmetric warp,
    Rotation (10 degrees), Random zoom, Brightness and contrast have been applied.

    b) Mixup Data Augmentation
    Mixup data augmentation was used, this gave a huge improve in performance of the deep learning model[Link].

2) **One Cycle Learning rate schedule**
    One Cycle learning rate schedule was used which helped in faster and better convergence on training.

3) **Architecture**
    Wide Resnet was used with k=6 and N=3 and a group of 3, totalling to 22 convolutional layers.

**Metric Score**

The scores have been calculated in *14-Compute-flops.ipynb.* The operations are counted for all the convolutional layer, linear layer and average pool layer. No operation is calculated for ReLU and MaxPool layer according to the competition rules. Also, no operations are being calculated for BatchNorm layer due to the optimization that all deep learning libraries do durind the inference mode as shown in this blog, where they explains how batch norm operations are optimised in the using the layer previous to batch norm. The computation of operations has been done using the torchstat library. I modified its code a bit so that it returns Multiplication and addition operation differently.

I modified two lines in torchstat/compute_madd.py
Line #27 and Line #124
Where instead of returning the sum of multiplication and addition operations I return both operations separately as a tuple.

**Total Parameters = 9693108**

1. **Convolutional Layers**
   Total Multiplication ops = 1384562688
   Total Addition ops = 1383342080
   *(Using the freebie quantization)*
   Total ops = (0.5 * Total Multiplication ops) + Total Addition ops
   = 2075623424

2. **Linear Layer**
   There is only one linear layer present in my network.
   Total Multiplication ops = 38400
   Total Addition ops = 38300
   *(Using the freebie quantization)*
   Total ops = (0.5 * Total Multiplication ops) + Total Addition ops
   = 57500

3. **Adaptive Average Pool layer**
   input_shape = (1, 384, 8, 8) # From model summary
   output_size = (1, 384, 1, 1) # From model summary

   Total Multiplication ops = 384
   Total Addition ops = 24576
   *(Using the freebie quantization)*
   Total ops = (0.5 * Total Multiplication ops) + Total Addition ops
   = 24768

**Total Math Operations = 2075705692** (2075623424 + 57500 + 24768)

**Metric Score**

(9693108 / 36.5 * 10^6) + (  2075705692  / 10.49 * 10^9) = **0.46343**