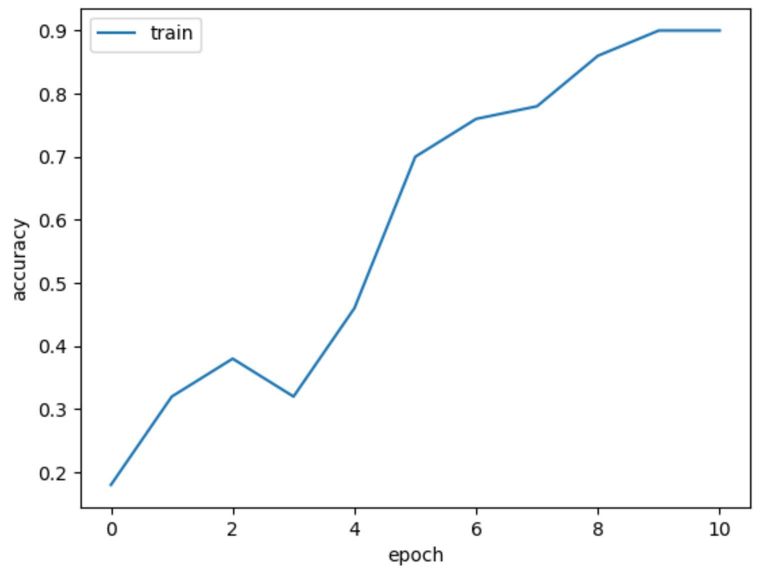
Assignment 2 Writeup

Name: Prateek Gupta
GT Email: pgupta353@gatech.edu

Put your learning curve here:



Part-1 ConvNet

My CNN Model

Describe your model design in plain text here:

1) The model uses a convolution block consisting of 1 Conv2D layer followed by a relu and a maxpool. Three of these convolution blocks are stacked on top of each other. The output is then flattened and passed through a series of linear layer for feature aggregation. Then it is connected to a final output layer. For more refinement, we can add linear layers of 64,32,16 which can further tune the network with limited parameters. We use a kernel size of 3, stride of 1 and padding of 0 in filters.

We donot use drop out and batch normalization as out of sample performance was able to beat the set benchmark.

The intention is to create stacks of Vanilla CNN and refine the architecture instead of starting from scratch. We add more filter, some of which may be redundant and further be pruned to reduce model size.

self.conv2d1=nn.Conv2d(in_channels=3,out_channels=**32**,kernel_size=3,stride=1,padding=0)

- self.conv2d2=nn.Conv2d(in_channels=32,out_channels=**64**,kernel_size=3,stride=1,padding=0)
- self.conv2d3=nn.Conv2d(in_channels=64,out_channels=**128**,kernel_size=3,stride=1,padding=0)
- self.relu=nn.ReLU()
- self.max_pool=nn.MaxPool2d(kernel_size=2,stride=2)
- self.linear1=nn.Linear(in_features=**512**,out_features=256)
- self.linear2=nn.Linear(in_features=**256**,out_features=128)
- self.linear3=nn.Linear(in_features=**128**,out_features=64)
- self.output=nn.Linear(in_features=**64**,out_features=10)

Describe your choice of hyper-parameters: For model training, we reduce the batch size from 128 to 32 for more frequent updates. Based on performance, the learning rate is increased from 0.0001 to 0.01. CE is used as the loss function and we train the data on regular imbalance.

The convolution block is build with an intent to increase the depth by increasing the number of filters. So, the first block has 32 filters, second block has 64 and third block as 128. With this, we aim to extract large number of feature maps. Relu and max pooling are used to reduce the dimensionality and improve gradient passing in back propagation. The kernel size is reduced as we can fit more parameters in the hardware and do not want to lose spatial information. After the three convolution blocks are able to extract the relevant features, we pass them through linear layer and gradually reduce the dimensionality. First linear layer provides 512*256 output, followed by 256*128 and 128*64. Then it is connected to the output layer with 10 nodes.

Stacking convolution blocks helps us learn more combination of features that may eventually led us to a dimension where separation between the 10 classes is easier.

The model is trained for 80 epochs. With 10 epochs, it is able to beat the easy and medium benchmark. Then we train it longer for passing the relevant benchmark. The scheduler and regularization is unchanged from vanilla CNN config.

- Train:
- batch_size: 32
- learning_rate: 0.01
- reg: 0.0005
- epochs: 80
- steps: [6, 8]
- warmup: 0
- momentum: 0.9
- network:
- model: MyModel
- data:
- imbalance: regular # regular or imbalance
- save_best: True
- loss:
- loss_type: CE # CE or Focal
- reweight: True
- beta: 0.99

What’s your final accuracy on validation set?

- **80.7%**

- **Please note that beating the gradescope benchmarking is taken as priority as opposed to beating all available benchmarks.**

Data Wrangling

What’s your result of training with regular CE loss on imbalanced CIFAR-10?

Fill in your per-class accuracy in the table

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
CE Loss	0.8290	0.6500	0.6040	0.2440	0.0040	0.0000	0.0010	0.0000	0.0000	0.0000

Batch Size:32

What’s your result of training with CB-Focal loss on imbalanced CIFAR-10?

Beta	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
0	0.91	0.911	0.58	0.639	0.37	0.115	0.315	0.152	0	0
0.1	0.887	0.887	0.546	0.665	0.398	0.126	0.281	0.111	0.001	0.003
0.2	0.91	0.915	0.597	0.595	0.315	0.156	0.284	0.112	0	0
0.3	0.895	0.912	0.578	0.563	0.354	0.2	0.201	0.156	0.006	0
0.4	0.904	0.902	0.57	0.561	0.347	0.176	0.213	0.156	0.001	0.001
0.5	0.83	0.897	0.467	0.687	0.396	0.183	0.358	0.185	0.004	0
0.6	0.9	0.9	0.558	0.584	0.384	0.147	0.243	0.149	0.001	0.001
0.7	0.899	0.899	0.543	0.642	0.367	0.103	0.224	0.137	0	0
0.8	0.898	0.92	0.553	0.634	0.314	0.151	0.266	0.135	0.006	0.001
0.9	0.907	0.899	0.575	0.624	0.332	0.151	0.27	0.14	0.001	0.004
0.99	0.895	0.893	0.537	0.613	0.349	0.162	0.253	0.206	0	0.001

Put your results of CE loss and CB-Focal Loss(best) together:

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
CE Loss	0.8290	0.6500	0.6040	0.2440	0.0040	0.0000	0.0010	0.0000	0.0000	0.0000
CB-Focal	0.898	0.92	0.553	0.634	0.314	0.151	0.266	0.135	0.006	0.001

Describe and explain your observation on the result:

We trained our model on a batch of 32 instead of 128 to allow more gradient updates. Softmax focal loss is trying to make a trade off between the most frequent and lesser frequent classes by assigning more weights in the loss function. In other words, we believe it is trying to force the model to learn better on the hard examples which are easily misclassified in normal cross entropy loss.

In our setup, we found the best combination had beta of 0.8. We did not tune gamma for reweighting as it was hardcoded to 1 in main.py. With respect to CE, we are able to improvement in out of sample performance for all the classes, including class 0. So, it seems, it is able to apply some regularization on most frequent classes, class 0 and class 1 in our table, which reduces overfitting as well. The loss function in focal loss is looking only at the softmax probability of the true class while CE is summing up losses for all the classes. This could be the reason for the regularization noticed where the model needs to give more attention on the high weight classes to reduce loss and not memorize the noise pattern on the most dominant class. But for class 2, the out of fold performance takes a hit.

The relationship between beta and accuracy may not be monotonically increasing/decreasing. This we can see from the table when beta is gradually increases from 0 to 0.99 where none of the classes increase/decrease in only one direction. But it can still be hard to get a good uniform distribution score on all the low frequency classes in out of sample dataset. It however, is able to mitigate the class imbalance problem for few of the categories, class 7,8 and 9, in our table has non zero accuracy scores now. We choose beta=0.8 as best model because as it is able to hold some predictive power in all of the classes.