Image Classification of CIFAR-100 dataset
Keras

The task this week was image classification for CIFAR-100 dataset.
This week we primarily did our exploration on some architectures (without going into dropouts and augmentation).
We basically tried 3 different architectures:

1. VGG-19
2. 'Network in Network' architecture
3. 'DenseNet' (and its variations)

All the codes were compiled and run on google collab. The validation scores mentioned are based on train validation split of 0.1 (5000 labeled data for validation and 45000 for training)
Results with various architectures are summarised below:

VGG-19:

This architecture implemented did not perform very well. We were not even able to achieve an accuracy of 2% in 20 iterations. Thus we simply disregarded this architecture.

'Network in Network' architecture:

The architecture is given below and is taken from ref[1].

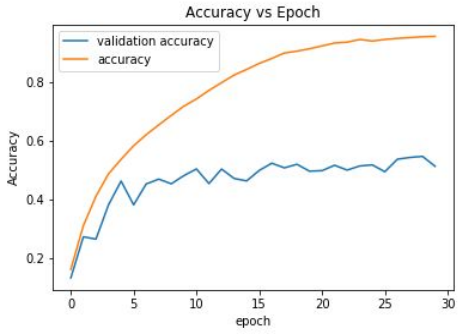
Input Size	NIN
32×32	5x5, 192
32×32	1x1, 160
32×32	1x1, 96
32×32	3x3 max pooling, /2
16×16	dropout, 0.5
16×16	5x5, 192
16×16	1x1, 192
16×16	1x1, 192
16×16	3x3,avg pooling, /2
8×8	dropout, 0.5
8×8	3x3, 192
8×8	1x1, 192
8×8	1x1, 10
8×8	8x8, avg pooling, /1
10 or 100	softmax

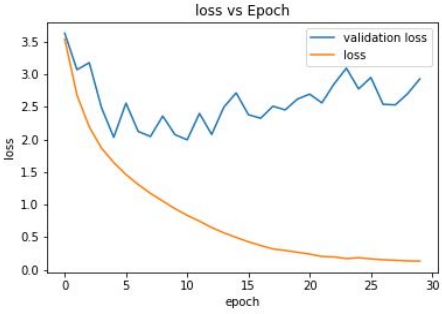
Table 1. CIFAR-10/CIFAR-100 network structure. Each layer is a convolutional layer if not otherwise specified. Activation function is followed by each convolutional layer.

Though our original idea was to actually try different ReLU functions with Network in Network architectures, we decided to try that next instead come up with just a reliable architecture this week.The results were much better than those obtained from VGG-19 network. An accuracy of **~0.41** was achieved on the validation set, without even trying different ReLU functions as cited in the paper.Training with network was very fast compared to the previous one and in ~70 epochs we were able to achieve a validation accuracy of **~0.41**

DenseNet:

Then we tried implemented ‘DenseNet’. The ref[4] and ref[5] were used primarily to understand and implement DenseNet. We tried a few networks. We firstly implemented DenseNet-100-12 (100 Dense layers and 12 growth rate). We set the compression equal to 1, thus each dense layer had same number of dense layers (= 16 in this case). Also We trained the model for 30 epochs and an accuracy of **0.54** was achieved on the validation set (submission values) (graphs included below, weight decay=0.0). We then tried with dropout for 20 epochs the results weren’t improving.(*3.5 hour training time on google collab*) We also implemented a custom network with just 51 layers (12 growth rate, 48 dense layers) the results were an accuracy of **~0.52.** This network was stated nowhere till now and its accuracy can be improved, if we use augmentation. This we’ll try this week.





Week: 2

What we TRIED

1. We tried various things such as we implemented a simple neural network architecture borrowed from Ref[6] with some variations such addition of a few extra layers and batch normalization with an Image augmentation technique. Adding Batch Normalization improved accuracy by 6%. Also, the simplicity of the network has reduced the time of execution of the code significantly. The validation accuracy improved to 60% but due to some reasons, the predictions on test data showed an accuracy of 20%. Without image augmentation, the same architecture produced an accuracy of 51% which was less than what we predicted last time.

Our architecture looked like:

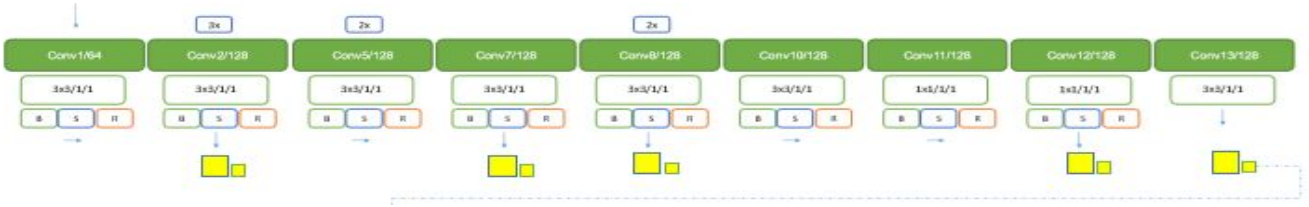
Layer type	# Filters	Window	Stride
Convolution	96	3×3	
Convolution	96	3×3	
Max pooling		3×3	2
Convolution	192	3×3	
Convolution	192	3×3	
Convolution	192	3×3	
Max pooling		3×3	2
Convolution	192	3×3	
Convolution	192	1×1	
Convolution	10/100	1×1	
Global averaging		6×6	
Softmax			

Image Source: Ref[6]

2. We also tried implementing the simple net architecture as suggested by Ref[7]. With variation in the number of layers(reduced) and the

addition of dropout at different locations. The architecture was comparatively slow and showed a slow increase in accuracy which increased in finally converged to around: 53% after 200 epoch with each epoch taking about 42 sec on google colab-GPU.

The architecture looked like:



Source: Ref[7]

Hyper Parameters used above: Optimizer: Adam, No. of Epochs: 70, Batch Size: 64, Loss function: categorical cross-entropy

Image Augmentation Parameters: rotation_range=20, width_shift_range=0.2, height_shift_range=0.2,

What we Implemented: We modified the architecture given to us in assignment 2 part b (CNN) to try something much simpler than what we were trying earlier. We added dropouts before each layer and Batch normalization as well.

Architecture (without dropout and BM**)**

Conv1: (64) 3*3 filter size
Pool1: 2*2 Max Pooling
Conv2: (128) 3*3 filter size
Pool2: 2*2 Max Pooling
Conv3: (256) 3*3 filter size
Pool3: 2*2 filter size
FC1: Fully Connected Layer with 512 outputs
FC2: Fully Connected Layer with 256 outputs
Softmax Layer

The results were pretty good as it gave us around 55-56% accuracy, with 150 epochs. We chose a batch size of 1000. Each epoch took around 19-20 seconds to execute on Collab. The Simplified architecture was efficient both in terms of time and storage since there were fewer layers, as a result, the weight and parameters were also not high. This also reduced the computations needed for neural net training.

Week 3:

Ensemble Parallel Neural networks

This time we were able to increase the accuracy to 63.5%. We implemented 3 different convolution neural network architecture on the same train data. The predictions from the three neural nets were taken and were weighted averaged with weights in the ratio of their individual validation accuracy.

Result: This method gave an accuracy better than any of the individual networks, the networks had accuracy in range 55-60 individually while the weighted average had an accuracy of 63.5. The neural nets implemented were not very deep but were of approximately 10 layers each running parallelly, as a result, the time of execution on google colab was around 1 hr only.

Reasoning: This happened because each neural net was able to extract certain features from the image and implementing different neural nets helped in extracting a set of important features to give a better result.

Observations: The accuracy improved when we used ‘elu’ as activation function instead of ‘relu’. Adding a convolution layer of (1,1) filter size also improved accuracy. Batch Normalization after every step was helpful in improving accuracy and train test difference.

Architecture 1(without dropout and BM**)**

Conv1: (128) 3*3 filter size
Conv1: (128) 1*1 filter size

Conv2: (128) 3*3 filter size
Pool1: 2*2 Max Pooling
Conv3: (256) 3*3 filter size
Conv3: (256) 1*1 filter size
Conv3: (256) 3*3 filter size
Pool2: 2*2 filter size
Conv2: (512) 3*3 filter size
Conv3: (512) 1*1 filter size
Conv3: (512) 3*3 filter size
Pool3: 2*2 filter size
FC1: Fully Connected Layer with 1024 outputs
Softmax Layer

Architecture 2(without dropout and BM**)**

Conv1: (64) 3*3 filter size
Conv1: (64) 1*1 filter size
Pool1: 2*2 Max Pooling
Conv2: (128) 3*3 filter size
Conv2: (128) 1*1 filter size
Pool2: 2*2 Max Pooling
Conv3: (256) 3*3 filter size
Conv3: (256) 1*1 filter size
Pool3: 2*2 filter size
FC1: Fully Connected Layer with 512 outputs
FC2: Fully Connected Layer with 256 outputs
Softmax Layer

Architecture 3(without dropout and BM**)**

Conv1: (128) 3*3 filter size
Conv1: (128) 1*1 filter size
Conv2: (128) 3*3 filter size
Pool1: 2*2 Max Pooling

Conv3: (256) 3*3 filter size
Conv3: (256) 1*1 filter size
Conv3: (256) 3*3 filter size
Pool2: 2*2 filter size
Conv2: (512) 3*3 filter size
Conv3: (512) 1*1 filter size
Conv3: (512) 3*3 filter size
Pool3: 2*2 filter size
FC1: Fully Connected Layer with 1024 outputs
Softmax Layer

Final pred =(6*pred1 + 5*pred2 + 4.5*pred3)/15.5

Week4:
This week we tried two things. First we tried incorporating the information given by broad classes (second last index). We tried building a model which will firstly predict the broad class, then based on the broad we will predict the class of the image. However we were not able to implement the complete thing properly.

The second thing we tried was similar to our last weeks approach of using ensemble modelling. This time we used a single architecture (Architecture1 of previous week) with different batch sizes and Adam optimizer. The number of epochs were 15 in all cases and the resulting probabilities from each model were averaged to give the final probability. The model gave us an accuracy of 66.32% on the test set which was higher than what we had last week. The batch sizes chosen were 1000,500,200 and 100.

Week 5:

This week we tried various things but were unable to increase the accuracy.
Things we tried:

1. Ensemble learning: We took a certain amount of random training data and trained it on various models (5 different models with 4 variations in hyper parameters.) The method took a lot of time and provided an accuracy of 67%
2. We tried adding image augmentation too but time taken for execution increased and increase in the accuracy was not very high
3. We also tried dividing data on the basis of labels and train each type of label but were unable to get promising results
4. We are thus submitting our old submission.

Week 6:

Due to other submission, quizzes and assignments we were unable to try new things. Hence we submit our previos weeks predictions

References:

VGG-19 :

https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

Network in Network:

Ref[1]: **'**Empirical Evaluation of Rectified Activations in Convolution Network**'** **https://arxiv.org/abs/1505.00853**

Ref[2]: 'Batch-normalized Maxout Network in Network' https://arxiv.org/abs/1511.02583

Ref[3]: 'Network in Network' https://arxiv.org/pdf/1312.4400.pdf

DenseNet:

Ref[4]:https://github.com/seasonyc/densenet/blob/master/densenet.py

Public source for implementing DenseNet

Ref[5]:'Densely Connected Convolutional Networks' https://arxiv.org/abs/1608.069

Tuned CNN:

Ref[6]:'arXiv:1502.05700v2 [stat.ML] 13 Jul 2015

Ref[7]: arXiv:1608.06037v7 [cs.CV] 14 Feb 2018