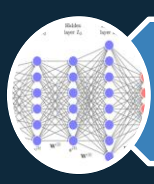
Enabling Deep learning for IoT

with efficiently scheduled Edge computing

-Shubhang Bhatnagar
N1903718K

Why use Deep learning?

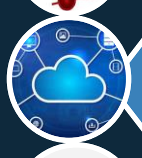


Deep learning is very efficient at many tasks traditional machine learning cannot do

It is increasingly being used in low power IoT devices for various uses

What are the problems encountered in using it?

It requires a lot of data and computational power to work



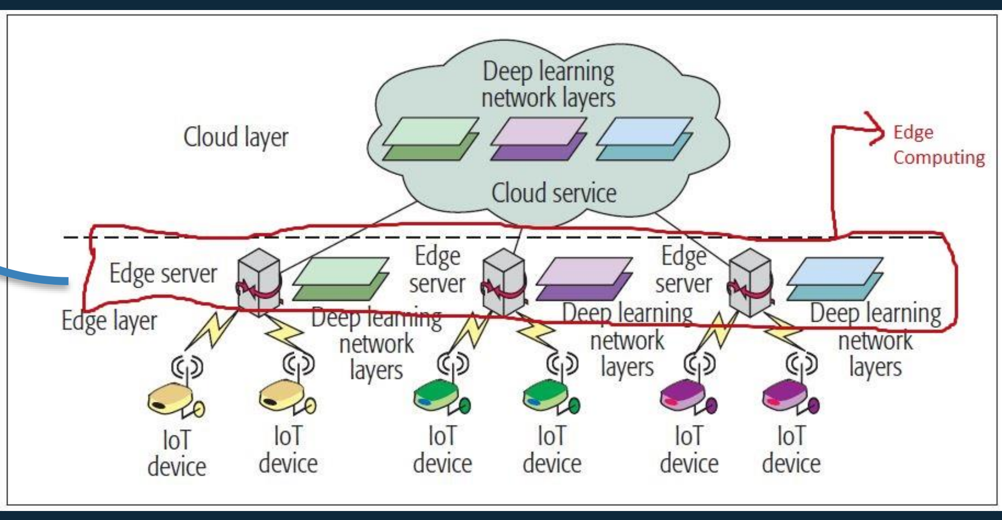
Leads to use of cloud servers for processing the huge amount of data

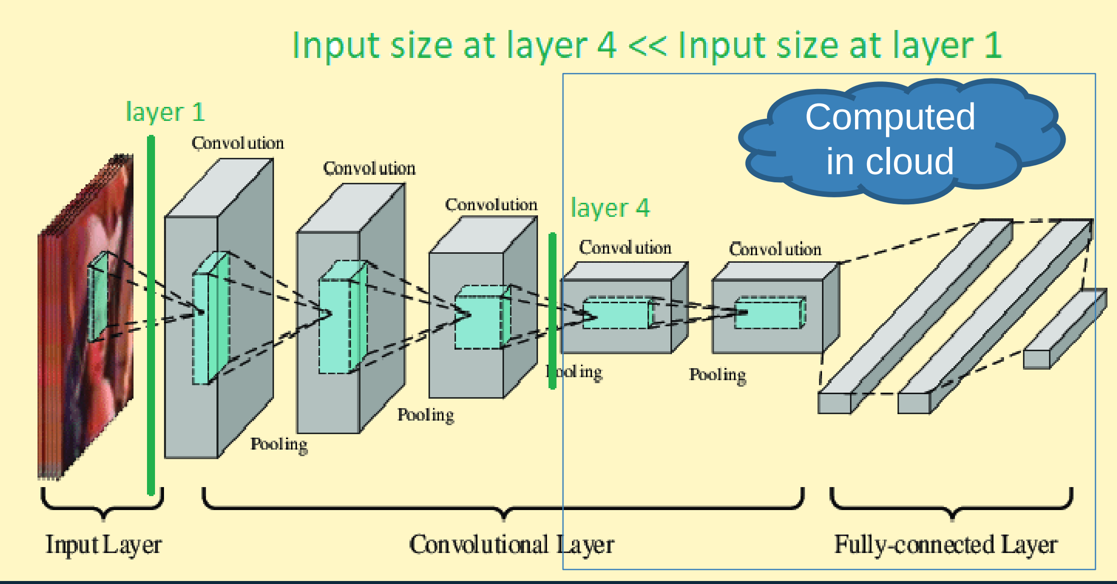
This leads to network congestion and high latency due to the large data flows

What's the solution ?

Use edge servers to perform a part of the computation at the edge

Structure of typical deep learning networks suitable for division into several independent layers





Data size and complexity decreases as we move to higher layers (closer to output)

So perform computations upto the first 'k' (=4 in figure) layers at the edge. Then send the reduced size input data to the cloud.

But, this leads to more questions

How to decide what to compute at edge?

There are a lot of variables to be considered including

- the currently available spare capacity in edge network
- the bandwidth to the cloud available
- the specific DL architecture and application being used

So, how to decide the 'k' for different deep learning architectures in different network traffic scenarios?

The solution- An efficient Scheduling algorithm

to decide k and allocate edge server capacity to different tasks

Constraints to be met

- Bandwidth between cloud and Edge network
- QoS requirements for different applications are met
- Total computing capacity of edge network not exceeded
- Efficiently schedule tasks of each deep net

Types of Scheduling employed

- Offline algorithms
- Online algorithms

Notation used

- k=number of layers counted from input at which architecture is split between edge and cloud
- c=remaining service capacity of edge server
- d=rate of data input for task
- Q=max acceptable latency
- l =remaining computational overhead after k layers
- b=bandwidth assigned to edge server for connecting to cloud
- r = ratio by which dt size is educed due to processing k layers in edge

Offline scheduling algorithm details

1. Find out the best split (value of k) for all possible different architectures used. It should give the best reduction of data size per unit computation in the edge.
2. Sort all tasks and servers in ascending order of input data size
3. Starting from task with largest data size, allocate each task to all servers if they have enough capacity and bandwidth to meet constraints
4. If the constraints are not satisfied, we vary the value of k to try to satisfy the constraints. If not satisfied for any k, then wont schedule that

Online Scheduling algorithm

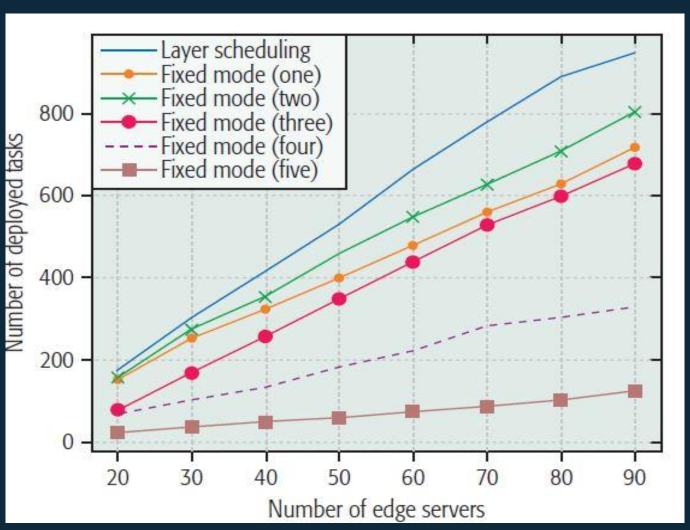
1. Find out the best split (value of k) for the incoming task. Also get B min and B max, the minimum and maximum bandwidths required for the task. Find the edge server with max incoming data for this task
2. For that edge server, calculate $\varphi(c\ ) = (B^{min}.e/B^{max})^{c}.B^{max}.e$
3. If $(b-d.r/Q)(c-d.l) \leq \varphi(c\ )$, Distribute the task onto the edge severs. Else send complete task to cloud. The edge server does not have enough capacity

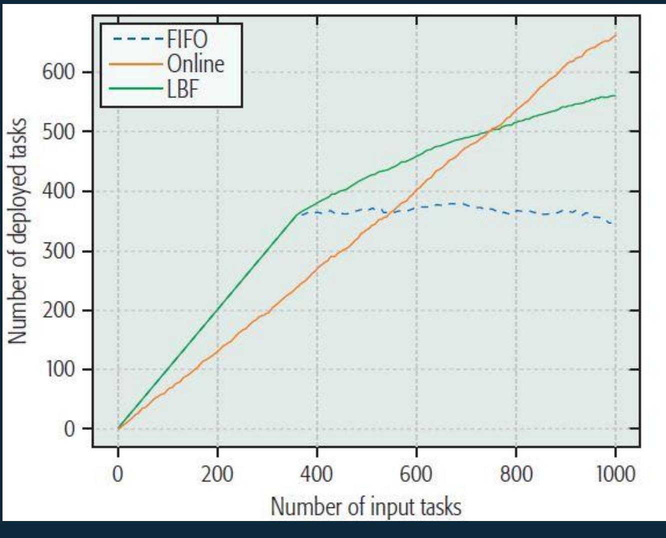
Does it really work?

To find out, the authors simulated the online algorithm using the networkx simulator and compared it with other contemporary techniques. Here is a short summary of the results-

First, the algorithm is compared with models where always a fixed number of layers are computed in the edge networks. The number of edge servers is varied.

The scheduling algorithm significantly outperforms the fixed mode models, and the gap increases as the scale of deployment of edge servers increases.





Testing it against commonly used online scheduling algorithms including FIFO and LBF(least bandwidth first)-

We compare the different algorithms as the number of tasks(deep learning) is gradually increased.

Again, the online scheduling algorithm outperforms both FIFO and LBF at large scales.

We can see that it really works well ! It consistently deploys a larger number of tasks on the edge than it's peers

Applications

Better health anomaly detection in wearables

Obstacle detection for Autonomous vehicles

Cheap in-Ear translators

Future Improvements

- Specially optimized deep neural networks, which are designed to allow cloud and edge networks both to do part of the computation while minimizing traffic on the network can be made for different applications
- More work can be done on finding the best scheduling algorithms for DL on the edge servers. This is one of the first papers to tackle this problem. More efficient or economical or quicker algorithms can be worked out, depending on the application need, taking into account costs for using edge infrastructure, cloud servers and the connecting networks
- Experiments about performance of such scheduling algorithms can be carried out on real networks, instead of being simulated as done in the paper. Real experiments would be more precise and take into consideration many other real world factors too