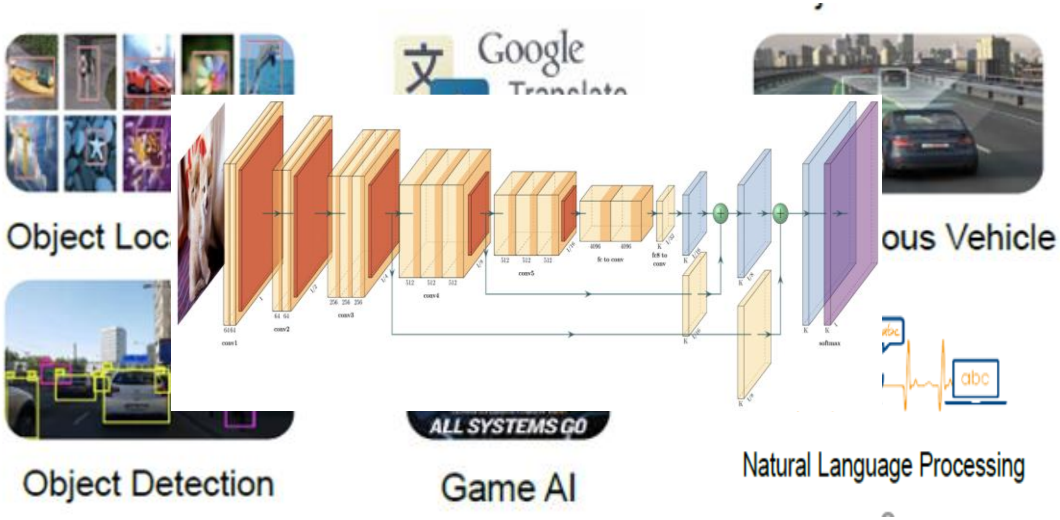
Edge^{n}AI: Distributed Inference with Local Edge Devices and Minimal Latency

Maedeh Hemmat, Azadeh Davoodi, and Yu Hen Hu

Department of Electrical and Computer Engineering
University of Wisconsin - Madison

Background



- Significant increase in memory and computational resources
- Relying heavily on sensors and IoT devices to gather data
  - Transferring raw data to cloud for processing
  - Incurring latency for real-time application

Motivation

- Solution: Distributed inference of complex DNNs across multiple edge devices

  Alleviate or remove reliance to cloud

  Avoid sending large data to cloud

  Incurring significant latency and communication overheads

- This work is inspired by two motivations:
  - Minimizing the overall latency of inference in a distributed network
    - Coupled with reducing the communication overheads
  - Utilizing as many edge devices as available
    - The number of IoT devices are projected to grow into billions[1]

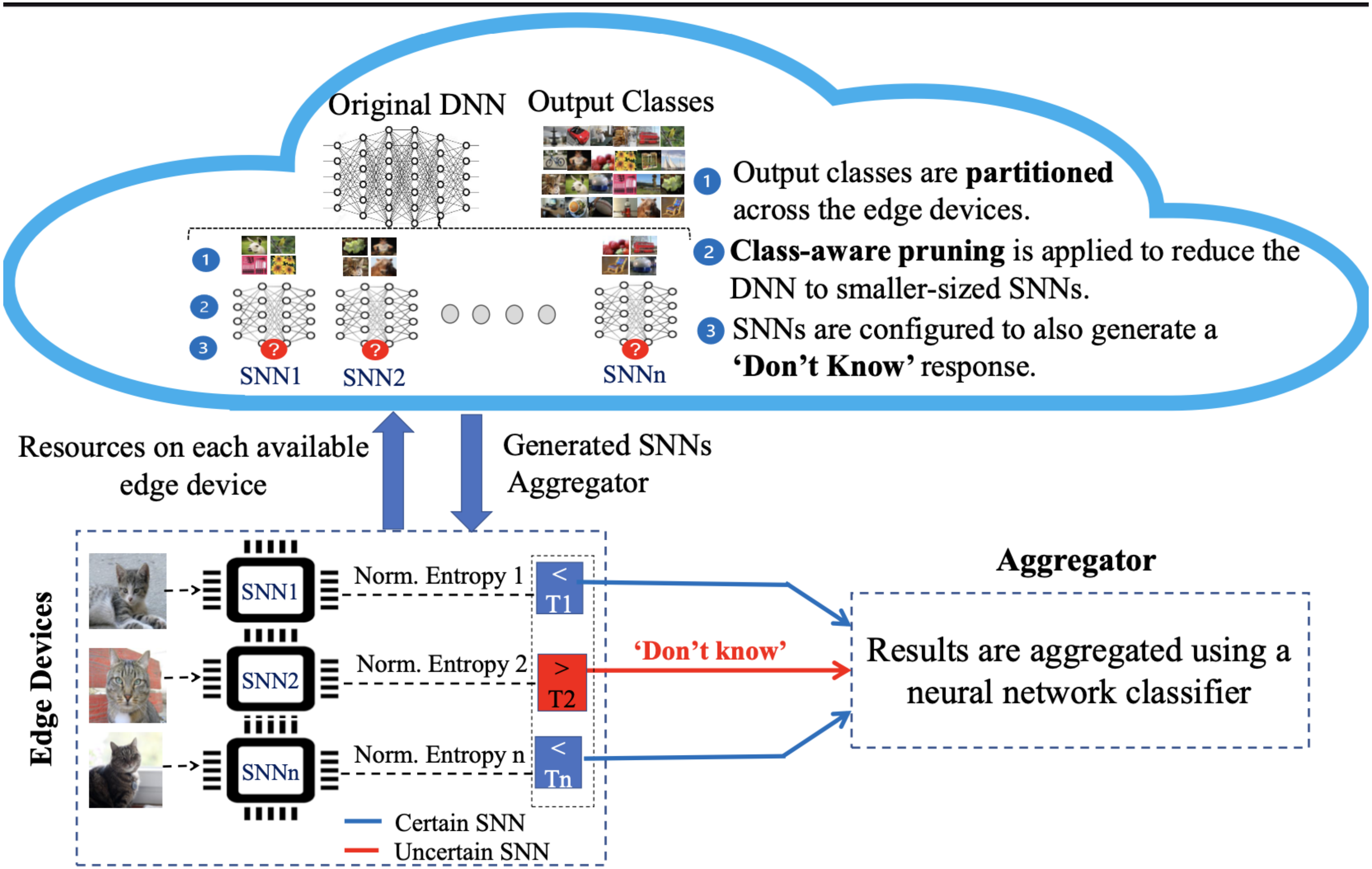
[1] Deeplearningmarketreport ,http://www.grandviewresearch.com/industry- analysis/deeplearning-market.

Solution: $Edge^{n}AI$

- $Edge^{n}AI$: Distributed inference with local devices and minimal latency

- Contributions:
  - Utilizing many parallel independent-running edge devices
  - Minimizing communication overheads with edge devices communicating only once to a back-end device
  - Maintaining accuracy of the distributed network
    - Partitioning the original network across output classes
    - Configuring each SNN to return a ‘Don’t know’ response when needed

Edge^{n}AI: Overview



Edge^{n}AI: Overview

- In Edge^{n}AI, following aspects should be considered:

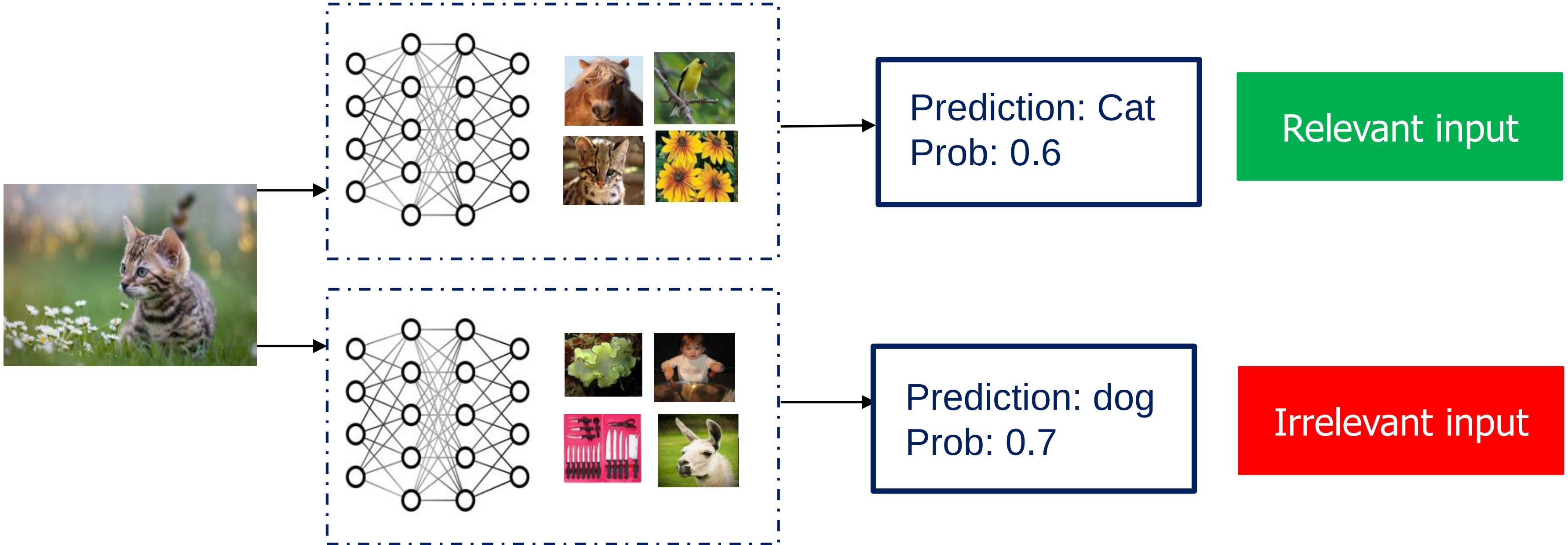
  1. Generation of ‘Don’t Know’ response
     - Each SNN is a reduced version of the original network

  2. Design of aggregator
     - The aggregator is responsible to make final prediction

  3. Efficient generation of SNNs
     - The SNNs are generated from decomposing a complex DNN across output classes

Generating ‘Don’t Know’

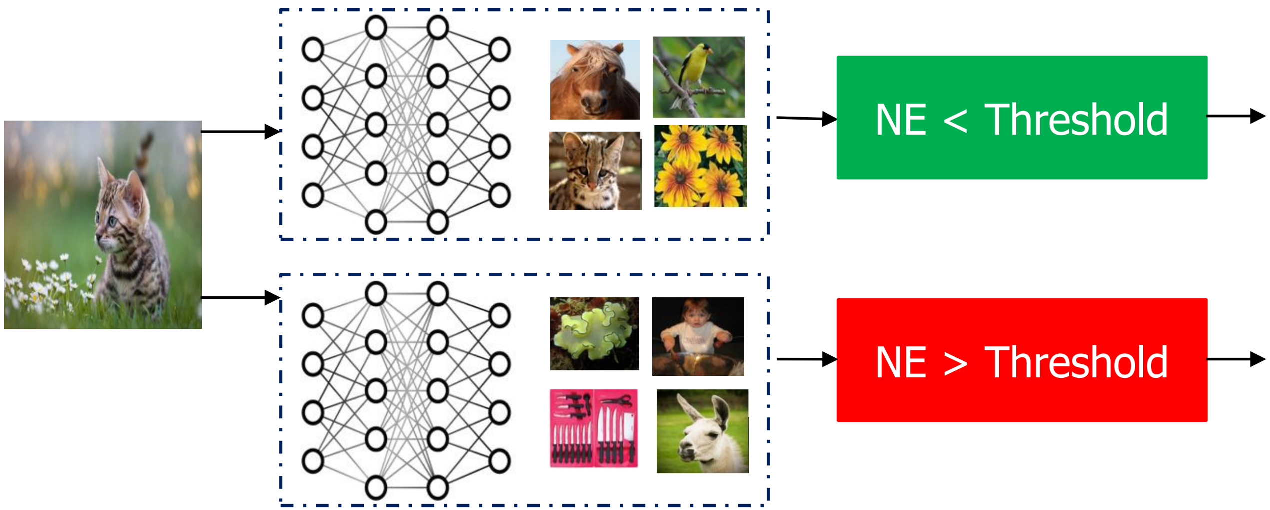
- Each SNN is a reduced version of the original network



- Normalized entropy (NE) metric is used to meaningfully compare the results from different SNNs
  - Used to define a ‘Don’t Know’ response

Generating 'Don't Know'

- SNNs are configured to predict if the received input is relevant or irrelevant
- NE is a smaller quantity for relevant inputs compared to irrelevant ones
- A 'Don't Know' response defined for each SNN by comparing its NE against a threshold



- SNN makes a prediction
- NE and prediction are sent to aggregator

- The input is rejected by SNN
- NE is set to one and sent to aggregator

Generating ‘Don’t Know’

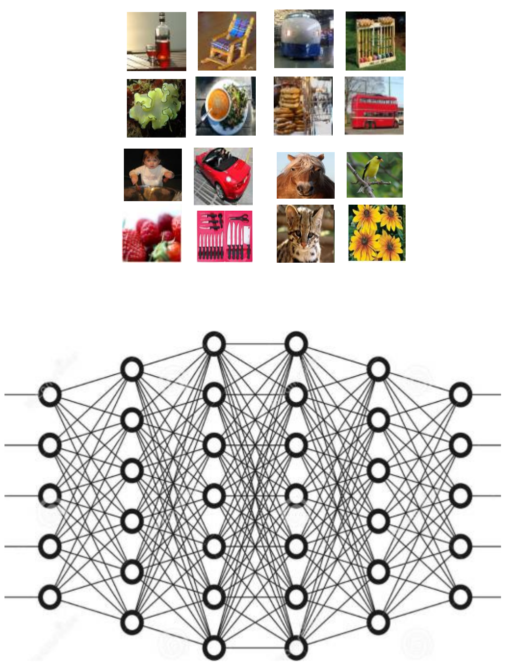
- Edge^{n}AI needs to calculate a threshold per SNN as a pre-processing step
- An algorithm is proposed to find the threshold
- The threshold is found such that the network is good at both making an inference for the relevant input and rejecting the irrelevant inputs
    - For more details, please refer to the paper

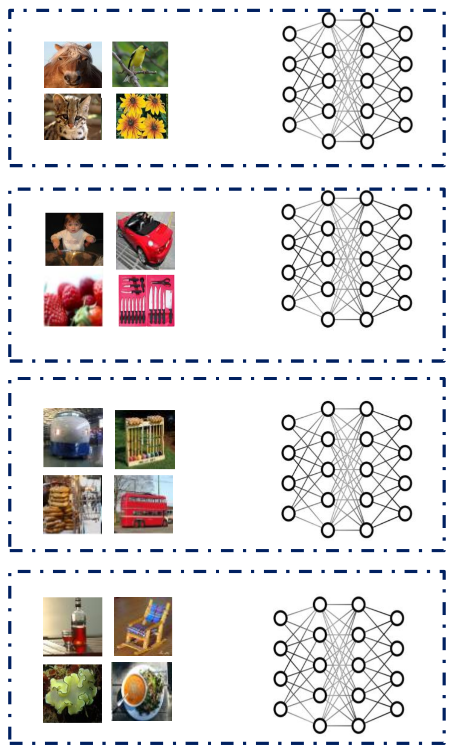
Aggregator Design

- Aggregator is responsible to make the final prediction
  - It receives the normalized entropy and the class with highest probability from each SNN
- Naïve approach is to eliminate the uncertain SNNs and pick the class with the lowest NE
  - Can’t maintain accuracy, specially with high number of SNNs
- We propose to implement aggregator as a lightweight neural network architecture
  - The aggregator has 3 layers with at most 60 neurons
  - It is trained on data collected from running distributed inference across SNNs

SNN Generation

- SNNs are generated via a two-step pre-processing approach:
  - Class partitioning
  - Class-aware pruning [2]





- Class-aware pruning:
  - Pruning the network for only a subset of classes
  - Exploiting the correlation between neurons and output classes

[2] CAP'NN: Class-aware Personalized Neural Network Inference, DAC, 2020.

SNN Generation

- To partition a network with $|C|$ classes on *n* devices, we have $\binom{|C|}{|S|}$ partition candidates ($|S| = \frac{|C|}{n}$)

- It is not feasible to implement all partitioning candidates and evaluate their performance
  - It requires to first prune the original network and generate SNNs
  - It requires to find the NE threshold for each SNN
  - It requires to measure the classification accuracy of each candidate

- We propose a scheme to efficiently estimate NE of SNNs
  - Only top 5% of candidates are implemented and evaluated
  - More details can be found in the paper

Results

- Effectiveness of Edge^{n}AI is assessed on VGG-16 and ResNet-152 networks

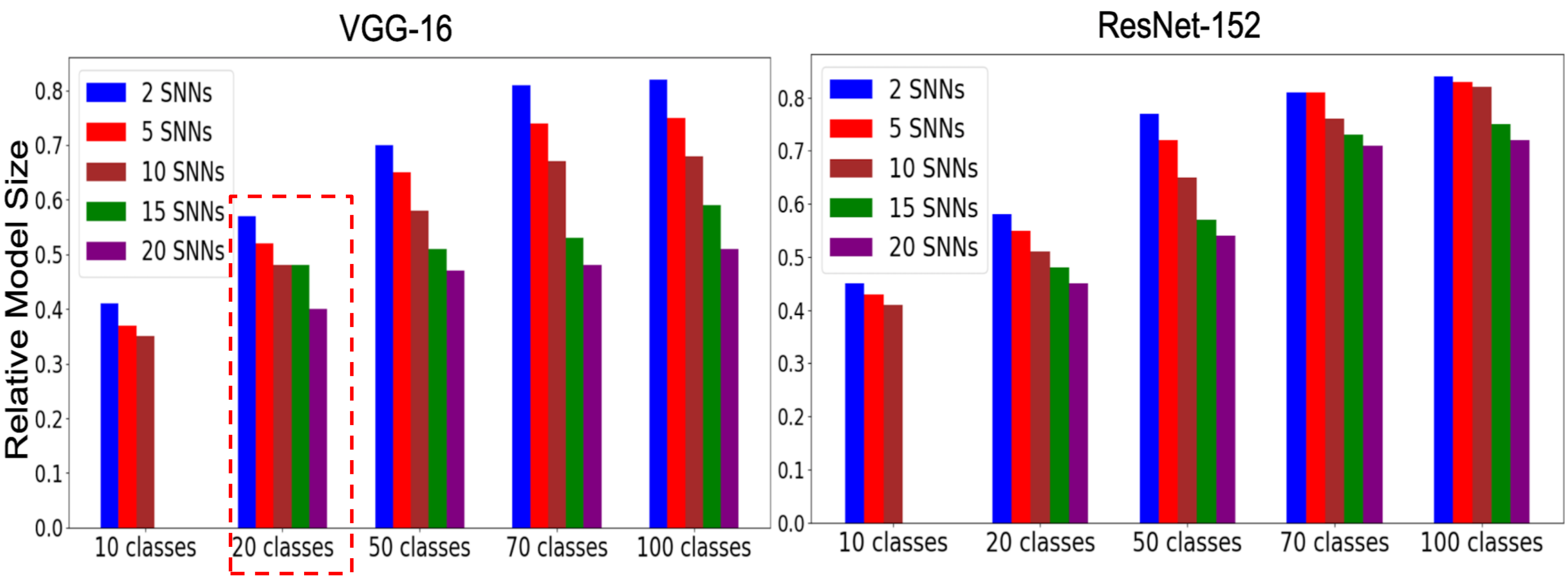
- Different variant of the networks are generated and used as base models
  - The base models have different number of output classes
    - $|C| = 10, 20, 50, 70, 100$

- Each base model is obtained by pruning the original model for a subset of classes

- Implemented base models with Edge^{n}AI when number of devices are varied

Results

- **Model size reduction:**
  - Model size corresponds to the largest SNN among all SNNs



- Model size is reduced for all combination of number of classes and number of devices

Results

- **Accuracy:**
  - Top-1 accuracy of the network with distributed implementation

Base Model	Base Accuracy	2 SNNs	5 SNNs	10 SNNs	15 SNNs	20 SNNs
VGG-16						
10 classes	88.3	87.8	86.7	86.2	–	–
20 classes	87.3	86.7	87.1	86.3	86.1	85.8
50 classes	85.1	84.9	84.7	83.1	83.3	83.5
70 classes	84.1	84.3	83.9	83.3	83.1	83.2
100 classes	82.4	81.9	81.6	82.1	82.2	82.3
ResNet-152						
10 classes	88.6	87.5	87.1	86.7	–	–
20 classes	87.9	87.2	86.8	86.4	86.1	86.3
50 classes	87.1	87.3	86.5	86.3	86.8	86.1
70 classes	85.6	85.1	85.3	84.8	84.3	84.6
100 classes	84.3	84.1	83.8	83.5	83.4	83.2

Results

- **Latency measurement:**
  - Inference latency is summation of 3 components:
    - Latency of the slowest SNN
    - Latency of the wireless communication network
    - Latency of the aggregator
  - We construct an analytical model to estimate the latency of SNNs and aggregator
    - Number of memory accesses and MAC operations are estimated based on the network architecture
  - Latency of the communication network is measured for the communication bandwidth of 100 Megabits per second

Results

- We measure latency of Edge^{n}AI on two hardware platforms:
  - Edge devices with at most 150 MB on-chip storage and no off-chip storage
  - Microcontrollers with at most 500 KB on-chip storage and a shared off-chip storage

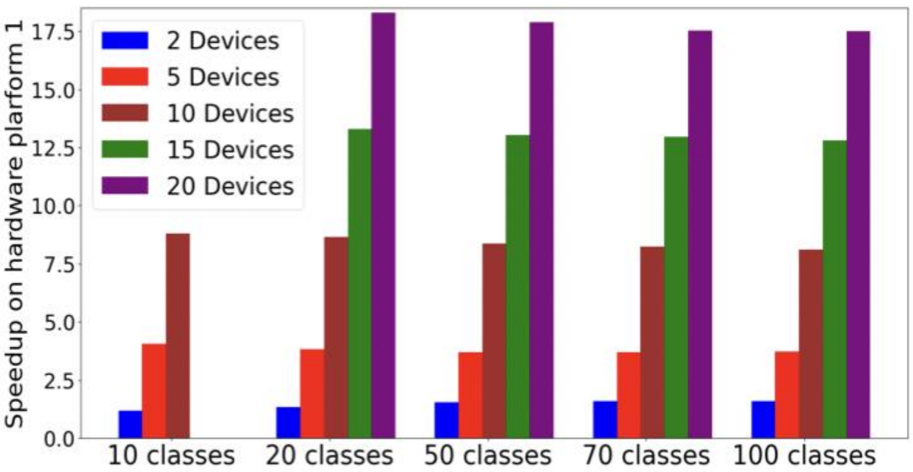
- The platforms have different number of on-chip and off-chip memory accesses and hence different latency

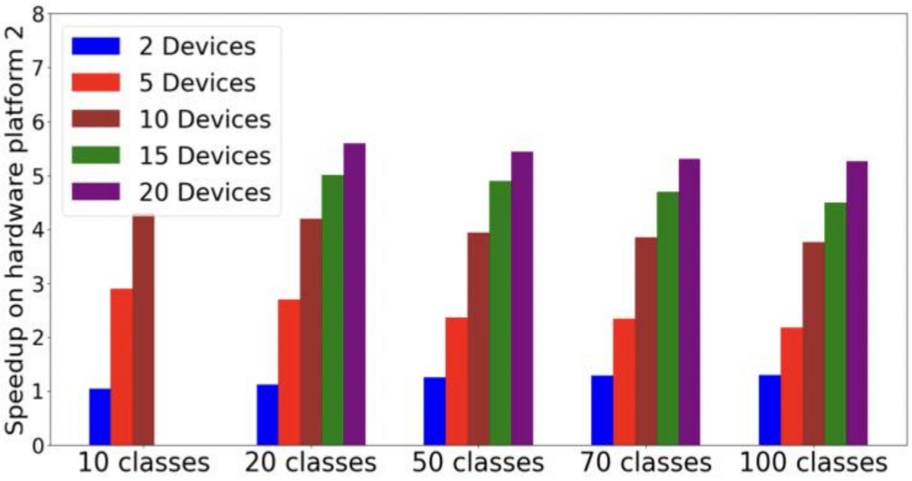
- Latency of Edge^{n}AI is compared against a recent work [3]

[3] Fully distributed deep learning inference on resource- constrained edge devices, International Conference on Embedded Computer Systems, 2019.

Results

- **VGG-16:**





- The speedup is increased with increase in number of devices
  - The communication overhead grows exponentially in [3] with increase in number of devices
- Speedup of 17X (5.5X) on platform 1 (platform 2) for VGG-16 with 100 output classes

Conclusions

- We proposed Edge^{n}AI to enable distributed inference of complex DNNs on local edge devices with minimal latency

- The effectiveness of Edge^{n}AI is evaluated on VGG-16 and ResNet-152 networks

- Edge^{n}AI reduces per-device model size and latency overheads while maintaining accuracy
  - up to 50% model size reduction and 17X speedup for a variant of VGG-16 with 100 output classes on 20 devices