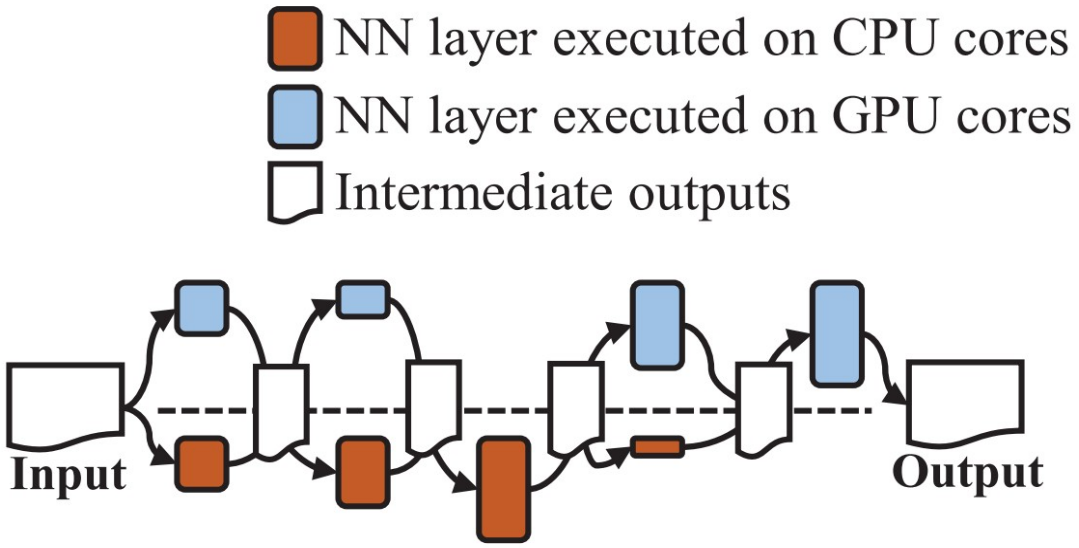
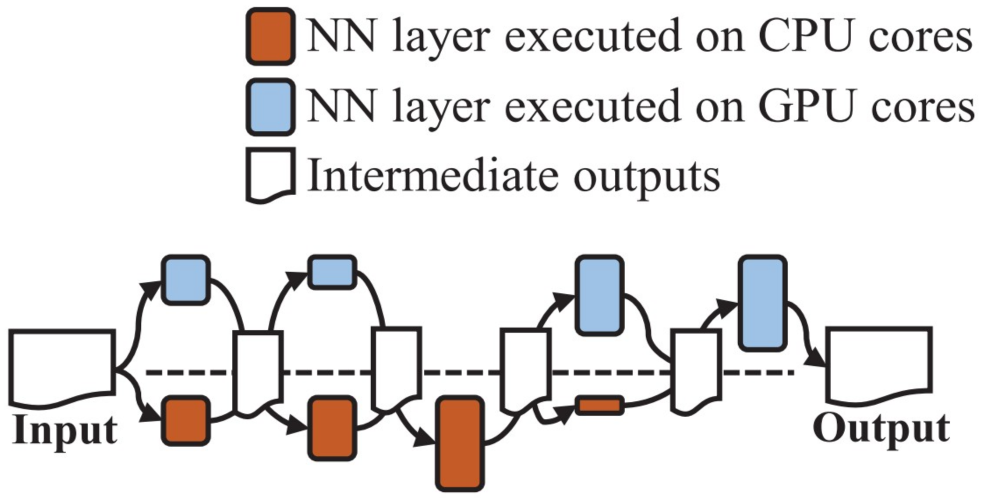
μLayer

Low Latency On-Device Inference Using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization


NN layer executed on CPU cores
NN layer executed on GPU cores
Intermediate outputs
Input
Output

Main Ideas

- On device inference with heterogeneous processors (CPUs, GPUs)
- By separating the layer operations of the neural network by channel
- Heterogeneous processors compute separate channels in parallel


NN layer executed on CPU cores
NN layer executed on GPU cores
Intermediate outputs
Input
Output

Motivation

Real-time service needs

Demand real-time response

Limits on real-time response spurred by communication delays

Security issues

Inference failure due to single processor failure

cloud-based GPU-dependent operation delay

Optimize operations for Inference on On-Device

μLayer Details

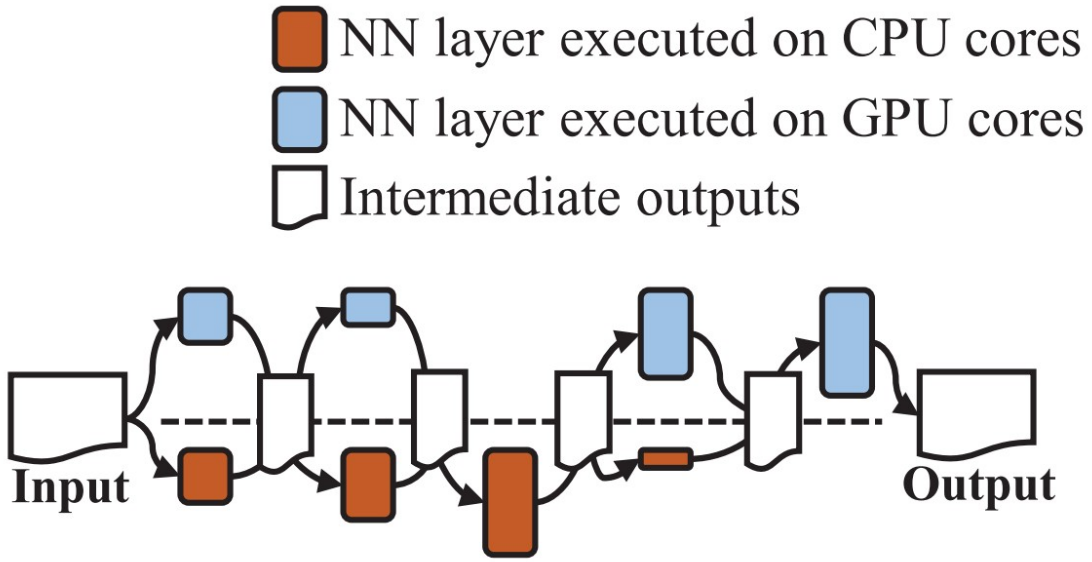
Network to Processor Mapping

Layer to Processor Mapping

Cooperative Single Layer Acceleration

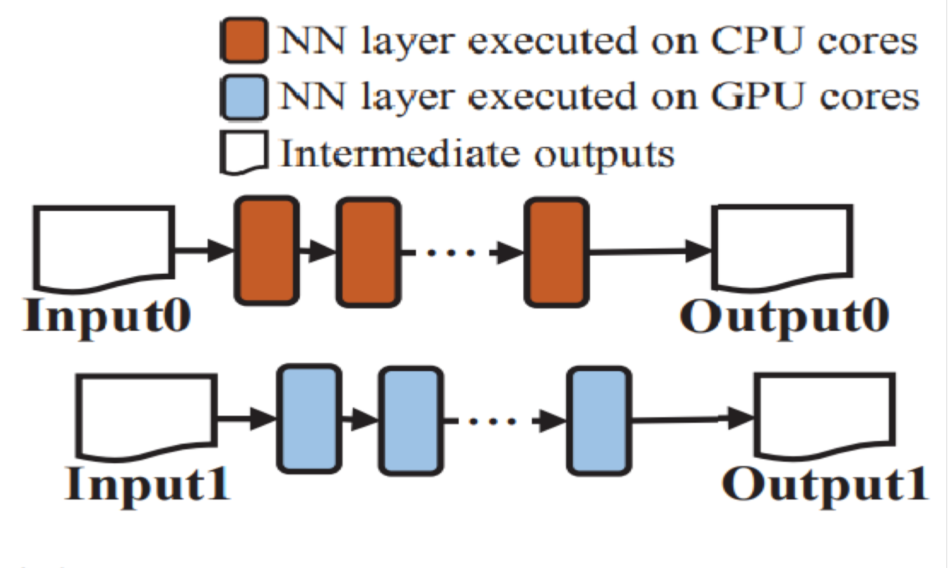
Processor Friendly Quantization

Branch Distribution


NN layer executed on CPU cores
NN layer executed on GPU cores
Intermediate outputs
Input
Output

Network to Processor Mapping

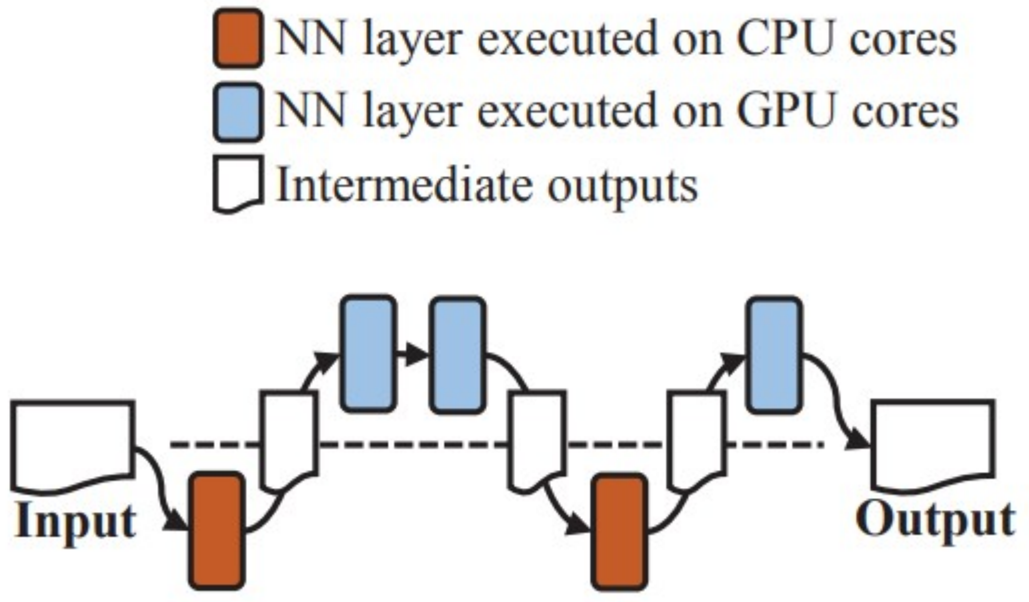
- Images are handled by different processors in some existing CNN arechitectures i.e. MCDNN
- Network to Processor Mapping mechanism has no benefit from inference delay in single input


NN layer executed on CPU cores
NN layer executed on GPU cores
Intermediate outputs
Input0
Output0
Input1
Output1

(a) Network-to-processor mapping

Layer to Processor Mapping

- The mechanism used to train a large-sized model in a distributed manner
- Layer to Processor Mapping mechanism performs better when compared to Network to Processor Mapping in a single input
- However, it still relies on single processor performance

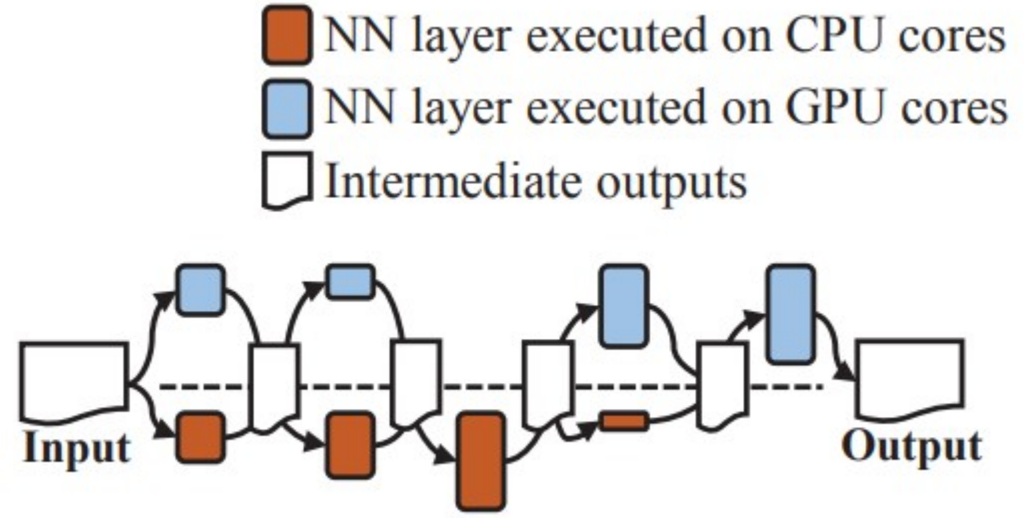


NN layer executed on CPU cores
NN layer executed on GPU cores
Intermediate outputs
Input
Output

(b) Layer-to-processor mapping

Cooperative Single Layer Acceleration

- How to accelerate(suggested in the paper)?
- How to split the output channels from the convolution network and calculate them by heterogeneous processor?
- Efficient, but has the overhead of managing multi-processors such as synchronizing memory from heterogeneous processors, GPU command issues

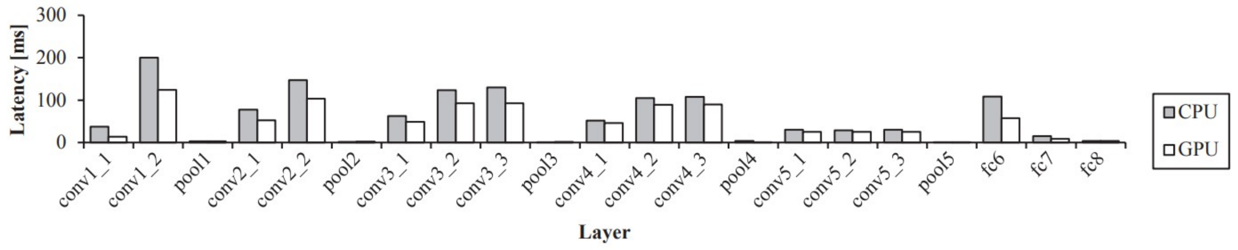

NN layer executed on CPU cores
NN layer executed on GPU cores
Intermediate outputs
Input
Output

(c) Cooperative single-layer acceleration (Ours)

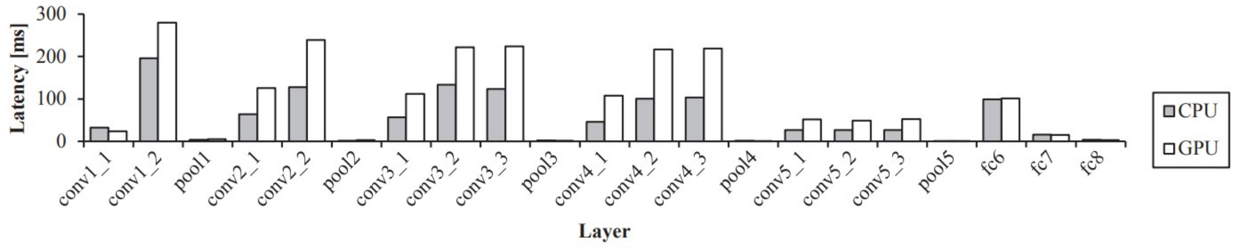
Cooperative Single Layer Acceleration

Profile the per layer execution latency

- Per-layer latency of VGG-16 shows multi-processor overhead does not affects inference performance
- Each node(CPU, GPU) has no significant difference in layer-specific computational speed
- For high-end SoC (Samsung Exynos 7420) due to low-power GPUs, the GPU is 1.40x faster than the CPU, and for mid-range SoC (Samsung Exynos 7880), the octa-core CPU is 1.26x faster.



(a) High-end (4 bCs + 8 GCs)

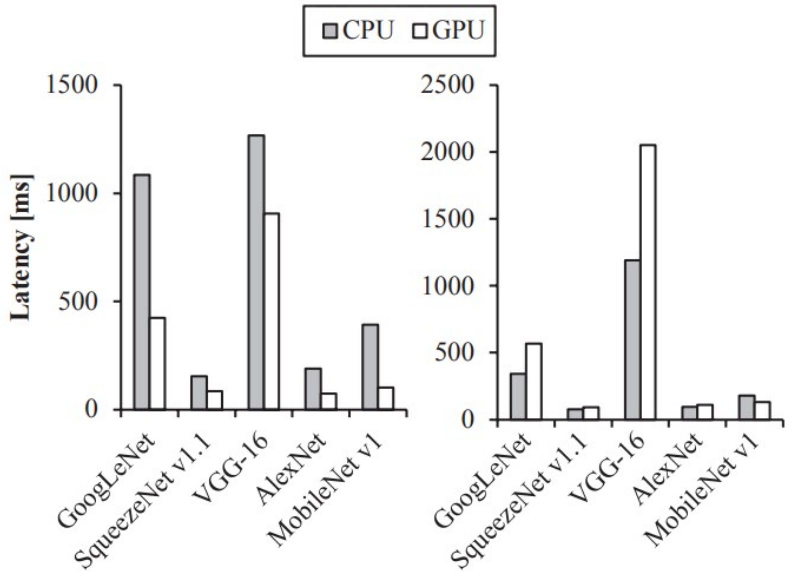


(b) Mid-range (8 LCs + 3 GCs)

Cooperative Single Layer Acceleration

Profile the per layer execution latency

- High-end SoC (Exynos 7420) and mid-range SoC (Exynos 7880) are the same in other neural networks (GoogLeNet, SqueezeNet, AlexNet, MobileNet)



(a) High-end (4 bCs + 8 GCs) **(b)** Mid-range (8 LCs + 3 GCs)

Figure 6. NN execution latency on modern SoCs

Cooperative Single Layer Acceleration

Channel Wise Workload Distribution

- Filters of the convolution network are split among the nodes w.r.t. the output channel and input data is shared among the nodes
- The percentage of channels responsible for CPU-to-GPU operations is calculated as $p_{cpu} = (1 - p_{gpu})$
- Output channels computed in each nodes merged later
- Pooling layers behave the same way; except input is distributed

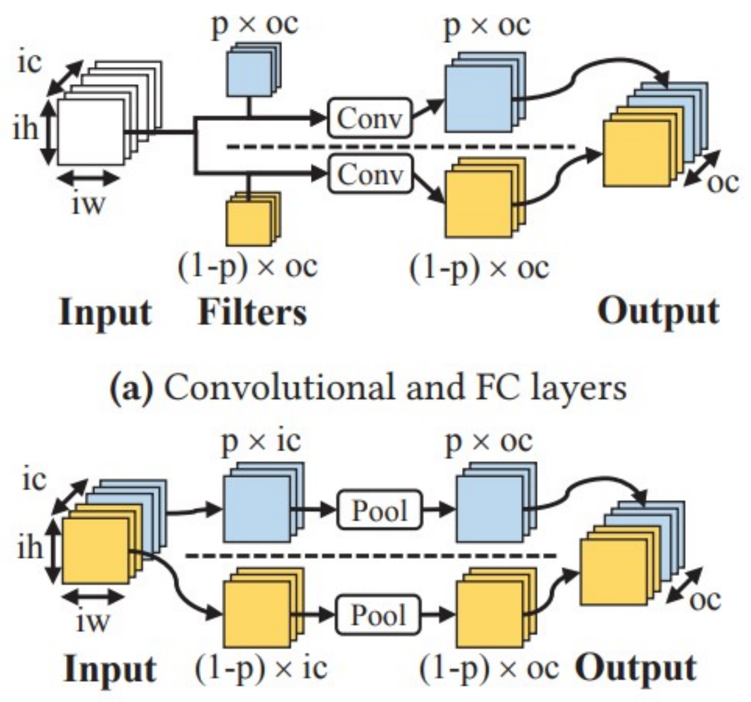


Figure 7. Channel-wise workload distribution of a layer

Processor Friendly Quantization

Bit Quantization

- The computational speed increases with weight quantization without degrading the accuracy
- The appropriate bit quantization technique depends on each SoC
- The GPU performs better with f16 due to the native support for high throughput floating points operations
- The CPU are equipped with ALUs capable of parallel-process 8-bit integers
- If the CPU does the operation based on F16, the operation will be forced to f32 due to the lack of ALU support for F16
- Performance characteristics vary from quantization to quantization

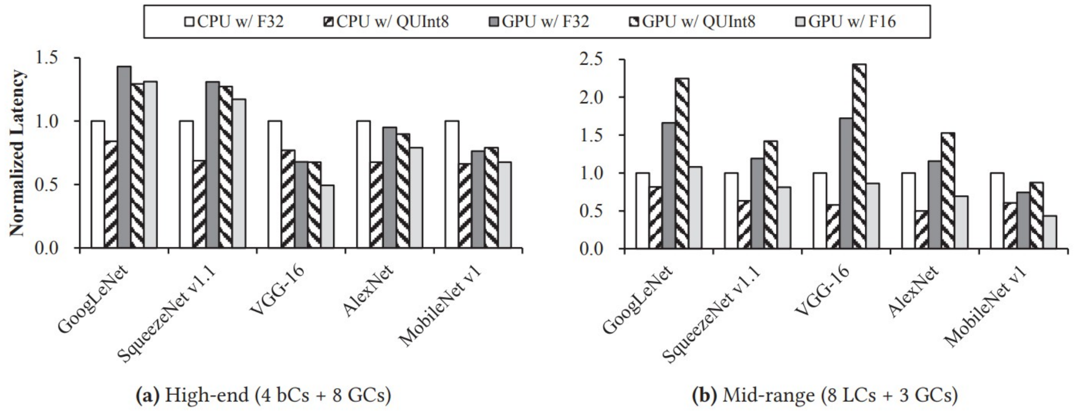


Figure 8. Impacts of quantization on inference latency; normalized to the latency of the CPU with F32.

Processor Friendly Quantization

Bit Quantization

- Input data, filters and output data are stored in QUInt8 to minimize memory size
- A linear scaling is used to convert F32 to QUint8
- In case of GPU, input and filters are converted from QUint8 to F16

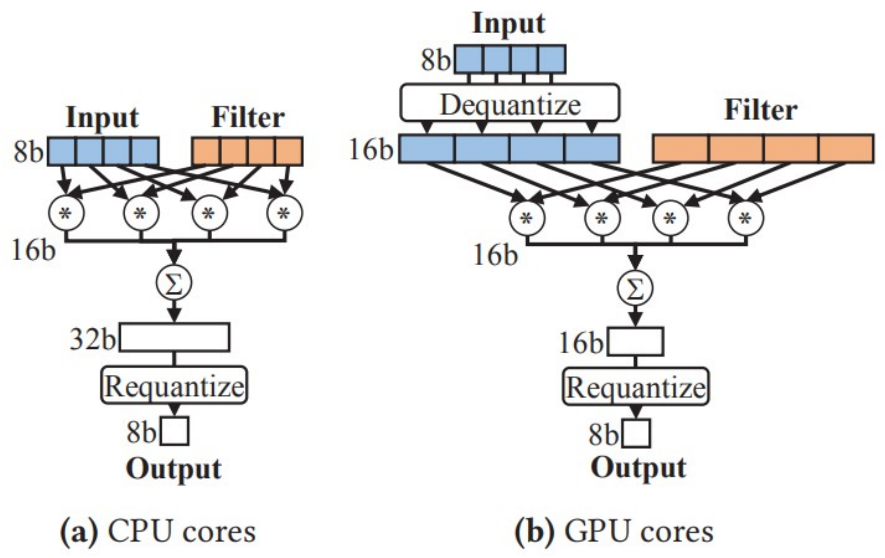


Figure 9. Processor-friendly quantization

Processor Friendly Quantization

Impacts on Inference Accuracy

- Re-train the network with 8-bit linear quantization to prevent large-event accuracy Loss
- After retraining, accuracy loss up to only 2.7%

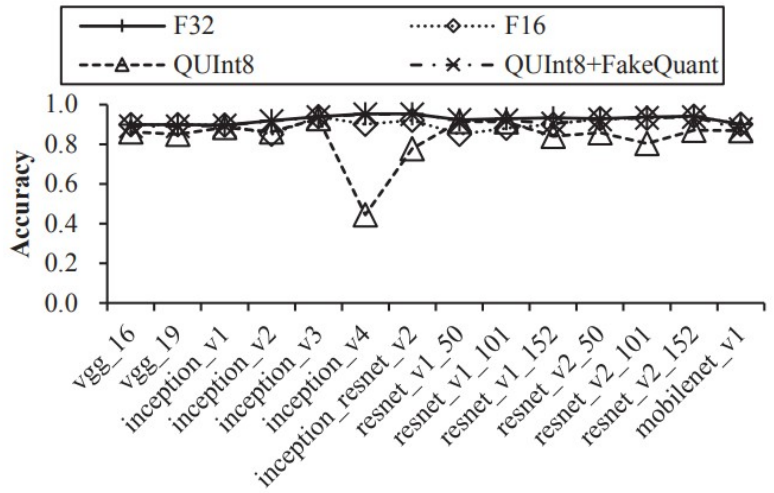
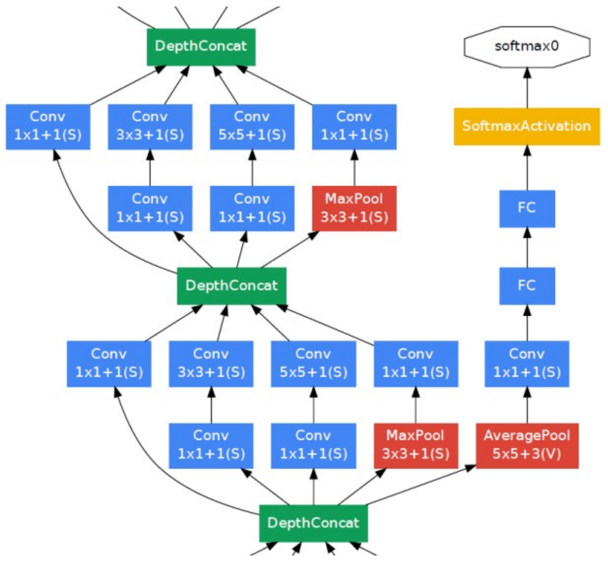


Figure 10. Impacts of quantization on the top-5 classification accuracy with the ImageNet dataset [62]

Branch Distribution

Branch Distribution

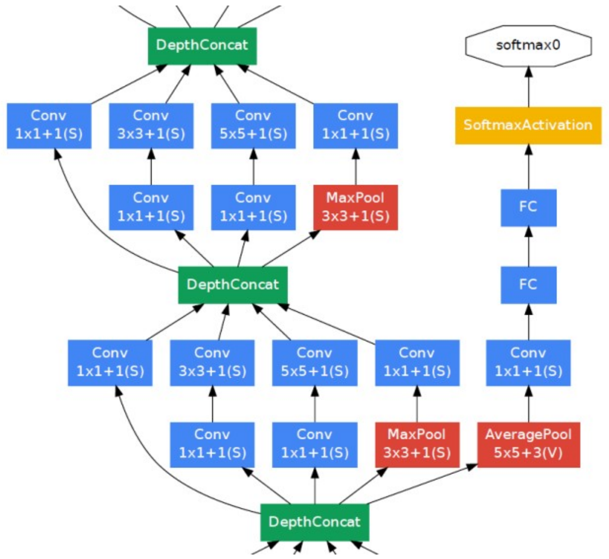
- Some neural network architectures have branch architectures i.e. Inception module in GoogleNet or fire module in SqueezeNet
- This results in different layer-to-layer computational latency, which increases the synchronization overhead between THE CPU and GPU due to channel wise workload distribution



Branch Distribution

Branch Distribution

- Branch Distribution is applied to optimize the synchronization overhead between CPUs and GPUs that result in different branches with computational latency
- Identify branches that can be branched
- Parallel processing is carried out by dividing the identified branches into CPU and GPU



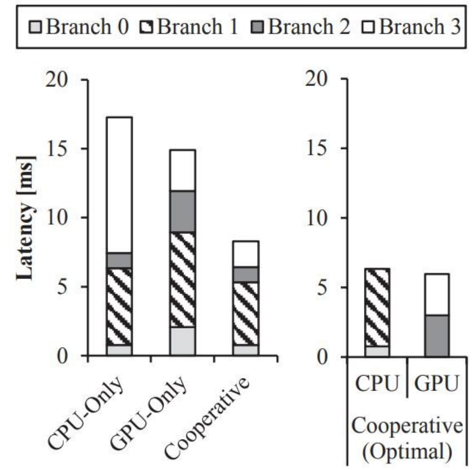


Figure 12. Potential latency benefits of branch distribution