MATLAB EXPO

인공지능 모델 경량화 및 배포

MathWorks®

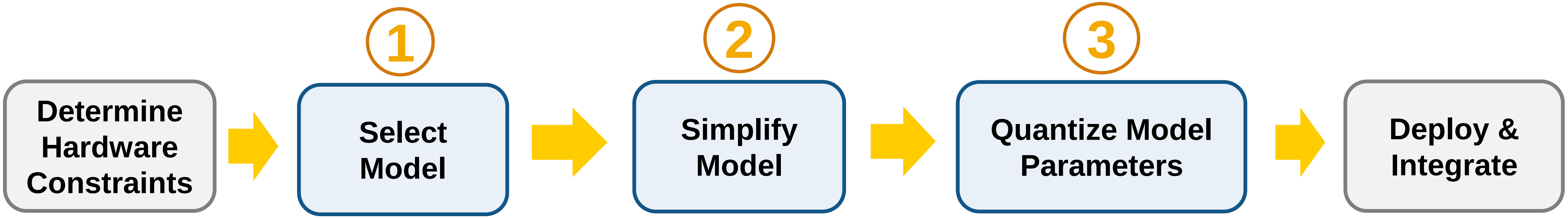
3 Outcomes of AI Model Compression

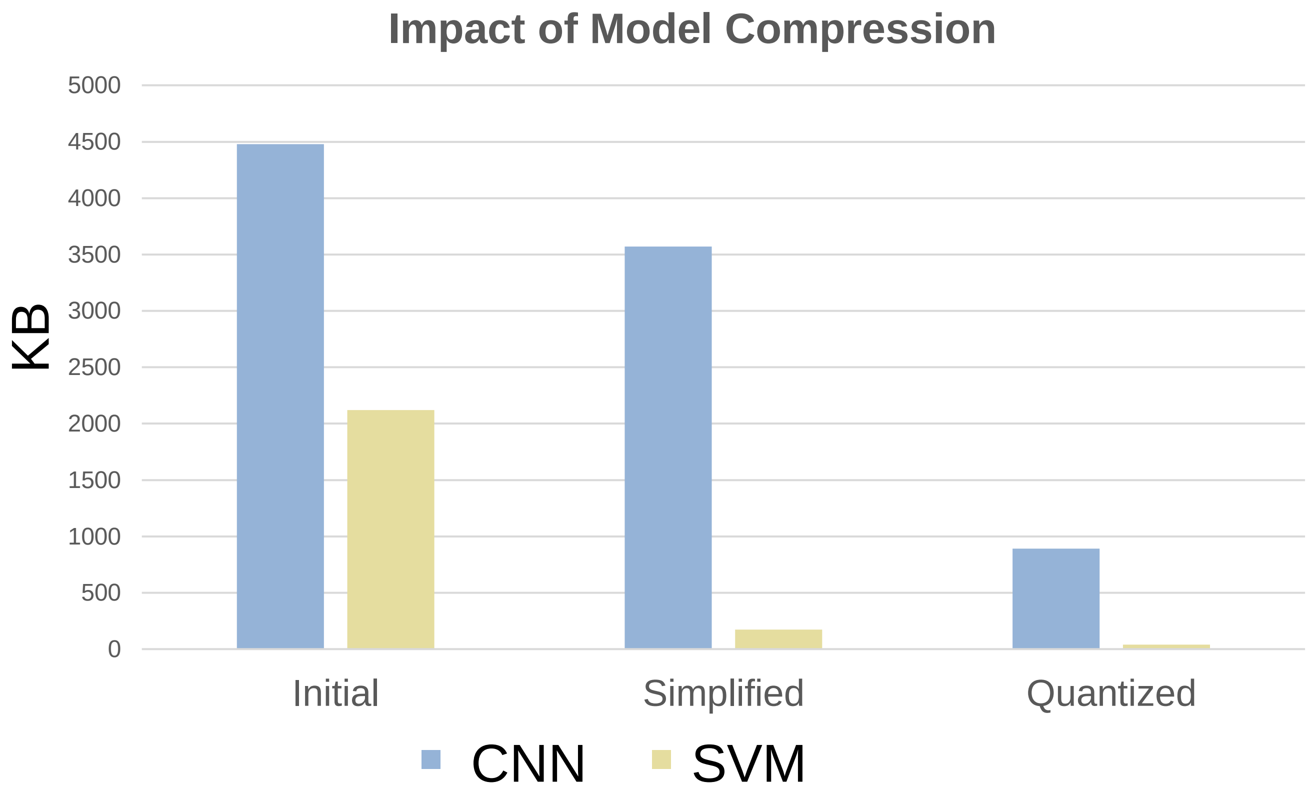
Smaller footprint and power consumption

Accelerate inference

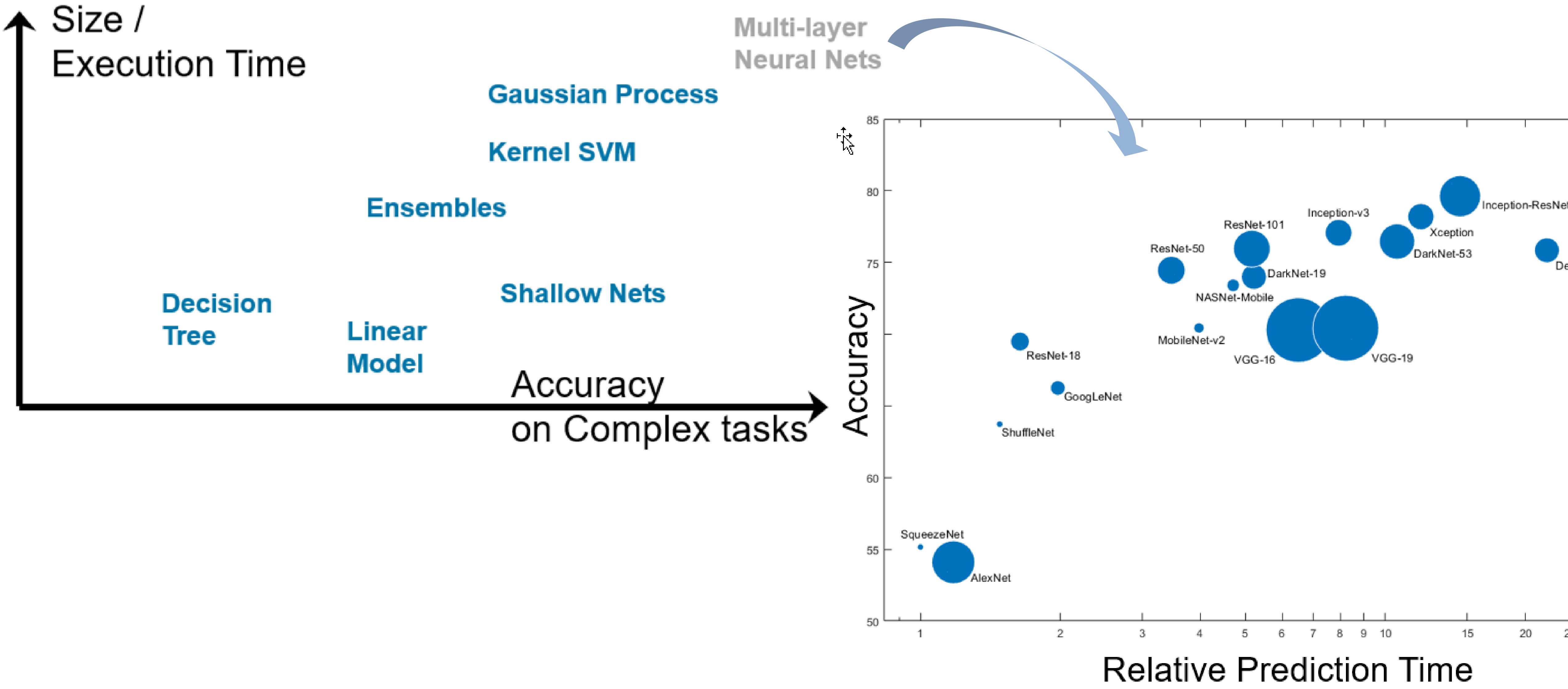
More robust model

The 3 Steps to Compress AI Models





Size Aware **Model Selection** - Step 1

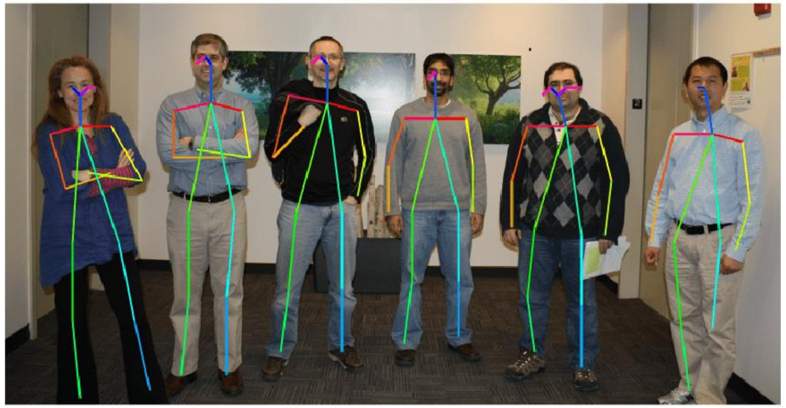


Size of Model Shown by Bubble Size

Simplify Model – Step 2
(Projection and Pruning)

CNNs (Convolutional Neural Networks) can be simplified via Taylor-based Filter Pruning

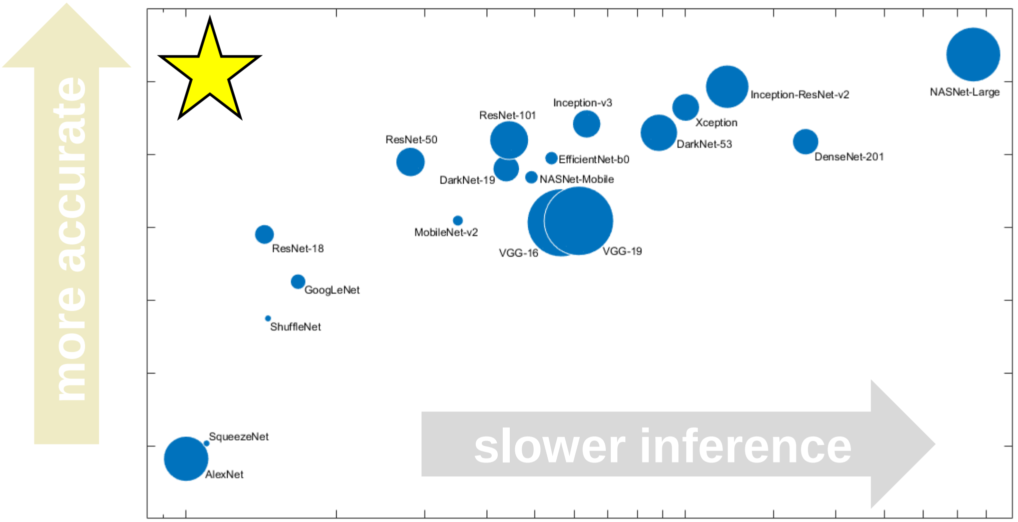
Application area - image-based problems, e.g. computer vision (object recognition)



CNNs can be simplified via Taylor-based Filter Pruning

Application area - image-based problems, e.g. computer vision (object recognition)

Challenge - transfer learning creates overparameterized networks



Pretrained Network	Depth	Size	Parameters (Millions)
squeezenet	18	**5.2 MB**	1.24
googlenet	22	**27 MB**	7.0
inceptionv3	48	**89 MB**	23.9
densenet201	201	**77 MB**	20.0
mobilenetv2	53	**13 MB**	3.5
resnet18	18	**44 MB**	11.7
resnet50	50	**96 MB**	25.6
resnet101	101	**167 MB**	44.6
xception	71	**85 MB**	22.9
inceptionresnetv2	164	**209 MB**	55.9
shufflenet	50	**5.4 MB**	1.4
nasnetmobile	*	**20 MB**	5.3
nasnetlarge	*	**332 MB**	88.9
darknet19	19	**78 MB**	20.8
darknet53	53	**155 MB**	41.6
efficientnetb0	82	**20 MB**	5.3
alexnet	8	**227 MB**	61.0
vgg16	16	**515 MB**	138
vgg19	19	**535 MB**	144

CNNs can be simplified via Taylor-based Filter Pruning

Application area - image-based problems, e.g. computer vision (object recognition)

Challenge - transfer learning creates overparameterized networks

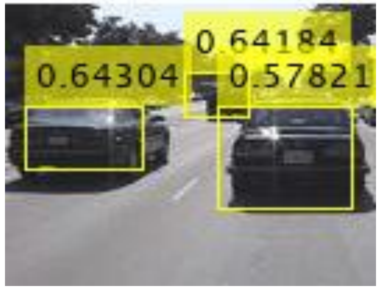
Solution - remove unimportant parts of weight matrices in 2D conv layers

In MATLAB R2022a

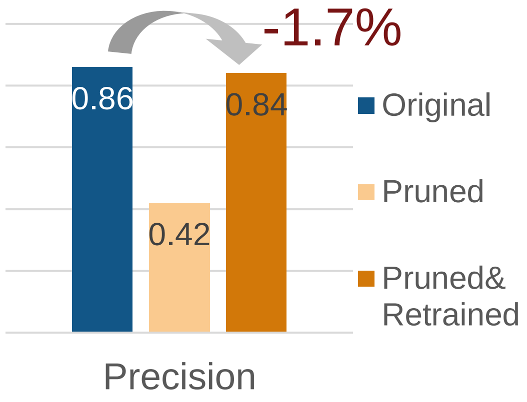
```
prunableNet = taylorPrunableNetwork(net)
```

Impact - reduces memory footprint and improves inference speed on all platforms

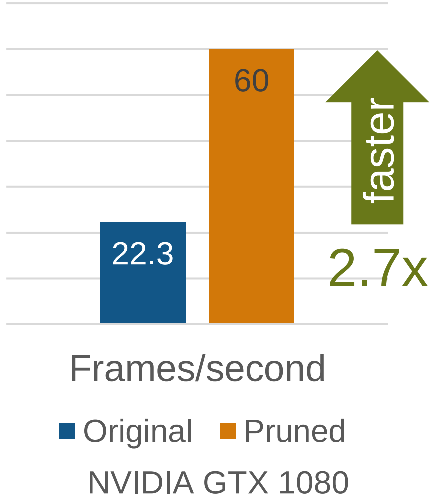
YOLO v3



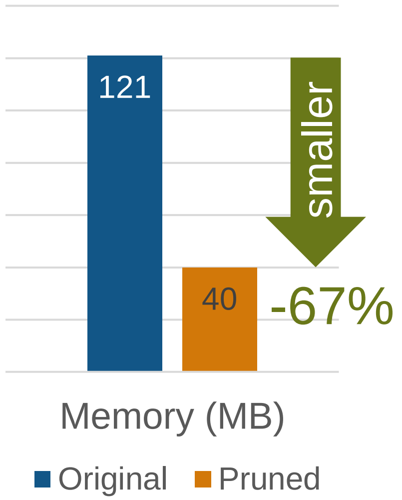
Avg. Precision



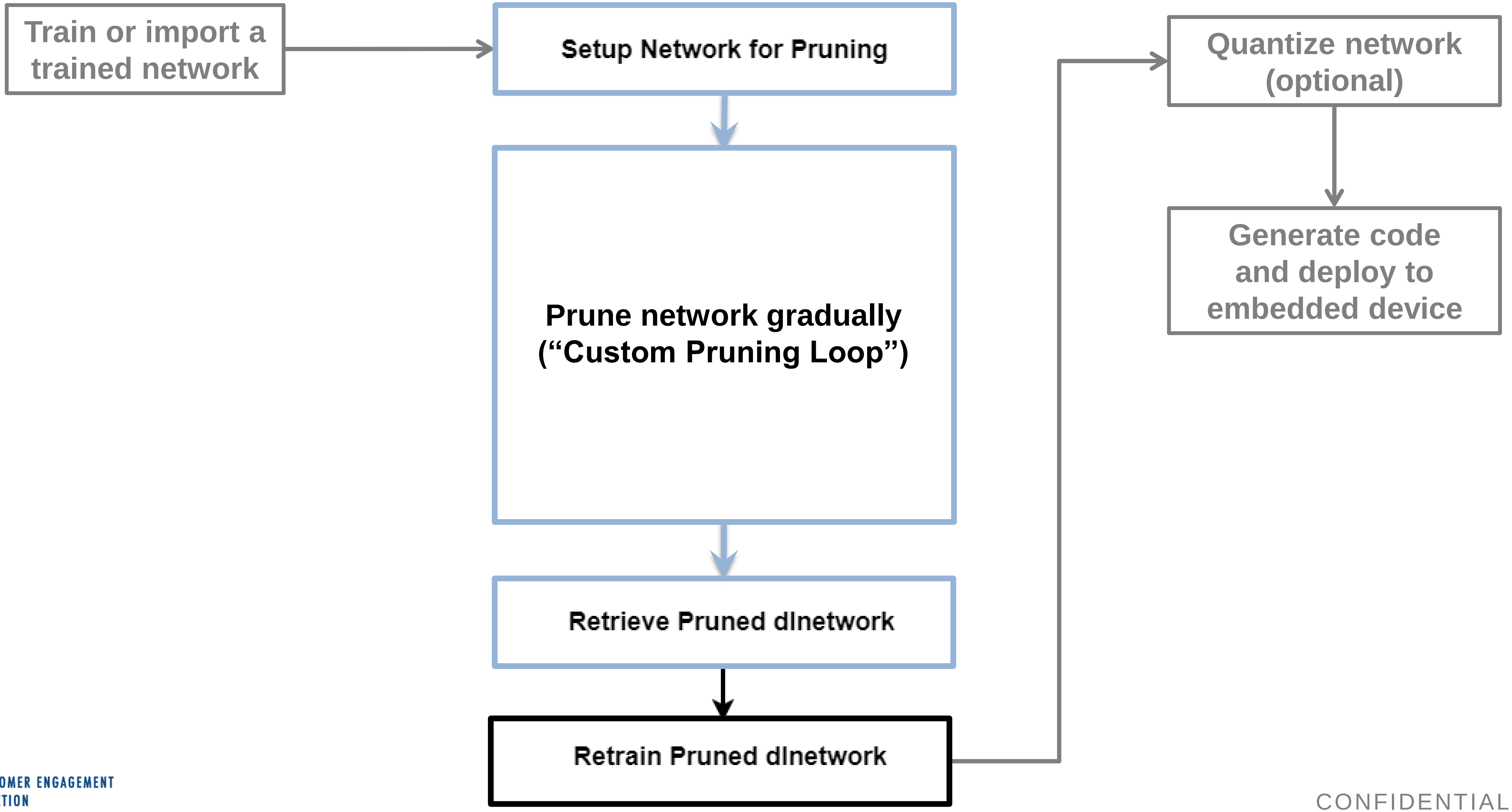
Inference Speed



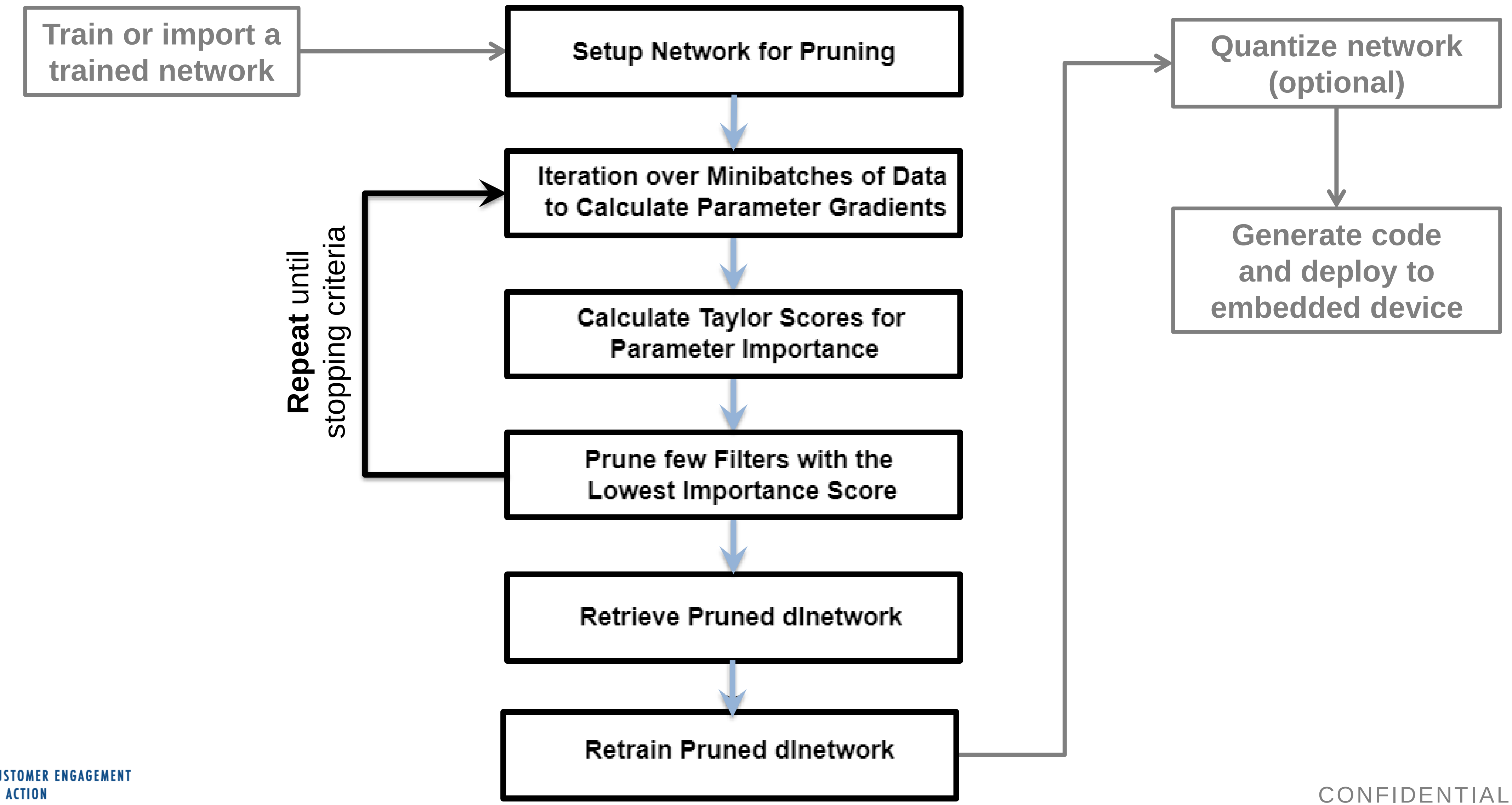
Model Size



Pruning, HOW?


Train or import a trained network
Setup Network for Pruning
Prune network gradually (“Custom Pruning Loop”)
Retrieve Pruned dlnetwork
Retrain Pruned dlnetwork
Quantize network (optional)
Generate code and deploy to embedded device

Pruning, HOW?


Train or import a trained network
Setup Network for Pruning
Iteration over Minibatches of Data to Calculate Parameter Gradients
Calculate Taylor Scores for Parameter Importance
Prune few Filters with the Lowest Importance Score
Repeat until stopping criteria
Retrieve Pruned dlnetwork
Retrain Pruned dlnetwork
Quantize network (optional)
Generate code and deploy to embedded device

Pruning in MATLAB uses custom ~~training~~ pruning loops

- Pruning requires the use of [`dlnetwork`] objects and custom training loops.
  See [Deep Learning Custom Training Loops] doc pages for help.

- Use `taylorPrunableNetwork` object to enable pruning:

```
>> prunableNet = taylorPrunableNetwork(dlnet)

prunableNet =

  TaylorPrunableNetwork with properties:

      Learnables: [86×3 table]
           State: [42×3 table]
      InputNames: {'input'}
     OutputNames: {'softmax'}
    NumPrunables: 637
```

`forward`	Compute deep learning network output for custom training
`predict`	Compute deep learning network output for prediction (inference)
`updateScore`	Compute first-order Taylor scores and accumulate the scores across previous minibatches of data.
`updatePrunables`	Remove filters from prunable layers
`dlnetwork`	Get pruned dlnetwork from the TaylorPrunableNetwork object

Custom pruning loop

- Create pruner
- Begin pruning loop
  - Begin fine-tuning loop
    - Fine-tune the network
    - Compute Taylor scores
  - End fine-tuning loop
  - Prune least important filters
- Repeat pruning loop
- Extract pruned dlnetwork

```
prunableNet = taylorPrunableNetwork(dlnet)

iteration = 0;
for pruningEpoch = 1:maxPruningEpochs
    shuffle(mbq);
    velocity = [];

    % Loop over mini-batches.
    fineTuningIteration = 0;
    while hasdata(mbq)
        iteration = iteration + 1;
        fineTuningIteration = fineTuningIteration + 1;
        [dlX, dlY] = next(mbq);

        % Evaluate activations, gradients, state, and loss
        [pruningActivations, pruningGradients, netGradients, state, loss] = ...
                        dlfeval(@modelGradientsPruning, prunableNet, dlX, dlY);

        % Update the network state
        prunableNet.State = state;

        % Update the network parameters using the SGDM optimizer
        [prunableNet, velocity] = sgdmupdate(prunableNet, ...
                                             netGradients, velocity, learnRate,
        momentum);

        % Compute first-order Taylor scores and accumulate the score
        prunableNet = updateScore(prunableNet, pruningActivations, pruningGradients);

        % Stop fine-tuning loop when numMinibatchUpdates is reached
        if (fineTuningIteration > numMinibatchUpdates)
            break
        end
    end
    % Prune filters based on previously computed Taylor scores
    prunableNet = updatePrunables(prunableNet, MaxToPrune = 8);
end

dlnetPruned = dlnetwork(prunableNet)
```

Typical parts of a custom training loop

Pruning-specific parts

Custom pruning loop

Typical parts of a custom training loop

Pruning-specific parts

- Create pruner
- Begin pruning loop
  - Begin fine-tuning loop
    - Fine-tune the network
    - Compute Taylor scores
  - End fine-tuning loop
  - Prune least important filters
- Repeat pruning loop
- Extract pruned dlnetwork

```
prunableNet = taylorPrunableNetwork(dlnet)

iteration = 0;
for pruningEpoch = 1:maxPruningEpochs
    shuffle(mbq);
    velocity = [];

    % Loop over mini-batches.
    fineTuningIteration = 0;
    while hasdata(mbq)
        iteration = iteration + 1;
        fineTuningIteration = fineTuningIteration + 1;
        [dlX, dlY] = next(mbq);

        % Evaluate activations, gradients, state, and loss
        [pruningActivations, pruningGradients, netGradients, state, loss] = ...
                    dlfeval(@modelGradientsPruning, prunableNet, dlX, dlY);

        % Update the network state
        prunableNet.State = state;

        % Update the network parameters using the SGDM optimizer
        [prunableNet, velocity] = sgdmupdate(prunableNet, ...
                                              netGradients, velocity, learnRate,
        momentum);

        % Compute first-order Taylor scores and accumulate the score
        prunableNet = updateScore(prunableNet, pruningActivations, pruningGradients);

        % Stop fine-tuning loop when numMinibatchUpdates is reached
        if (fineTuningIteration > numMinibatchUpdates)
            break
        end
    end
    % Prune filters based on previously computed Taylor scores
    prunableNet = updatePrunables(prunableNet, MaxToPrune = 8);
end

dlnetPruned = dlnetwork(prunableNet)
```

Custom pruning loop

Typical parts of a custom training loop

Pruning-specific parts

- Create pruner
- Begin pruning loop
  - Begin fine-tuning loop
    - Fine-tune the network
    - Compute Taylor scores
  - End fine-tuning loop
  - Prune least important filters
- Repeat pruning loop
- Extract pruned dlnetwork

```
prunableNet = taylorPrunableNetwork(dlnet)

iteration = 0;
for pruningEpoch = 1:maxPruningEpochs
      shuffle(mbq);
      velocity = [];

      % Loop over mini-batches.
      fineTuningIteration = 0;
      while hasdata(mbq)
            iteration = iteration + 1;
            fineTuningIteration = fineTuningIteration + 1;
            [dlX, dlY] = next(mbq);

            % Evaluate activations, gradients, state, and loss
            [pruningActivations, pruningGradients, netGradients, state, loss] = ...
                           dlfeval(@modelGradientsPruning, prunableNet, dlX, dlY);

            % Update the network state
            prunableNet.State = state;

            % Update the network parameters using the SGDM optimizer
            [prunableNet, velocity] = sgdmupdate(prunableNet, ...
                                                  netGradients, velocity, learnRate,
momentum);

            % Compute first-order Taylor scores and accumulate the score
            prunableNet = updateScore(prunableNet, pruningActivations, pruningGradients);

            % Stop fine-tuning loop when numMinibatchUpdates is reached
            if (fineTuningIteration > numMinibatchUpdates)
                break
            end
      end
      % Prune filters based on previously computed Taylor scores
      prunableNet = updatePrunables(prunableNet, MaxToPrune = 8);
end

dlnetPruned = dlnetwork(prunableNet)
```

Custom pruning loop

- Create pruner
- Begin pruning loop
  - Begin fine-tuning loop
    - Fine-tune the network
    - Compute Taylor scores
  - End fine-tuning loop
  - Prune least important filters
- Repeat pruning loop
- Extract pruned dlnetwork

```
prunableNet = taylorPrunableNetwork(dlnet)

iteration = 0;
for pruningEpoch = 1:maxPruningEpochs
    shuffle(mbq);
    velocity = [];

    % Loop over mini-batches.
    fineTuningIteration = 0;
    while hasdata(mbq)
        iteration = iteration + 1;
        fineTuningIteration = fineTuningIteration + 1;
        [dlX, dlY] = next(mbq);

        % Evaluate activations, gradients, state, and loss
        [pruningActivations, pruningGradients, netGradients, state, loss] = ...
                    dlfeval(@modelGradientsPruning, prunableNet, dlX, dlY);

        % Stop fine-tuning loop when numMinibatchUpdates is reached
        if (fineTuningIteration > numMinibatchUpdates)
            break
        end
    end
    % Prune filters based on previously computed Taylor scores
    prunableNet = updatePrunables(prunableNet, MaxToPrune = 8);
end

dlnetPruned = dlnetwork(prunableNet)
```

```
function [pruningActivations, pruningGradient, netGradients, state, loss] = …
  modelGradientsPruning(net, dlX, Y)

  [dlYPred, state, pruningActivations] = forward(net, dlX);

  loss = crossentropy(dlYPred,Y);

  [pruningGradient, netGradients] = dlgradient(loss, ...
     pruningActivations, net.Learnables);

end
```

Typical parts of a custom training loop

Pruning-specific parts

Custom pruning loop

Typical parts of a custom training loop

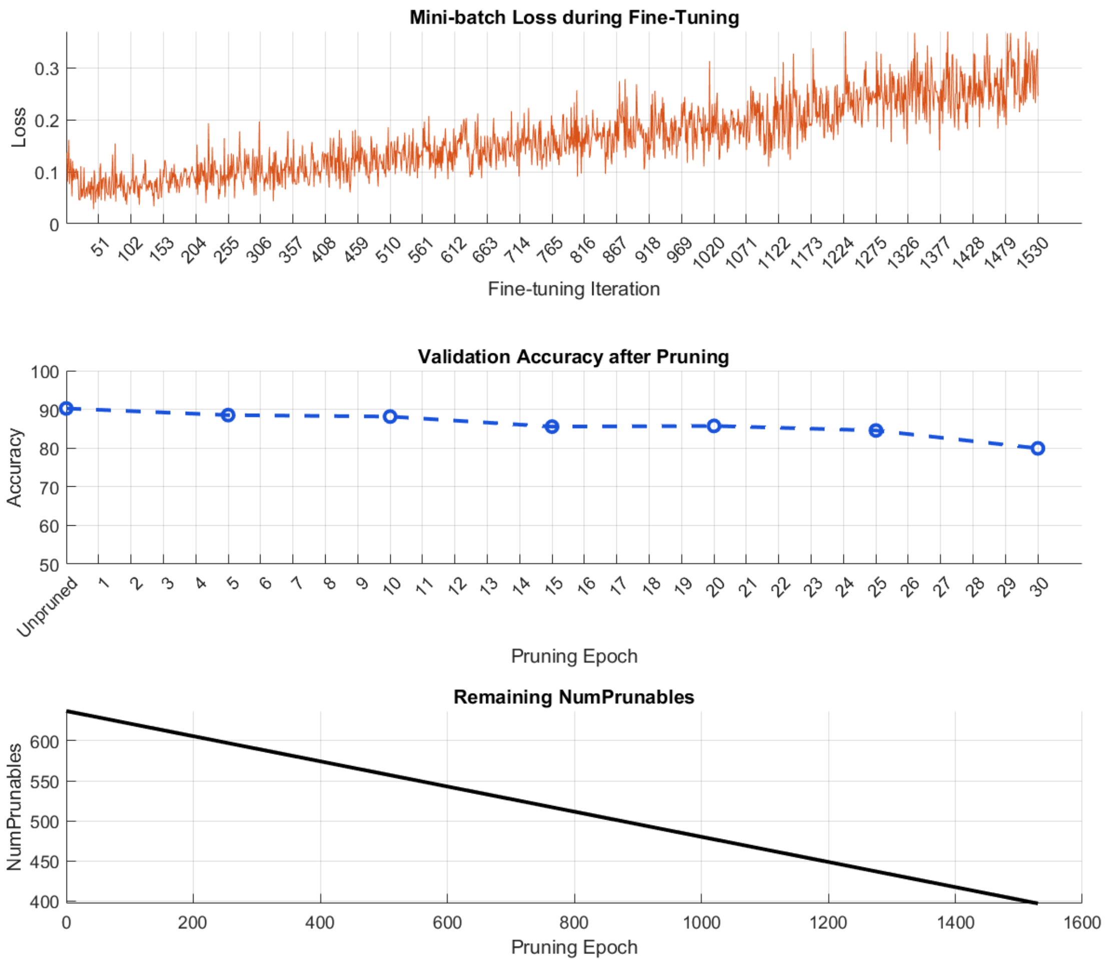
Pruning-specific parts

- Create pruner
- Begin pruning loop
  - Begin fine-tuning loop
    - Fine-tune the network
    - Compute Taylor scores
  - End fine-tuning loop
  - Prune least important filters
- Repeat pruning loop
- Extract pruned dlnetwork

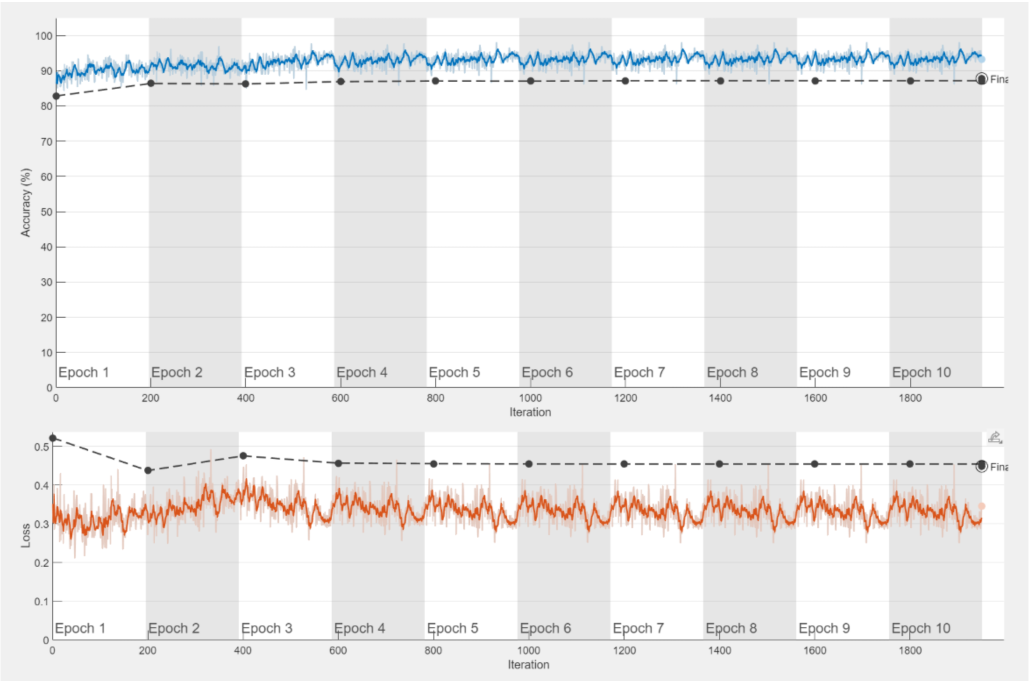
```
prunableNet = taylorPrunableNetwork(dlnet)

iteration = 0;
for pruningEpoch = 1:maxPruningEpochs
      shuffle(mbq);
      velocity = [];

      % Loop over mini-batches.
      fineTuningIteration = 0;
      while hasdata(mbq)
            iteration = iteration + 1;
            fineTuningIteration = fineTuningIteration + 1;
            [dlX, dlY] = next(mbq);

            % Evaluate activations, gradients, state, and loss
            [pruningActivations, pruningGradients, netGradients, state, loss] = ...
                               dlfeval(@modelGradientsPruning, prunableNet, dlX, dlY);

            % Update the network state
            prunableNet.State = state;

            % Update the network parameters using the SGDM optimizer
            [prunableNet, velocity] = sgdmupdate(prunableNet, ...
                                                   netGradients, velocity, learnRate,
            momentum);

            % Compute first-order Taylor scores and accumulate the score
            prunableNet = updateScore(prunableNet, pruningActivations, pruningGradients);

            % Stop fine-tuning loop when numMinibatchUpdates is reached
            if (fineTuningIteration > numMinibatchUpdates)
                break
            end
      end
      % Prune filters based on previously computed Taylor scores
      prunableNet = updatePrunables(prunableNet, MaxToPrune = 8);
end

dlnetPruned = dlnetwork(prunableNet)
```

Pruning reduces network accuracy but retraining recovers it

Pruning Progress

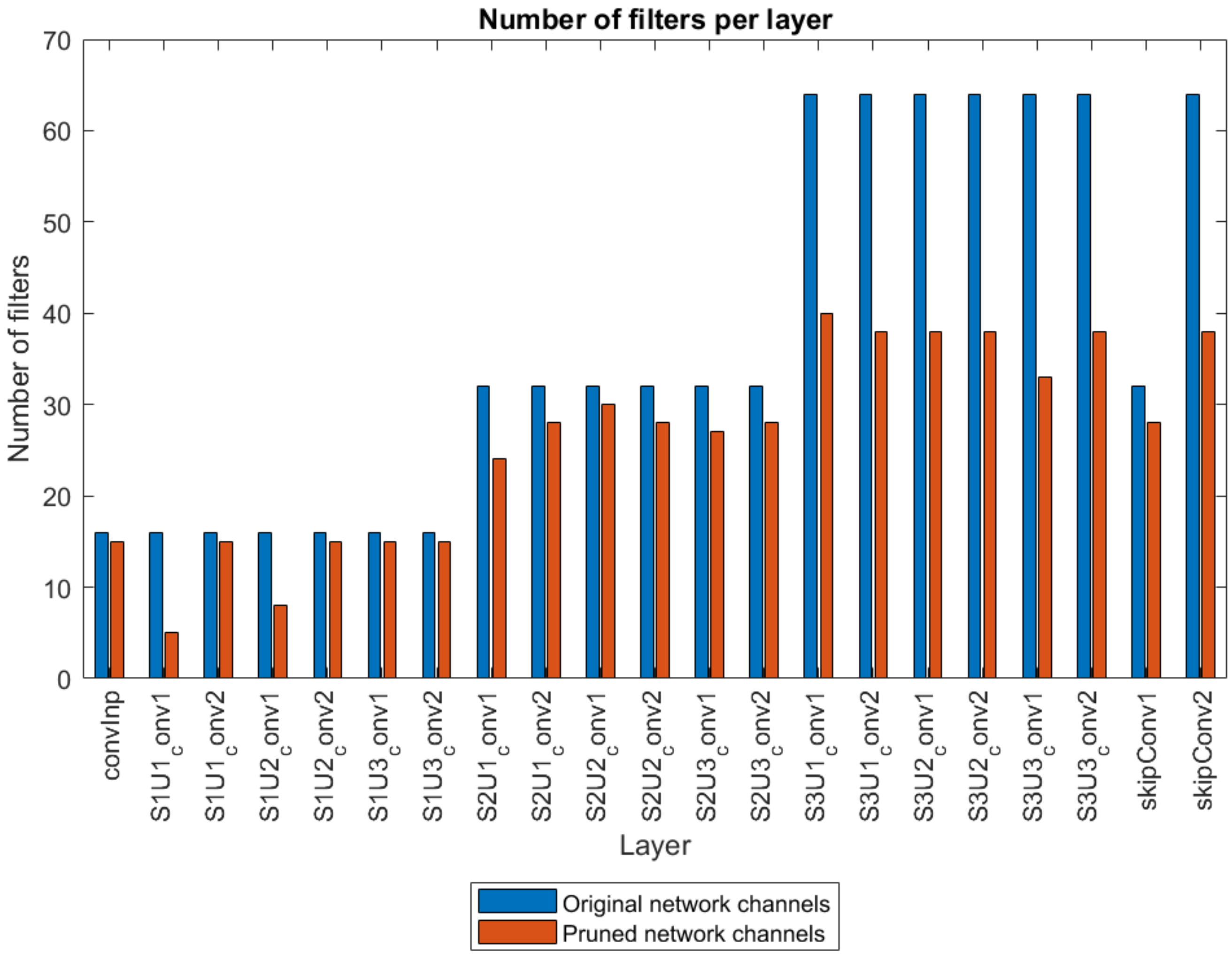


Retraining Progress



	Test Set Accuracy
Original Network	90.24% (+-0%)
Pruned Network	79.93% (-12%)
After Retraining Pruned Network	87.62% (-2.9%)

Pruning reduces network size and improves inference speed



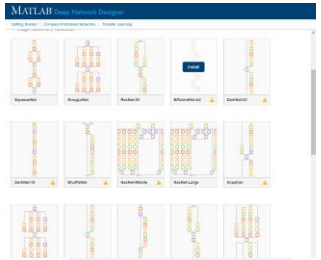

```
>> analyzeNetwork(prunedNet)
>> estimateNetworkMetrics(prunedNet)
```

	Test Set Accuracy	**Number of Learnables**	**Approx. Parameter Memory (MB)**
Original Network	90.24% (+-0%)	271,690 (+-0%)	1.0364 (-56%)
Pruned Network	87.62% (-2.9%)	120,723 (-56%)	0.4605 (-56%)

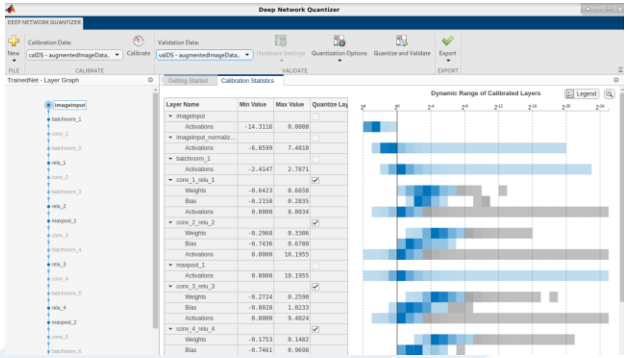
Quantize Model – Step 3

Different Routes for Compressing AI Models

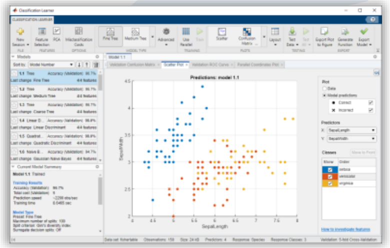
Deep Learning



Deep Learning Toolbox Model Quantization Library SPKG

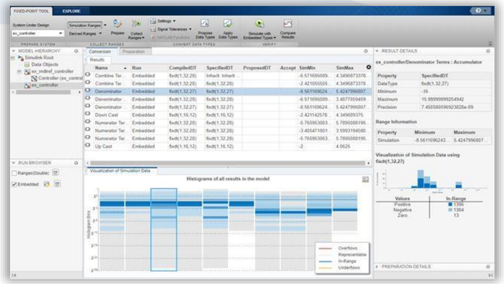


Determine Hardware Constraints → **Select Model** → **Simplify Model** → **Quantize Model Parameters** → **Deploy & Integrate**

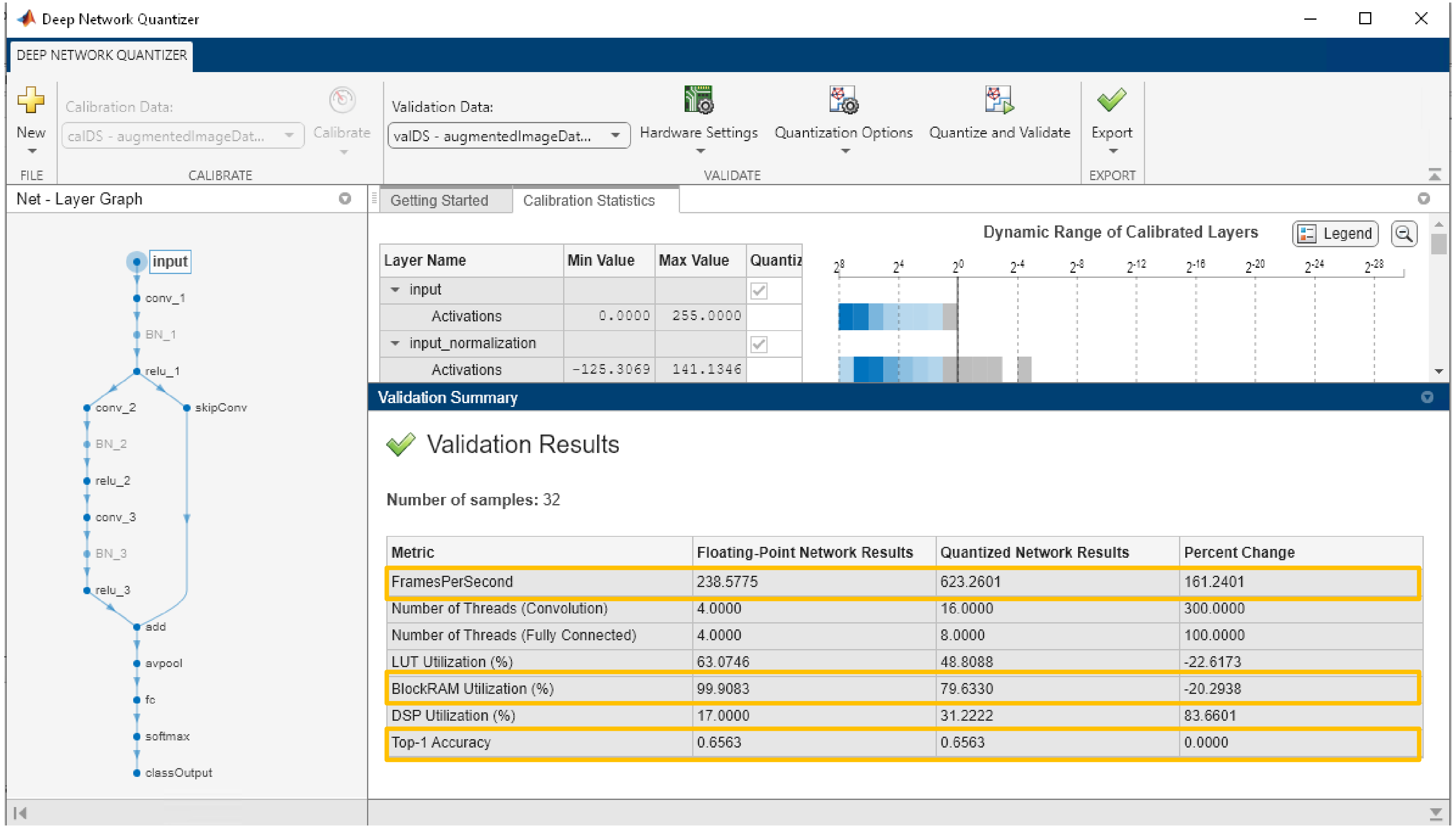


Machine Learning

Fixed-Point Designer



Quantization of Deep Learning Networks increases computational peak performance



Screenshot: Data collected on Xilinx Zynq-7000 SoC ZC706

- Improves inference speed
- Reduces storage space
- Minimal effect on accuracy

Key Takeaways

- MathWorks provides solution for whole AI development workflow

- Compress model for deployment
  - Purning & Quantization

Thank You