# SUPPLEMENTARY MATERIAL TO "MOTHERNETS: RAPID DEEP ENSEMBLE LEARNING"

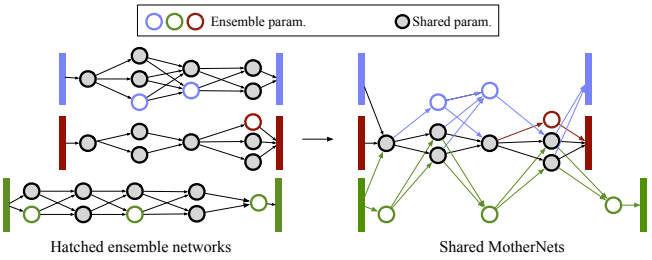


Figure A: Shared-MotherNets reduces inference time by sharing parameters between ensembles.

## A ALGORITHMS FOR CONSTRUCTING THE MOTHERNET

We outline algorithms for constructing the MotherNet given a cluster of neural networks. We describe the algorithms for both fully-connected and convolutional neural networks.

**Fully-Connected Neural Networks.** Algorithm A describes how to construct the MotherNet for a cluster of fully-connected neural networks. We proceed layer-by-layer selecting the layer with the least number of parameters at every position.

**Convolutional Neural Networks.** Algorithm B provides a detailed strategy to construct the MotherNet for a cluster of convolutional neural networks. We proceed block-by-block, where each block is composed of multiple convolutional layers. The MotherNet has as many blocks as the network with the least number of blocks. Then, for every block, we proceed layer-by-layer and construct the MotherNet layer at every position as follows: First, we compute the least number of convolutional filters and convolutional filter sizes at that position across all ensemble networks. Let these be $F_{min}$ and $S_{min}$ respectively. Then, in MotherNet, we include a convolutional layer with $F_{min}$ filters of $S_{min}$ size at that position.

## B SHARED-MOTHERNETS

We explain how MotherNets improve the efficiency of ensemble inference.

**Ensemble inference.** Inference in an ensemble of neural networks proceeds as follows: First, the data item (e.g., an image or a feature vector) is passed through every network in the ensemble. These forward passes produce multiple predictions – one prediction for every network in the ensemble. The prediction of the ensemble is then computed by combining the individual predictions using some averaging or voting function. As the size of the ensemble grows, the inference cost in terms of memory and time required for inference increases linearly. This is because for every additional ensemble network, we need to maintain its parameters as well as do an additional forward pass on them.

**Shared-MotherNets.** We introduce shared-MotherNets to reduce inference time and memory requirement of ensembles trained through MotherNets. In shared-MotherNets, after the process of hatching (step 2 from §2), the parameters originating from the MotherNet are incrementally trained in a shared manner. This yields a neural network ensemble with a single copy of MotherNet parameters reducing both inference time and memory requirement.

**Constructing a shared-MotherNet.** Given an ensemble $E$ of $K$ hatched networks (i.e., those networks that are obtained from a trained MotherNet), we construct a shared-MotherNet $S$ as follows: First, $S$ is initialized with $K$ input and output layers, one for every hatched network. This allows $S$ to produce as many as $K$ predictions. Then, every

**Algorithm A** Constructing the MotherNet for fully-connected neural networks

**Input:** E: ensemble networks in one cluster;
**Initialize:** M: empty MotherNet;

```
// set input/output layer sizes
 M.input.num_param ← E[0].input.num_param;
 M.output.num_param ← E[0].output.num_param;
 M.num_hidden ← getSmallestNetwork(E).num_hidden;

// set hidden layer sizes
for i ← 0 ... M.num_hidden-1 do
 └ M.hidden[i].num_param ← getMin(E,i);
return M;


// Get the min. size layer at posn
Function getMin(E,posn)
    min ← E[0].hidden[posn].num_param;
     for j ← 0 ... len(E)-1 do
        if E[j].hidden[posn].num_param < min then
           └ min ← E[j].hidden[posn].num_param
    └ return min;
```

**Algorithm B** Constructing the MotherNet for convolutional neural networks block-by-block.

**Input:** E: ensemble networks in one cluster;
**Initialize:** M: empty MotherNet;

```
// set input/output layer sizes and number of blocks
M.input.num_param ← E[0].input.num_param;
 M.output.num_param ← E[0].output.num_param;
 M.num_blocks ← getShallowestNetwork(E).num_blocks;

// set hidden layers block-by-block
for k ← 0 ... M.num_blocks-1 do
    M.block[k].num_hidden ← getShallowestBlockAt(E,k).num_hidden; // select the shallowest block
    for i ← 0 ... M.block[k].num_hidden-1 do
        M.block[k].hidden[i]..num_filters, M.block[k].hidden[i]..filter_size ← getMin(E,k,i)
return M;

// Get minimum number of filters and filter size at posn
Function getMin(E,blk,posn)
    min_num_filters ← E[0].block[blk].hidden[posn].num_filters;
     min_filter_size ← E[0].block[blk].hidden[posn].filter_size;
     for j ← 0 ... len(E) do
        if E[j].block[blk].hidden[posn].num_filters < min_num_filters then
            min_num_filters ← E[j].block[blk].hidden[posn].num_filters;
        if E[j].block[blk].hidden[posn].filter_size < min_filter_size then
            min_filter_size ← E[j].block[blk].hidden[posn].filter_size;
    return min_num_filters, min_filter_size;
```

hidden layer of $S$ is constructed one-by-one going from the input to the output layer and consolidating all neurons across all of $E$ that originate from the MotherNet. To consolidate a MotherNet neuron at layer $l_i$, we first reduce the $k$ copies of that neuron (across all $K$ networks in H) to a single copy. All inputs to the neuron that may originate from various other neurons in the layer $l_{i-1}$ across different hatched networks are added together. The output of this consolidated neuron is then forwarded to all neurons in the next layer $l_{i+1}$ (across all hatched networks) which were connected to the consolidated neuron.

Figure A shows an example of how this process works for a simple ensemble of three hatched networks. The filled circles represent neurons originating from the MotherNet and the colored circles represent neurons from ensemble networks. To construct the shared-MotherNet (shown on the right), we go layer-by-layer consolidating MotherNet neurons.

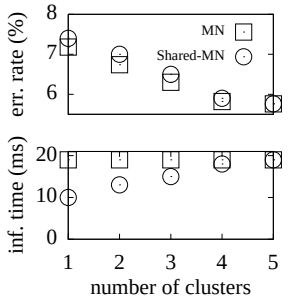


Figure B: Improving inference time

The shared-MotherNet is then trained incrementally. This proceeds similarly to step 3 from §2, however, now through the shared-MotherNet, the neurons originating from the MotherNet are trained jointly. This results in an ensemble that has $K$ outputs, but some parameters between the networks are shared instead of being completely independent. This reduces the overall number of parameters, improving both the speed and the memory requirement of inference.

**Memory reduction.** Assume an ensemble $E = \{N_0, N_1, \ldots N_{K-1}\}$ of $K$ neural networks (where $N_i$ denotes a neural network architecture in the ensemble with $|N_i|$ number of parameters) and its MotherNet $M$. The number of parameters in the ensemble is reduced by a factor of $\chi$ given by:

$$\chi = 1 - \frac{k|M|}{\sum_{i=0}^{K-1} |N_i|}$$

**Results.** Figure B shows how shared-MotherNets improves inference time for an ensemble of 5 variants of VGGNet as described in Table 1. This ensemble is trained on the CIFAR-10 data set. We report both overall ensemble test error rate and the inference time per image. We see an improvement of 2× with negligible loss in accuracy. This

improvement is because shared-MotherNets has a reduced number of parameters requiring less computation during inference time. This improvement scales with the ensemble size.

## C Model Covariance and Ensemble Predictive Accuracy

We can analyze how model covariance effects ensemble performance by using Chebyshev's Inequality to bound the chance that a model predicts an example incorrectly. By showing that lower covariance between models makes this bound on the probability smaller, we give an intuitive reason why ensembles with lower covariance between models perform better. The proof shows as well that the average model's predictive accuracy is important; finally, no assumptions need to be made for the proof to hold. The individual models can be of different quality and have different chances of getting each example correct.

Given a fixed training dataset, let $Y_i$ be the softmax value of model $i$ in the ensemble for the correct class, and let $\hat{Y} = \frac{1}{m}\sum_{i=1}^{m} Y_i$ be the ensemble's average softmax value on the correct class. Both are random variables with the randomness of $\hat{Y}$ and $Y_i$ coming through the randomness of neural network training. Under the mild assumption that $E[\hat{Y}] > \frac{1}{2}$, so that the a one vs. all softmax classifier would say on average that the correct class is more likely, than Chebyshev's Inequality bounds the probability of incorrect prediction. Namely, the correct prediction is made with certainty if $\hat{Y} > \frac{1}{2}$ and so the probability of incorrect prediction is less than

$$P(|\hat{Y} - E[\hat{Y}]| \geq E[\hat{Y}] - \frac{1}{2}) \leq \frac{Var(\hat{Y})}{E[\hat{Y}] - \frac{1}{2}}$$

From the form of the equation, we immediately see that keeping the average model accuracy $E[Y_i]$ high is important, and that degradation in model quality can offset reductions in variance. If we expand out $Var(\hat{Y})$ Since the variance of $\hat{Y}$ decomposes into $\frac{1}{m^2}(\sum_{i=1}^{m} Var(Y_i) + \sum_{i \neq i'} Cov(Y_i, Y_{i'})$, we see that low model covariance keeps the variance of the ensemble low, and that models which have covariance other models provides little benefit to the ensemble.