Supplementary Meta-Learning: Towards a Dynamic Model For Deep Neural Networks

Feihu Zhang
The Chinese University of Hong Kong, Shatin, Hong Kong
hi.yexu@gmail.com

Benjamin W. Wah
bwah@cuhk.eud.hk

Abstract

Data diversity in terms of types, styles, as well as radiometric, exposure and texture conditions widely exists in training and test data of vision applications. However, learning in traditional neural networks (NNs) only tries to find a model with fixed parameters that optimize the average behavior over all inputs, without using data-specific properties. In this paper, we develop a meta-level NN (MLNN) model that learns meta-knowledge on data-specific properties of images during learning and that dynamically adapts its weights during application according to the properties of the images input. MLNN consists of two parts: the dynamic supplementary NN (SNN) that learns meta-information on each type of inputs, and the fixed base-level NN (BLNN) that incorporates the meta-information from SNN into its weights at run time to realize the generalization for each type of inputs. We verify our approach using over ten network architectures under various application scenarios and loss functions. In low-level vision applications on image super-resolution and denoising, MLNN has 0.1∼0.3 dB improvements on PSNR, whereas for high-level image classification, MLNN has accuracy improvement of 0.4∼0.6% for Cifar10 and 1.2∼2.1% for ImageNet when compared to convolutional NNs (CNNs). Improvements are more pronounced as the scale or diversity of data is increased.

1. Introduction

A lot of attention has been directed to deep neural networks (DNN), with numerous breakthroughs on both low-level computer-vision applications, such as image super-resolution [8] and denoising [4, 15, 40], and high-level classification [11, 21], recognition [32] and detection [37].

Training neural networks (NNs) for computer-vision applications often requires a large amount of diverse training data for the NN to learn its features and to generalize them to unseen inputs. These diversities include those at the image-level (such as type, style and radiometric), region-level (smooth or textured), and pixel/patch-level (such as noise, location and artifact).

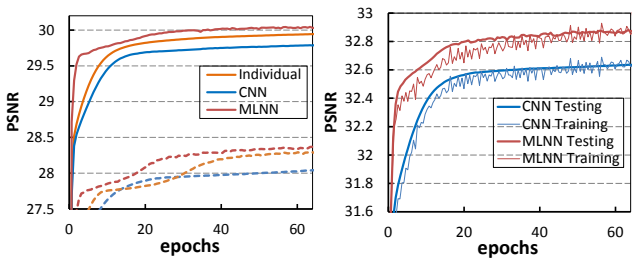

Figure 1: Using image super-resolution (Sec. 4) to illustrate the benefit of dynamic model adaptation to inputs. *Left*: Test errors for each type of inputs with a mixture of nature images (solid lines) and screenshots (dashed lines) as training data. Our MLNN (red lines) with model adaptation performs better than training with static manual classification (orange lines) and original CNNs without data classification (blue lines). *Right*: Training and testing results for a mixture of five types of images.

Traditional NNs are trained by maximizing an objective based on a loss function across all training data in order to find a model with fixed structure and weights. It optimizes the average behavior across all training data, without specializing to their variations. NNs with fixed structure and weights may under-perform when compared to NNs that can dynamically adapt to training data. Fig. 1 shows reduced errors when we manually classify training data into multiple classes according to their meta-level differences, and train each to learn a unique model. Obviously, such an approach is onerous. In some low-level vision tasks in which each pixel is an input as well as a target, it is impossible to manually classify them. Alternatively, we can train a second classifier to categorize the data before selecting the proper model. Its drawback is that it is hard for the new classifier to accurately classify every input, leading to possibly an improper model selected. Moreover, the number of classes is unknown and can be infinitely many.

To address the problem of simultaneous automatic classification and dynamic model selection with generalization,

Demo code and models are available at https://github.com/feihuzhang/MLNN.

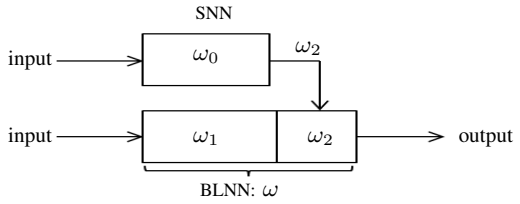

Figure 2: Proposed meta-learning NN (MLNN) using SNN (with ω_0 weights and ω_2 outputs) and BLNN (with ω weights).

we propose to use meta-learning to integrate the two steps together. Meta-learning systems [39] adapt to specific situations by dynamically searching for the best learning strategy in response to data diversity. It differs from base-level learning (like CNNs) in terms of adaptation: meta-level learning studies how to choose the right model dynamically, as opposed to base-level learning whose model is fixed with a priori assumptions (or inductive bias [26]).

Fig. 2 depicts our proposed *meta-learning NN* (MLNN). The top *supplementary NN* (SNN) with a small number of weights ω_0 (around 5$\sim$10% of ω) learns and outputs meta-level information ω_2 to the *base-level NN* (BLNN) that uses it as part of its weights. The SNN is a traditional NN that abstracts meta-information ω_2 important for the BLNN to adapt to different inputs. The BLNN maintains generalization using ω_1 as in traditional NN, as well as meta-information ω_2 from SNN for dynamic input adaptation. Our experience shows that the system can be trained by backward propagation and achieves far better performance when compared to traditional CNNs, especially on complex problems with large data diversity.

In this paper, we focus on realizing MLNN and study its behaviors in several high- and low-level computer vision applications. Significant improvements were found in our experiments on comparing MLNN with traditional CNNs using training and test data with significant diversities.

2. Related Work

2.1. Meta-Learning

In applications with diverse features, it is desirable to have algorithms that dynamically adapt to input features [30, 34, 35]. Meta-learning [23, 33] is an approach that focuses on learning the characteristics of a problem or its inputs and that allows the system to select a suitable model for new or unseen scenarios. By accumulating meta-knowledge [39], meta-learning systems build self-adaptive learners using algorithms that improve their bias dynamically.

Many existing approaches combine known models into a system and use meta-knowledge to select a proper model. Chan *et al*. [5] proposed to combine the results of multiple learning algorithms, each applied to a different set of training data, in order to adapt to diverse situations. Alexandros *et al*. [1] used decision trees as inducers at the meta level and mapped dataset characteristics to inducer performance in order to adapt inducers to datasets. Ali *et al*. [2] used meta-learning in automatic kernel selection for support vector machines. Ferrari *et al*. [10] developed a new method on distance-based problem characterization and ranking combination for selecting clustering algorithms.

When applied to NNs, existing meta-learning approaches can be used to select among learned models in different scenarios. Such an approach is inadequate because it is hard to enumerate all possible cases, and an incorrect selection may result in even worse performance.

In this paper, we propose a novel approach that integrates meta-knowledge into NN learning and that realizes selection and learning together. This integration avoids the segregation of the accumulation of incomplete meta-knowledge and the learning of general characteristics of inputs.

2.2. Dynamic Neural Networks

Dynamic learning has been studied for a long time. Recent work explored the idea of introducing more flexibility in the network structure and their weights. Jaderberg *et al*. [14] proposed the Spatial Transformer that allows the spatial manipulation of data in a NN. Kalchbrenner *et al*. [18] proposed structure-dynamic k-Max Pooling in order to handle input sentences of varying lengths.

There were recent studies on weight-dynamic NNs. Noh *et al*. [27] introduced a layer to generate dynamic outputs and to supply them as parameters of another fully connected layer. Klein *et al*. [19] designed a dynamic convolutional layer and used it for short-range weather prediction. Bert and Xu *et al*. [16] proposed dynamic filter networks that generated different filters with adaptive weights.

These weight-dynamic as well as traditional NNs are actually special cases of our model in Fig. 2. Our model consists of three parts: SNN with ω_0 for extracting meta-information from inputs, ω_1 in BLNN for generalization across inputs, and ω_2 output by SNN for dynamic adaptation in BLNN to different inputs. Noh *et al*.'s dynamic fully-connected layer [27], Klein *et al*.'s dynamic convolutional layer [19] and Xu *et al*.'s dynamic filter [16] all assume $\omega_1 = \varnothing$, whereas traditional CNNs assume $\omega_2 = \varnothing$.

To get similar modeling power as CNNs with ω, Klein *et al*. [19] and Noh *et al*. [27] need a complex ω_0 to generate enough parameters (ω_2) to run the convolutional computation. As a result, the size of the model is often too large for existing back propagation solvers, leading to overfitting or getting stuck in local minima. For example, when we implement the dynamic convolutional layer [19] for image super-resolution [8] with three convolutional layers, the network needs a hundred times of parameters in order to construct a structure comparable to the original CNNs. Such a network

can easily get stuck into an all-zero local minimum, and the memory for storing hidden layers increases quadratically.

In contrast, Bert *et al*. [16] tried to control model complexity by limiting ω_0 in order to avoid over-fitting. However, it comes at the cost of performance, since only a few parameters in ω_2 can be generated. The resulting parameters can only be used for simple 1D or 2D filters that are not as powerful as the original 3D convolutional layer for feature extraction. In our experiments, we observed a 10% degradation when compared to the original CNN method.

Some methods use a small number of weights to predict the remaining weights [7], whereas MLNN uses a small number of parameters to learn meta-knowledge and makes some weights dynamic. There are teacher-student networks [3, 31] with a similar guided scheme to improve modeling power or learning capability. However, none tried to realize the dynamic behavior of their filter kernels. In short, traditional NNs cannot dynamically adapt to diverse inputs and only achieve good average performance across all inputs. Hence, it may perform well in one case but not in another.

2.3. Dynamic Adaptation in Vision Applications

In low-level vision applications, such as image denoising [4] and super-resolution [8, 13], the objective is to improve image quality. To get similar or better results when compared to non-learning based methods, the NNs used must be able to precisely manipulate each pixel. When each pixel is a target, it is much more complex considering the diverse conditions involved. Although one solution is to use different models in smooth and textured regions, it is not as flexible as pixel-level meta-learning networks.

Unlike low-level vision tasks, NN models for high-level vision applications, such as image classification [21], detection [29] and recognition [32], are usually complex with millions of parameters and hundreds of layers. These models are designed to have good classification ability for handling possibly different view angles, resolutions and object locations. However, they would hit a performance ceiling even when the number of parameters and layers is increased. Moreover, future applications may involve complex or even mixtures of different kinds of images, rather than only nature images. In such applications, an effective scene-level meta-learning network would be essential.

3. Supplementary Meta-Learning Network

In implementing BLNN in Fig. 2 directly as a traditional NN like CNN, output $\mathbf{Y}_i$ for input $\mathbf{X}_i$ can be computed using $\mathbf{W}$ (convolutional filter) and $\mathbf{b}$ (bias vector) as follows:

$$\mathbf{Y}_i = \mathbf{W} \cdot \mathbf{X}_i + \mathbf{b}. \tag{1}$$

In traditional NNs, the solver optimizes an average behavior over input $\mathbf{X} = \{\mathbf{X}_0, ..., \mathbf{X}_n\}$ to result in fixed $\mathbf{W}$ and $\mathbf{b}$ independent of input. That is, all inputs share the same model despite their individual features.

3.1. Learning Meta-Knowledge

In contrast to traditional NNs, we propose to learn meta-knowledge in order to increase our model's flexibility. Since no model can perform well under all situations, it is best to use a design that selects the best model for each case. We integrate selection and generalization together by combining the learned meta-knowledge into the model itself.

Let $\mathbf{F}(\mathbf{X}_i)$ (ω_2 in Fig. 2) be the learned meta-knowledge for $\mathbf{X}_i$, where $\mathbf{F}$ is the function to be learned (the function form of ω_0 in Fig. 2). To combine $\mathbf{F}$ into the original convolutional model, we split it into two parts as follows.

$$\mathbf{F}(\mathbf{X}_i) = \{\mathbf{F}_\mathbf{w}(\mathbf{X}_i), \mathbf{F}_\mathbf{b}(\mathbf{X}_i)\}. \tag{2}$$

We then combine $\mathbf{F}$ into the two parts of Eq. (1) by the simple dot product and addition.

$$\begin{aligned} &\mathbf{Y}_i = [\mathbf{F}_\mathbf{w}(\mathbf{X}_i) \cdot \mathbf{W}]\mathbf{X}_i + [\mathbf{F}_\mathbf{b}(\mathbf{X}_i) + \mathbf{b}] \\ &\text{where } \ \mathbf{W}_i = \mathbf{F}_\mathbf{w}(\mathbf{X}_i) \cdot \mathbf{W}; \quad \mathbf{b}_i = \mathbf{F}_\mathbf{b}(\mathbf{X}_i) + \mathbf{b}. \end{aligned} \tag{3}$$

The meta-knowledge introduced in Eq. (3) allows different situations to be classified accordingly, whereas the fixed part $\{\mathbf{W}, \mathbf{b}\}$ in Eq. (1) enables generalization of each class to unseen cases. The model is unique and can adapt to different inputs, as the meta-knowledge captured by SNN through ω_0 and realized as $\{\mathbf{F}_\mathbf{w}, \mathbf{F}_\mathbf{b}\}$ will change with different inputs. Since the learning of both parts is done simultaneously in our model, the classification and generalization of each input can be realized in an integrated manner.

3.2. Matrix Setting

We discuss in this subsection the setting of $\{\mathbf{F}_\mathbf{w}, \mathbf{F}_\mathbf{b}\}$ in Eq. (3). For $\{\mathbf{W}, \mathbf{b}\}$ in ω_1, we assume $\mathbf{W}$ to be an $N \times M$ matrix, and $\mathbf{b}$, an $N \times 1$ vector. To keep the matrix sizes consistent, we set $\{\mathbf{F}_\mathbf{w}(\mathbf{X}_i), \mathbf{F}_\mathbf{b}(\mathbf{X}_i)\}$ in ω_2 as an $N \times N$ matrix and an $N \times 1$ vector, respectively.

Since the output of $\mathbf{F}_\mathbf{b}$ is an $N \times 1$ vector that can be easily learned, we focus on $\mathbf{F}_\mathbf{w}$. Learning such a function is expensive as it needs many new parameters to fit $\mathbf{F}_\mathbf{w}$ as well as N^2 memory to store intermediate and final outputs. To reduce the burden in learning, we need to simplify $\mathbf{F}_\mathbf{w}$ by reducing the number of elements in ω_2. There are three possibilities here: a) using a matrix with repeated elements, like a circulant matrix, b) choosing a matrix with special element distribution, like each row satisfying the Gaussian distribution that requires learning its mean and standard deviation, c) employing a sparse matrix.

We encountered some difficulties when using the first two alternatives. We observe that convergence is not stable during learning, with a speed that is slower than that of the original CNNs and learning can easily get stuck in local

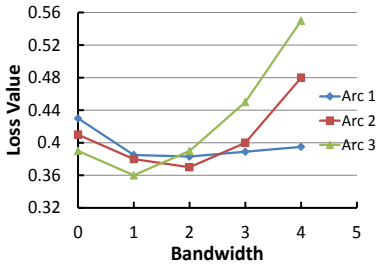

Figure 3: Image supper resolution (Sec. 4) used to illustrate the effect of bandwidth on loss values. The graph plots the performance (loss values) of different trained network architectures (Arcs 1-3) as a function of bandwidth.

minima. This could be caused by the dependence of matrix elements that counteract each other. For example, in a convolutional model, each element can be used to adjust more than one filters.

We, therefore, focus on using sparse matrices with independent elements. Learning in this case is more efficient as the number of elements is small, although its complexity is highly affected by the structure of $\mathbf{F_w}(\mathbf{X}_i)$.

To better utilize the learned meta-knowledge, it is obvious that $\mathbf{F_w}(\mathbf{X}_i)$ should have full rank, namely, $rank(\mathbf{F_w}(\mathbf{X}_i)) = N$. For $\mathbf{W}_i = (\mathbf{w}_1, ..., \mathbf{w}_n)^T$, we know that $rank(\mathbf{W}_i) \leq rank(\mathbf{F_w}(\mathbf{X}_i))$. If $rank(\mathbf{F_w}(\mathbf{X}_i)) < N$, then there exists $j \neq k$ such that $\mathbf{w}_k = \alpha \mathbf{w}_j$, which makes $\mathbf{w}_k$ homogeneous with $\mathbf{w}_j$ and contributes nothing new to the model. For example, in an extreme case where $rank(\mathbf{F_w}(\mathbf{X}_i)) = 1$, $(N-1)/N$ of the elements in $\mathbf{W}_i$ are redundant because all the N filters are homogeneous. This setting performs no better than directly using a $1 \times M$ matrix as $\mathbf{W}_i$. During learning, although we cannot control the values of the weights, we can eliminate redundant information in $\mathbf{F_w}(\mathbf{X}_i)$ to reduce the complexity in learning.

In sparse matrices, a band matrix is a special case with full rank regardless of bandwidth. We, therefore, use a band matrix to implement $\mathbf{F_w}(\mathbf{X}_i)$, as it has full rank and learning complexity can be controlled by adjusting its bandwidth.

A band matrix has many good properties when realizing DNNs. Firstly, we only need to learn no more than $3N$ (*resp.* $5N$) parameters for a 1-band (*resp.* 2-band) matrix. Secondly, learning is fast and memory efficient, since band matrices lend themselves to more efficient computations than dense ones. Lastly, band matrices have full rank and are more effective in utilizing the learned elements when compared to other sparse matrices.

When implementing the output of $\mathbf{F_w}$ as a band matrix, its bandwidth will significantly affect the speed, memory usage and accuracy in learning. When bandwidth is increased, more elements will need to be learned and the SNN becomes more complex. Limited by the ability of back propagation, it will soon lead to either over-fitting or getting stuck in local minima. Fig. 3 illustrates this phenomenon in which learning performance reaches an optimum at a particular bandwidth. In most cases, this optimum is at a bandwidth between 1 and 3, with little difference among them. As a result, we set the bandwidth to 1 in all our experiments.

3.3. Efficient Implementation of MLNN

The original Eq. (3) is not flexible when used in DNNs. In this subsection, we simplify the model by employing simple transformations to $\mathbf{F_w}$ and $\mathbf{F_b}$ to facilitate its implementation in existing deep-learning platforms. We first assume $\mathbf{F_w}(\mathbf{X}_i)$ and $\mathbf{F_b}(\mathbf{X}_i)$ to be near linear transformation of $\mathbf{X}_i$, which can be implemented by several convolutional layers. Since $\mathbf{F_b}$ has no fixed form, we give it a new form:

$$\mathbf{F_b}(\mathbf{X}_i) = [(\mathbf{F_w}(\mathbf{X}_i) - \mathbf{I}) \cdot \mathbf{b}]^T + \mathbf{F}'_\mathbf{b}(\mathbf{X}_i). \tag{4}$$

Note that $\mathbf{F}'_\mathbf{b}(\mathbf{X}_i)$ and $[(\mathbf{F_w}(\mathbf{X}_i) - \mathbf{I}) \cdot \mathbf{b}]^T$ can be treated as a base shift vector. In that case, $\mathbf{F}'_\mathbf{b}(\mathbf{X}_i)$ is still dynamic to inputs but now becomes the target of learning. As the original $\mathbf{F_b}$ and $\mathbf{F_w}$ are learned as a set of convolutional layers, $\mathbf{F}'_\mathbf{b}$ can also be learned using these layers.

By substituting Eq. (4) into Eq. (3), we have

$$\mathbf{Y}_i = \mathbf{F_w}(\mathbf{X}_i) \cdot (\mathbf{W}\mathbf{X}_i + \mathbf{b}) + \mathbf{F}'_\mathbf{b}(\mathbf{X}_i). \tag{5}$$

We further introduce a linear transformation of $\mathbf{X}_i$ to $\mathbf{F_w}(\mathbf{X}_i)$ and $\mathbf{F}'_\mathbf{b}(\mathbf{X}_i)$,

$$\begin{aligned} \mathbf{Y}_i = \mathbf{F}'_\mathbf{w}(\mathbf{W}\mathbf{X}_i + \mathbf{b}) \cdot (\mathbf{W}\mathbf{X}_i + \mathbf{b}) + \mathbf{F}''_\mathbf{b}(\mathbf{W}\mathbf{X}_i + \mathbf{b}) \\ \text{where } \mathbf{F_w}(\mathbf{X}_i) = \mathbf{F}'_\mathbf{w}(\mathbf{W} \cdot \mathbf{X}_i + \mathbf{b}) \\ \text{and } \mathbf{F}'_\mathbf{b}(\mathbf{X}_i) = \mathbf{F}''_\mathbf{b}(\mathbf{W} \cdot \mathbf{X}_i + \mathbf{b}). \end{aligned} \tag{6}$$

As indicated above, the fixed part $\mathbf{W}\mathbf{X}_i + \mathbf{b}$ in Eq's (5)-(6) can be implemented by convolutional layers. In this paper, we assume that $\mathbf{F}'_\mathbf{w}$, $\mathbf{F}'_\mathbf{b}$ and $\mathbf{F}''_\mathbf{b}$ can all be fitted by some convolutional layers. We further restrict the values of $\mathbf{F}'_\mathbf{w}(\mathbf{X}_i)$ in a range $(-1,1)$ or $(0,1)$ using an activation function like $Tanh$ or $Sigmoid$. This approach helps limit the search space and improves convergence, while avoiding getting stuck in local minima. Finally, the outputs of $\mathbf{F}'_\mathbf{w}$ are reshaped to the required band matrices.

Fig. 4 shows that $\mathbf{W}\mathbf{X}_i + \mathbf{b}$ can be achieved in BLNN with ω_1. We further implement SNN using two sub-branches for $\mathbf{F}'_\mathbf{w}$ and $\mathbf{F}'_\mathbf{b}/\mathbf{F}''_\mathbf{b}$, respectively. The difference between their implementations is that Eq. (5) needs to use the output of the previous layers in Stage (a) as inputs to the sub-branches, whereas Eq. (6) directly uses the output of the major branch in Stage (b). When the BLNN layer is a $k \times k$ convolutional layer, to guarantee shape and range consistency, the two branches of SNN also need similar $k \times k$ convolutional scope. For Eq. (6), we can always use 1×1 filters to improve efficiency. For these reasons, we use Eq. (6) to realize the dynamic model in our experiments. Eq. (5) will only be used in fully-connected layers or when the inputs of SNN is different from those of BLNN.

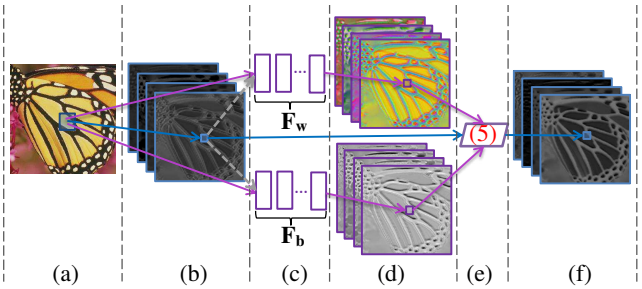

Figure 4: Implementation of MLNN, where pixel-level meta knowledge $\{\mathbf{F_w}(\mathbf{X}_i), \mathbf{F_b}(\mathbf{X}_i)\}$ is extracted in SNN by two sub-branches. (a) Input sample; (b) Output of ω_1; (c) ω_0: two-branch SNN; (d) Learned meta-knowledge ω_2: $\{\mathbf{F_w}(\mathbf{X}_i), \mathbf{F_b}(\mathbf{X}_i)\}$; (e) Combining the meta-knowledge ω_2 by Eq. (5); (f) Output of MLNN. For implementation of Eq. (6), the input to SNN in Stage (c) is changed to the gray dashed arrows.

3.4. Understanding Performance Improvements

This subsection explains the source of performance improvements in MLNN from two aspects.

Meta-level learning. As discussed earlier, traditional NN optimizes the average performance across all inputs, without tailoring its behavior to address the diversity of inputs. In contrast, MLNN uses SNN to learn input-specific meta-knowledge and provides this information in the form of dynamic weights to BLNN. Stage (d) of Fig. 4 visualizes the meta-knowledge learned, where SNN learns the meta-information for each pixel and different colors correspond to different classes of inputs. By introducing meta-knowledge into BLNN, ω can adapt itself to different types of inputs, leading to better adaptability of MLNN.

Dynamic weights. An important difference between traditional CNN and MLNN models lies in the dynamic nature of their weights/parameters. During training, inputs for both are varying and weights are updated. However, during testing, inputs for CNNs are varying but weights are fixed. For MLNNs, inputs are varying and weights are adaptive. This significantly increases their modeling power. Our MLNN successfully introduces flexibility into a NN by using dynamic weights. As shown in Fig. 5, the filter kernels of MLNN are always adaptive to inputs, whereas those of traditional NNs are fixed. As a result, even when using simpler 48-channel outputs for MLNN, its performance is still better than a fixed model with 64-channel outputs.

As discussed in Sec. 2.2, existing weight-dynamic NNs are actually special cases of MLNN shown in Fig. 2. They do not have a term ω_1 to realize the balance between generalization (ω_1) and input-adaptation (ω_2). In contrast, MLNN uses a small number of additional parameters in SNN to make the model dynamic to inputs. Most of the parameters are used in the generalization term ω_1 to avoid overfitting and to generalize to unknown scenes.

We can find evidence of MLNN's good generalization

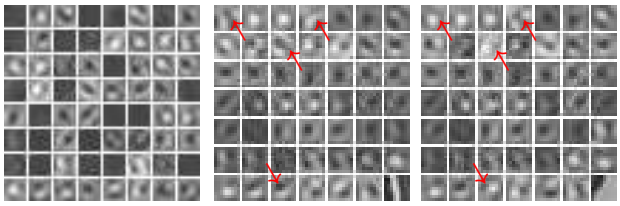

Figure 5: Super-resolution (Sec. 4) used to compare filter kernels (Arc 1 trained on dataset-5). (a) The fixed 64 first-layer filters of the original CNN model. (b) and (c) MLNN model: adaptive filter kernels on two different input patches (last one). Most of the filters are slightly different, whereas some are significantly different (indicated by red arrows). MLNN, based on adaptive kernels, can use a simpler 48 first-layer model to achieve better performance.

behavior by comparing the filter kernels of different inputs. Fig. 5 visualizes the filter kernels of the first layer in a learned MLNN model. For the two different input patches, most of the filters are slightly different from each other and only some are significantly different. It is likely that those similar filters realize the generalization of the MLNN model, whereas those that are significantly different adapt the model's behavior to different inputs.

3.5. Relation to Polynomial and Residual Nets

Polynomial NNs (PNNs) [28, 38] are designed to learn a polynomial model and possess more powerful capabilities to represent or classify data. The popular CNNs can be understood as a near linear system, although they use activation functions to increase their modeling power.

Our MLNN can be treated as one kind of well-controlled and organized PNNs. Since we utilize the dot product to combine meta-knowledge $\mathbf{F_w}(\mathbf{X}_i)$ into the original convolutional layers, $\mathbf{F_w}(\mathbf{X}_i)$ can be represented as a near-linear transformation of $\mathbf{X}_i$. After each MLNN layer, the degree of the model will increase quadratically, which significantly increases its complexity when compared to CNNs. Meanwhile, due to the well controlled architecture and parameters, MLNN overcomes the common problems of PNNs [9,24] that are hard to implement and can easily overfit.

In a more general sense, the MLNN layer can also be used as one kind of quadratic activation function to effectively increase the search space of the NN model.

Deep Residual Nets [11] insert shortcut connections and turn a network into its counterpart residual version. This successfully solves the convergence problem in very deep NNs and significantly improves the classification accuracy.

Similar to residual learning, we use element-wise additions to combine meta-knowledge $\mathbf{F_b}(\mathbf{X}_i)$. In particular, in implementing Eq's (5) and (6), when we remove the branch of $\mathbf{F_w}$, the SNN (as shown in Fig. 4) is directly transformed to residual connections as those in [41] and [11]. In general, a residual net partially realizes dynamic learning in the bias

Table 1: Baseline Network Architectures

Layer Set	Low-Level Vision Applications				High-Level Classification Applications	
	Arc 1 3-layer [8]	Arc 2 3-layer [8]	Arc 3 4-layer [8]	Arc 4 21-layer [42]	Arc 5 4-layer [17]	Arc 6 4-layer
set 1	$\begin{bmatrix}\text{conv}, 9\times 9, 64\\ \text{ReLU}, 64\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 9\times 9, 128\\ \text{ReLU}, 128\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 9\times 9, 128\\ \text{ReLU}, 128\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 3\times 3, 64\\ \text{ReLU}, 64\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 5\times 5, 20\\ \text{Maxpool}, 2, 2\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 5\times 5, 32\\ \text{Maxpool}, 2, 2\\ \text{ReLU}, 32\\ \text{BatchNorm}\end{bmatrix}\times 1$
set 2	$\begin{bmatrix}\text{conv}, 1\times 1, 32\\ \text{ReLU}, 32\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 1\times 1, 64\\ \text{ReLU}, 64\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 5\times 5, 64\\ \text{ReLU}, 64\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 3\times 3, 64\\ \text{BatchNorm}\\ \text{ReLU}, 64\end{bmatrix}\times 19$	$\begin{bmatrix}\text{conv}, 5\times 5, 50\\ \text{Maxpool}, 2, 2\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 5\times 5, 32\\ \text{ReLU}, 32\\ \text{Maxpool}, 2, 2\\ \text{BatchNorm}\end{bmatrix}\times 1$
set 3	$[\text{conv}, 5\times 5, 1]\times 1$	$[\text{conv}, 5\times 5, 1]\times 1$	$\begin{bmatrix}\text{conv}, 1\times 1, 64\\ \text{ReLU}, 64\end{bmatrix}\times 1$	$[\text{conv}, 3\times 3, 1]\times 1$	$\begin{bmatrix}\text{fc}, 500\\ \text{ReLU}, 500\end{bmatrix}\times 1$	$\begin{bmatrix}\text{conv}, 5\times 5, 64\\ \text{ReLU}, 64\\ \text{Maxpool}, 2, 2\end{bmatrix}\times 1$
set 4			$[\text{conv}, 5\times 5, 1]\times 1$		$[\text{fc}, 10]\times 1$	$[\text{fc}, 10]\times 1$

term. We believe that our MLNN helps partially understand the performance of residual nets. On the other hand, the convergence behavior of our MLNN can also be attributed to the realization of residual learning in MLNN.

4. Experimental Results

In this section, we verify the performance of our MLNN design under multiple application scenarios.[1]

4.1. Parameters and Network Settings

Table 1 shows the baseline network architectures under different applications. We replace some convolutional or fully connected layers in these architectures by MLNN in order to get new dynamic models. The performance of these new models are then measured and compared to the original standard models under different applications.

Given an original $[k\times k, n]$ convolutional layer, to ensure that the number of parameters are not significantly changed, we discuss their setting when realizing the layer MLNN.

For BLNN, we employ a $[k\times k, 3n/4]$ convolutional layer, whereas for SNN, in each SNN branch ($\mathbf{F_w}$ and $\mathbf{F_b}$), we use two 1×1 convolutional layers with ReLU activation between them. Specifically, we implement one $[1\times 1, n/4]$ and one $[1\times 1, 3n-2]$ convolutional layers for $\mathbf{F_w}$; for $\mathbf{F_b}$, we use $[1\times 1, n/4]$ and $[1\times 1, n]$ layers.[2]

To combine $\mathbf{F_w}$, we first employ $TanH$ (for classification tasks) or $Sigmoid$ (for low-level vision tasks) to restrict the output of $\mathbf{F_w}$, before reshaping it to a 1-band matrix. In contrast to gated structures [36], an activation function after $\mathbf{F_w}$ is used to avoid the influence of initializations. Configurations without activation perform similar to or slightly better (<0.15%) than those with activations, but their performance highly depends on the initialization of the bias term in the $\mathbf{F_w}$ layers.

[1]More details and demonstrations are available in the supplementary material at http://www.feihuzhang.com/Publication.html

[2]For better performance, we can use more complex layers (e.g. two 3×3 convolutional layers) in SNN with 20-50% extra computational cost.

Table 2: Evaluations using image supper-resolution (PSNR: dB)

Data set	Model	Dataset-1	Dataset-2	Dataset-5
Arc 1	Original	29.97	29.01	32.64
	CNN-c	30.01	29.07	32.71
	MLNN	**30.11**	**29.20**	**32.87**
Arc 2	Original	30.05	29.08	32.73
	MLNN	**30.18**	**29.26**	**32.93**
Arc 3	Original	30.21	29.26	32.88
	MLNN	**30.31**	**29.39**	**33.03**
Arc 4	Original	30.34	29.38	32.99
	MLNN	**30.43**	**29.50**	**33.11**

4.2. Low-Level Image Quality Improvement

In almost all low-level vision applications, each pixel will be a target and the training data can contain more than ten million samples. Hence, data diversity is very common and significant, and dynamic MLNN models will have apparent advantages over standard models.

Image super-resolution, which aims at recovering a high-resolution image from a single low-resolution image, is a classical problem in low-level computer vision. Dong *et al.* show that traditional super-resolution algorithms can be replaced by a deep CNN to get better performance and faster speed [8]. Due to the convenience of collecting and controlling training data, this is one of the most suitable applications to verify the performance of MLNN.

To control data diversity, we build mixed datasets with five types of images, including natural, depth and cartoon images, as well as screenshots and oil paintings (with about 100,000 51×51 patches for each type, 9/10 for training and 1/10 for testing). We refer natural image as "dataset-1," mixture of natural images and screenshots as "dataset-2" and mixture of all five types of images as "dataset-5." We use three kinds of network structures proposed in [8] and another much deeper 21-layer residual net [42] as the base models for evaluations and comparisons.

In each comparison, we implement $3\times$ super-resolution. The first (for Arc 1-2), the first two (for Arc 3) or the first five (for Arc 4) original convolutional layers are replaced by MLNN. All other settings in training are set the same as those in [8]. Table 2 shows the PSNR for measuring the difference between testing results and ground truths after train-

Table 3: Evaluations of Arc1 on 5 types of images (PSNR: dB)

Data Set	CNN	Individual Models	MLNN
natural images	29.81	29.97	**30.02**
screenshots	28.09	28.30	**28.38**
depth images	43.89	**44.23**	44.11
cartoon images	32.77	32.92	**33.02**
oil paintings	30.22	30.32	**30.38**

Table 4: Evaluations using image denoising (PSNR: dB)

Architecture	With CNN Model	With MLNN Model
Arc 1	28.71	**28.87**
Arc 3	28.87	**28.98**
Arc 4	29.02	**29.14**

ing for 64 epochs. Fig. 1 further compares the convergence curves of Arc 1. We observe a) that in all the architectures, MLNN helps get 0.1-0.3 dB improvement in PSNR; b) that the improvements are more pronounced as data diversity is increased. For example, in Arc 1, MLNN only improves 0.14 dB for "dataset-1," whereas MLNN gets 0.23 dB improvement for "dataset-5" with significant data diversity.

Table 3 shows the detailed results on each type of images in "dataset-5" for Arc 1. When compared to a single CNN trained on "dataset-5," training the same CNN for each type of images to get five models leads to better performance. Our MLNN performs even better in four out of the five types than the CNNs trained for each type. This happens because MLNN learns pixel-level meta-knowledge for each pixel and can differentiate input diversities of each pixel even within a single type of images.

Image denoising is another low-level vision application that benefits from the development of DNNs. With the Berkeley Segmentation Database [25] as the ground truth, random noises are added to the images, using Gaussian noises with standard deviation in [0, 50]. 240 images are randomly chosen for training and 60 for testing. We employ simple fast 3-layer Arc 1-2 and the much deeper 21-layer residual net in [42] as the base network settings, while keeping all the other settings the same as those in [42] as well as our MLNN the same as those in image super-resolution. Table 4 shows that all the three structures with MLNNs get about 0.11-0.16 dB improvements when compared to those with the original CNNs after training for 64 epochs.

4.3. High-Level Image Classification

In scene classifications, data diversity is not as significant as low-level vision applications since existing datasets only contain no more than one million samples. However, MLNN still helps get 0.4∼1.5% improvements in accuracy on the following three datasets. Moreover, its improvement will be more pronounced as data scale is increased.

MNIST database of handwritten digits [22] of 0-9 has a training set of 60,000 examples and a test set of 10,000 examples. We use a 4-layer network [17] (Arc 5 in Table 1)

Table 5: Performance comparisons in image classification

Data Set	Model	Top-1 Error (%)	Top-5 Error (%)
Mnist	Arc 5	1.11	——
	MLNN	0.71	——
Cifar-10	Arc 6	18.7	——
	MLNN	18.3	——
	ResNet-44	7.61	——
	Res-MLNN-44	6.97	——
	ResNet-110	6.38	——
	Res-MLNN-110	5.82	——
ImageNet	AlexNet	42.6	19.6
	MLNN-1	42.2	19.0
	MLNN-2	41.3	18.5
	MLNN-3	41.1	18.2
	ResNet-50	24.53	7.89
	Res-MLNN-50	23.27	7.02

as the base setting and get 1.11% error rate on the test set after 10,000 iterations in training. After changing the first fully-connected layer to MLNN of Eq. (5), the error rate is reduced to 0.71%.

CIFAR-10 dataset [20] consists of 60,000 32×32 colour images in 10 classes, with 6,000 images per class (50,000 for training and 10,000 for testing). The CIFAR10 Caffe model [17] (Arc 6 in Table 1) was trained for the CIFAR-10 classification task. Without any data augmentation, we trained it for 70,000 iterations with a batch size of 100. Table 5 shows the error rates obtained with and without MLNN. The original Arc 6 achieves a top-1 error of 18.7%. After replacing the fist two convolutional layers by MLNN (as shown in Fig. 4) with nothing else changed, the error rate is reduced to 18.3%. We also tested the more complicated 44-layer and 110-layer residual nets [11]. By replacing the first 3×3 layer in each of the residual blocks by MLNN and training them with data augmentation, the accuracy has been improved by 0.6%. Since the CIFAR-10 dataset contains only 60,000 samples, data diversity is not as obvious as that in low-level vision applications where each pixel is a target. The performance of MLNN is, therefore, limited, but we still can achieve 0.4∼0.6% accuracy improvement.

ImageNet [6] is a data set with over 15 million labeled high-resolution images belonging to roughly 22,000 categories. We use the ILSVRC-2012 which is a subset of ImageNet with 1000 images in each of 1000 categories for training and another 50 in each category for validation.

We first use AlexNet [21] as the base setting of the network architecture, which contains five convolutional layers, three fully connected layers and some non-linear activations. For MLNN-1, we replace the fist fully connected layer of AlexNet by the implementation of Eq. (5); for MLNN-2, we change the 3rd and the 4th convolutional layers to the MLNN implementation of Eq. (6); and for MLNN-3, we include all the changes in MLNN-1 and MLNN-2. We fixed all other settings and trained the models for 310,000 iterations without data augmentation. We then tested the more advanced

ResNet-50 [11] with data augmentation to increase data diversity and to avoid overfitting. We replaced the bottleneck 3×3 layer of each residual block by MLNN layers (with 16 layers changed).

Table 5 reports the top-1 and top-5 error rates on the validation data. Convolutional nets with MLNN outperform the base models by a large margin (0.4%-1.5% in top-1 error rate and 0.6%-1.4% in top-5 error rate). Moreover, the improvement is more pronounced as we increase the number of MLNN layers from one to three in the base model.

4.4. Effects on Number of SNN Branches

We first study whether improvements are caused by adding extra branches to the original network. In the CNN-c model (Table 2), we use concatenation to replace the dot product and element-wise addition in Eq. (6). That is, CNN-c has the same network branches and even more parameters when compared to MLNN. However, CNN-c has very limited improvements (0.04∼0.06 dB) when compared to MLNN (0.14∼0.23 dB). This illustrates that the improvements brought by MLNN are attributed to the its dynamic behavior with respect to each input.

Since the meta-level knowledge is split into two parts ($\mathbf{F_w}$ and $\mathbf{F_b}$) in our implementation, we test the effects of each branch in the SNN implementation individually. Firstly, we replace $\mathbf{F_w}$ by traditional fixed-weight structures and add $\mathbf{F_b}$ directly to $\mathbf{F_w}$ with the output of $\mathbf{F_b}$ adjusted to the same shape as $\mathbf{F_w}$. We find that for image upsampling, PSNR drops from 32.87 dB to 32.68 dB, and for AlexNet classification, the top-1 error rate increases by 0.61%. These results support our claim that adaptive weights are effective in MLNN. Secondly, we change $\mathbf{F_b}$ to an identity mapping [12]. We observe that for image upsampling, PSNR drops from 32.87 dB to 32.7 dB and for AlexNet classification, the top-1 error rate increases by 0.73%. It is obvious that both SNN branches are indispensable in our model to get good performance.

4.5. Effects on Number of MLNN Layers

The number of the MLNN layers also affects the model's performance. In general, as the number of MLNN layers is increased, the accuracy of the model will first increase and then begin to drop. For example, in AlexNet [21], if we continue to change the 5$^{\text{th}}$ convolutional layer to MLNN to get MLNN-4, we get similar accuracy as MLNN-3. However, if we further change the second fully connected layer to get MLNN-5, the accuracy begins to drop by 0.2%. This happens because after adding a new MLNN layer, the degree of the model will increase quadratically. As we ceaselessly add more MLNN layers to a model, overfitting begins to occur at some point. Our experience shows that in a neural net, we change 1/4∼1/2 of the traditional layers to MLNN layers in order to get good performance.

Table 6: Complexity and Efficiency Comparisons

Arch.	Model	Params	FLOPs	Speed
Arc 1	Original	8K	8.0×10^3	2.5 MPixel/s
	MLNN	8.5K	8.5×10^3	2.3 MPixel/s
Arc 4	Original	0.7M	7.0×10^5	21 KPixel/s
	MLNN	0.69M	6.9×10^5	21 KPixel/s
AlexNet	Original	60M	7.3×10^8	675 fps
	MLNN-2	60.6M	7.8×10^8	633 fps
	MLNN-3	66M	7.9×10^8	619 fps

4.6. Efficiency Comparisons and Analysis

In this subsection, we compare the efficiency of MLNN with standard CNN models. We implement MLNN based on the deep learning platform Caffe [17] using a TESLA K40C GPU. Table 6 compares the computational complexity and the speed of the standard CNN and the dynamic MLNN models. For low-level vision applications, the computational complexity is directly proportional to the number of parameters in the model. During the implementation, we control the number of parameters by reducing the number of channels of the output after the MLNN layers to offset the extra parameters used in learning meta-knowledge. As a result, efficiency does not decrease too much. For image classification, the running time of the new MLNN model is only sightly increased by 5-10%.

5. Conclusions

MLNN studied in this paper introduces meta-knowledge to DNNs in order to allow them to adapt to inputs with 5%∼10% extra computation costs. Meta-knowledge is first learned by SNN to differentiate various types of inputs and then combined in BLNN to make the MLNN model adaptive to inputs. We verify its performance improvement using several high- and low-level vision applications. By replacing some standard convolutional or fully-connected layers with our MLNN layers in a traditional NN, our new model can outperform the base model by a large margin.

The MLNN layers are beneficial when the dataset has large diversity, such as different image types, textures, and radiometric and noise conditions. Usually, data diversity is larger as the data scale is increased, and the improvements brought by MLNN will be more pronounced.

The limitation of MLNN is that in very deep NNs, overfitting may occur when too many MLNN layers are used (like in a pure MLNN net), as they significantly increase the degree of the model. As a result, MLNN layers should be used in conjunction with standard layers (convolutional or fully-connected layers) in a DNN model.

Acknowledgement

Research supported in part by the National Grand Fundamental Research 973 Program of China No. 2014CB340401.

References

[1] K. Alexandros and H. Melanie. Model selection via meta-learning: a comparative study. *Int'l Journal on Artificial Intelligence Tools*, 10(04):525–554, 2001. 2

[2] S. Ali and K. A. Smith-Miles. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(1):173–186, 2006. 2

[3] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2014. 3

[4] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2392–2399. IEEE, 2012. 1, 3

[5] P. K. Chan and S. J. Stolfo. Experiments on multistrategy learning by meta-learning. In *Proc. of Int'l Conf. on Information and Knowledge Management*, pages 314–323. ACM, 1993. 2

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009. 7

[7] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013. 3

[8] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016. 1, 2, 3, 6

[9] M. Donini and F. Aiolli. Learning deep kernels in the space of dot product polynomials. *Machine Learning*, pages 1–25, 2016. 5

[10] D. G. Ferrari and L. N. De Castro. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Information Sciences*, 301:181–194, 2015. 2

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, June 2016. 1, 5, 7, 8

[12] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conf. on Computer Vision (ECCV)*, pages 630–645. Springer, 2016. 8

[13] T.-W. Hui, C. C. Loy, and X. Tang. Depth map super-resolution by deep multi-scale guidance. In *Proc. of European Conf. on Computer Vision (ECCV)*, pages 353–369. Springer, 2016. 3

[14] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 2

[15] V. Jain and S. Seung. Natural image denoising with convolutional networks. In *Advances in Neural Information Processing Systems*, pages 769–776, 2009. 1

[16] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016. 2, 3

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. of ACM Int'l Conf. on Multimedia*, pages 675–678. ACM, 2014. 6, 7, 8

[18] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014. 2

[19] B. Klein, L. Wolf, and Y. Afek. A dynamic convolutional layer for short range weather prediction. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 4840–4848. IEEE, 2015. 2

[20] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset, 2014. 7

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. 1, 3, 7, 8

[22] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits, 1998. 7

[23] C. Lemke, M. Budka, and B. Gabrys. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*, 44(1):117–130, 2015. 2

[24] R. Livni, S. Shalev-Shwartz, and O. Shamir. An algorithm for training polynomial networks. *arXiv preprint arXiv:1304.7045*, 2013. 5

[25] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. of IEEE Int'l Conf. on Computer Vision (ICCV)*, volume 2, pages 416–423. IEEE, 2001. 7

[26] T. M. Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980. 2

[27] H. Noh, P. H. Seo, and B. Han. Image question answering using convolutional neural network with dynamic parameter prediction. *arXiv preprint arXiv:1511.05756*, 2015. 2

[28] S.-K. Oh, W. Pedrycz, and B.-J. Park. Polynomial neural networks architecture: analysis and design. *Computers & Electrical Engineering*, 29(6):703–725, 2003. 5

[29] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015. 3

[30] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. 2

[31] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 3

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 3

[33] M. R. Smith, L. Mitchell, C. Giraud-Carrier, and T. Martinez. Recommending learning algorithms and their associated hyperparameters. In *Proc. of Int'l Conf. on Meta-learning and Algorithm Selection*, pages 39–40. CEUR-WS. org, 2014. 2

[34] K. A. Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *Proc. of IEEE Int'l Joint Conf. on Neural Networks (IJCNN)*, pages 4118–4124. IEEE, 2008. 2

[35] K. A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6, 2009. 2

[36] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. 6

[37] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013. 1

[38] I. V. Tetko, T. I. Aksenova, V. V. Volkovich, T. N. Kasheva, D. V. Filipov, W. J. Welsh, D. J. Livingstone, and A. E. Villa. Polynomial neural network for linear and non-linear model selection in quantitative-structure activity relationship studies on the internet. *SAR and QSAR in Environmental Research*, 11(3-4):263–280, 2000. 5

[39] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002. 2

[40] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010. 1

[41] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016. 5

[42] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *arXiv preprint arXiv:1608.03981*, 2016. 6, 7