A BRIEF REVIEW OF HYPERNETWORKS IN DEEP LEARNING

Vinod Kumar Chauhan[1]*, **Jiandong Zhou**[1], **Ping Lu**[1], **Soheila Molaei**[1] **and David A. Clifton**[1,2]
[1]Institute of Biomedical Engineering, University of Oxford, OX3 7DQ, UK
[2]Oxford-Suzhou Institute of Advanced Research (OSCAR), Suzhou, China

July 16, 2024

ABSTRACT

Hypernetworks, or hypernets for short, are neural networks that generate weights for another neural network, known as the target network. They have emerged as a powerful deep learning technique that allows for greater flexibility, adaptability, dynamism, faster training, information sharing, and model compression. Hypernets have shown promising results in a variety of deep learning problems, including continual learning, causal inference, transfer learning, weight pruning, uncertainty quantification, zero-shot learning, natural language processing, and reinforcement learning. Despite their success across different problem settings, there is currently no comprehensive review available to inform researchers about the latest developments and to assist in utilizing hypernets. To fill this gap, we review the progress in hypernets. We present an illustrative example of training deep neural networks using hypernets and propose categorizing hypernets based on five design criteria: inputs, outputs, variability of inputs and outputs, and the architecture of hypernets. We also review applications of hypernets across different deep learning problem settings, followed by a discussion of general scenarios where hypernets can be effectively employed. Finally, we discuss the challenges and future directions that remain underexplored in the field of hypernets. We believe that hypernetworks have the potential to revolutionize the field of deep learning. They offer a new way to design and train neural networks, and they have the potential to improve the performance of deep learning models on a variety of tasks. Through this review, we aim to inspire further advancements in deep learning through hypernetworks.

Keywords Hypernetworks · Deep learning · Neural Networks · Parameter generation · Weight generation

1 Introduction

Deep learning has revolutionized the field of artificial intelligence by enabling remarkable advancements in various domains, including computer vision [12], natural language processing [18], causal inference [11], and reinforcement learning [35]. Standard deep neural networks (DNNs) have proven to be powerful tools for learning complex representations from data. However, despite their success, standard DNNs remain restrictive in certain conditions. For example, once a DNN is trained, its weights as well as its architecture are fixed [55, 73], and any changes to weights or architecture require re-training the DNN. This lack of adaptability and dynamism restricts the flexibility of DNNs, making them less suitable for scenarios where dynamic adjustments or data adaptivity are required [24, 8]. DNNs generally have a large number of weights and need substantial amounts of data to optimize those weights [3]. This can be challenging in situations where large amounts of data are not available. For example, in healthcare, collecting sufficient data for rare diseases can be particularly difficult due to the limited number of patients available per year [76]. Finally, uncertainty quantification in DNNs' predictions is essential as it provides a measure of confidence, enabling better decision-making in high-stakes applications [13]. Existing uncertainty quantification techniques have limitations, such as the need to train multiple models [1], and uncertainty quantification is still considered an open problem [32]. Similarly, domain adaptation, domain generalization, adversarial defence, neural style transfer, and neural architecture

*Accepted to **Artificial Intelligence Review** (Springer Nature). Corresponding author: Vinod Kumar Chauhan (vinod.kumar@eng.ox.ac.uk)

search are important problems that remain unsolved, where hypernets can provide effective solutions as discussed in Section 4.

Hypernetworks (or hypernets in short) have emerged as a promising architectural paradigm to enhance the flexibility (through data adaptivity and dynamic architectures) and performance of DNNs. Hypernets are a class of neural networks that generate the weights/parameters of another neural network called the target/main/primary network, where both networks are trained in an end-to-end differentiable manner [24]. Hypernets complement existing DNNs and provide a new framework to train DNNs, resulting in a new class of DNNs called HyperDNNs (please refer to Section 2 for details). The key characteristics and advantages of hypernets that offer applications across different problem settings are discussed below.

(a) Soft weight sharing: Hypernetworks can be trained to generate the weights of multiple DNNs for solving related tasks [14, 49]. This is called soft weight sharing because, unlike hard weight sharing which involves shared layers among tasks (e.g., in multitasking), different DNNs are generated by a common hypernet through task conditioning. This helps share information among tasks and can be used for transfer learning or dynamic information sharing [14].

(b) Dynamic architectures: Hypernetworks can be used to generate the weights of a network with a dynamic architecture, where the number of layers or the structure of the network changes during training or inference. This can be particularly useful for tasks where the target network structure is not known at training time [24].

(c) Data-adaptive DNNs: Unlike standard DNNs whose weights are fixed at inference time, HyperDNNs can generate a target network customized to the needs of the data. In such cases, hypernets are conditioned on the input data to adapt to the data [69].

(d) Uncertainty quantification: Hypernets can effectively train uncertainty-aware DNNs by leveraging techniques like sampling multiple inputs from the noise distribution [33] or incorporating dropout within the hypernets themselves [15]. By generating multiple sets of weights for the main network, hypernets create an ensemble of models, each with different parameter configurations. This ensemble-based approach aids in estimating uncertainty in the model predictions, a crucial aspect for safety-critical applications like healthcare, where having a measure of confidence in predictions is essential.

(e) Parameter efficiency: HyperDNNs, i.e., DNNs trained with hypernets, can have fewer weights than the corresponding standard DNNs, resulting in weight compression [81]. This can be particularly useful when working with limited resources, limited data, or high-dimensional data and can result in faster training than the corresponding DNN [45].

Ha et. al [24] coined the term hypernets (also referred to as meta-networks or meta-models) and trained the target network and hypernet in an end-to-end differentiable way. However, the concept of learnable context-dependent weights was discussed even earlier, such as *fast weights* in [59, 60] and HyperNEAT [68]. Our discussion on hypernets focuses on neural networks generating weights for the target neural network due to their popularity, expressiveness, and flexibility [73, 12]. Recently, hypernets have gained significant attention and have produced state-of-the-art (SOTA) results across several deep learning problems, including ensemble learning [32], multitasking [71], neural architecture search [80], continual learning [49], weight pruning [40], Bayesian neural networks [17], generative models [17], hyperparameter optimization [41], information sharing [14], adversarial defence [69], and reinforcement learning (RL) [54] (please refer to Section 4 for more details).

Despite the success of hypernets across different problem settings, to the best of our knowledge, there is no review of hypernets to guide researchers about the developments and to help in utilizing hypernets. To fill this gap, we provide a brief review of hypernets in deep learning. We illustrate hypernets using an example and differentiate HyperDNNs from DNNs (Section 2). To facilitate better understanding and organization, we propose a systematic categorization of hypernets based on five distinct design criteria, resulting in different classifications that consider factors such as (i) input characteristics, (ii) output characteristics, (iii) variability of inputs, (iv) variability of outputs, and (v) the architecture of hypernets (Section 3). Furthermore, we offer a comprehensive overview of the diverse applications of hypernets in deep learning, spanning various problem settings (Section 4). By examining real-world applications, we aim to demonstrate the practical advantages and potential impact of hypernetworks. Additionally, we discuss some scenarios and pose direct questions to understand if we can apply hypernets to a given problem (Section 5). Finally, we discuss the challenges and future directions of hypernet research (Section 6). This includes addressing initialization, stability, and complexity concerns, as well as exploring avenues for enhancing the theoretical understanding and uncertainty quantification of DNNs. By providing a comprehensive review of hypernetworks, this paper aims to serve as a valuable resource for researchers and practitioners in the field. Through this review, we hope to inspire further advancements in deep learning by leveraging the potential of hypernets to develop more flexible, high-performing models.

Contributions: This review paper makes the following key contributions:

- To the best of our knowledge, we present the first review on hypernetworks in deep learning, which have shown impressive results across several deep learning problems.
- We propose categorizing hypernets based on five design criteria, leading to different classifications of hypernets, such as based on inputs, outputs, variability of inputs and outputs, and architecture of hypernets.
- We present a comprehensive overview of applications of hypernetworks across different problem settings, such as uncertainty quantification, continual learning, causal inference, transfer learning, and federated learning, and summarize our review, as per our categorization, in a table (Table 2).
- We explore broad scenarios for hypernet applications, drawing from existing use cases and hypernet characteristics. This exploration aims to equip researchers with actionable insights into when to leverage hypernets in their problem setting.
- Finally, we identify the challenges and future directions of hypernetwork research, including initialization, stability, scalability, and efficiency concerns, and the need for theoretical understanding and interpretability of hypernetworks. By highlighting these areas, we aim to inspire further advancements in hypernetworks and provide guidance for researchers interested in addressing these challenges.

The rest of the paper is organized as follows: Section 2 provides a comprehensive background on hypernets, while Section 3 introduces a novel categorization scheme for hypernets. The diverse applications of hypernets across various problems are discussed in Section 4, followed by an exploration of specific scenarios where hypernets can be effectively employed in Section 5. Addressing challenges and delineating future research directions is the focus of Section 6, and finally, the concluding remarks are discussed in Section 7.

2 Background

In this section, we discuss and differentiate the workings of standard deep neural networks (DNNs) and DNNs trained with hypernetworks, referred to as HyperDNNs, using a generic example. Fig. 1 illustrates the structural differences and gradient flows in DNNs and HyperDNNs. Both solve the same problem using the same DNN architecture at inference time. However, differences exist in their training processes, specifically in gradient flow and weight optimization, making hypernets an alternative way of training DNNs.

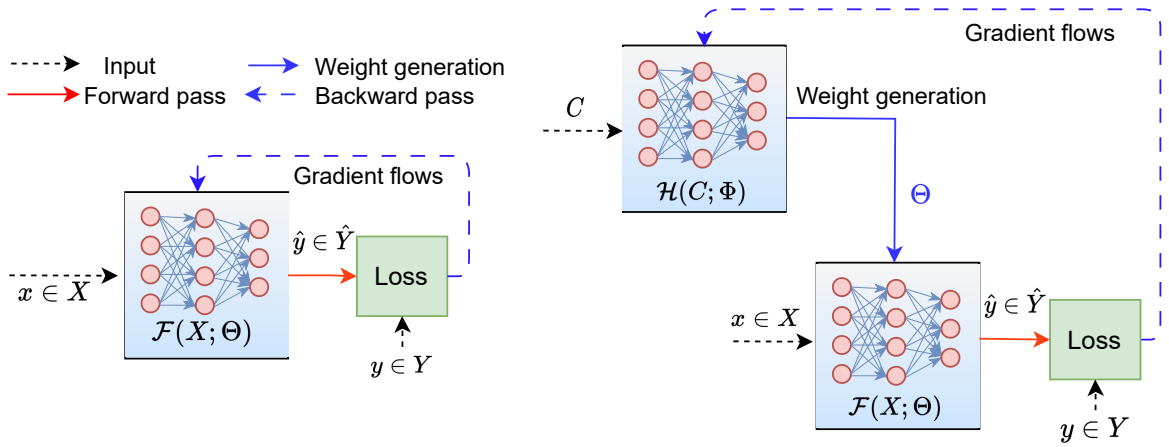

Figure 1: An overview of the architectures and gradient flows for a standard DNN $\mathcal{F}(X;\Theta)$ and the same DNN implemented with hypernets, referred to as HyperDNN $\mathcal{F}(X;\Theta) = \mathcal{F}(X;\mathcal{H}(C;\Phi))$. For the DNN, gradients flow through the DNN, and DNN weights Θ are learned during training. For the HyperDNN, gradients flow through the hypernet, and hypernet weights Φ are learned during training to produce DNN weights Θ as outputs.

Let us denote a dataset using X, Y to solve a general task $\mathcal{T}$, where X is a matrix of features and Y is a vector of labels, and $x \in X$ denotes one data point and $y \in Y$ is the corresponding label. Let a DNN be denoted as a function $\mathcal{F}(X;\Theta)$, where X denotes the inputs and Θ represents the weights of the DNN. During the forward pass, inputs $x \in X$ pass through the layers of $\mathcal{F}$ to produce predictions $\hat{y} \in \hat{Y}$, which are then used along with true labels $y \in Y$ to calculate an objective function that measures the discrepancy between actual values and the values predicted by the

model using a loss function $\mathcal{L}(Y, \hat{Y})$. During the backward pass, DNNs typically use backpropagation to propagate the error backwards through the layers and calculate gradients of $\mathcal{L}$ with respect to Θ. Optimization algorithms, such as Adam [30], use these gradients to update the weights. At the end of the training, we receive optimized weights Θ that are used at inference time in the DNN $\mathcal{F}(X; \Theta)$ to make predictions with the test data for solving task $\mathcal{T}$. Thus, in standard DNNs, Θ are the learnable weights.

Hypernets provide an alternative way of learning weights Θ of the DNN $\mathcal{F}(X; \Theta)$ to solve task $\mathcal{T}$, where Θ are not directly learned but are generated by another neural network. In this framework, we solve the same task using the same DNN architecture but with a different training approach. Let a hypernet be denoted as $\mathcal{H}(C; \Phi)$ which generates the task-specific weights of the DNN $\mathcal{F}(X; \Theta)$, where C is a task-specific context vector that acts as input to $\mathcal{H}$ and Φ are weights of the hypernet $\mathcal{H}$. That is, $\Theta = \mathcal{H}(C; \Phi)$ where Φ are the only learnable weights in the overall architecture. The context vector C can be generated from the data [2], sampled from a noise distribution [33], or correspond to task identity/embedding [4]. During the forward pass, a task-specific context vector C is passed to the hypernet $\mathcal{H}$ which generates weights Θ for the DNN $\mathcal{F}$. Then, like a standard DNN, an input $x \in X$ is passed through the DNN $\mathcal{F}$ to predict the output Y, and the loss is calculated as $\mathcal{L}(Y, \hat{Y})$. However, during the backward pass, the error is backpropagated through the hypernet $\mathcal{H}$ and gradients of $\mathcal{L}$ are calculated with respect to the weights of the hypernet Φ. The learning algorithm optimizes Φ to generate Θ so that performance on the target task $\mathcal{T}$ is optimized. At test time, Θ generated from the optimized hypernet $\mathcal{H}$ are used in the DNN $\mathcal{F}(X; \Theta)$ to make predictions with the test data for solving task $\mathcal{T}$. The optimization problems for the standard DNN and the HyperDNN can be written as follows (ignoring regularization terms for simplicity):

$$\text{DNN:} \quad \min_{\Theta} \ \mathcal{F}(X; \Theta), \qquad \text{HyperDNN:} \quad \min_{\Phi} \ \mathcal{F}(X; \Theta) = \mathcal{F}(X; \mathcal{H}(C; \Phi)). \tag{1}$$

Thus, DNNs learn their weights[2] directly from the data, while in HyperDNNs the weights of the hypernet are learned, and the weights of the DNN are generated by the hypernet. For a specific example of a comparison of DNN and HyperDNN architectures and their workings, please refer to our work in causal inference [14].

As discussed in Section 1, training a DNN with a hypernet, i.e., HyperDNN presents several advantages over directly training a DNN. However, these advantages are application-specific and cannot be generalized across all tasks or applications. For instance, a key feature of hypernets is soft-weight sharing, which enables information sharing among related components. This information sharing is particularly valuable in settings with limited data, leading to performance improvements for HyperDNNs in such scenarios. In general, HyperDNNs are beneficial for applications with limited data, problems requiring data-adaptive networks, dynamic network architectures, parameter efficiency, and uncertainty quantification. A detailed discussion of scenarios where HyperDNNs can be useful is provided in Section 5.

In general, if a task can be solved using standard DNNs, it is advisable to use them instead of hypernets. As depicted in Figure 1, HyperDNNs require an additional DNN to solve the same task. Despite the advantages offered by hypernets, this additional DNN introduces complexities in training and implementing HyperDNNs. For example, the initialization of HyperDNNs is more challenging than DNNs because the weights of the target network are generated at the output layer of the hypernet. Classical initialization techniques do not guarantee that the weights of the target network are initialized within the same range. However, adaptive optimizers, such as Adam [30], can mitigate this issue to some extent. Another significant challenge with HyperDNNs is their scalability. Since the weights of the target network are generated at the output layer of the hypernet, this approach can present difficulties when dealing with large target networks. Scalability issues can be managed using various weight generation strategies. Therefore, when using HyperDNNs, practitioners should consider employing adaptive optimizers, implementing different weight generation strategies, and using approaches to stabilize training, such as spectral norms. For a detailed discussion on the challenges associated with HyperDNNs, please refer to Section 6.

3 Categorization of Hypernetworks

In this section, we propose to categorize the hypernetworks based on five design criteria, as depicted in Fig. 2 and as given below:

(a) Input-based, i.e., what kind of input is taken by the hypernetworks to generate the target neural network weights?

(b) Output-based, i.e., how are the outputs, that is, the target weights generated?

(c) Variability of inputs, i.e., are the inputs of hypernet fixed?

[2]we have used weights and parameters interchangeably

(d) Variability of outputs, i.e., does the target network have a fixed number of weights? and

(e) Architecture-based, i.e., what kind of architecture does hypernet use to generate the target weights?

We discuss these in the following subsections. One can categorize hypernets based on the architecture of the target network but that is not considered because hypernets mostly generate target weights independent of their architecture.

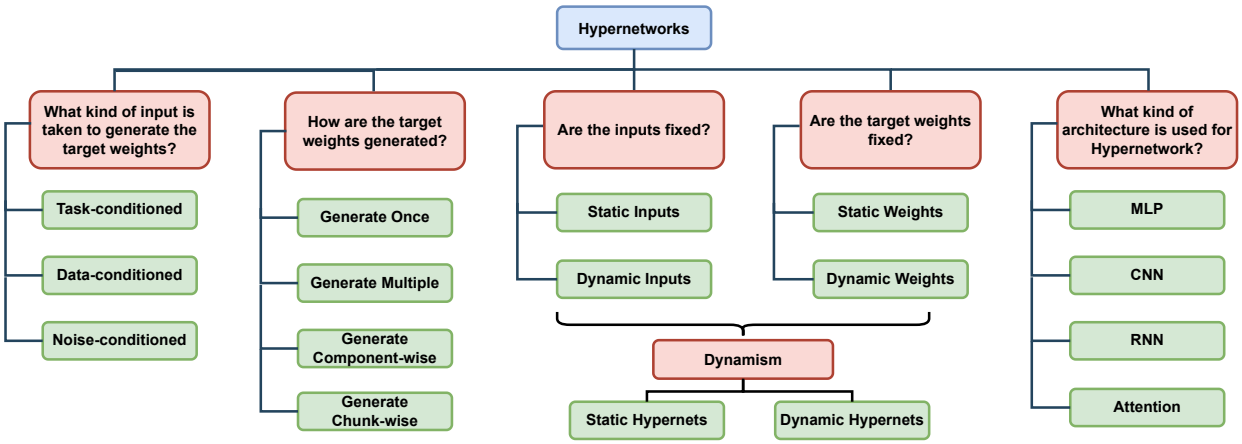

Figure 2: Proposed categorization of hypernets based on five design criteria.

3.1 Input-based Hypernetworks

Hypernetworks take a context vector as an input and generate weights of the target DNN as output. Depending on what context vector is used, we can have the following types of hypernetworks.

Task-conditioned hypernetworks: These hypernetworks take task-specific information as input. The task information can be in the form of task identity/embedding, hyperparameters, architectures, or any other task-specific cues. The hypernetwork generates weights that are tailored to the specific task. This allows the hypernet to adapt its behavior accordingly and allows information sharing, through soft weight sharing of hypernets, among the tasks, resulting in better performance on the tasks. For example, Chauhan et al. [14] applied hypernets to solve treatment effects estimation problem in causal inference that uses an identity or embedding of potential outcome (PO) functions to generate weights corresponding to the PO function. The hypernetworks enabled dynamic end-to-end inter-treatment information sharing among treatment groups and helped to calculate reliable treatment estimates in observational studies with limited-size datasets. Similarly, task-conditioned hypernets have been used to solve other problems, including multitasking [45], natural language processing (NLP) [24], and continual learning [49].

Data-conditioned hypernetworks: These hypernetworks are conditioned on the data that the target network is being trained on. The hypernetwork generates weights based on the characteristics of the input data. This enables the neural network to dynamically adjust its behavior based on the specific input pattern or features, leading to more flexible and adaptive models, and resulting in better generalization to unseen data. For example, Alaluf et al. [2] applied hypernets for image editing where the input of hypernet is based on the input images and initial approximation of reconstruction to generate modulations to the weights of the pre-trained generator. Similarly, data-conditioned hypernets have been used to solve other problems, such as adversarial defence [69], knowledge graphs learning [5] and shape learning [39].

Noise-conditioned hypernetworks: These hypernetworks are not conditioned on any input data or task cues, but rather on randomly sampled noise. This makes them more general-purpose and helps in predictive uncertainty quantification for DNNs, but it also means that they may not perform as well as task-conditioned or data-conditioned hypernetworks on multiple tasks or datasets. For example, Krueger et al. [33] applied hypernetworks to approximate Bayesian inference in the DNNs and evaluated the approach for active learning, model uncertainty, regularization, and anomaly detection. Similarly, noise-conditioned hypernets have been used to solve other problems, such as manifold learning [17] and uncertainty quantification [53].

These different types of conditioning enable hypernetworks to enhance the flexibility (through adaptability and dynamic architectures), and performance of deep learning models in various contexts. The specific type of hypernetwork that is used will depend on the specific task or application. For example, task-conditioned hypernets are suitable for information sharing among multiple tasks, data-conditioned hypernets are suitable to deal with conditions where DNN need to adapt to input data, and noise-conditioned hypernets are suitable for uncertainty quantification in the predictions.

3.2 Output-based Hypernetworks

Based on the outputs of hypernets, i.e., weight generation strategy, we classify hypernetworks according to whether all weights are generated together or not. This classification of hypernetworks is important because it controls the scalability and complexity of the hypernetworks, as typically DNNs have a large number of weights, and producing all of them together can make the size of the last layer of hypernets large. So, there are ways to manage the complexity of the hypernets that lead to different strategies of weight generation, as discussed below. It is possible to train HyperDNN with fewer weights than the target DNN – this is called weight compression [81]. We compared and summarized the characteristics of various weight generation strategies in Table 1. The first column represents the considered characteristic for comparison, while the following three columns correspond to three different weight generation strategies. The values in each row indicate whether a particular weight generation strategy provides the specified feature or not.

Generate Once: These hypernetworks generate weights of the entire target DNN altogether. This approach uses all the generated weights, and weights of each layer are generated together, unlike the other weight generation strategies. However, this weight generation approach is not suitable for large target networks because that can lead to complex hypernets. For example, Shamsian et al. [63], Galanti and Wolf [22], Zhang et al. [80] used generate once weight generation.

Generate Multiple: These hypernetworks have multiple heads for producing weights (sometimes referred to as split/multi-head hypernets) and this weight generation approach can complement the other approaches. This simplifies the complexity and reduces the number of weights required in the last layer of the hypernets by the number of head times. This approach does not need additional embeddings, and in general, uses all the generated weights, unlike component-wise and chunk-wise weight generation approaches where some weights remain unused. For example, Beck et al. [6], Rezaei-Shoshtari et al. [54], Chauhan et al. [14] used generate multiple strategy to produce target weights.

Generate Chunk-wise: Chunk-wise hypernetworks generate weights of the target network in chunks. This can lead to not using some of the generated weights because the weights are generated as per the chunk size, which may not match the layer sizes. If the chunk size is smaller than the layer size, then all the weights of a layer may not be generated together. Moreover, these hypernets need additional embeddings to distinguish different chunks and to produce specific weights for the chunks. However, overall chunk-wise weight generation leads to reducing complexity and improving the scalability of hypernets. For example, Chauhan et al. [14], Oswald et al. [49] used chunk-wise weight generation.

Generate Component-wise: Component-wise weights generation strategy generates weights for each individual component (such as layer or channel) of the target model separately. This is helpful in generating specific weights because different layers or channels represent different features or patterns in the network. However, similar to the chunk-wise approach, component-wise hypernets need an embedding for each component to distinguish among different components and produce weights specific to that component. They also help to reduce the complexity and improve the scalability of hypernets. Since the weights are generated as per the size of the largest layer so this weight generation approach can lead to not using some of weights in smaller layers. This strategy can be seen as a special case of a chunk-wise weight generation approach, where one chunk is equal to the size of one component. For example, Zhao et al. [81], Alaluf et al. [2], Mahabadi et al. [43] used component-wise weight generation.

By classifying hypernetworks based on their weight generation strategy, we can make informed choices that may help control the scalability and complexity of the hypernetworks effectively. Each type of weight generation strategy offers unique benefits and considerations based on the specific characteristics and requirements of the task at hand. The comparative study of characteristics of different weight generation approaches is summarized in Table 1.

3.3 Variability of Inputs

We can categorize hypernets based on the variability of the inputs. We have two classes, static inputs and dynamic inputs, as discussed below.

Static Inputs: If the inputs are predefined and are fixed then the hypernet is called static with respect to the inputs. For example, multitasking [43] has fixed number of tasks leading to fixed number of inputs. It is to be noted that here fixed input only means fixed tasks identities, however hypernets can learn embeddings for different tasks.

Dynamic Inputs: If the inputs change and generally are dependent on data on which the target network is trained, then the hypernet is called dynamic with respect to the inputs. Dynamic inputs help hypernetworks to introduce a new level of adaptability by dynamically generating the weights of the target network. This dynamic weight generation enables hypernetworks to respond to input-dependent context and adjust their behavior accordingly. By generating network weights based on specific inputs, hypernetworks can capture intricate patterns and dependencies that may vary across different instances of data. This adaptability leads to enhanced model performance, especially in scenarios with

Table 1: Comparison of different weight generation strategies, i.e., output-based hypernetworks.

Weight-generation→ /Characteristics↓	**Generate Once**	**Generate Component-wise**	**Generate Chunk-wise**	**Generate Multiple**
Weight generation	Generates all target weights together	Generates target weights for one component at a time	Generates target weights in chunks	Complements all other weight generation strategies so can generate weights like any of the other
Efficient use of generated weights	Yes	No as some weights can stay unused	Yes	Depends on the base strategy
Are all weights of a layer generated together	Yes	Yes	No	Depends on the base strategy
The complexity of output space	Highest	Lower than generate once	Lowest	Can further improve Chunk-wise generation
The complexity of input space	Lowest	More complex than 'generate once' but lower than chunk-wise generation if the number of target layers is fewer than the number of chunks	Highest (assuming the number of chunks is more than the number of target layers)	Does not have any effect on input space complexity

complex and evolving data distributions [75]. Thus, dynamic input-based hypernets help in domain adaptation [75], density estimation [28] and knowledge graph learning [5] etc.

This can be seen as a super categorization over input-based hypernets where task-conditioned hypernets fall in the static inputs category while random-noise and data-conditioned hypernets fall in the dynamic category. Both the categories have their own advantages as static inputs help in information sharing [14], transfer learning [49], and are suitable where we have multiple tasks to solve [63]. On the other hand, dynamic inputs give hypernets adaptability to new conditions unknown during training [5].

3.4 Variability of Outputs

When classifying hypernetworks based on the nature of the target network's weights, we can categorize them into two types, static outputs or dynamic outputs, as discussed below.

Static Outputs: If weights of the target network are fixed in size, then the hypernet is called static with respect to the outputs. In this case, the target network is also static. For example, Pan et al. [50], Szatkowski et al. [70] produce static weights.

Dynamic Outputs: If weights of the target network are not fixed, i.e., the architecture varies in size, then the hypernet is called dynamic with respect to the outputs, and the target network is also a dynamic network as it can have different architecture depending on the input of the hypernet. The dynamic weights can be generated, mainly, in two situations, first when the hypernet architecture is dynamic, e.g., Ha et al. [24] used recurrent neural network (RNN) to propose HyperRNN based on non-shared weights. Second, the dynamic weights can be generated when the inputs are dynamic, i.e., hypernet adapts as per the input data, e.g., Littwin and Wolf [39] applied convolutional neural network (CNN) based hypernet to generate dynamic weights for shape learning from an image of a shape. Similarly, Peng et al. [51], Li et al. [36] also produce dynamic weights.

3.5 Dynamism in Hypernetworks

This is a super categorization of Subsection 3.3 and 3.4 into broader category based on the dynamism in inputs or outputs of the hypernets, as discussed below.

Static Hypernets: If input of a hypernet is fixed, i.e., predefined and number of weights produced by hypernet for the target network are fixed, i.e., the architecture is fixed, then the hypernet is called as a static hypernet. This kind of hypernets work with predefined inputs, e.g., task identities, which can be learned as embeddings, but the tasks being

solved remain same. For example, heterogeneous treatment effect estimation [14] where number of treatment groups or potential outcome functions are fixed, and architecture of the target network (in this case potential outcome functions) is also fixed.

Dynamic Hypernets: If input of a hypernet is based on input of target network, i.e., input data, or number of weights produced by hypernet for the target network are variable, i.e., the architecture is dynamic, then the hypernet is called as a dynamic hypernet. For example, Sendera et al. [61] applied data-conditioned hypernet to few-shot learning by combining kernels and hypernets. The kernels were used to extract support information from data of different tasks that act as input to the hypernet which generates weights for the target task. Zhang et al. [80] applied hypernetworks for neural architecture search where they modeled neural architectures of a DNN as graph and used them as input to hypernet to generate the target network weights. So, the target network has variable architecture, and is a dynamic hypernet based on the dynamic outputs.

3.6 Architecture of Hypernetworks

In the categorization of hypernetworks based on their architectures, we can classify them into four major types: multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention-based networks, as given below.

MLPs: MLP based hypernetworks employ a dense and fully connected architecture, allowing every input neuron to connect with every output neuron. This architecture enables a comprehensive weight generation process by considering the entire input information, e.g., [14].

CNNs: CNN hypernetworks, on the other hand, leverage convolutional layers to capture local patterns and spatial information. These hypernetworks excel in tasks involving spatial data, such as an image or video analysis, by extracting features from the input and generating weights or parameters accordingly, e.g., Nirkin et al. [47] employed MLP to implement hypernets.

RNNs: RNN hypernetworks incorporate recurrent connections in their architecture, facilitating feedback loops and sequential information processing. They dynamically generate weights or parameters based on previous states or inputs, making them well-suited for tasks involving sequential data, such as natural language processing or time series analysis, e.g., Ha et al. [24] employed RNN to implement hypernets.

Attention Attention-based hypernetworks incorporate attention mechanisms [73] into their architecture. By selectively focusing on relevant input features, these hypernetworks generate weights for the target network, allowing them to capture long-range dependencies and improve the quality of generated outputs, e.g., Volk et al. [75] employed attention to implement hypernets.

Each type of architecture has its own strengths and applicability, enabling hypernetworks to adapt and generate weights in a manner that aligns with the specific characteristics and demands of the target network and the data being processed.

4 Applications of Hypernetworks

Hypernetworks have demonstrated their effectiveness and versatility across a wide range of domains and tasks in deep learning. In this section, we discuss some of the important applications[3] of hypernetworks and highlight their contributions to advancing the SOTA in these areas. We summarize the applications of hypernets as per our proposed categorization and also provide links to code repositories for the benefit of the researchers, wherever available, in Table 2.

Continual Learning: Continual learning, also known as lifelong learning or incremental learning, is a machine learning paradigm that focuses on the ability of a model to learn and adapt continuously over time, in a sequential manner, without forgetting previously learned knowledge. Unlike traditional batch learning, which assumes static and independent training and testing sets, continual learning deals with dynamic and non-stationary data distributions, where new data arrives incrementally, and the model needs to adapt to these changes while retaining previously acquired knowledge. The challenge in continual learning lies in mitigating *catastrophic forgetting*, which refers to the tendency of a model to forget previously learned information when it is trained on new data. To address this, various strategies have been proposed, including regularization techniques, rehearsal methods, dynamic architectures, and parameter isolation. Oswald et al. [49] modeled each incrementally obtained dataset as a task and applied task-conditioned hypernets for continual learning – this helped to share information among tasks. To address the catastrophic forgetting issue, they

[3]We have explored 50 important papers (arranged by publication year) while considering at least one application in each distinct problem setting. This is not an exhaustive list and it is possible that we may have missed important references.

proposed a regularizer for rehearsing task-specific weight realizations rather than the data from previous tasks. They achieved SOTA results on benchmarks and empirically showed that the task-conditioned hypernets have a long capacity to retain memories of previous tasks. Similarly, Huang et al. [29], Ehret et al. [20] applied task-conditioned hypernets to continual learning in reinforcement learning (RL).

Federated Learning: Federated Learning is a decentralized approach to machine learning where the training process is distributed across multiple devices or edge devices, without the need to centralize data in a single location. In this paradigm, each device or edge node locally trains a model using its own data, and only the model updates, rather than the raw data, are shared and aggregated on a central server. This enables collaborative learning while preserving data privacy and security. It also reduces communication costs and latency, making it suitable for scenarios with limited bandwidth or intermittent connectivity. Shamsian et al. [63] modeled each client machine as a task and applied task-conditioned hypernets to federated learning problem. They trained a central hypernet to generate the weights for the client models. This allowed information sharing across different clients while making the hypernet size independent of communication cost, as hypernet weights are never transmitted. The hypernet-based federated learning achieved the SOTA results and also showed better generalization to new clients whose distributions were different than the existing clients. Litany et al. [37] extended this work to heterogeneous clients, i.e., clients with different neural architectures, using graph hypernetworks [80].

Table 2: Important applications of hypernetworks, arranged by ascending publication year, and their categorization based on Input: (i) task-conditioned, (ii) noise-conditioned, and (iii) data-conditioned; output, i.e., weight generation: (i) generate once, (ii) generate component-wise, (iii) generate chunk-wise, and (iv) generate multiple; Input variability: (i) static inputs, and (ii) dynamic inputs; Output variability: (i) static weights, and (ii) dynamic weights; and architecture of hypernets (SN: Serial Number, Ref.: Reference, DL: Deep Learning, RL: Reinforcement learning).

SN	Ref.	DL Problem	Input			Output				Input Var.		Out Var.		Architecture	Code
			(i)	(ii)	(iii)	(i)	(ii)	(iii)	(iv)	(i)	(ii)	(i)	(ii)		
1	[24]	Image classification, NLP	✓				✓				✓	✓	✓	RNN, MLP	
2	[33]	Uncertainty quantification		✓		✓					✓	✓		MLP	
3	[69]	Adversarial defence			✓		✓				✓		✓	MLP	
4	[41]	Hyperparameter optimization	✓			✓					✓	✓		MLP	
5	[8]	Neural architecture search	✓					✓			✓		✓	CNN	Link
6	[50]	Spatio-temporal learning			✓	✓					✓	✓		MLP, RNN, CNN	
7	[80]	Neural architecture search	✓			✓					✓	✓		MLP	
8	[17]	Manifold learning		✓			✓				✓		✓	CNN	
9	[53]	Uncertainty quantification		✓			✓		✓		✓	✓		GAN	
10	[40]	Weight pruning	✓			✓					✓	✓		MLP	Link
11	[5]	Knowledge graphs learning			✓		✓				✓	✓		MLP	Link
12	[39]	Shape learning			✓				✓		✓		✓	CNN	Link
13	[32]	Uncertainty quantification			✓	✓					✓	✓		MLP	
14	[31]	Image processing			✓				✓		✓	✓		CNN	
15	[49]	Continual learning, transfer learning	✓			✓	✓	✓		✓		✓		MLP	Link
16	[81]	Few-shot learning	✓					✓		✓		✓		MLP	
17	[22]	Complexity of NN			✓	✓					✓	✓		MLP	
18	[36]	Weight pruning	✓				✓				✓		✓	MLP	Link
19	[51]	Neural architecture search	✓				✓				✓		✓	CNN	Link
20	[45]	Pareto-Front Learning (multi-tasking, fairness, image segmentation)	✓						✓	✓		✓		MLP	Link
21	[63]	Federated Learning	✓			✓				✓		✓		MLP	Link
22	[47]	Semantic segmentation			✓		✓		✓		✓		✓	CNN	Link
23	[43]	Multitasking, NLP, language model	✓				✓			✓		✓		MLP	Link
24	[58]	RL	✓			✓				✓		✓		MLP	Link
25	[29]	Continual RL	✓			✓				✓		✓		MLP	Link
26	[64]	Density estimation	✓				✓			✓		✓		MLP	
27	[44]	Neural image enhancement			✓	✓					✓		✓	MLP	
28	[26]	Continual learning	✓			✓				✓		✓		MLP	
29	[20]	Continual learning	✓					✓		✓		✓		MLP	Link
30	[34]	Adaptation of neural network architectures			✓	✓					✓		✓	MLP	
31	[46]	Network compression	✓			✓					✓	✓		MLP	
32	[7]	Internal learning (computer vision)			✓				✓		✓	✓		CNN	Link
33	[16]	Learning differential equations	✓			✓					✓	✓		MLP	
34	[2]	Image editing			✓		✓				✓	✓		CNN	Link
35	[75]	Domain adaptation, NLP	✓		✓	✓					✓	✓		Attention	Link
36	[48]	Autonomous driving			✓				✓		✓		✓	Attention, RNN, CNN, MLP	
37	[78]	NLP	✓		✓		✓			✓		✓	✓	MLP	Link
38	[52]	Domain generalization	✓			✓				✓		✓		MLP	
39	[66]	3D point cloud processing			✓	✓					✓	✓		MLP	
40	[19]	Image processing			✓	✓					✓	✓		CNN	Link
41	[79]	Few-shot learning			✓	✓					✓	✓		CNN	Link

Table 2: Important applications of hypernetworks, arranged by ascending publication year, and their categorization based on Input: (i) task-conditioned, (ii) noise-conditioned, and (iii) data-conditioned; output, i.e., weight generation: (i) generate once, (ii) generate component-wise, (iii) generate chunk-wise, and (iv) generate multiple; Input variability: (i) static inputs, and (ii) dynamic inputs; Output variability: (i) static weights, and (ii) dynamic weights; and architecture of hypernets (SN: Serial Number, Ref.: Reference, DL: Deep Learning, RL: Reinforcement learning).

SN	Ref.	DL Problem	Input			Output				Input Var.		Out Var.		Architecture	Code
			(i)	(ii)	(iii)	(i)	(ii)	(iii)	(iv)	(i)	(ii)	(i)	(ii)		
42	[14]	Treatment effects estimation	✓			✓	✓	✓	✓	✓		✓		MLP	
43	[6]	Meta-RL	✓						✓	✓		✓		MLP	
44	[54]	Zero-shot RL	✓						✓		✓	✓		MLP	Link
45	[70]	Sound representation			✓	✓					✓	✓		CNN	
46	[28]	Density estimation			✓	✓					✓	✓		CNN	
47	[9]	Quantum computing	✓			✓					✓	✓		MLP	Link
48	[57]	Neural style transfer			✓	✓					✓		✓	MLP	
49	[21]	Camera pose localization			✓		✓				✓	✓		Attention, MLP	Link
50	[77]	Knowledge distillation, visualization	✓			✓					✓	✓		MLP	

Few-shot Learning: Few-shot learning is a sub-field of machine learning that focuses on training models to learn new concepts or tasks with only a limited number of training examples. Unlike traditional machine learning approaches that typically require large amounts of labeled data for each task, few-shot learning aims to generalize knowledge from a small support set of labeled examples to classify or recognize new instances. To address the practical difficulties of existing techniques to operate in high-dimensional parameter spaces with extremely limited-data settings, Rusu et al. [56] applied data-conditioned hypernets. They employed encoder-decoder based hypernet which learns a data-dependent latent generative representation of model parameters that shares information between different tasks through soft weight sharing of hypernets. They also achieved SOTA results and showed that the proposed technique can capture uncertainty in the data. Sendera et al. [61] also applied data-conditioned hypernet to few-shot learning by combining kernels and hypernets. The kernels were used to extract support information from data of different tasks that act as input to the hypernet which generates weights for the target task. Similarly, Zhao et al. [81], Zięba [82], Sendera et al. [62] also applied hypernets, and utilized soft weight sharing, for few-shot learning.

Manifold Learning: Manifold learning is a sub-field of machine learning that focuses on capturing the underlying structure or geometry of high-dimensional data in lower-dimensional representations or manifolds. It aims to uncover the intrinsic relationships and patterns within the data by mapping it to a lower-dimensional space, enabling better visualization, clustering, or classification. Hypernetworks can be utilized in the context of manifold learning to enhance the representation learning process. By generating weights or parameters for the target network based on the input, hypernetworks can adaptively learn a manifold that captures the intricate data structure [63]. Deutsch et al. [17] applied noise-conditioned hypernetworks to map latent vectors for generating target network weights that generalize mode connectivity in loss landscape to higher dimensional manifolds.

AutoML: AutoML, short for Automated Machine Learning, refers to the development of algorithms, systems, and tools that automate various aspects of the machine learning pipeline, e.g., neural architecture search (NAS) and automated hyperparameter optimization. Zhang et al. [80] applied hypernetworks for NAS where they modeled neural architectures of a DNN as graph and used them as input to hypernet to generate the target network weights. They achieved about 10 times faster results than the SOTA. Similarly, Brock et al. [8], Peng et al. [51] present another example of application of hypernets to NAS, where they exploit soft weight sharing property of hypernets for information sharing among different architectures. For hyperparameter optimization, Lorraine and Duvenaud [41] applied hypernets that take hyperparameters of the target network as input and generate optimal weights for the target network, and hence perform joint training for target network parameters and hyperparameters which are otherwise trained in nested optimization loops. The authors proved the efficacy of the proposed technique against the SOTA to train thousands of hyperparameters.

Pareto-front Learning: Pareto-front learning, also known as multi-objective optimization, is a technique that addresses problems with multiple conflicting objectives, e.g., multitasking has multiple tasks that may have conflicting gradients. It aims to find a set of solutions that represent the trade-off among different objectives, rather than a single optimal solution. In Pareto-front learning, the goal is to identify a set of solutions that cannot be improved in one objective without sacrificing performance in another objective. These solutions are referred to as Pareto-optimal or non-dominated solutions and lie on the Pareto-front, which represents the best possible trade-off between objectives. Navon et al. [45] applied hypernets to learn the entire Pareto-front, which at inference time takes a preferential point on the Pareto-front and generates Pareto-front weights for the target network whose loss vector is in the direction of the ray. They showed that the proposed hypernets are computationally very efficient as compared with the SOTA and can scale to large models, such as ResNet18. This work is further extended in Hoang et al. [27], where hypernet generates multiple solutions, and Tran et al. [72], which consider completed scalarization functions in the Pareto-front learning.

Domain Adaptation: Domain adaptation refers to the process of adapting a machine learning model trained on a source domain to perform well in a different target domain. It is a crucial challenge in machine learning when there is a shift or discrepancy between the distribution of the source and the target data. Hypernets can play a valuable role in domain adaptation by dynamically generating or adapting model parameters, architectures, or other components to effectively handle domain shifts. For example, Volk et al. [75] were the first to propose hypernets for domain adaptation. They used data-conditioned hypernets where examples from the target domains are used as input to hypernet that generates weights for the target network. This gives hypernets ability to learn and share information from existing domains with target domain through shared training.

Causal Inference: Causal inference is a field of study that focuses on understanding and estimating causal relationships between variables. It aims to uncover the cause-and-effect relationships within a system by leveraging observational or experimental data. Causal inference is particularly important when inferring the impact of treatments/ interventions/ policies on outcomes of interest. Recently, we were the first to apply hypernets to heterogeneous treatment effects (HTE) estimation problem [14]. We applied task-conditioned hypernets where each potential outcome (PO) function is considered as a task. Embeddings of PO functions are used as input to hypernet that generates parameters for the corresponding PO function, i.e., factual and counterfactual models. Based on soft weight sharing of hypernets, this work presents the first general mechanism to train HTE learners that enables end-to-end inter-treatment information sharing among the PO functions and helps to get reliable estimates, especially with limited-size observational data. The proposed framework also incorporates dropout in the hypernet that allows to generate multiple sets of parameters for the PO functions and helps in uncertainty quantification.

Uncertainty Quantification: Uncertainty quantification is a critical aspect of deep learning and decision-making that involves estimating and understanding the uncertainty associated with model predictions or outcomes. It provides a measure of confidence or reliability in the predictions made by a model, particularly in situations where the model encounters unseen or uncertain data. Hypernets can effectively train uncertainty aware DNNs by leveraging techniques like sampling multiple inputs from the noise distribution [33] or incorporating dropout within the hypernets themselves [15]. By generating multiple sets of weights for the main network, hypernets create an ensemble of models, each with different parameter configurations. This ensemble-based approach aids in estimating uncertainty in the model predictions. Krueger et al. [33] proposed Bayesian hypernets that take random noise as input to produce distributions over the weights of the target network and showed competitive performance for uncertainty. Ratzlaff and Fuxin [53] also applied noise-conditioned hypernets for uncertainty quantification and showed that the proposed technique provides a better estimate of uncertainty as compared to the ensemble learning technique. In addition, Chauhan et al. [15] used dropout in the task-conditioned hypernets to generate multiple sets of weights for the target network and thus helping to estimate uncertainty.

Adversarial Defence: Adversarial defence in deep learning refers to the techniques used to enhance the robustness and resilience of models against adversarial attacks. Adversarial attacks involve making carefully crafted perturbations to input data in order to deceive or mislead deep learning models [42]. By incorporating hypernetworks, models can enhance their ability to detect and defend against adversarial attacks by dynamically generating or adapting their weights or architectures. For example, Sun et al. [69] generated data-dependent adaptive convolution kernels to improve the robustness of CNNs against adversarial attacks and were successful in spontaneously detecting attacks generated by Gaussian noise, fast gradient sign methods, and black-box attack methods. The models developed with hypernets are highly adaptive and customized to the data. Similarly, Kristiadi et al. [32], Ratzlaff and Fuxin [53], Krueger et al. [33] also found noise-conditioned hypernets robust to adversarial examples as compared with the SOTA.

Multitasking: Multitasking refers to the capability of a model to perform multiple tasks or learn multiple objectives simultaneously. It involves leveraging shared representations and parameters across different tasks to enhance learning efficiency and overall performance. Hypernets can be applied in the context of multitasking to facilitate the joint learning of multiple tasks by dynamically generating or adapting the model's parameters or architectures. Specifically, we can train task-conditioned hypernets for multitasking where embedding of a task act as input to the hypernet that generates weights for the corresponding task. We can either generate entire model for each of the tasks or can only generate non-shared parts of a multitasking network. The hypernets facilitate such models to share information across different tasks as well as have specific personalized model for each task. For example, Mahabadi et al. [43] applied task-conditioned hypernets that share knowledge across the tasks as well as generate task-specific models and achieved benchmark results. Navon et al. [45] also studied task-conditioned hypernets for Pareto-front learning to address the conflicting gradients among different objectives and obtained impressive results on multitasking, including fairness and image segmentation.

Reinforcement Learning: Reinforcement Learning (RL) focuses on training agents to make sequential decisions in an environment to maximize a cumulative reward. RL operates through an interaction loop where the agent takes actions, receives feedback in the form of rewards, and learns optimal policies through trial and error. Hypernets can be used to

dynamically generate or adapt network architectures, model parameters, or exploration strategies in RL agents. By using a hypernetwork, the RL agent can effectively learn to customize its internal representations or policies based on the specific characteristics of the environment or task. For example, Sarafian et al. [58] applied hypernets to generate the building blocks of RL, i.e., policy networks and Q-functions, rather than using MLPs. They showed faster training and improved performance on different algorithms for RL and in meta-RL. Similarly, noise-conditioned hypernets are used in [74] to generate weights of each Bellman iteration with HyperRNN, and task-conditioned hypernets were used in RL for generalization across tasks [6], continual RL [29], and zero-shot learning [54].

Natural Language Processing: Natural language processing (NLP) is a sub-field of artificial intelligence that focuses on the interaction between computers and human language. It involves various tasks, such as language generation, sentiment analysis, machine translation, and question answering, among others. In the context of NLP, hypernets can be used to generate or adapt neural network architectures, tuning hyperparameters, for neural architecture search, and for transfer learning and domain adaptation etc. For example, Volk et al. [75] applied data-conditioned hypernet for out-of-distribution (OOD) generalization. They used T5 encoder-decoder framework to generate a unique signature for each example from different source domains. This signature acts as input to the hypernet and generates parameters for the target network – a dynamic and adaptive network. As discussed above, Mahabadi et al. [43] applied task-conditioned hypernets to fine-tune the pre-trained language models by generating weights for the bottleneck adapters. In the multitasking setting, they modeled task, adapter location and layer id as different tasks and used embedding of these tasks as input to the hypernet that helps in shared learning and achieving parameter efficiency.

Computer Vision: Computer vision focuses on enabling computers to understand and interpret visual information from images or videos. Computer vision algorithms aim to replicate human visual perception by detecting and recognizing objects, understanding their spatial relationships, extracting features, and making sense of the visual scene. Some applications of hypernets in computer vision are: Ha et al. [24], in their pioneering work, first applied task-conditioned hypernets for image classification, Alaluf et al. [2], Muller [44] applied data-conditioned hypernets, where image acts as input to hypernet, for image enhancement, and Ratzlaff and Fuxin [53] applied noise-conditioned hypernets for image classification. Data-conditioned hypernets are also applied to semantic segmentation in [47]. Some other applications of hypernets in computer vision are camera pose estimation [21], neural style transfer [57], image processing/editing [2], and neural image enhancement [44]. It is to be noted that computer vision is a vast subject and encompasses many problem settings discussed earlier so they can be used as such with change of domain related data or models. For example, hypernets developed for AutoML, domain adaption, continual learning, and federated learning etc. can be applied to computer vision problems as well.

The above applications of hypernets are not exhaustive and some other interesting areas where hypernets have produced the SOTA results are knowledge graph learning [5], shape learning [39], network compression [46], learning differential equations [16], 3D point cloud processing [65], speech processing [70], quantum computing [9], and knowledge distillation [77] etc. These applications demonstrate the wide-ranging potential of hypernetworks in deep learning, enabling adaptive and task-specific parameter generation for improved model performance and generalization.

5 When can we use Hypernets?

After discussing what a hypernet is, how it works, its different types, and its current applications, the most important question is when and where to utilize hypernets. This will help researchers and practitioners fully harness the benefits of this versatile technique in deep learning. One straightforward answer to the question, 'When can we use Hypernets?' is 'in all those application areas where it is already applied'. There is a long list of application areas where hypernets are already in use, and the reader's area of interest is likely covered. Based on the characteristics and applications of hypernets discussed above, we have generalized and formulated some questions/scenarios for readers to check if hypernets can be applied to a specific area/problem setting. If our answer is yes to any of the scenarios, then we can apply hypernets to the problem setting under consideration.

Are there any related components in the problem setting under consideration?
Here, a component can refer to a task, dataset, or neural network. This is one of the most important scenarios/questions, and several applications, as discussed above, fall under this scenario. If the answer to this question is yes, then we can employ task-conditioned hypernets to solve the problem under consideration, where task identity is used to generate the target network for the component. By conditioning on the component (task, dataset, or network), we can perform joint training of different components by exploiting the soft weight sharing of hypernets. This enables the hypernets to share information among components, leading to improved performance [14]. Thus, sharing information is the key to achieving better results for related components. The question can be reformulated as, 'Do we need information sharing in our problem setting?'. All the task-conditioned applications of hypernets discussed in Table 2 fall under this scenario. For example, multitasking [43] has related tasks (as components), and hypernets help in shared learning while

having personalized networks for each task. Similarly, continual learning [49], federated learning [63], heterogeneous treatment effects estimation [14], transfer learning [49], and domain adaptation [75] fall under this scenario.

Do we need a data-adaptive neural network?
This is another important scenario with several applications across different problem settings. In other words, we can ask, 'Are we working in a setting where the target network has to be customized to the input data?' or 'Are the data changing regularly?'. In this scenario, we can employ data-conditioned hypernets that take data as input and adaptively generate the parameters of the target network. During training, the hypernet takes the available data and learns the intrinsic characteristics of the data to generate the target network. Then, at inference time, it can take new data with slightly different characteristics and generate the target network based on the learned characteristics of the existing data. It is noted that there is some similarity between task-conditioned and data-conditioned settings, so some problems may be modelled using either technique. From existing research, it is unclear when to model a problem as data-conditioned or task-conditioned, and it needs to be explored. However, it will depend on the problem under consideration, the availability of data, and the number of tasks. All the data-conditioned applications of hypernets discussed in Table 2 fall under this scenario. For example, in neural image enhancement [44], we are interested in improving the quality of an image, so we need a target network specific to the image for a good quality output. Thus, data-conditioned hypernets are suitable for this application. Similarly, adversarial defence [69], shape learning [39], camera pose estimation [21], neural style transfer [57], few-shot learning [79], and 3D point cloud processing [66] fall under this scenario.

Do we need a dynamic neural network architecture?
Here, dynamic neural network architecture means the architecture of the target network is not known or fixed at training time. This scenario has limited but important applications. In this case, a hypernet takes some information about the architecture of the target network and generates the parameters accordingly. For example, neural architecture search [80] is such an application, which uses graph hypernetworks that take the computation graph of the target network as input to generate the network parameters. Similarly, another example of this scenario is when recurrent neural networks are implemented with hypernets [24], which need a dynamic network architecture to account for a variable number of time-steps.

Do we need faster training/parameter efficiency? As discussed earlier, hypernets can achieve parameter efficiency or weight compression, which means that the 'learnable' weights of HyperDNN are fewer than the corresponding DNN. This is expected to achieve faster training as well. This could be useful for limited resource settings and would depend on the problem setting as well as the architecture of the hypernets. For example, as discussed earlier, Mahabadi et al. [43] applied task-conditioned hypernets to fine-tune pre-trained language models by generating weights for the bottleneck adapters. In the multitasking setting, they modelled task, adapter location, and layer identity as different tasks and used embeddings of these tasks as input to the hypernet that helps in shared learning and achieved parameter efficiency. Similarly, Zhao et al. [81] also demonstrated parameter efficiency in a few-shot learning setting.

Do we need uncertainty quantification? This is a specific application scenario for hypernets. Hypernets can be used for uncertainty quantification either using noise-conditioned hypernets [33] or by using dropout in the hypernets [15]. As discussed earlier, in some settings, hypernets can produce better uncertainty estimates, e.g., [33, 53]. However, if uncertainty estimation is the sole purpose of the study, then existing uncertainty estimation techniques must be explored first. However, using dropout [67] in the hypernet architecture, similar to using dropout in standard DNNs, can complement the existing hypernets and help in uncertainty quantification.

The scenarios discussed have overlaps, so multiple scenarios can fit a problem under consideration. For example, Mahabadi et al. [43] considered fine-tuning language models using hypernets, which achieved parameter efficiency and used task-conditioning (related component setting) to solve multiple tasks. Thus, by thinking about these broad scenarios, one can determine if hypernets apply to a problem setting under consideration.

6 Challenges and Future Directions

Hypernetworks have shown enormous potential in enhancing deep learning models with increased flexibility, efficiency, and generalization. However, several challenges and opportunities for future research and development remain under-explored. In this section, we discuss some of the key challenges and propose potential directions for future exploration.

Initialization Challenge: The initialization challenge in hypernetworks refers to the difficulty of initializing the hypernetwork parameters effectively, as finding suitable initial values for the hypernetwork parameters is far from being resolved. One reason for the initialization challenge is that the weights of the target network are generated at the output layer of hypernet, and weights generation does not consider layer-wise architecture of the target network. So, initialization of hypernet weights using classical initialization techniques, such as Xavier [23] and Kaiming initialization

[25], does not guarantee that weights of target network are initialized in the same range. The performance of the hypernetwork is highly influenced by the initial state of the target network and its parameters that are generated at the output layer of the hypernet. If the target network is poorly initialized, it can propagate errors or uncertainties to the hypernetwork, affecting its ability to generate or adapt parameters effectively. Chang et al. [10] were the first to discuss the challenge of initializing hypernets. They showed that classical techniques of initializing DNNs do not work well with hypernets, however, adaptive optimizers, such as Adam [30], can address the issue to some extent. The authors suggested initializing the hypernet weights in a manner that allows the target network weights to approximate the conventional initialization of DNNs. However, it is difficult to adopt this because the weights of the target network are typically generated together. We may solve this challenge if weight generation process is aware of the layer-wise architecture of the target network. Moreover, recently, Beck et al. [6] also showed that initialization challenge of hypernets occurs even in meta-RL and classical initialization techniques fail.

Complexity/Scalability: One of the primary challenges in hypernetworks is scalability and efficiency of hypernetwork-based models. As the size and complexity of target DNNs increase, hypernetworks also become very complex, e.g., the size of the output layer is typically $m \times n$ where m is the number of neurons in the penultimate layer of hypernet and n is the number of weights in the target network. So, hypernets may not be suitable for large models unless appropriate weight-generation strategies are developed and used. Although, there are some approaches, such as multiple weight generation [14] and chunk-wise weight generation [8] to manage the complexity of hypernets but it needs more research to address the scalability challenge and make hypernetworks more practical for real-world applications.

Numerical Stability: Numerical stability in hypernetworks refers to the ability of the model to maintain accurate and reliable computations throughout the training and inference process. Hypernets, like standard neural networks, can encounter numerical stability issues [58]. One common numerical stability issue in hypernetworks is the vanishing or exploding gradients problem. During the training process, gradients can become extremely small or large, making it difficult for the model to effectively update the parameters. This can result in slow convergence or unstable training dynamics. To address numerical stability issues in hypernets, various techniques can be employed, such as careful initialization of the model's parameters, the use of gradient clipping, which bounds the gradient values to prevent them from becoming too large, and different regularization techniques such as weight decay, dropout, and spectral norm [14] that help improve numerical stability by preventing overfitting and promoting smoother optimization. Furthermore, similar to standard DNNs, using appropriate activation functions, such as ReLU or Leaky ReLU, can help alleviate the vanishing gradient problem by providing non-linearities that allow for more effective gradient propagation. It is also important to choose appropriate optimization algorithms that are known for their stability, such as Adam [30], which can handle the training dynamics of hypernetworks more effectively [10].

Theoretical Understanding: Theoretical analysis of hypernetworks involves studying their representational capacity, learning dynamics, and generalization properties. By understanding the theoretical foundations of hypernetworks, researchers can gain insights into the underlying principles that drive their effectiveness and explore new avenues for improving their performance. Just like DNNs, understanding the working of hypernets is far from being solved. Although, there are some works that provide theoretical insights into hypernets, e.g., Littwin et al. [38] highlighted that infinitely wide hypernetworks may not converge to a global minimum using gradient descent, but convexity can be achieved by increasing the dimensionality of the hypernetwork's output. Galanti and Wolf [22] also studied the modularity of hypernets and showed that hypernets can be more efficient than the embedding-based method for mapping an input to a function. Intuitively, hypernets map an input to one point on a low-dimensional manifold for weights of target network [63] – theoretical insights into the connection between two can be very helpful. Thus, more research into the theoretical properties of hypernets will help to make them more popular and will also attract more research.

Uncertainty-aware Deep Learning: Uncertainty-aware neural networks allow for more reliable and robust predictions, especially in scenarios where uncertainty estimation is crucial, such as decision-making under uncertainty, safety-critical applications, or when working with limited or noisy data [1]. Despite the success of DNNs and the development of different uncertainty quantification techniques, it still remains an open problem to quantify the prediction uncertainty [32]. Hypernets have opened a new door to uncertainty quantification as noise-conditioned hypernets can generate distribution on target network weights and have been shown to have better uncertainties than the SOTA [33, 53]. Similarly, Chauhan et al. [15] used task-conditioned hypernets with dropout to generate multiple sets of weights for the target network. Further research into this can provide computationally efficient and effective techniques as compared with other techniques, such as ensemble methods, which need to train multiple models.

Interpretability Enhancement: It will be helpful for the community to develop methods for visualizing, analyzing, and explaining the task-specific weights generated by hypernetworks. This includes developing intuitive visualization methods, and feature relevance analysis techniques that provide deeper insights into the weight generation and decision-making process of hypernetwork-based models.

Model Compression and Efficiency: Hypernetworks can aid in model compression and efficiency in some problem settings [81, 43], where smaller hypernets are trained to generate larger target networks that can reduce the memory footprint and computational requirements of the model. This is particularly useful in resource-constrained environments where memory and computational resources are limited, and hypernets can be studied specifically for such settings.

Usage Guidelines: Hypernetworks add additional complexity to solving problems. As with HyperDNN, we have an additional network to generate weights for the target DNN. Hypernets introduce additional hyperparameters related to the weight generation process, e.g., what kind of weight generation should be used and how many chunks should be used. Some research and guidelines are needed to guide the researchers through these choices, stressing the need for a comparative study of different approaches under varying problem settings.

Thus, the field of hypernetworks in deep learning presents several challenges and opportunities for future research. The advancements in these areas will pave the way for the widespread adoption and effective utilization of hypernetworks in various domains of deep learning.

7 Conclusion

Hypernetworks have emerged as a promising approach to enhance deep learning models with increased flexibility, efficiency, generalization, uncertainty awareness, and information sharing. They have opened new avenues for research and applications across various domains. In this paper, we presented the first review of hypernetworks in the context of deep learning. We provided an illustrative example to explain the workings of hypernetworks and proposed a categorization based on five design criteria: inputs, outputs, variability of inputs and outputs, and the architecture of hypernets. We discussed some of the important applications of hypernets to different deep learning problems, including multitasking, continual learning, federated learning, causal inference, and computer vision. Additionally, we presented scenarios and questions to help readers understand whether hypernets can be applied to a given problem setting. Finally, we highlighted challenges that need to be addressed in the future. These challenges include initialization, stability, scalability, efficiency, and the need for theoretical insights. Future research should focus on tackling these challenges to further advance the field of hypernetworks and make them more accessible and practical for real-world applications. By addressing these issues, the potential of hypernetworks can be fully realized, leading to more robust and versatile deep learning models.

Acknowledgements

This work was supported in part by the National Institute for Health Research (NIHR) Oxford Biomedical Research Centre (BRC) and in part by InnoHK Project Programme 3.2: Human Intelligence and AI Integration (HIAI) for the Prediction and Intervention of CVDs: Warning System at Hong Kong Centre for Cerebro-cardiovascular Health Engineering (COCHE). DAC was supported by an NIHR Research Professorship, an RAEng Research Chair, the InnoHK Hong Kong Centre for Cerebro-cardiovascular Health Engineering (COCHE), the NIHR Oxford Biomedical Research Centre (BRC), and the Pandemic Sciences Institute at the University of Oxford. The views expressed are those of the authors and not necessarily those of the NHS, the NIHR, the Department of Health, the InnoHK – ITC, or the University of Oxford.

Statements and Declarations

Competing Interests

The authors declare that they have no competing interests.

Data Availability

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Contributions

V.K.C. conceptualized the study, analyzed the literature and wrote the first draft. J.Z., P.L. and S.M. helped to filter out the literature and prepare Table 2. D.A.C did supervision and funding acquisition. All authors reviewed and approved the final manuscript.

References

[1] Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., et al. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297.

[2] Alaluf, Y., Tov, O., Mokady, R., Gal, R., and Bermano, A. (2022). Hyperstyle: Stylegan inversion with hypernetworks for real image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18511–18521.

[3] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., and Farhan, L. (2021). Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8:1–74.

[4] Armstrong, J. and Clifton, D. (2021). Continual learning of longitudinal health records. *arXiv preprint arXiv:2112.11944*.

[5] Balažević, I., Allen, C., and Hospedales, T. M. (2019). Hypernetwork knowledge graph embeddings. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Workshop and Special Sessions: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings 28*, pages 553–565. Springer.

[6] Beck, J., Jackson, M. T., Vuorio, R., and Whiteson, S. (2023). Hypernetworks in meta-reinforcement learning. In *Conference on Robot Learning*, pages 1478–1487. PMLR.

[7] Bensadoun, R., Gur, S., Galanti, T., and Wolf, L. (2021). Meta internal learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20645–20656. Curran Associates, Inc.

[8] Brock, A., Lim, T., Ritchie, J., and Weston, N. (2018). SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*.

[9] Carrasquilla, J., Hibat-Allah, M., Inack, E., Makhzani, A., Neklyudov, K., Taylor, G. W., and Torlai, G. (2023). Quantum hypernetworks: Training binary neural networks in quantum superposition. *arXiv preprint arXiv:2301.08292*.

[10] Chang, O., Flokas, L., and Lipson, H. (2020). Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*.

[11] Chauhan, V. K., Molaei, S., Tania, M. H., Thakur, A., Zhu, T., and Clifton, D. A. (2023a). Adversarial deconfounding in individualised treatment effects estimation. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206, pages 837–849. PMLR.

[12] Chauhan, V. K., Singh, S., and Sharma, A. (2024a). HCR-Net: A deep learning based script independent handwritten character recognition network. *Multimedia Tools and Applications*, pages 1–35.

[13] Chauhan, V. K., Thakur, A., O'Donoghue, O., Rohanian, O., Molaei, S., and Clifton, D. A. (2024b). Continuous patient state attention model for addressing irregularity in electronic health records. *BMC Medical Informatics and Decision Making*, 24(1):117.

[14] Chauhan, V. K., Zhou, J., Ghosheh, G., Molaei, S., and A Clifton, D. (2024c). Dynamic inter-treatment information sharing for individualized treatment effects estimation. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238, pages 3529–3537. PMLR.

[15] Chauhan, V. K., Zhou, J., Molaei, S., Ghosheh, G., and Clifton, D. A. (2023b). Dynamic inter-treatment information sharing for heterogeneous treatment effects estimation.

[16] de Avila Belbute-Peres, F., fan Chen, Y., and Sha, F. (2021). HyperPINN: Learning parameterized differential equations with physics-informed hypernetworks. In *The Symbiosis of Deep Learning and Differential Equations*.

[17] Deutsch, L., Nijkamp, E., and Yang, Y. (2019). A generative model for sampling high-performance and diverse weights for neural networks. *arXiv preprint arXiv:1905.02898*.

[18] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[19] Dinh, T. M., Tran, A. T., Nguyen, R., and Hua, B.-S. (2022). Hyperinverter: Improving stylegan inversion via hypernetwork. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11389–11398.

[20] Ehret, B., Henning, C., Cervera, M., Meulemans, A., Oswald, J. V., and Grewe, B. F. (2021). Continual learning in recurrent neural networks. In *International Conference on Learning Representations*.

[21] Ferens, R. and Keller, Y. (2023). Hyperpose: Camera pose localization using attention hypernetworks. *arXiv preprint arXiv:2303.02610*.

[22] Galanti, T. and Wolf, L. (2020). On the modularity of hypernetworks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10409–10419. Curran Associates, Inc.

[23] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

[24] Ha, D., Dai, A. M., and Le, Q. V. (2017). Hypernetworks. In *International Conference on Learning Representations*.

[25] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

[26] Henning, C., Cervera, M., D'Angelo, F., Oswald, J. V., Traber, R., Ehret, B., Kobayashi, S., Grewe, B. F., and Sacramento, J. (2021). Posterior meta-replay for continual learning. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

[27] Hoang, L. P., Le, D. D., Tuan, T. A., and Thang, T. N. (2023). Improving pareto front learning via multi-sample hypernetworks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37(7), pages 7875–7883.

[28] Höfer, T., Kiefer, B., Messmer, M., and Zell, A. (2023). Hyperposepdf - hypernetworks predicting the probability distribution on so(3). In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2369–2379.

[29] Huang, Y., Xie, K., Bharadhwaj, H., and Shkurti, F. (2021). Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 799–805. IEEE.

[30] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[31] Klocek, S., Maziarka, Ł., Wołczyk, M., Tabor, J., Nowak, J., and Śmieja, M. (2019). Hypernetwork functional image representation. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Workshop and Special Sessions: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings 28*, pages 496–510. Springer.

[32] Kristiadi, A., Däubener, S., and Fischer, A. (2019). Predictive uncertainty quantification with compound density networks. *arXiv preprint arXiv:1902.01080*.

[33] Krueger, D., Huang, C.-W., Islam, R., Turner, R., Lacoste, A., and Courville, A. (2018). Bayesian hypernetworks.

[34] Lamb, A., Saveliev, E., Li, Y., Tschiatschek, S., Longden, C., Woodhead, S., Hernández-Lobato, J. M., Turner, R. E., Cameron, P., and Zhang, C. (2021). Contextual hypernetworks for novel feature adaptation. *arXiv preprint arXiv:2104.05860*.

[35] Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

[36] Li, Y., Gu, S., Zhang, K., Van Gool, L., and Timofte, R. (2020). Dhp: Differentiable meta pruning via hypernetworks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 608–624. Springer.

[37] Litany, O., Maron, H., Acuna, D., Kautz, J., Chechik, G., and Fidler, S. (2022). Federated learning with heterogeneous architectures using graph hypernetworks. *arXiv preprint arXiv:2201.08459*.

[38] Littwin, E., Galanti, T., Wolf, L., and Yang, G. (2020). On infinite-width hypernetworks. *Advances in neural information processing systems*, 33:13226–13237.

[39] Littwin, G. and Wolf, L. (2019). Deep meta functionals for shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1824–1833.

[40] Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., and Sun, J. (2019). Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305.

[41] Lorraine, J. and Duvenaud, D. (2018). Stochastic hyperparameter optimization through hypernetworks.

[42] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

[43] Mahabadi, R. K., Ruder, S., Dehghani, M., and Henderson, J. (2021). Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576.

[44] Muller, L. K. (2021). Overparametrization of hypernetworks at fixed flop-count enables fast neural image enhancement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 284–293.

[45] Navon, A., Shamsian, A., Fetaya, E., and Chechik, G. (2021). Learning the pareto front with hypernetworks. In *International Conference on Learning Representations*.

[46] Nguyen, P., Tran, T., Le, K., Gupta, S., Rana, S., Nguyen, D., Nguyen, T., Ryan, S., and Venkatesh, S. (2021). Fast conditional network compression using bayesian hypernetworks. In Oliver, N., Pérez-Cruz, F., Kramer, S., Read, J., and Lozano, J. A., editors, *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 330–345, Cham. Springer International Publishing.

[47] Nirkin, Y., Wolf, L., and Hassner, T. (2021). Hyperseg: Patch-wise hypernetwork for real-time semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4061–4070.

[48] Oh, G. and Peng, H. (2022). Cvae-h: Conditionalizing variational autoencoders via hypernetworks and trajectory forecasting for autonomous driving. *arXiv preprint arXiv:2201.09874*.

[49] Oswald, J. V., Henning, C., Grewe, B. F., and Sacramento, J. (2020). Continual learning with hypernetworks. In *International Conference on Learning Representations*.

[50] Pan, Z., Liang, Y., Zhang, J., Yi, X., Yu, Y., and Zheng, Y. (2018). Hyperst-net: Hypernetworks for spatio-temporal forecasting. *arXiv preprint arXiv:1809.10889*.

[51] Peng, H., Du, H., Yu, H., Li, Q., Liao, J., and Fu, J. (2020). Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. *Advances in Neural Information Processing Systems*, 33:17955–17964.

[52] Qu, J., Faney, T., Wang, Z., Gallinari, P., Yousef, S., and de Hemptinne, J.-C. (2022). Hmoe: Hypernetwork-based mixture of experts for domain generalization. *arXiv preprint arXiv:2211.08253*.

[53] Ratzlaff, N. and Fuxin, L. (2019). Hypergan: A generative model for diverse, performant neural networks. In *International Conference on Machine Learning*, pages 5361–5369. PMLR.

[54] Rezaei-Shoshtari, S., Morissette, C., Hogan, F. R., Dudek, G., and Meger, D. (2023). Hypernetworks for zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:2211.15457*.

[55] Rohanian, O., Jauncey, H., Nouriborji, M., Chauhan, V. K., Gonalves, B. P., Kartsonaki, C., Clinical Characterisation Group, I., Merson, L., and Clifton, D. (2023). Using bottleneck adapters to identify cancer in clinical notes under low-resource constraints. In *The 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks*, pages 62–78, Toronto, Canada. Association for Computational Linguistics.

[56] Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2019). Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*.

[57] Ruta, D., Gilbert, A., Motiian, S., Faieta, B., Lin, Z., and Collomosse, J. (2023). Hypernst: Hyper-networks for neural style transfer. In Karlinsky, L., Michaeli, T., and Nishino, K., editors, *Computer Vision – ECCV 2022 Workshops*, pages 201–217, Cham. Springer Nature Switzerland.

[58] Sarafian, E., Keynan, S., and Kraus, S. (2021). Recomposing the reinforcement learning building blocks with hypernetworks. In *International Conference on Machine Learning*, pages 9301–9312. PMLR.

[59] Schmidhuber, J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.

[60] Schmidhuber, J. (1993). A 'self-referential'weight matrix. In *ICANN'93: Proceedings of the International Conference on Artificial Neural Networks Amsterdam, The Netherlands 13–16 September 1993 3*, pages 446–450. Springer.

[61] Sendera, M., Przewięźlikowski, M., Karanowski, K., Zięba, M., Tabor, J., and Spurek, P. (2023a). Hypershot: Few-shot learning by kernel hypernetworks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2469–2478.

[62] Sendera, M., Przewięźlikowski, M., Miksa, J., Rajski, M., Karanowski, K., Zięba, M., Tabor, J., and Spurek, P. (2023b). The general framework for few-shot learning by kernel hypernetworks. *Machine Vision and Applications*, 34(4):53.

[63] Shamsian, A., Navon, A., Fetaya, E., and Chechik, G. (2021). Personalized federated learning using hypernetworks. In *International Conference on Machine Learning*, pages 9489–9502. PMLR.

[64] Shih, A., Sadigh, D., and Ermon, S. (2021). Hyperspns: Compact and expressive probabilistic circuits. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8571–8582. Curran Associates, Inc.

[65] Spurek, P., Winczowski, S., Tabor, J., Zamorski, M., Zieba, M., and Trzciński, T. (2020). Hypernetwork approach to generating point clouds. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9099–9108.

[66] Spurek, P., Zieba, M., Tabor, J., and Trzcinski, T. (2022). General hypernetwork framework for creating 3d point clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):9995–10008.

[67] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[68] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

[69] Sun, Z., Ozay, M., and Okatani, T. (2017). Hypernetworks with statistical filtering for defending adversarial examples. *arXiv preprint arXiv:1711.01791*.

[70] Szatkowski, F., Piczak, K. J., Spurek, P., Tabor, J., and Trzcinski, T. (2022). Hypersound: Generating implicit neural representations of audio signals with hypernetworks. In *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*.

[71] Tay, Y., Zhao, Z., Bahri, D., Metzler, D., and Juan, D.-C. (2021). Hypergrid transformers: Towards a single model for multiple tasks. In *International Conference on Learning Representations*.

[72] Tran, T. A., Hoang, L. P., Le, D. D., and Tran, T. N. (2023). A framework for controllable pareto front learning with completed scalarization functions and its applications. *arXiv preprint arXiv:2302.12487*.

[73] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

[74] Vincent, T., Metelli, A. M., Belousov, B., Peters, J., Restelli, M., and D'Eramo, C. (2023). Parameterized projected bellman operator.

[75] Volk, T., Ben-David, E., Amosy, O., Chechik, G., and Reichart, R. (2022). Example-based hypernetworks for out-of-distribution generalization. *arXiv preprint arXiv:2203.14276*.

[76] Wiens, J., Guttag, J., and Horvitz, E. (2014). A study in transfer learning: leveraging data from multiple hospitals to enhance hospital-specific predictions. *Journal of the American Medical Informatics Association*, 21(4):699–706.

[77] Wu, Q., Bauer, D., Chen, Y., and Ma, K.-L. (2023). Hyperinr: A fast and predictive hypernetwork for implicit neural representations via knowledge distillation. *arXiv preprint arXiv:2304.04188*.

[78] Wullach, T., Adler, A., and Minkov, E. (2022). Character-level hypernetworks for hate speech detection. *Expert Systems with Applications*, 205:117571.

[79] Yin, L., Perez-Rua, J. M., and Liang, K. J. (2022). Sylph: A hypernetwork framework for incremental few-shot object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9035–9045.

[80] Zhang, C., Ren, M., and Urtasun, R. (2019). Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*.

[81] Zhao, D., Kobayashi, S., Sacramento, J., and Von Oswald, J. (2020). Meta-learning via hypernetworks. In *4th Workshop on Meta-Learning at NeurIPS 2020 (MetaLearn 2020)*. NeurIPS.

[82] Zięba, M. (2022). Hypermaml: Few-shot adaptation of deep models with hypernetworks. *arXiv preprint arXiv:2205.15745*.