Journal of Intelligent Systems and Future Computing

ISSN 3117 - 4698

https://jisfc.com/index.php/jisfc/index

Lightweight Convolutional Architectures for On-Device Computer Vision Inference

Ramanathan P[1*], Mahalingam S[2] and Vaidyanathan K[3]

[1]Department of Robotics and Automation, Meenachil Institute of Distributed Intelligence, Malaysia Email: pallavi.ramanathan@meenachil.edu

ORCID: https://orcid.org/8643-9406-7425-2934

[2]Department of Robotics and Automation, Muvattupuzha Center for Computational Cognition, Bhutan Email: sundar.mahalingam@muvattupuzha.edu

ORCID: https://orcid.org/9618-2407-4743-3961

[3]Department of Information Technology, Chaliyar Institute of Quantum Computing, Singapore Email: kavitha.vaidyanathan@chaliyar.edu

ORCID: https://orcid.org/5066-9066-6286-8096

ABSTRACT

On-device computer vision now depends on convolutional architectures deliberately engineered to fit within the memory, latency, and power budgets of phones, embedded boards, and microcontrollers, rather than repurposed from server-scale networks. This article synthesizes the literature that produced those architectures: depthwise separable convolutions, popularized by Xception and operationalized for mobile deployment by the MobileNet family; the channel-shuffle and inverted-residual-bottleneck refinements introduced by ShuffleNet and MobileNetV2; the platform-aware neural architecture search that produced MnasNet, MobileNetV3, and the EfficientNet compound-scaling family; and the complementary, largely orthogonal literature on pruning, quantization, and knowledge distillation that compresses these architectures further once trained. We give particular attention to the most recent phase of this work, from 2021 onward, in which structural reparameterization (RepVGG, MobileOne) and hybrid convolution-attention designs (MobileViT, EdgeNeXt, FastViT) have extended the same efficiency-first design philosophy to compete with, and selectively borrow from, vision transformers. Read together, the literature supports lightweight convolutional design as a cumulative, empirically validated research program rather than a series of disconnected tricks, while leaving genuinely open the questions of how architecture, quantization, and target hardware should be jointly optimized, and whether pure convolutional, pure transformer, or hybrid designs will dominate the next generation of on-device vision models.

Keywords: lightweight convolutional neural networks; on-device inference; mobile computer vision; neural architecture search; model compression

Citation: Ramanathan P, Mahalingam S and Vaidyanathan K [2023]. Lightweight Convolutional Architectures for On-Device Computer Vision Inference. **DOI:** https://doi.org/10.5281/zenodo.22843875

Copyright: © 2023 by the authors. Open access under CC BY 4.0 license.

Article Information: *Received:* 24 February 2023 *Accepted:* 16 May 2023 *Published:* 11 August 2023

Article Class: *Research Article*

1. Introduction

A smartphone camera app that recognizes a plant species before the shutter finishes clicking, a warehouse robot that must tell a dented box from an intact one without a round trip to a data center, a security camera that flags an intruder without ever uploading a frame of video: all of these depend on a convolutional neural network running entirely on the device that captured the image, under a power, memory, and latency budget that a cloud-hosted model never has to think about. For most of the deep learning era, "run a good vision model" and "run a small, fast vision model" were treated as separate, sometimes conflicting, goals: the field's headline architectures — deep VGG-style stacks of 3×3 convolutions and the residual networks that followed them — were designed and evaluated with essentially no efficiency constraint in mind, optimized purely for ImageNet accuracy on server-class GPUs with tens or hundreds of millions of parameters [1,2]. That separation has narrowed considerably over the past decade. A distinct architectural research program, aimed specifically at convolutional networks that are accurate enough to be useful and small enough to run on a phone, a microcontroller, or an embedded accelerator, has produced a fairly coherent body of design ideas: depthwise separable convolutions, inverted residual bottlenecks, channel shuffling, hardware-aware neural architecture search, and, more recently, structural reparameterization and hybrid convolution-attention blocks.

This article synthesizes that literature. Its subject is not vision transformers in general, nor edge computing infrastructure in general, but specifically the architectural lineage of lightweight convolutional networks built for on-device inference — the sequence of design decisions, starting roughly with SqueezeNet and MobileNet in 2016-2017, that made it possible to run competent image classification, detection, and segmentation models within a few megabytes of memory and a few hundred million multiply-accumulate operations. We also cover the complementary, non-architectural techniques — pruning, quantization, knowledge distillation, and neural architecture search — that are so tightly interwoven with the architecture literature that treating them separately would misrepresent how the field actually developed; MobileNetV3 and MnasNet, for instance, are architecture search outputs as much as they are hand-designed architectures, and int8 quantization is now assumed rather than optional for most deployed mobile CNNs.

The efficient-CNN literature emerged from a specific, well-documented tension. Sze et al., in their widely cited tutorial and survey on efficient deep neural network processing, frame the core problem as one of data movement: for the convolutional workloads that dominate computer vision, moving weights and activations between memory and compute units, not the arithmetic itself, dominates both latency and energy consumption on real hardware, which means that a network's architecture — how much data it has to move, and how often — is inseparable from how efficiently it can run [3]. That framing matters for how this article is organized. A model can look cheap on paper (few floating-point operations) and still run slowly on a phone if its architecture is memory-access-bound rather than compute-bound; a point Ma et al. make explicit and central to their own design work [4]. We take that lesson seriously throughout: wherever the cited literature reports actual on-device latency rather than only parameter counts or FLOPs, we report the latency figure, because that is ultimately the number a deployment engineer cares about.

We organize the discussion into five parts. First, we cover the foundational architectural moves — depthwise separable convolutions and the SqueezeNet/MobileNet lineage — that established lightweight CNN design as a distinct subfield. Second, we trace the compound-scaling and neural-architecture-search literature that turned architecture design itself into an optimization problem. Third, we cover complementary compression techniques — pruning, quantization, and distillation — that are applied on top of, and increasingly co-designed with, these architectures. Fourth, we discuss the more recent generation of hybrid CNN-transformer and reparameterized architectures that have emerged since 2021, and what they change about the design space. Fifth, we discuss open challenges: benchmarking inconsistency, the gap between FLOPs and real latency, and the genuinely unresolved question of how CNN-based efficiency techniques should relate to the transformer architectures that increasingly dominate vision research at the high-accuracy end of the spectrum. We close with a conclusion assessing what the literature, read as a

whole, actually supports.

A note on scope, in the same spirit as the caveat any honest literature synthesis owes its reader: almost all of the accuracy and efficiency numbers reported in the papers we cite come from standardized benchmarks — principally ImageNet-1K classification, and to a lesser extent COCO detection and Cityscapes or ADE20K segmentation — evaluated on specific reference hardware (a particular phone SoC, a particular embedded board) that the original authors chose. Cross-paper comparisons of exact numbers should be read with that caveat in mind: a claimed "x% faster than architecture Y" is a statement about one hardware target under one measurement methodology, not a universal law. We flag this explicitly rather than smoothing over it, because the efficient-CNN literature itself is generally careful about this caveat and a synthesis that erased it would misrepresent the sources.

2. Depthwise Separable Convolutions and the First Wave of Mobile Architectures

Standard convolution layers apply a full-depth filter — one that spans every input channel simultaneously — at every spatial location, which means the computational cost of a single convolutional layer scales with the product of the kernel's spatial extent, the number of input channels, and the number of output channels. Depthwise separable convolution factors that operation into two cheaper steps: a depthwise convolution that applies a single spatial filter per input channel independently, followed by a pointwise (1×1) convolution that mixes information across channels. Chollet's Xception architecture demonstrated that this factorization, pushed to its logical extreme by replacing essentially every Inception module with a depthwise separable convolution, could match or slightly exceed Inception V3's accuracy on ImageNet at a comparable parameter count while using the freed-up capacity more efficiently, establishing depthwise separability as a serious architectural primitive rather than a niche optimization [5].

Howard et al.'s original MobileNet paper took the same primitive and built an architecture explicitly optimized for mobile and embedded deployment around it, introducing two global hyperparameters — a width multiplier that scales the number of channels in every layer, and a resolution multiplier that scales the input image size — that let a practitioner trade accuracy for latency and model size along a continuous, controllable spectrum rather than needing an entirely new architecture for each operating point [6]. The paper reported competitive ImageNet accuracy at a fraction of the parameter count of contemporary full-size CNNs, and, notably, validated the architecture across a range of downstream tasks beyond classification — object detection, fine-grained recognition, face attributes, and large-scale geolocalization — establishing depthwise separable MobileNets as a general-purpose lightweight backbone rather than a classification-specific trick.

MobileNetV2 refined this design with two ideas that became standard vocabulary across the field. First, the inverted residual: where a classic ResNet bottleneck compresses channels before the expensive spatial convolution and expands afterward, MobileNetV2 does the opposite, expanding channels before the (cheap, depthwise) spatial convolution and compressing afterward, on the reasoning that the memory-expensive tensors should be the thin ones and the wide, information-rich intermediate representation should exist only briefly, inside the block, where a depthwise convolution can process it cheaply [7]. Second, the linear bottleneck: the paper showed, both theoretically and empirically, that applying a nonlinearity like ReLU to the low-dimensional output of a bottleneck destroys information disproportionately, because ReLU's zeroing behavior is much more damaging in a narrow channel space than a wide one, so MobileNetV2 removes the nonlinearity from the final projection layer of each block. The resulting architecture improved both accuracy and efficiency over the original MobileNet on ImageNet classification, and the paper also demonstrated the architecture's effectiveness as a backbone for object detection (via an SSDLite head) and semantic segmentation (via a mobile DeepLabv3 head), reinforcing the pattern of validating a lightweight backbone across the full range of dense-prediction vision tasks, not classification alone.

SqueezeNet, developed roughly concurrently with the earliest of these ideas, approached the same efficiency goal via a different structural primitive: the "fire module," which squeezes the number of input channels with 1×1 convolutions before expanding again with a mix of 1×1 and 3×3 filters, deliberately minimizing the number of expensive 3×3 filters in the network. Iandola et al. reported that this design achieved AlexNet-level

ImageNet accuracy with 50 times fewer parameters, and that with additional deep compression the resulting model could be stored in under half a megabyte — small enough, the authors argued explicitly, to matter for practical concerns like over-the-air model updates and FPGA deployment with limited on-chip memory, rather than being an abstract parameter-counting exercise [8]. Iandola and Han's own subsequent involvement in the pruning and compression literature (discussed below) reflects how tightly the "design a small architecture" and "compress an existing architecture" research threads were connected from the outset [9].

ShuffleNet pursued efficiency from yet another angle: group convolutions reduce computation by having each output channel depend on only a subset of input channels, but naively stacking group convolutions creates a channel-isolation problem, where information from one group of channels never mixes with another. Zhang et al.'s ShuffleNet addressed this directly with a channel shuffle operation that interleaves the outputs of a grouped convolution before the next grouped layer consumes them, restoring cross-group information flow at essentially no additional computational cost [10]. The paper reported a substantial accuracy improvement over MobileNet at a comparable computational budget, and importantly measured actual inference speed on an ARM-based mobile device rather than relying on FLOPs alone, reporting roughly a 13-fold speedup over AlexNet on that hardware. ShuffleNet V2 revisited this design two years later with a more systematic methodology, arguing — in direct alignment with the data-movement framing from Sze et al. discussed above — that FLOPs alone are a poor proxy for actual latency, because memory access cost and the degree of parallelism a given operation permits on real hardware matter just as much as raw operation count [4]. From this analysis the authors derived four practical design guidelines (favoring equal channel widths, limiting group convolution usage, avoiding network fragmentation, and reducing elementwise operations) and used them to redesign ShuffleNet into a version that achieved a better speed-accuracy tradeoff on both GPU and ARM hardware than the original.

3. Compound Scaling and Neural Architecture Search

A second, related strand of the literature treats "what is the best lightweight CNN architecture" less as a hand-design problem and more as a search or optimization problem over a well-defined design space. MnasNet, from Tan et al., was among the first widely adopted examples: rather than optimizing architecture search purely against a proxy metric like FLOPs, the authors incorporated real, on-device inference latency, measured by executing candidate architectures directly on Pixel phones, as part of the search objective. The resulting architecture was reported to run 1.8 times faster than a comparable MobileNetV2 configuration while achieving 0.5 percentage points higher ImageNet top-1 accuracy, and 2.3 times faster than a comparable NASNet configuration at 1.2 points higher accuracy, directly on the accuracy-latency tradeoff that matters for deployment rather than on a FLOPs proxy that might not correlate with it [11]. This "hardware-in-the-loop" search philosophy became a recurring theme in subsequent work.

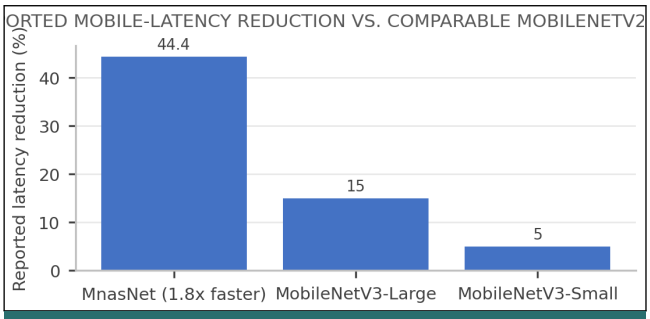


Figure 1: Reported on-device latency reduction of hardware-aware search and NAS-refined architectures relative to a comparable MobileNetV2 baseline at matched or better accuracy, as stated by each paper's authors -- MnasNet's reported 1.8x speedup, and MobileNetV3-Large and MobileNetV3-Small's reported latency reductions on mobile hardware [11,12].

Howard et al.'s MobileNetV3 combined this platform-aware search methodology (specifically, a MnasNet-style search complemented by the NetAdapt algorithm for fine-grained layer-by-layer refinement) with several hand-designed architectural improvements — a redesigned, more efficient "last stage" of the network, and the reintroduction of squeeze-and-excitation attention blocks and the h-swish activation function, chosen because they are cheaper to compute in quantized, fixed-point mobile inference than a naive sigmoid or swish implementation [12]. The resulting MobileNetV3-Large variant was reported to be 3.2 percentage points more accurate on ImageNet than the comparable MobileNetV2 configuration while also reducing latency by roughly 15%, and the smaller MobileNetV3-Small variant achieved a 4.6-point accuracy improvement alongside a 5% latency

reduction; on object detection the authors reported roughly 25% faster inference at comparable COCO detection accuracy. These are exactly the kind of paired accuracy-and-latency numbers that the earlier caveat about FLOPs-versus-real-latency argues a reader should look for, and MobileNetV3 is unusually careful about reporting both.

EfficientNet took the scaling question in a different, complementary direction. Tan and Le observed that prior work typically scaled a baseline network up along a single dimension — more layers (depth), wider layers (width), or higher input resolution — and argued instead for compound scaling: increasing depth, width, and resolution together, governed by a single compound coefficient, on the grounds that these three dimensions interact and none of them alone is sufficient at large scale [13]. Applying compound scaling to a baseline architecture discovered via neural architecture search (structurally related to the MnasNet search space), the authors reported that their largest EfficientNet variant achieved 84.3% top-1 ImageNet accuracy while being 8.4 times smaller and 6.1 times faster than the best previously reported model achieving comparable accuracy, and that smaller EfficientNet variants transferred well to smaller benchmark datasets with substantially fewer parameters than prior transfer-learning baselines. EfficientNetV2 revisited the family with training efficiency, not just inference efficiency, as an explicit target: Tan and Le used training-aware neural architecture search to jointly optimize for both parameter efficiency and training speed, and introduced progressive learning — increasing image resolution over the course of training while adaptively strengthening regularization to compensate for the harder optimization problem that larger images at later training stages create [14]. The resulting models were reported to be up to 6.8 times smaller than comparable-accuracy prior EfficientNet models while training substantially faster, and the largest configuration reported 87.3% top-1 ImageNet accuracy — outperforming a comparable Vision Transformer configuration by 2 accuracy points while training 5 to 11 times faster, a data point worth flagging because it is one of relatively few direct, matched-condition CNN-versus-transformer efficiency comparisons in the literature we reviewed.

Once-for-All, from Cai et al., addressed a different practical bottleneck in this line of work: even efficient neural architecture search is expensive to run separately for every new hardware target, and mobile deployment increasingly means many different targets (different phone SoCs, different accelerator chips) rather than one. Once-for-All trains a single large "supernet" that supports many architectural configurations simultaneously, using a progressive shrinking training procedure across depth, width, kernel size, and resolution, from which a specialized sub-network for a specific hardware target can then be extracted without any additional training [15]. The authors reported that sub-networks extracted this way achieved up to a 4-percentage-point ImageNet accuracy improvement over comparable MobileNetV3 configurations at similar computational cost, while reducing the search-and-train cost of deploying to a new hardware target by orders of magnitude relative to running architecture search from scratch for each target — an efficiency argument aimed at the deployment pipeline itself, not just the resulting model.

GhostNet approached the architecture-design problem from a feature-redundancy angle rather than a search angle. Han et al. observed that trained CNNs typically produce feature maps with substantial redundancy — many channels are near-linear transformations of other channels — and proposed the Ghost module, which generates a smaller set of "intrinsic" feature maps via ordinary convolution and then generates the remaining feature maps via a series of cheap linear operations (inexpensive depthwise-style transformations) applied to those intrinsic maps, rather than generating every channel via full convolution [16]. Stacked into Ghost bottlenecks, this design achieved a reported 75.7% ImageNet top-1 accuracy at computational cost comparable to MobileNetV3, matching or exceeding it, and demonstrated that exploiting redundancy directly, rather than searching over an architecture space or scaling a fixed template, is a viable and complementary route to the same goal.

4. Pruning, Quantization, and Distillation as Co-Design Tools

The techniques discussed so far design a small architecture from the outset. A parallel and older literature instead starts from a larger, already-trained network and compresses it — and in modern practice, the two approaches are routinely combined, so treating

them as separate research programs risks understating how integrated they have become. Han, Mao, and Dally's Deep Compression paper is the foundational reference for the compress-after-training approach: a three-stage pipeline of magnitude-based weight pruning, k-means-based weight quantization with shared codebooks, and Huffman coding of the resulting quantized indices, reported to reduce model storage by 35 to 49 times on architectures like AlexNet and VGG-16 without accuracy loss, small enough in the resulting footprint to fit entirely in on-chip SRAM rather than off-chip DRAM on suitable embedded hardware [9]. Li et al.'s filter-pruning work addressed a specific weakness of unstructured, weight-level pruning like Deep Compression's: it produces irregular sparsity patterns that general-purpose hardware and standard BLAS libraries cannot exploit efficiently without specialized sparse-matrix support. Their solution — removing entire filters identified as contributing least to the network's output, rather than individual weights — preserves dense, regular computation and reported inference cost reductions of up to 34% for VGG-16 and 38% for ResNet-110 on CIFAR-10, after a brief retraining step to recover the resulting accuracy drop [17]. Liu et al.'s network slimming pursued a related structured-pruning goal via a different mechanism: adding an L1 sparsity penalty directly to the scaling factors already present in batch normalization layers during training, which drives many channels' effective contribution toward zero and lets the network be pruned at the channel level afterward with a straightforward, principled criterion for which channels to remove, demonstrated across VGGNet, ResNet, and DenseNet architectures [18].

Table 1: Representative compression and efficiency figures reported by foundational pruning, quantization, and knowledge-distillation papers relevant to lightweight CNN deployment. Figures are as originally reported by each source and are not directly comparable across rows, since they were measured on different architectures, hardware, and benchmarks. [9,17,18,19,20,21].

Technique	Source	Mechanism	Reported Effect
Pruning + trained quantization + Huffman coding (Deep Compression)	Han et al. [9]	Weight-level pruning, codebook quantization, entropy coding	35-49x storage reduction on AlexNet/VGG-16, no accuracy loss
Structured filter pruning	Li et al. [17]	Remove whole filters ranked by L1 norm	Up to 34% (VGG-16) and 38% (ResNet-110) inference cost reduction on CIFAR-10
Channel-level slimming via BN scaling factors	Liu et al. [18]	L1 sparsity penalty on batch-norm gammas, then channel pruning	Substantial model size and FLOPs reduction across VGGNet/ResNet/DenseNet with minimal accuracy loss
Integer-only quantization	Jacob et al. [19]	8-bit weights and activations, quantization-aware training	Meaningful CPU latency reduction even on already-efficient MobileNets
Binary weights and activations (XNOR-Net)	Rastegari et al. [20]	1-bit weights/activations, XNOR + popcount convolution	~58x faster convolution, 32x memory savings on CPU
Teacher-student knowledge distillation	Hinton et al. [21]	Student matches softened teacher output distribution	Compact student models recover much of teacher accuracy at a fraction of the size

Quantization compresses along a different axis: reducing the numerical precision used to represent weights and activations, typically from 32-bit floating point down to 8-bit integers or lower. Jacob et al.'s widely deployed integer-arithmetic-only quantization scheme is particularly relevant to this article's subject because the authors explicitly evaluated it on MobileNets — an architecture already optimized for efficiency along other axes — and showed that quantization still yielded meaningful additional latency improvements on standard mobile CPUs, evaluated on both ImageNet classification and COCO detection, while requiring specific training-time adjustments (simulated quantization during training, so the network learns weights that are robust to the eventual precision reduction) to avoid the accuracy collapse that naive post-training quantization can cause [19]. Rastegari et al.'s XNOR-Net pushed precision reduction to its logical extreme, binarizing both weights and, in the XNOR-Net variant, activations as well, replacing floating-point multiply-accumulate operations with bitwise XNOR and popcount operations that are dramatically cheaper on standard CPU hardware; the paper reported roughly 58 times faster convolutional operations and 32 times memory savings relative to full-precision networks, at a real but bounded accuracy cost on ImageNet classification, and the resulting binary networks were fast enough to run classification in real time on a CPU rather than requiring a GPU [20]. Binarization this aggressive has not become the default choice for mainstream mobile deployment, largely because the accuracy gap at full 1000-class ImageNet scale remains substantial, but the technique defines one useful end of

the precision-versus-accuracy tradeoff curve that the rest of the quantization literature operates within.

Knowledge distillation offers a third, complementary compression mechanism that operates during training rather than after it. Hinton, Vinyals, and Dean's foundational distillation paper trains a small "student" network to match the softened output probability distribution of a larger, already-trained "teacher" network, rather than only the hard ground-truth labels, on the reasoning that the teacher's relative confidence across incorrect classes carries useful information about the structure of the problem that hard labels discard [21]. In the lightweight-CNN context, distillation is frequently used as a training-time booster layered on top of an already-small architecture — training a MobileNet-scale student against a larger, more accurate teacher tends to close part of the accuracy gap between them — rather than as a standalone compression technique on its own, and several of the architecture papers discussed elsewhere in this article report using distillation as one ingredient in their own training recipes. Table 1 summarizes the reported compression or efficiency figures from several of these techniques side by side, to make clear how differently each one is measured and how they are not directly interchangeable numbers.

5. Reparameterization and Hybrid CNN-Transformer Architectures

The most recent phase of this literature, roughly since 2021, has been shaped by the rise of vision transformers as a competing high-accuracy architecture family, and by a renewed interest in a design idea that is almost the opposite of the multi-branch complexity that characterized much of the preceding decade of CNN research. Dosovitskiy et al.'s original Vision Transformer paper demonstrated that a nearly unmodified transformer encoder, applied to a sequence of image patches rather than a sequence of word tokens, could match or exceed convolutional networks on ImageNet classification given sufficient pretraining data, establishing the transformer as a credible alternative to convolution for vision tasks rather than a curiosity [22]. Vision transformers are not, on their own, well suited to the tightest edge deployment targets — self-attention's computational cost scales quadratically with the number of tokens, and standard ViT implementations tend to have higher latency than comparably-accurate CNNs on mobile hardware — but the architectural ideas they introduced have influenced lightweight CNN design considerably, mostly by way of hybrid architectures that combine convolutional stages with a small number of attention-based stages.

Mehta and Rastegari's MobileViT is a representative example: rather than treating convolution and self-attention as competing choices, MobileViT uses convolutional layers for local spatial feature extraction and a lightweight transformer block, applied to spatially unfolded patches, for global feature mixing, combining the two within a single MobileNet-scale architecture rather than picking one paradigm exclusively [23]. The stated goal, and the paper's central empirical claim, is that this hybrid captures a meaningful share of the representational advantages transformers bring to vision tasks while retaining a parameter and latency budget suitable for mobile deployment, evaluated across classification, detection, and segmentation tasks. Maaz et al.'s EdgeNeXt pursues a similar hybrid goal with a different specific mechanism — a "split depth-wise transpose attention" encoder that applies self-attention along the channel dimension rather than the spatial dimension, which is substantially cheaper computationally — and reported 71.2% ImageNet top-1 accuracy from a model with only 1.3 million parameters, a favorable accuracy-per-parameter figure that the authors position explicitly against both pure-CNN and pure-transformer mobile baselines [24].

A second, structurally distinct recent idea is reparameterization: designing a network with a complex, multi-branch structure during training, where the extra branches help optimization and gradient flow, and then mathematically folding those branches into a single, simple structure for inference, so the deployed model pays none of the multi-branch structure's runtime cost. Ding et al.'s RepVGG is the architecture that established this idea clearly for the efficient-CNN literature: during training, each block is a multi-branch combination of a 3×3 convolution, a 1×1 convolution, and an identity shortcut, but because all three branches are linear operations, they can be algebraically merged into a single 3×3 convolution at inference time, producing a network whose deployed structure is literally "a stack of 3×3 convolutions and ReLU," a structure about as simple as VGG's but achieving considerably better speed-accuracy tradeoffs than VGG,

and competitive results against ResNet and EfficientNet variants on standard benchmarks [25]. Vasu et al.'s MobileOne and FastViT, both from Apple's on-device machine learning group, extend the reparameterization idea directly into the mobile deployment context. MobileOne applies training-time multi-branch reparameterization to a pure-CNN mobile architecture, explicitly targeting sub-millisecond inference latency on an iPhone 12 while remaining competitive with MobileNetV3-scale accuracy on ImageNet, detection, and segmentation benchmarks [26]. FastViT applies the same structural reparameterization principle to a hybrid CNN-transformer architecture, using reparameterizable convolutional blocks in place of some of the more expensive token-mixing operations that pure vision transformers rely on, and reporting favorable latency-accuracy tradeoffs specifically measured on-device rather than only in FLOPs [27]. Read together, RepVGG, MobileOne, and FastViT represent a coherent recent thread: rather than searching for or hand-designing a network that is simple and fast at both training and inference time simultaneously, let training-time structure be as complex as helps optimization, and pay its cost only once, offline, when folding it into the deployed network.

6. Open Challenges

Several recurring gaps are worth stating plainly rather than leaving implicit, because they shape how confidently this literature's numbers should be used in a real deployment decision. First, benchmarking inconsistency: FLOPs, parameter count, and on-device latency are all reported inconsistently across the papers synthesized here, and, as ShuffleNet V2's own central argument makes explicit, these three metrics do not reliably track each other [4]. A reader comparing two architectures purely by parameter count, without checking whether either paper also reports measured latency on comparable hardware, risks drawing the wrong practical conclusion. Sze et al.'s survey makes essentially the same point at the hardware level: because data movement, not arithmetic, tends to dominate energy and latency for these workloads, an architecture's true efficiency is inseparable from the specific memory hierarchy and dataflow of the chip it runs on, which means a single, hardware-agnostic efficiency number is somewhat of a fiction [3].

Second, the mobile-benchmark reference set itself is narrower than the field's stated ambitions. The overwhelming majority of the accuracy figures cited throughout this article come from ImageNet-1K classification, evaluated on a handful of reference phone chipsets that happened to be available to each paper's authors at the time — Pixel phones and specific Qualcomm or Apple silicon recur across many of these papers, but not consistently the same hardware from paper to paper, and genuinely resource-constrained microcontroller-class targets (the domain MCUNet addresses directly) receive comparatively little attention from the mainstream mobile-CNN architecture literature, which tends to assume a smartphone-class SoC as the deployment floor rather than an 8-bit or Cortex-M microcontroller with kilobytes, not megabytes, of RAM [28]. Lin et al.'s MCUNet is explicit that this is a genuinely different regime requiring co-designed architecture search and a purpose-built inference engine, not simply a smaller MobileNet, which suggests that "lightweight" as used across most of this literature and "lightweight enough for a microcontroller" are not the same claim and should not be conflated. Ray's broader review of the TinyML movement reaches a similar conclusion from a systems-survey perspective, noting that the great majority of deployments at true microcontroller scale remain inference-only, with the architecture, quantization scheme, and toolchain co-designed together rather than a mobile-scale CNN simply ported downward [29].

Third, the relationship between compression techniques (pruning, quantization, distillation) and architecture-search or hand-design techniques is under-characterized as a joint optimization problem. Most of the papers in this synthesis apply one or two of these techniques in isolation, or apply quantization as a fairly standard final step without deeply co-designing the architecture around it; there is comparatively little work rigorously establishing, for a fixed hardware and accuracy target, which combination of architecture choice, pruning ratio, quantization bit-width, and distillation strategy is actually Pareto-optimal, as opposed to which combination happened to be tried together in a given paper. Fourth, the CNN-versus-hybrid-transformer question remains genuinely open rather than settled: EfficientNetV2's direct comparison against a matched Vision Transformer configuration is one of the few controlled comparisons we found in this literature, and it favors the

CNN-based approach on both accuracy and training speed at the scale tested [14], while MobileViT, EdgeNeXt, and FastViT's own results suggest that some hybridization with attention mechanisms is worth its added complexity at the mobile scale specifically [23,24,27]. We do not think the literature currently supports a confident general answer to "should a new lightweight vision architecture in 2026 be a pure CNN, a pure transformer, or a hybrid," and readers should treat any strong claim in either direction with some skepticism until more controlled, matched-hardware comparisons exist.

Fifth, and more mundane but genuinely limiting for practitioners: very few of the papers synthesized here report energy consumption directly, as opposed to latency or FLOPs, despite energy being the actual binding constraint for battery-powered edge devices in many real deployments. Latency and energy are correlated but not identical, particularly once quantization and specialized accelerator hardware are involved, and the gap between "fast" and "energy-efficient" is a real one that this literature, taken as a whole, addresses less thoroughly than it addresses raw speed.

7. Conclusion

The literature reviewed here traces a genuinely cumulative architectural lineage rather than a series of disconnected proposals. Depthwise separable convolutions, established by Xception and operationalized for mobile deployment by MobileNet, remain the computational backbone of essentially every architecture discussed in this article, including the most recent hybrid and reparameterized designs [5,6]. The inverted residual and linear bottleneck ideas from MobileNetV2, the channel shuffle from ShuffleNet, and the hardware-aware neural architecture search that produced MnasNet and MobileNetV3 each addressed a specific, well-documented shortcoming of what preceded them, and each subsequent paper's reported gains were measured against, and generally validated relative to, that prior baseline [7,10,11,12]. Compound scaling and once-for-all supernets turned architecture selection itself into a more systematic search problem, while pruning, quantization, and distillation supply a complementary, largely orthogonal set of tools that compress an already-chosen architecture further, and modern deployment pipelines increasingly apply several of these tools together rather than choosing among them [13,15,9,19,21].

The most recent architectural developments — reparameterization, as demonstrated by RepVGG and extended to mobile deployment by MobileOne, and hybrid convolution-attention designs like MobileViT, EdgeNeXt, and FastViT — suggest the field has not settled into a single dominant paradigm even after a decade of concentrated effort, and that both "make the deployed network as structurally simple as possible" and "borrow the most useful parts of transformer attention while keeping convolutional efficiency" remain active, competing design philosophies with genuine empirical support behind each [25,26,23,24,27]. What the evidence does support with reasonable confidence is a general design principle that recurs across nearly every paper synthesized here: efficiency in deployment is a property of the joint system — architecture, numerical precision, and target hardware together — and not a property of an architecture considered in isolation, a point Sze et al. and the ShuffleNet V2 authors both make as an explicit methodological argument rather than an incidental observation [3,4]. Future architectures in this space, whatever specific mechanism they use, will likely continue to be judged by that same joint standard: not parameter count or FLOPs alone, but measured accuracy, latency, and energy consumption on the actual constrained hardware the model is meant to run on.

8. References

1. Simonyan, K., Zisserman, A. (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition. International Conference on Learning Representations (ICLR). DOI: 10.48550/arXiv.1409.1556

2. He, K., Zhang, X., Ren, S., Sun, J. (2016) Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.48550/arXiv.1512.03385

3. Sze, V., Chen, Y.-H., Yang, T.-J., Emer, J.S. (2017) Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE, 105(12): 2295-2329. DOI: 10.1109/JPROC.2017.2761740

4. Ma, N., Zhang, X., Zheng, H.-T., Sun, J. (2018) ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. Proceedings of the European Conference on Computer Vision (ECCV). DOI: 10.48550/arXiv.1807.11164

5. Chollet, F. (2017) Xception: Deep Learning with Depthwise

Separable Convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.48550/arXiv.1610.02357

6. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H. (2017) MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint. DOI: 10.48550/arXiv.1704.04861

7. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C. (2018) MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.48550/arXiv.1801.04381

8. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K. (2016) SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size. arXiv preprint. DOI: 10.48550/arXiv.1602.07360

9. Han, S., Mao, H., Dally, W.J. (2016) Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. International Conference on Learning Representations (ICLR). DOI: 10.48550/arXiv.1510.00149

10. Zhang, X., Zhou, X., Lin, M., Sun, J. (2017) ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.48550/arXiv.1707.01083

11. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V. (2019) MnasNet: Platform-Aware Neural Architecture Search for Mobile. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.1109/CVPR.2019.00293

12. Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H. (2019) Searching for MobileNetV3. Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). DOI: 10.48550/arXiv.1905.02244

13. Tan, M., Le, Q.V. (2019) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning (ICML), PMLR 97: 6105-6114. DOI: 10.48550/arXiv.1905.11946

14. Tan, M., Le, Q.V. (2021) EfficientNetV2: Smaller Models and Faster Training. Proceedings of the 38th International Conference on Machine Learning (ICML), PMLR 139: 10096-10106. DOI: 10.48550/arXiv.2104.00298

15. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S. (2020) Once-for-All: Train One Network and Specialize it for Efficient Deployment. International Conference on Learning Representations (ICLR). DOI: 10.48550/arXiv.1908.09791

16. Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C. (2020) GhostNet: More Features from Cheap Operations. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.48550/arXiv.1911.11907

17. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P. (2017) Pruning Filters for Efficient ConvNets. International Conference on Learning Representations (ICLR). DOI: 10.48550/arXiv.1608.08710

18. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C. (2017) Learning Efficient Convolutional Networks through Network Slimming. Proceedings of the IEEE International Conference on Computer Vision (ICCV). DOI: 10.48550/arXiv.1708.06519

19. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D. (2018) Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.48550/arXiv.1712.05877

20. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A. (2016) XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. Proceedings of the European Conference on Computer Vision (ECCV). DOI: 10.48550/arXiv.1603.05279

21. Hinton, G., Vinyals, O., Dean, J. (2015) Distilling the Knowledge in a Neural Network. arXiv preprint. DOI: 10.48550/arXiv.1503.02531

22. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N. (2021) An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. International Conference on Learning Representations (ICLR). DOI: 10.48550/arXiv.2010.11929

23. Mehta, S., Rastegari, M. (2022) MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. International Conference on Learning Representations (ICLR). DOI: 10.48550/arXiv.2110.02178

24. Maaz, M., Shaker, A., Cholakkal, H., Khan, S., Zamir, S.W., Anwer, R.M., Khan, F.S. (2022) EdgeNeXt: Efficiently Amalgamated CNN-Transformer Architecture for Mobile Vision Applications. Proceedings of the European Conference on Computer Vision Workshops (ECCVW). DOI: 10.48550/arXiv.2206.10589

25. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J. (2021) RepVGG: Making VGG-style ConvNets Great Again. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI:

10.48550/arXiv.2101.03697

26. Vasu, P.K.A., Gabriel, J., Zhu, J., Tuzel, O., Ranjan, A. (2023) MobileOne: An Improved One Millisecond Mobile Backbone. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.48550/arXiv.2206.04040

27. Vasu, P.K.A., Gabriel, J., Zhu, J., Tuzel, O., Ranjan, A. (2023) FastViT: A Fast Hybrid Vision Transformer using Structural Reparameterization. Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). DOI: 10.48550/arXiv.2303.14189

28. Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., Han, S. (2020) MCUNet: Tiny Deep Learning on IoT Devices. Advances in Neural Information Processing Systems (NeurIPS) 33. DOI: 10.48550/arXiv.2007.10319

29. Ray, P.P. (2022) A Review on TinyML: State-of-the-Art and Prospects. Journal of King Saud University - Computer and Information Sciences, 34(4): 1595-1623. DOI: 10.1016/j.jksuci.2021.11.019

9. Declarations

Funding

This literature synthesis was not supported by any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflict of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported here.

Data Availability Statement

This article is a literature synthesis and did not generate or analyse new primary data; all figures and comparisons are drawn from the cited publications, and readers should consult those original sources for their underlying data, code, and model availability.

Ethical Approval

Not applicable. This work involved no human participants, animal subjects, or collection of primary data, and required no ethical review.