Rewrite the Stars ⭐

Supplementary Material

This supplementary document elaborates on the implementation details of DemoNet, as showcased in Figure 1, Table 3, and Table 2. It also covers the simple network used in visualizing the decision boundary, illustrated in Figure 2, and the implementation of our StarNet presented in Table 6 (details can be found in Section A). Moreover, we provide a more granular view of decision boundary visualization in Section B. Sections C delves into our exploratory studies on ultra-compact models.The analysis of activations is presented in Sec. D. Additionally, a detailed examination of block design is discussed in Section E.

A. Implementation Details

A.1. Model Architecture

DemoNet In our isotropic DemoNet, featured in Fig. 1, Table 3, and Table 2, detailed implementation is provided in Algorithm 1. We adjust the depth or width values to facilitate the experiments showcased in the aforementioned figures and tables.

Algorithm 1 PyTorch codes of network illustrated in Fig. 1.

```
# demo-code for our DemoNet in Fig. 1.
# We build an isotropic network with depth 12.
# params: dim: width of network; mod: "star" or "sum".

import torch.nn as nn
from torch.nn import Sequential as Seq
import torch.nn.functional as F

class Blk(nn.Module):
  def __init__(self, dim, mod="sum"):
    super().__init__()
    self.mod=mod
    self.norm = nn.LayerNorm(dim)
    self.dwconv = nn.Conv2d(dim,dim,7,1,3,groups=dim)
    self.f = nn.Linear(dim, 6 * dim)
    self.g = nn.Linear(3*dim, dim)

    def forward(self, x):
      input = x
      x = self.dwconv(self.norm(x).permute(0,3,1,2))
      x = self.f(x.permute(0,2,3,1))
      x1, x2 = x.split(x.size(-1)//2, dim=-1)
      x = F.gelu(x1)+x2 if self.mod == "sum" \\
        else F.gelu(x1)*x2
      x = self.g(x)
      x = input + x
      return x


class DemoNet(nn.Module):
  def __init__(self, dim=128, mod="star", depth=12):
    super().__init__()
    self.num_classes = 1000
    self.stem = nn.Conv2d(3, dim, 16, 16)
    self.net=Seq(*[Blk(dim,mod) for _ in range(depth)])
    self.norm = nn.LayerNorm(dim)
    self.head = nn.Linear(dim, self.num_classes)

  def forward(self, x):
    x = self.net(self.stem(x).permute(0,2,3,1))
    return self.head(self.norm(x.mean([1, 2])))
```

DemoNet for 2D Points For the illustration of 2D points, as shown in Fig. 2, we have further simplified the DemoNet structure. In this adaptation, all convolutional layers within the original DemoNet have been removed. Additionally, we replaced the GELU activation function with ReLU to streamline the architecture. The details of this simplified DemoNet are outlined in Algorithm 2.

Algorithm 2 PyTorch codes of network illustrated in Fig. 2.

```
# demo-code for Network in Fig. 2.
# We build an isotropic network with depth 4, width
    100
# params: mod: "star" or "sum".

import torch.nn as nn
from torch.nn import Sequential as Seq
import torch.nn.functional as F

class Blk(nn.Module):
  def __init__(self, dim, mod="sum"):
    super().__init__()
    self.mod=mod
    self.norm = nn.LayerNorm(dim)
    self.f = nn.Linear(dim, 6 * dim)
    self.g = nn.Linear(3*dim, dim)

    def forward(self, x):
      input = x
      x = self.f(self.norm(x))
      x1, x2 = x.split(x.size(-1)//2, dim=-1)
      x = F.relu(x1)+x2 if self.mod == "sum" \\
        else F.relu(x1)*x2
      x = self.g(x)
      x = input + x
      return x


class DemoNet2D(nn.Module):
  def __init__(self, dim=100, mod="star", depth=4):
    super().__init__()
    self.num_classes = 2
    self.stem = nn.Linear(2, dim)
    self.net=Seq(*[Blk(dim,mod) for _ in range(depth)])
    self.norm = nn.LayerNorm(dim)
    self.head = nn.Linear(dim, self.num_classes)

  def forward(self, x):
    x = self.net(self.stem(x))
    return self.head(self.norm(x))
```

StarNet For ease of reproduction, we include a separate file in the supplementary materials dedicated to our StarNet. Detailed information regarding its architecture is also available in Section 4.1.

A.2. Training Recipes

We next provide detailed training recipe for each experiment.

DemoNet For all DemoNet variants, we utilize a consistent and standard training recipe. While it's acknowledged that specialized and finely-tuned training recipes could better suit different model sizes and potentially yield enhanced perfor-

mance, as in the cases of DemoNet(width=96, depth=12) (approximately 1.26M parameters) and DemoNet(width=288, depth=12) (approximately 9.68M parameters), achieving superior performance with DemoNet is not the primary goal of this work. Our objective is to provide a fair comparison among the various DemoNet variants; hence, the same training recipe is applied across all. Details of this training recipe for DemoNet are presented in Table 10.

config	value
image size	224
optimizer	AdamW [39]
base learning rate	4e-3
weight decay	0.05
optimizer momentum	β_1, β_2=0.9, 0.999
batch size	2048
learning rate schedule	cosine decay [38]
warmup epochs	5
training epochs	300
AutoAugment	rand-m9-mstd0.5-inc1 [11]
label smoothing [41]	0.1
mixup [64]	0.8
cutmix [62]	1.0
color jitter	0.4
drop path [31]	0.
AMP	False
ema	0.9998

Table 10. **DemoNet training setting.**

DemoNet for 2D Points Given the inherent simplicity of 2D points, we have eliminated all data augmentation processes and reduced the number of training epochs. The specific training recipe employed for this streamlined process is detailed in Table 11.

config	value
optimizer	SGD
base learning rate	0.1
minimal learning rate	0.005
learning rate schedule	cosine decay [38]
weight decay	2e-4
optimizer momentum	0.9
batch size	32
training epochs	30

Table 11. **Simplified DemoNet for 2D points training setting.**

StarNet Owing to its small model size and straightforward architectural design, StarNet necessitates fewer augmentations for training regularization. **Importantly, we opted not to use the commonly employed Exponential Moving Average (EMA) and learnable layer-scale technique** [54]. While these methods could potentially enhance performance, they may obscure the unique contributions of our work. The detailed training recipe for various StarNet models is provided in Table 12.

config	value
image size	224
optimizer	AdamW [39]
base learning rate	3e-3
weight decay	0.025
optimizer momentum	β_1, β_2=0.9, 0.999
batch size	2048
learning rate schedule	cosine decay [38]
warmup epochs	5
training epochs	300
AutoAugment	rand-m1-mstd0.5-inc1 [11]
label smoothing [41]	0.1
mixup [64]	0.8
cutmix [62]	0.2
color jitter	0.
drop path [31]	0.(S1/S2/S3), 0.1(S4)
AMP	False
EMA	None
layer-scale	None

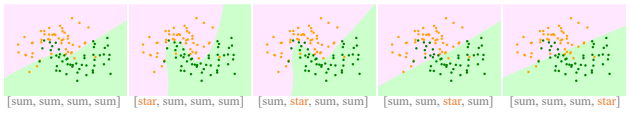
Table 12. **StarNet variants training setting.**

A.3. Latency Benchmark Settings

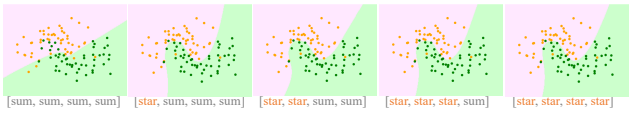
All models listed in Table 6 have been converted from Pytorch code to the ONNX format [13], to enable latency evaluations on different hardware: CPU (Intel Xeon CPU E5-2680 v4 @ 2.40GHz) and GPU (P100). We have conducted benchmarks with a batch size of one, mirroring real-world application scenarios. The benchmark involves a warm-up of 50 iterations, followed by the computation of the average latency over 500 iterations. It's important to note that all models were benchmarked on the same device to ensure fairness in comparisons. For CPU benchmarks, we utilize 4 threads to optimize performance. In the case of GPU evaluations, we've adapted the pointwise convolutional layer in StarNet to a linear layer, incorporating a permutation operation. This modification was prompted by observations of marginally faster inference speeds. It's important to note that, mathematically, a pointwise convolutional layer is equivalent to a linear layer. Therefore, this change does not lead to any variances in performance or alterations in the architecture of our StarNet.

Thanks to MobileOne [55], we have utilized their open-sourced iOS benchmark application for all CoreML models. Our latency benchmark settings follow those used in Mo-

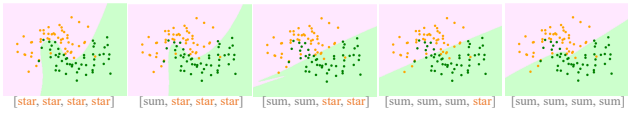
Figure 5. 4-run results of sum and star operations.



(a) Replace sum with star for single block, from first to last.



(b) Progressively replace sum with star, from first to last.



(c) Progressively replace star with sum, from first to last.

Figure 6. More comprehensive analyses on decision boundary.

bileOne, with the only difference being the inclusion of additional models in our tests. We have observed that the initial run of the iOS benchmark consistently yields slightly faster results. To account for this, each model is run three times, and we report the latency of the final run. Although there are minor latency variations when testing on iPhone, these discrepancies (typically less than 0.05 ms) do not affect our analysis.

B. Decision Boundary Visualization

We provide more analyses for the decision boundary visualization, as shown in Fig. 2.

Firstly, as a supplement to Fig. 2, we present more comprehensive results. The decision boundary can exhibit significant variation due to randomness in different tests. To illustrate this, we display the decision boundaries from an additional four runs (with no fixed seed) in Fig. 5. As evidenced, the star operation not only surpasses the summation operation in representational capability, but it also demonstrates greater robustness, exhibiting minimal variance.

Next, our investigation extends to a more in-depth analysis of the decision boundary. Considering the network has 4 blocks, we experimented with a mix of summation and star operations, applying them in various combinations as

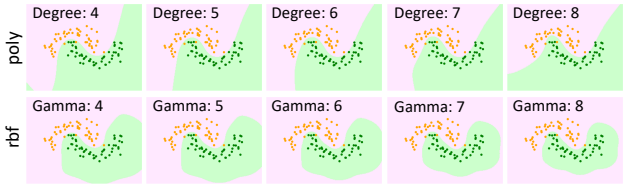


Figure 7. Adjusting hyper-parameters for rgb-SVM and poly-SVM.

Variant	width	depth	expand	Params	FLOPs
StarNet-050	16	[1, 1, 3, 1]	3	0.54M	92.8M
StarNet-100	20	[1, 2, 4, 1]	4	1.04M	187.1M
StarNet-150	24	[1, 2, 4, 2]	3	1.56M	229.0M

Table 13. **Configurations of very small StarNets**. We vary the embed width, depth, and MLP expansion ratio to StarNets at 0.5M, 1.0M, and 1.5M parameters.

demonstrated in Figure 6. The visual results from this exploration suggest that employing star operations in the early blocks of the network yields the most significant benefits.

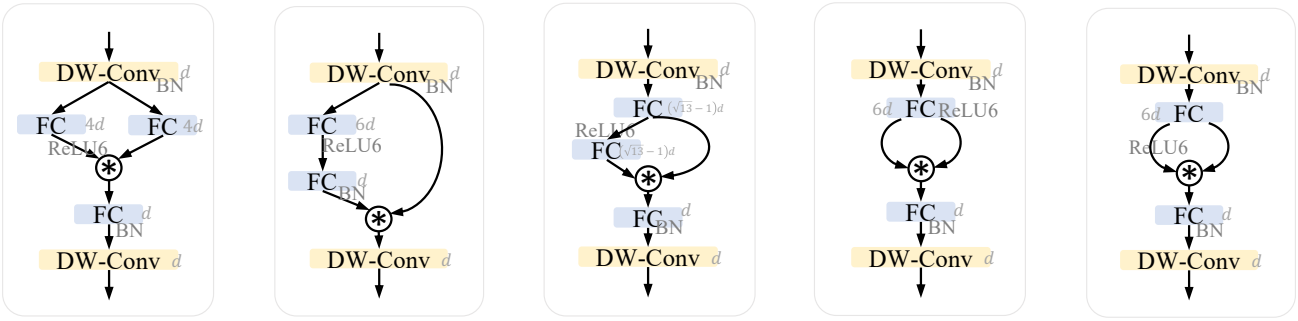
Impact of hyper-parameters in SVM. It should be noted that parameters adjustments will lead to different visualization results, which may challenge our claim based on Fig. 2. However, the change is not significant due to the intrinsic differences between ploy and rbf kernels, as shown in Fig. 7. Our star-based network decision boundary is consistently similar to the decision boundary of poly-SVM. Hence, not only theoretical proof, our visual experiments also hold firmly.

C. Exploring Extremely Small Models

In this section, we explore the performance of StarNet under extremely small parameters (around 0.5M, 1.0M, and 1.5M). For these extremely small variants, we further tune the MLP expansion ratio besides block number and base embedding width. The detailed configurations of these very small variants are presented in Table 13.

In this section, we delve into the performance of StarNet when configured with extremely small number of parameters, specifically around 0.5M, 1.0M, and 1.5M. For these ultra-compact variants, we go beyond just adjusting the block number and base embedding width; we also finely tune the MLP expansion ratio. The specific configurations for these very small StarNet variants are detailed in Table 13.

The results presented in Table 14 demonstrate that our ultra-compact variants of StarNet are also promising in performance. When compared to MobileNetv2-050, which is trained for longer training period and introduces approximately 25% more parameters (1.97M *vs.* 1.56M), our StarNet-150 variant still outperforms in both top-1 accuracy and speed on mobile devices.



(a) Block I (default) (b) Block II (c) Block III (d) Block IV (e) Block V

Figure 8. We provide more studies on the block design of StarNet. **Block I** is the default design used in our StarNet, and the standard star operation as we discussed. **Block II** and **Block III** can be considered as instantiations of special case II as presented in Sec.3.3. **Block IV** and **Block V** can be considered as instantiations of special case III. We vary the expansion ratio to ensure all the block variants have same parameters and FLOPs for fair comparison. Skip connection is ignored for simplicity.

Model	Top-1 (%)	Params (M)	FLOPs (M)	Latency (ms)		
				Mobile	GPU	CPU
MobileNetv2-050*	66.0	1.97	104.5	0.7	1.4	1.5
StarNet-050	**55.7**	0.54	92.8	**0.5**	1.0	1.3
StarNet-100	**64.3**	1.04	187.1	**0.6**	1.3	2.3
StarNet-150	**68.0**	1.56	229.0	**0.6**	1.5	2.6

Table 14. **Performance of extremely small StarNet variants on ImageNet-1k**. All our StarNet are trained with 300 epochs following the training recipe of StarNet-S1. MobileNetv2-050* is taken from the timm library [59], trained with 450 epochs.

Width	96	128	160	192	224	256	288
w/ act.	57.6	64.0	68.2	71.7	73.9	75.3	76.1
w/o act.	57.6	64.0	67.5	70.5	73.0	75.3	75.7
acc. gap	-	-	0.7↓	1.2↓	0.9↓	-	0.4↓

Table 15. **Comparison of removing all activations in DemoNet (using star operation) with different widths.** We set the depth to 12. We gradually increase the width by a step of 32.

Depth	10	12	14	16	18	20	22
w/ act.	70.3	71.8	72.9	72.9	73.9	75.4	75.4
w/o act.	69.4	70.5	72.6	73.5	74.2	74.7	75.5
acc. gap	0.9↓	1.3↓	0.3↑	0.6↑	0.3↑	0.7↓	0.1↑

Table 16. **Comparison of removing all activations in DemoNet (using star operation) with different depths.** We set the width to 192. We gradually increase the depth by a step of 2.

Act.	ReLU	GELU	LeakyReLU	HardSwish	ReLU6
	$\max(0, x)$	$\frac{x}{2}\left(1+\text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$	$\max(0.01x, x)$	$x\frac{\text{ReLU6}(x+3)}{6}$	$\min(\max(0, x), 6)$
Acc.	77.6	77.8	77.7	77.7	**78.4**

Table 17. **Performance of different activations in StarNet-S4.**

D. Activation Analysis

D.1. Analysis on removing all activations

In Sec. 3.5, we explored the possibility of eliminating all activation functions, presenting preliminary findings in Table 4 and Table 9. This section delves deeper into the **detailed analyses of removing all activations**. Utilizing DemoNet as our model for illustration, we provide further results in Table 15 and Table 16.

In most cases, we observe slightly performance drop when removing all activation from DemoNet (using star operation). However, an intriguing observation emerged: in some instances, the removal of all activations resulted in similar or even better performance. This was notably evident when the depth values were set to 14, 16, 18, and 22, as detailed in Table 16. This phenomenon implies that the star operation may inherently provide sufficient non-linearity, akin to what is typically achieved through activation layers. We believe that a more thorough investigation in this direction could yield valuable insights.

D.2. Exploring activation types

In our StarNet design, we adopt the ReLU6 activation function, following MobileNetv2. Additionally, we have experimented with various other activation functions, with the results of these explorations presented in Table 17. Empirically, we found that StarNet-S4 delivers the best performance when equipped with the ReLU6 activation function.

E. Exploring Block Designs

We provide detailed ablation studies on the design of StarNet block. To this end, we present five implementation of star operation in StarNet, as illustrated in Fig. 8. Of note is that Block I can be considered as a standard implementation of star operation, Block II and Block III can be considered as instantiations of special case II, which is discussed in Sec.3.3, Block IV and Block V can be considered as different imple-

Star Op.	Design	Top-1
Standard	Block I	78.4
Case II	Block II	74.4
	Block III	74.4
Case III	Block IV	78.5
	Block V	**78.6**

Table 18. Performance of different block designs. The detailed implementation of each block can be found in Fig. 8.

mentations of special case III. All the block variants have same parameters and FLOPs via varying expansion ratio, ensuring same computational complexity for fair comparison. We test all these block variants based on StarNet-S4 architecture, and report the performance in Table 18. Empirically, we see strong performance for Block I, Block IV, and Block V, while Block II and Block III perform worse. The detailed results suggest that the strong performance stems from the star operation rather than specific block design. We consider the default star operation (Block I) as the essential building block in our StarNet.

We have conducted detailed ablation studies on the design of the StarNet block. To this end, we explored five different implementations of the star operation in StarNet, as depicted in Fig.8. Notably, Block I represents a standard implementation of the star operation. Blocks II and III are instantiations of the special case II, discussed in Sec.3.3, while Blocks IV and V offer alternative takes on special case III. All these block variants maintain the same number of parameters and FLOPs by adjusting the expansion ratio, ensuring same computational complexity for a fair comparison. These block variants were tested within the StarNet-S4 architecture framework, with their performance detailed in Table18. Empirically, Blocks I, IV, and V demonstrated strong performance, with minimal performance gap. These findings suggest that the effectiveness is more attributable to the star operation itself rather than to the specific design of the blocks. Therefore, we use the default star operation (Block I) as the foundational block in our StarNet.

F. More Latency Analysis on StarNet

CPU latency Visualization. To better understand the latency of the proof-of-concept model StarNet, we further plot the CPU-latency trade-off in Fig. 9.

Mobile Device latency Robustness. In Table 6, we presented the mobile device latency, which is tested on a iPhone13 mobile phone. We further conducted latency testing on 4 different mobile devices, including iPhone12, iPhone12 Pro Max, iPhone 13, and iPhone14, to test the latency stability of different models. Results in Tab. 19 demonstrate that, despite varying latency results for some models, StarNet always shows stable inference across different phones, attributed to its simple design.

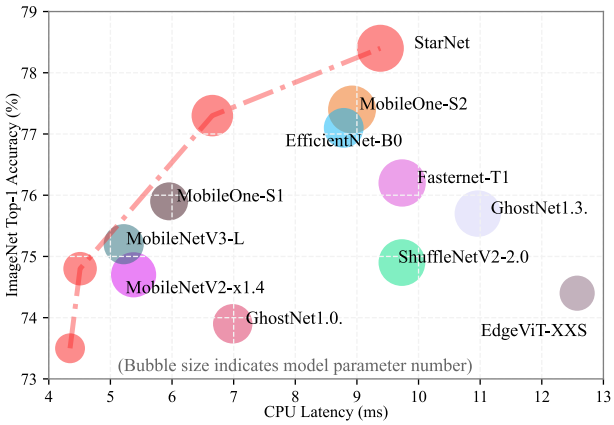


Figure 9. CPU Latency vs. ImageNet Accuracy. Low-accuracy or high-latency models are removed for better visualization.

Table 19. **Supplementary of Table 6**. We further test model latency on four iPhone devices, including iPhone12, iPhone12Pro Max, iPhone13, and iPhone14. The average latency and variance are reported in the last column. We mark most stable models in green color and most unstable models in red.

Model	**Top-1**	**iPhone devices Latency (ms)**				
	(%)	**12**	**12PM**	**13**	**14**	**avg±std**
MobileOne-S0	71.4	0.7	0.7	0.7	0.7	0.7±0.02
ShuffleV2-1.0	69.4	4.1	4.1	0.8	0.8	2.4±1.66
MobileV3-S0.75	65.4	5.5	5.3	6.5	6.6	6.0±0.59
GhostNet0.5	66.2	10.0	7.3	9.7	8.8	8.9±1.05
MobileV3-S	67.4	6.5	5.8	7.5	7.7	6.9±0.75
StarNet-S1	**73.5**	0.7	0.7	0.7	0.7	0.7±0.00
MobileV2-1.0	72.0	0.9	0.9	1.0	0.9	0.9±0.04
ShuffleV21.5	72.6	5.9	5.9	1.2	1.2	3.5±2.34
M-Former-52	68.7	6.6	6.3	8.3	8.2	7.4±0.93
FasterNet-T0	71.9	0.7	0.7	0.8	0.7	0.7±0.02
StarNet-S2	**74.8**	0.7	0.8	0.8	0.8	0.8±0.01
MobileV3-L0.75	73.3	10.9	11.4	12.4	14.2	12.2±1.26
EdgeViT-XXS	74.4	1.8	1.8	1.2	1.2	1.5±0.31
MobileOne-S1	75.9	0.9	0.9	0.9	0.9	0.9±0.03
GhostNet1.0	73.9	7.9	7.4	10.1	10.2	8.9±1.28
EfficientNet-B0	77.1	1.6	1.5	1.7	1.7	1.6±0.10
MobileV3-L	75.2	11.4	12.9	15.1	14.9	13.6±1.52
StarNet-S3	**77.3**	0.9	0.9	0.9	0.9	0.9±0.02
EdgeViT-XS	77.5	3.5	3.5	1.6	1.6	2.5±0.96
MobileV2-1.4	74.7	1.1	1.2	1.3	1.3	1.2±0.08
GhostNet1.3	75.7	9.7	9.0	12.4	12.5	10.9±1.57
ShuffleV2-2.0	74.9	19.9	16.9	1.8	1.5	10.0±8.44
Fasternet-T1	76.2	0.9	0.9	1.0	1.0	1.0±0.03
MobileOne-S2	77.4	1.0	1.1	1.2	1.2	1.1±0.07
StarNet-S4	**78.4**	1.0	1.0	1.1	1.1	1.1±0.03