The Trade-off Between Compilation and Execution of Unfused Operations

Leonardo Kamei, Pedro Henrique Ramos

PUC Minas

8 de outubro de 2026

Motivação e objetivos

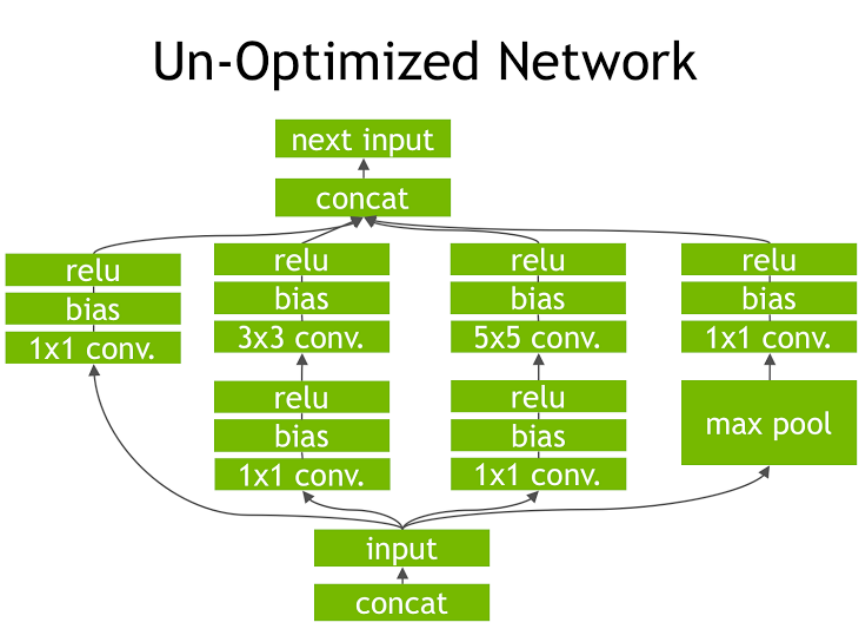
- Frameworks de DL de alto nível unificam aceleradores e automatizam a criação de modelos.
- Compilador embutido aumenta **tempo de compilação** e diminui **tempo de execução**.
- Compilar só compensa com execuções suficientes, e n depende da aplicação:

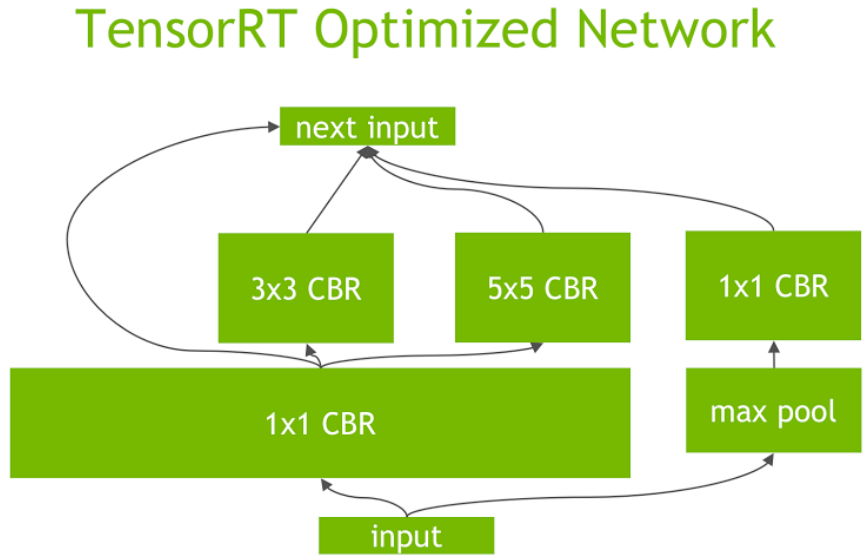
Cenário	n	**Por que importa**
P&D automatizado em IA (RSI), NAS	pequeno	Compilação domina o ciclo
LLMs com formas dinâmicas	variável	Recompilação paga b_b de novo
Edge e NPUs embarcadas	grande	Otimização agressiva se paga

Objetivo: medir o trade-off **compilação** $\times$ **execução** da **fusão de operadores** em frameworks e propor $T_b(n) = a_b n + b_b$ para escolher o backend.

Fusão de operadores

- Unir 2+ operações em um único kernel para:
  - reduzir a comunicação CPU–GPU;
  - reduzir sincronizações;
  - evitar materializar tensores intermediários.
- Condições para a fusão:
  - dependência e ordem dos dados;
  - compatibilidade de formas;
  - tipo de operação (elementwise, convolução, matmul, . . .);
  - restrições do hardware/backend.





Metodologia

- **Backends**: PyTorch 2/TorchInductor, XLA (jaxlib 0.6.2) e TVM 0.22.dev0, na mesma máquina e sistema operacional, com CUDA 12.6.
- **Modelos** (inferência):
  - ResNet-18/50;
  - BERT e GPT-2.
- **Protocolo**: $K = 5$ processos frios isolados por (modelo, entrada, backend); 10 aquecimentos + 50 execuções cronometradas e sincronizadas; caches de compilador privados por processo.

Resnets

		Compilação (ms)		Execução (ms)		Unidades (int. + ext.)	
Modelo	**Entrada**	Inductor	XLA	Inductor	XLA	Inductor	XLA
ResNet-18	(1,3,224,224)	11765	**1829**	**1,88**	2,13		
ResNet-18	(64,3,224,224)	13116	**4560**	42,43	**38,71**	18 + 21	**8 + 20**
ResNet-18	(1,3,1024,1024)	16502	**2663**	15,03	**12,94**		
ResNet-50	(1,3,224,224)	24424	**3503**	**3,78**	4,19		
ResNet-50	(64,3,224,224)	21689	**9352**	112,51	**105,22**	50 + 54	**11 + 53**
ResNet-50	(1,3,1024,1024)	23834	**5011**	40,19	**34,54**		

int. = geradas pelo backend, ext. = cuDNN/cuBLAS.

Ambos utilizam de bibliotecas externas. XLA utiliza uma custom-call que reduz tanto a granularidade quanto o codegen.

CNNs: onde a fusão acontece

```
Backend: hlo
  Kernels totais: 25
  - Fusions: 5 (únicos: 5)
    por kind: kLoop: 2, kInput: 2, kCustom: 1
  - Custom-calls: 20 (targets únicos: 1)
  Custom-call targets:
    - __cudnn$convBiasActivationForward → 20

=== Resumo agregado ===
  hlo_custom_total: 20
  hlo_fusion_total: 5
  hlo_kernels_totais: 25
```

(a) XLA (HLO): Conv + BN + ReLU dentro de uma única *custom call*, com epílogo fundido da biblioteca.

```
Backend: triton
  Triton launches: 19 (kernels únicos: 14)
  Detalhes Triton:
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_2 → 2
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_3 → 2
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_4 → 2
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_7 → 2
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_10 → 2
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_0 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_max_pool2d_with_indices_relu_1 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_5 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_6 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_8 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_9 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_11 → 1
    - triton_red_fused__native_batch_norm_legit_no_training_add_mean_relu_12 → 1
    - triton_per_fused__native_batch_norm_legit_no_training_add_mean_relu_13 → 1
  extern_kernels calls: 21 (funções únicas: 2)
  Detalhes extern_kernels:
    - convolution → 20
    - addmm → 1
```

(b) TorchInductor (Triton): convolução no cuDNN; BN + ReLU em kernels Triton à parte.

Transformers

		Compilação (ms)		**Execução (ms)**		**Unidades (int. + ext.)**	
Modelo	**Entrada**	Inductor	XLA	Inductor	XLA	Inductor	XLA
BERT	(1,128)	9033	**2360**	8,07	8,02	**49 + 72**	135 + 72
BERT	(8,128)	10466	**2779**	41,76	41,99		
GPT-2	(1,128)	8939	**1854**	**7,29**	7,65	**37 + 48**	87 + 72
GPT-2	(8,128)	9677	**2555**	**39,79**	42,24		

No BERT, o Inductor emite apenas **4 kernels Triton distintos**, reutilizados nas 12 camadas; o XLA gera 135 fusões.

Fold Conv–BN no TorchInductor

Hipótese: o Inductor paga caro na compilação pelos kernels Triton extras em volta da convolução.
Experimento: fold offline dos parâmetros **BN→Conv** em modo `eval()`, antes da captura do grafo (custo de pré-processamento: 100–120 ms).
Isso não irá **diminuir a quantidade de kernels**, mas sim a **geração de codegen**.

Modelo	Entrada	Comp. sem fold	Comp. com fold	Δ Comp.	Exec. sem fold	Exec. com fold
ResNet-18	(1,3,224,224)	11765	6929	−41,1%	1,88	1,86
ResNet-18	(64,3,224,224)	13116	9246	−29,5%	42,43	42,26
ResNet-18	(1,3,1024,1024)	16502	10187	−38,3%	15,03	14,89
ResNet-50	(1,3,224,224)	24424	17427	−28,6%	3,78	3,71
ResNet-50	(64,3,224,224)	21689	21280	−1,9%	112,51	110,15
ResNet-50	(1,3,1024,1024)	23834	21031	−11,8%	40,19	38,91

- Compilação: −29,5% a −41,1% (ResNet-18) e −1,9% a −28,6% (ResNet-50), sem piorar a execução.

Visão macro do fold

```
Backend: triton
  Triton launches: 50 (kernels únicos: 18)
  Detalhes Triton:
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_10 → 11
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_6 → 7
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_2 → 6
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_12 → 5
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_14 → 5
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_8 → 3
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_4 → 2
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_0 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_max_pool2d_with_indices_relu_1 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_3 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_5 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_7 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_9 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_11 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_relu_13 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_15 → 1
    - triton_poi_fused__native_batch_norm_legit_no_training_add_relu_16 → 1
    - triton_red_fused__native_batch_norm_legit_no_training_add_mean_relu_17 → 1
  extern_kernels calls: 54 (funções únicas: 2)
  Detalhes extern_kernels:
    - convolution → 53
    - addmm → 1
```

(a) Antes do fold

```
Backend: triton
  Triton launches: 50 (kernels únicos: 18)
  Detalhes Triton:
    - triton_poi_fused_convolution_relu_10 → 11
    - triton_poi_fused_convolution_relu_6 → 7
    - triton_poi_fused_convolution_relu_2 → 6
    - triton_poi_fused_add_convolution_relu_12 → 5
    - triton_poi_fused_convolution_relu_14 → 5
    - triton_poi_fused_add_convolution_relu_8 → 3
    - triton_poi_fused_add_convolution_relu_4 → 2
    - triton_poi_fused_convolution_relu_0 → 1
    - triton_poi_fused_convolution_max_pool2d_with_indices_relu_1 → 1
    - triton_poi_fused_add_convolution_relu_3 → 1
    - triton_poi_fused_convolution_relu_5 → 1
    - triton_poi_fused_add_convolution_relu_7 → 1
    - triton_poi_fused_convolution_relu_9 → 1
    - triton_poi_fused_add_convolution_relu_11 → 1
    - triton_poi_fused_convolution_relu_13 → 1
    - triton_poi_fused_add_convolution_relu_15 → 1
    - triton_poi_fused_add_convolution_relu_16 → 1
    - triton_red_fused_add_convolution_mean_relu_17 → 1
  extern_kernels calls: 54 (funções únicas: 2)
  Detalhes extern_kernels:
    - convolution → 53
    - addmm → 1
```

(b) Depois do fold

Visão micro do fold

```
@triton.jit
def triton_(in_out_ptr0, in_ptr0, in_ptr1, in_ptr2, in_ptr3, xnumel, XBLOCK : tl.constexpr):
    xnumel = 16777216
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = tl.full([XBLOCK], True, tl.int1)
    x2 = xindex
    x0 = xindex % 64
    tmp0 = tl.load(in_out_ptr0 + (x2), None)
    tmp1 = tl.load(in_ptr0 + (x0), None, eviction_policy='evict_last')
    tmp3 = tl.load(in_ptr1 + (x0), None, eviction_policy='evict_last')
    tmp12 = tl.load(in_ptr2 + (x0), None, eviction_policy='evict_last')
    tmp14 = tl.load(in_ptr3 + (x0), None, eviction_policy='evict_last')
    tmp2 = tmp0 - tmp1
    tmp4 = 1e-05
    tmp5 = tmp3 + tmp4
    tmp6 = libdevice.sqrt(tmp5)
    tmp7 = tl.full([1], 1, tl.int32)
    tmp8 = tmp7 / tmp6
    tmp9 = 1.0
    tmp10 = tmp8 * tmp9
    tmp11 = tmp2 * tmp10
    tmp13 = tmp11 * tmp12
    tmp15 = tmp13 + tmp14
    tmp16 = tl.full([1], 0, tl.int32)
    tmp17 = triton_helpers.maximum(tmp16, tmp15)
    tl.store(in_out_ptr0 + (x2), tmp17, None)
''', device_str='cuda')
```

(a) Antes do fold

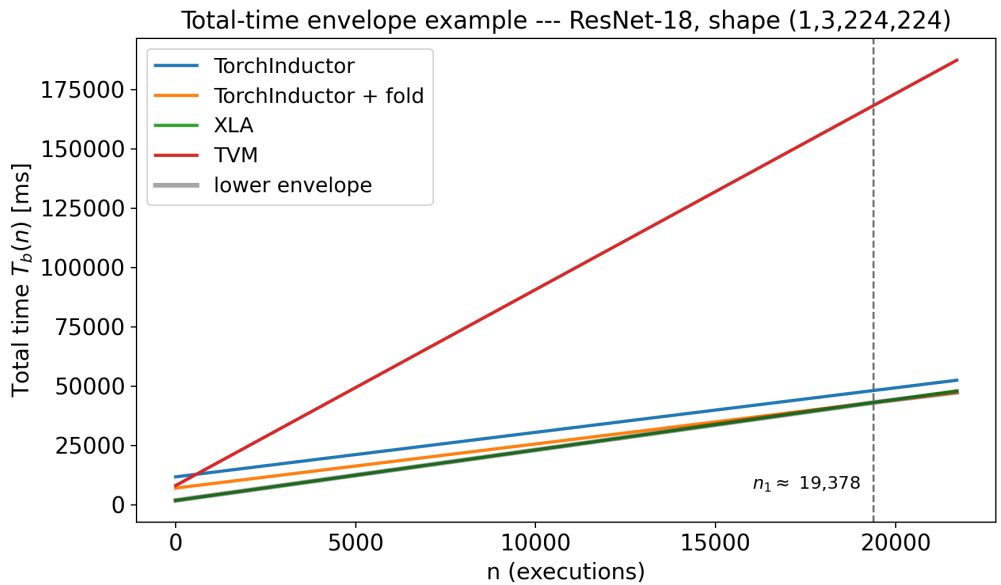
```
@triton.jit
def triton_poi_fused_convolution_relu_0(in_out_ptr0, in_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 16777216
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = tl.full([XBLOCK], True, tl.int1)
    x2 = xindex
    x0 = (xindex % 64)
    tmp0 = tl.load(in_out_ptr0 + (x2), None)
    tmp1 = tl.load(in_ptr0 + (x0), None, eviction_policy='evict_last')
    tmp2 = tmp0 + tmp1
    tmp3 = tl.full([1], 0, tl.int32)
    tmp4 = triton_helpers.maximum(tmp3, tmp2)
    tl.store(in_out_ptr0 + (x2), tmp4, None)
''', device_str='cuda')
```

(b) Depois do fold

Modelo $T_b(n)$: qual backend para n execuções?

$T_b(n) = a_b n + b_b = T_C + n\,T_X$

- $a_b = T_X$: latência por execução;
- $b_b = T_C$: custo de compilação.
- O **envelope inferior** das retas indica o melhor backend para cada n:
  - poucas execuções $\rightarrow$ vence a compilação mais barata;
  - muitas execuções $\rightarrow$ vence o menor a_b.
- Com o fold, $b_f < b_b$ e $a_f \leq a_b$ nas dez entradas das ResNets: o fold domina desde $n = 0$.



ResNet-18 em $(1, 3, 224, 224)$: XLA tem o menor custo até $n \approx 19\,378$; depois, o Inductor com fold.

Conclusões e trabalhos futuros

Conclusões

- Análise da fusão de operadores pelas IRs.
- O **fold BN→Conv** reduz a compilação do Inductor em 9 de 10 entradas, mesmo pagando o pré-processamento.
- O backend preferido muda com n: dependendo da aplicação (P&D automatizado, formas dinâmicas, co-design), compilar menos vale mais do que executar mais rápido. Granularidade e custo de compilação não preveem a ordem de execução.

Limitações: uma única máquina e versões específicas de CUDA/cuDNN; o modelo linear ignora caches, autotuning e recompilações por formas dinâmicas.

Trabalhos futuros

- Especialização com pesos congelados: fold BN→Conv + epílogo ReLU via cuDNN, com fallback seguro.
- Passe de casamento de padrões para Conv→BN→ReLU.

Slides de apoio

Apoio: pilha CUDA / cuDNN / Python

Framework	**Python**	**Compilador / runtime**	**CUDA / cuDNN**
TorchInductor (ResNets)	3.10.18	PyTorch 2	CUDA 12.6 / cuDNN 9.8
XLA (ResNets)	3.10.18	jaxlib 0.6.2	CUDA 12.6 / cuDNN 9.8

Campanha	**Modelo**	**TorchInductor**	**XLA**
Pilhas nativas	ResNet-18/50	TF32	TF32
	BERT, GPT-2	FP32	FP32
Fold (Conv–BN)	ResNet-18/50	TF32	—
Fold (LN–Linear)	BERT	FP32	—

Apoio: ResNet-18 — tabela completa

Framework	Entrada (N,C,H,W)	Compilação (ms)	Execução (ms)
TorchInductor	(1,3,224,224)	11765 ± 1036	**1,88 ± 0,03**
TorchInductor	(16,3,224,224)	14958 ± 907	11,86 ± 0,16
TorchInductor	(64,3,224,224)	13116 ± 791	42,43 ± 0,13
TorchInductor	(1,3,512,512)	15544 ± 1364	4,85 ± 0,08
TorchInductor	(1,3,1024,1024)	16502 ± 1110	15,03 ± 0,17
XLA	(1,3,224,224)	**1829 ± 12**	2,13 ± 0,04
XLA	(16,3,224,224)	**2408 ± 27**	11,61 ± 0,05
XLA	(64,3,224,224)	**4560 ± 100**	**38,71 ± 0,12**
XLA	(1,3,512,512)	**2599 ± 54**	**4,23 ± 0,02**
XLA	(1,3,1024,1024)	**2663 ± 76**	**12,94 ± 0,01**

Média ± desvio sobre $K = 5$ processos frios. **Negrito** = menor média cujo IC de 95% não se sobrepõe ao do concorrente.

Apoio: ResNet-50 — tabela completa

Framework	Entrada (N,C,H,W)	Compilação (ms)	Execução (ms)
TorchInductor	(1,3,224,224)	24424 ± 1539	**3,78 ± 0,08**
TorchInductor	(16,3,224,224)	22519 ± 1094	31,09 ± 0,11
TorchInductor	(64,3,224,224)	21689 ± 586	112,51 ± 0,24
TorchInductor	(1,3,512,512)	23221 ± 1411	11,85 ± 0,09
TorchInductor	(1,3,1024,1024)	23834 ± 548	40,19 ± 0,14
XLA	(1,3,224,224)	**3503 ± 85**	4,19 ± 0,05
XLA	(16,3,224,224)	**4800 ± 33**	**29,29 ± 0,13**
XLA	(64,3,224,224)	**9352 ± 121**	**105,22 ± 0,26**
XLA	(1,3,512,512)	**4013 ± 27**	**10,38 ± 0,04**
XLA	(1,3,1024,1024)	**5011 ± 73**	**34,54 ± 0,02**

Apoio: BERT e GPT-2 — tabelas completas

BERT

Framework	Entrada	Compilação (ms)	Execução (ms)
Inductor	(1,64)	9076 ± 259	**4,95 ± 0,06**
Inductor	(1,128)	9033 ± 262	8,07 ± 0,06
Inductor	(8,128)	10466 ± 759	41,76 ± 0,03
XLA	(1,64)	**2401 ± 72**	5,77 ± 0,09
XLA	(1,128)	**2360 ± 44**	8,02 ± 0,20
XLA	(8,128)	**2779 ± 109**	41,99 ± 0,31

GPT-2

Framework	Entrada	Compilação (ms)	Execução (ms)
Inductor	(1,16)	9389 ± 510	**3,20 ± 0,11**
Inductor	(1,128)	8939 ± 187	**7,29 ± 0,04**
Inductor	(8,128)	9677 ± 242	**39,79 ± 0,15**
Inductor	(8,256)	9621 ± 148	**82,78 ± 0,10**
XLA	(1,16)	**1729 ± 99**	5,03 ± 0,10
XLA	(1,128)	**1854 ± 29**	7,65 ± 0,06
XLA	(8,128)	**2555 ± 15**	42,24 ± 0,05
XLA	(8,256)	**2629 ± 11**	86,24 ± 0,15

Apoio: Transformers — análise qualitativa

TorchInductor

- Delega todos os GEMMs ao cuBLAS: 72 chamadas externas, uma por projeção linear das 12 camadas do encoder (BERT).
- Emite só 4 kernels Triton distintos, reutilizados nas 12 camadas: embedding+norma, residual+norma, projeção+ativação e o corpo da atenção.
- No GPT-2, mesma estratégia: epílogos estendidos que reúnem normalização, ativação e softmax; menos unidades, porém kernels grandes e complexos.

XLA

- BERT: 207 unidades = 135 fusões (12 de atenção, 25 de redução para LayerNorm, 98 de laço) + 72 GEMMs via *custom call*.
- GPT-2: 87 fusões + 72 chamadas `__cublas$gemm`, a mesma divisão vista no BERT.

Apoio: fold Conv–BN — tabela completa

Modelo	Entrada	Comp. sem fold	Comp. com fold	Δ Comp.	Exec. sem fold	Exec. com fold
ResNet-18	(1,3,224,224)	11765 ± 1036	6929 ± 235	−41,1%	1,88	1,86
ResNet-18	(16,3,224,224)	14958 ± 907	9026 ± 214	−39,7%	11,86	11,79
ResNet-18	(64,3,224,224)	13116 ± 791	9246 ± 215	−29,5%	42,43	42,26
ResNet-18	(1,3,512,512)	15544 ± 1364	10201 ± 1132	−34,4%	4,85	4,80
ResNet-18	(1,3,1024,1024)	16502 ± 1110	10187 ± 328	−38,3%	15,03	14,89
ResNet-50	(1,3,224,224)	24424 ± 1539	17427 ± 661	−28,6%	3,78	3,71
ResNet-50	(16,3,224,224)	22519 ± 1094	19700 ± 475	−12,5%	31,09	30,63
ResNet-50	(64,3,224,224)	21689 ± 586	21280 ± 188	−1,9%	112,51	110,15
ResNet-50	(1,3,512,512)	23221 ± 1411	19435 ± 295	−16,3%	11,85	11,39
ResNet-50	(1,3,1024,1024)	23834 ± 548	21031 ± 273	−11,8%	40,19	38,91

Custo de pré-processamento do fold: 100–106 ms (ResNet-18) e 114–120 ms (ResNet-50). Cruzamento entre variantes: $n_{\text{cross}} = (b_b - b_f)/(a_f - a_b)$, para $a_f \neq a_b$.