Teste de escalabilidade Forte:

Parâmetros setados:

"num_epochs": 1, → para facilitar os testes

"batch_size": 16,

"learning_rate": 0.001,

"dropout": 0.5,

"weight_decay": 0.0001,

"filters_conv1": 16,

"filters_conv2": 32,

"kernel_size": 3,

"pool_size": 2,

"fc_neurons": 512,

Divido em dois cenário:

a) Uso de 10 % do database(1 , 2 , 4 , 8 threads)
b) Uso de 100% do database (1 , 2 , 4 , 8 threads)

Resultados:

A)

2. Análise dos Resultados para 10% da Base de Dados

Nº de Processos	Tempo de Execução (s)	Ganho (Speedup)	Eficiência
1	30,20	1,000	1,000
2	27,04	1,117	0,559
4	26,56	1,137	0,284
8	26,55	1,138	0,142

Ganho: Ao aumentar o número de processos, observamos um pequeno ganho na velocidade, mas esse ganho não é proporcional ao número de processos

Eficiência: A eficiência cai significativamente, indicando que, ao usar mais processos, há uma redução proporcional nos ganhos. Isso ocorre devido ao

overhead na sincronização entre processos e à parte do código que não pode ser paralelizada.

B)

3. Análise dos Resultados para 100% da Base de Dados

Nº de Processos	Tempo de Execução (s)	Ganho (Speedup)	Eficiência
1	1.157,12	1,000	1,000
2	1.107,10	1,045	0,523
4	959,01	1,207	0,302
8	958,08	1,208	0,151

Ganho: Com o aumento do número de processos, há um ganho na velocidade, mas ele também é limitado. Comparando com a base de 10%, vemos que o código tem dificuldades em escalar de maneira linear mesmo com um conjunto de dados maior. Isso pode ser devido ao maior volume de dados e a limitações no hardware (ex.: largura de banda).

Eficiência: A eficiência cai ainda mais do que para a base de 10%, refletindo o aumento do overhead e da comunicação entre processos. Isso sugere que o modelo atinge um ponto de retorno decrescente, onde adicionar mais processos não resulta em um aumento significativo de desempenho.

Teste de escalabilidade Forte:

Parâmetros setados:

"num_epochs": 1, → para facilitar os testes

"batch_size": 16,

"learning_rate": 0.001,

"dropout": 0.5,

"weight_decay": 0.0001,

"filters_conv1": 16,

"filters_conv2": 32,

"kernel_size": 3,

"pool_size": 2,

"fc_neurons": 512,

Divido em um cenário:

A) Aumento de therads em proporcional ao aumento da % do database (1 , 2 ,4 ,8 threads e 1,2,4,8 % da base usada)

Resultados :

% da Base	Nº de Processos	Tempo de Execução (s)	Ganho (Speedup)	Eficiência
1x	1	26,43	1,000	1,000
2x	2	37,27	0,709	0,355
4x	4	36,57	0,709	0,177
8x	8	132,85	0,723	0,090

A análise da **escalabilidade fraca** avalia como o sistema se comporta quando aumentamos simultaneamente o número de dispositivos (ou processos) e o tamanho do problema, com o objetivo de manter o tempo de execução constante.

Mostrar que o melhor resultado foi :

2x a 4x da Base:

- O tempo de execução entre 2 e 4 processos praticamente se mantém, mas não há ganho adicional em velocidade, refletindo-se numa eficiência de 0,177.
- Ao analisar os resultados de escalabilidade fraca, o melhor desempenho relativo foi observado na transição entre 2 e 4 processos. Nesse intervalo, houve uma menor variação no tempo de execução, apesar do aumento do tamanho da base, o que sugere que o código conseguiu lidar razoavelmente bem com o aumento de carga.

Tempo de Execução: Ao aumentar de 2 para 4 processos, o tempo de execução praticamente se manteve estável (de 37,27 s para 36,57 s). Isso indica que o código foi capaz de suportar o dobro da carga de dados com uma diferença mínima no tempo, refletindo um bom balanceamento de carga para essa faixa.

Ganho (Speedup): Embora o ganho absoluto (speedup) não seja ideal, ele se manteve igual (0,709), o que mostra uma consistência na capacidade de paralelização para essas configurações. Esse resultado é particularmente interessante, já que não houve uma queda significativa no desempenho ao duplicar o número de processos.

Eficiência: A eficiência foi de 0,355 para 0,177, o que representa uma queda, mas ainda assim é superior ao que foi observado ao passar de 4 para 8 processos. Esse nível de eficiência é aceitável em comparação aos outros valores, pois mostra uma melhor utilização dos recursos.

Conclusões finais

Positivos:

- Aumento Gradual do Speedup em Baixa Escala: Em ambos os cenários de escalabilidade (forte e fraca), o sistema demonstrou uma melhoria no speedup ao passar de 1 para 2 processos, especialmente no cenário de 10% da base. Isso indica que o código foi capaz de aproveitar o paralelismo para alcançar ganhos de desempenho quando a carga não é excessiva.

- Desempenho Consistente em Escalabilidade Fraca para 2 a 4 Processos: O sistema mostrou um bom equilíbrio entre carga e desempenho nessa faixa, especialmente ao processar dados maiores, com impacto mínimo no tempo de execução. Isso sugere que o código foi bem configurado para aproveitar o paralelismo até certo ponto.

- Aproximação do Speedup Ideal em Escalabilidade Forte: Para a escalabilidade forte com 100% da base, o código conseguiu um speedup próximo de 1,2 entre 4 e 8 processos, o que sugere que o sistema foi capaz de lidar bem com uma grande carga inicial e ainda obter um ganho moderado. Embora o speedup não seja linear, a melhora em comparação com o uso de 1 processo é um ponto positivo, considerando a carga total.

- Aproveitamento Eficaz de Recursos em Cargas Moderadas: Os resultados indicam que o código consegue aproveitar bem os recursos para escalas de paralelismo moderadas. Isso se reflete no speedup obtido em cargas menores, onde a diminuição do tempo de execução é visível sem um crescimento expressivo do overhead.

Negativos:

- Queda de Eficiência com o Aumento Excesivo de Processos: A eficiência cai substancialmente quando o número de processos passa de 4, especialmente em escalabilidade forte. Isso indica que o código enfrenta desafios com overhead e sincronização entre dispositivos.
- Overhead de Comunicação: O aumento do número de processos resulta em um custo adicional para sincronização e troca de dados, limitando o ganho em configurações de alto paralelismo.

- Número de Épocas Fixado em 1 Limita a Análise Completa do Paralelismo: Definir o número de épocas para apenas 1 restringe a capacidade de avaliar completamente os benefícios do paralelismo ao longo do tempo. Com apenas uma época, o sistema não tem a oportunidade de demonstrar ganhos de desempenho sustentados e os efeitos do paralelismo em execuções mais longas.