IJHSR
ijhighschoolresearch.org
■ RESEARCH ARTICLE

Generative AI for Image Synthesis - A Study on CPUs

Arin P. Thakkar

American High School, 36300 Fremont Blvd, Fremont, California, 94555, USA
Mentor: Jashwant Raj Gunasekaran

ABSTRACT: With the immense growth of Artificial Intelligence (AI), especially in image processing, many people have taken advantage of its numerous capabilities. It is used in different domains, including generative imaging (GenI). Some of the crucial applications of GenI include content creation, shadow synthesis, and in-painting. However, image synthesis comes with flaws. Due to the complexity of generating images, the process can have very high processing times. This paper analyzes the shortcomings by profiling LaMa,[1] an in-painting model run on CPUs. We provide insights by highlighting the bottlenecks that hinder performance. Our main finding is that the mkldnn_convolution layer takes up most of the time, indicating that speeding up this layer's computations could drastically improve overall processing time.
KEYWORDS: Robotics And Intelligent Machines; Machine Learning; Image Synthesis AI; GAN.

■ Introduction

With a rise in the capabilities of AI, image synthesis models have exploded in popularity. Image synthesis is the process in which images are generated with content chosen by the user. Image synthesis AI takes in data in the form of images or videos and removes, adds, or replaces objects from the supplied images. AI helps automate this process. An interesting example of this is LaMa,[1] an image synthesis model. This model uses a new method called large mask inpainting. Large mask inpainting is where the model takes information on the entire image as context to fill in a gap in the image. Unlike other generative models, LaMa stands out due to its efficiency and performance. Many models have problems with high memory usage and struggle with producing high-quality results when dealing with large inpainting tasks.

Although inpainting is a useful method, it has limitations. For example, inpainting struggles with complex images and generates unrealistic images to fill the gaps. A Graphical Processing Unit (GPU) is a type of hardware that helps show graphics on a screen. Some synthesis models are mostly run on GPUs because they have more computational power on image tasks. On the other hand, during inference, inpainting usually uses the Central Processing Unit (CPU) as it provides stability, flexibility, and low latency. However, one downside to using CPUs is that LaMa's execution time is much higher than when using GPUs. There are various ways to improve CPU performance, including parallelization and algorithmic improvements. This paper analyzes the reasons for the low speed and CPU inefficiency. We conducted experiments to calculate the time to run the LaMa model on CPUs.

Furthermore, we profiled the CPU metrics to determine the processing time each layer takes in the model. We did not modify the original LaMa procedure, so the image generation and fidelity of the results should remain the same. Finally, our observations showcased the inefficiencies of executing an in-painting model on the CPU. Our insights could potentially be used to address these bottlenecks in future works.

■ Literature Review

Image synthesis is used in various applications. Facelab[2] is a generative AI that creates a 3D facial rig and animates the movement of a face using masks. Another example is "From Shadow Generation to Shadow Removal," which uses a Shadow Generator, a tool that creates, removes, and refines shadows in an image. Finally, Bau *et al.* introduce an image-specific GAN model for semantic photo manipulation[3], a generative AI method that can do many things, such as adding, removing, and changing objects in a scene with a prompt from the user. It uses Generative Adversarial Networks (GANs) to create objects and images while manipulating a photo using mathematical equations. GANs are made out of two parts: a generator and a discriminator. The generator tries to make data realistic while the discriminator determines if the data is real or generated. They both improve upon each other until the generator creates data that is impossible to distinguish from real data.[4]

These models have limitations that hinder the AI's performance. For Facelab, the AI struggles when the video has inadequate lighting. Shadow Generation heavily relies on its masks to tell the AI where specific objects are and what objects to remove, which can lead to errors if the masking process is incorrect. Finally, Semantic Photo Manipulation could give unnatural results and take up a lot of CPU processing power.

In the context of this paper, we limit our scope to inpainting applications with a specific focus on the well-known LaMa model. LaMa has several pitfalls, from the application layer to the hardware stack. It usually struggles when strong perspective distortion gets involved, which occurs when a picture is taken at an angle that makes objects look askew. LaMa can construct good quality masks for resolutions up to 1090x1092 due to using Fast Fourier convolutions (FFCs).[1] FFCs are fully

DOI: 10.36838/v6i11.4

differentiable and are an easy-to-use replacement for conventional convolutions, improving the network's quality and efficiency.

However, from a hardware perspective, LaMa's execution time is about 10-20 times slower on CPUs than on GPUs. An explanation for this is that FFCs are memory-bound rather than compute-bound.[1] Memory-bound, in this case, means that the model's speed is based on the speed at which data can be read and written, and compute-bound means the speed is limited to how fast the model can complete the mathematical operations it needs to generate the image. While providing limited insight into how LaMa performs for complex images, we mainly focus on the CPU slowdown of running LaMa models in this paper.

Methods

The LaMa model is an inpainting network designed for large inpainting tasks using FFCs. LaMa was created to fill in missing parts of an image with something realistic. It comprises three types of layers: downscaling layers, convolution layers, and upscaling layers.[1] The model's files include a dataset of test images, including pictures of buildings and places with many objects and a mask for each image.

LaMa's primary goal is inpainting, and it uses FFCs in its methodology to achieve this goal. When processing an image, LaMa goes through three crucial steps. First, the input image is downscaled, reducing the image size to reduce noise. Next, the model uses FFCs to understand the patterns that occur in the image. Finally, the image is upscaled back to its original size.[1]

To determine how efficient LaMa was in modifying each image, we needed to find a specific way to gain information on what the model did with the CPU and how long each layer took to complete. The hardware used to run this experiment is a MacBook Pro 2023 with an Apple M2 Pro chip. It ran on macOS 14 Sonoma; the metrics were run using PyTorch 1.2.9. Specifically, we found metrics on how long each layer took to finish processing using the self-CPU time. Furthermore, PyTorch[5] has a library called torchvision,[6] which provides the metrics needed. It profiled the model, giving us the amount of memory it consumes and how many CPU and GPU resources it uses. Using the psutil library,[7] we measured the CPU utilization and discovered what percentage of CPU each layer took up while generating. With this information, we are able to provide graphs on the efficiency of the model. To get these graphs, we used a Python library called Matplotlib.[8] Using the averages from 5 runs of each layer, we used Matplotlib to plot the data on a bar graph.

Results

In Figures 1 and 2, the graphs show the average processing time for each layer. Figure 1 represents the y-axis in the log scale. We did this because the mkldnn_convolution layers had disproportionately large times compared to the rest of the layers, and representing the y-axis in the log scale allows all the bars to appear on the graph. The *x-axis* is each layer, and the *y-axis* is the time taken to process each in microseconds. In both Figures 1 and 2, the time taken to process the mkldnn_convolutions layers was significantly higher than the other

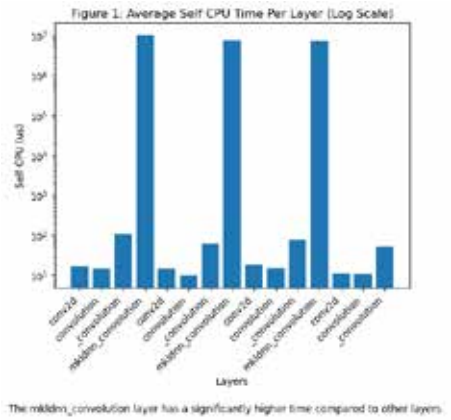

Figure 1: Average Self CPU Time Per Layer (Log Scale).

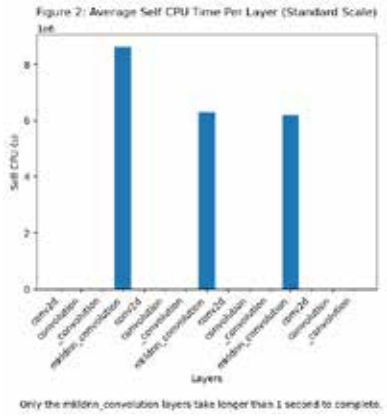

Figure 2: Average Self CPU Time Per Layer (Standard Scale).

Table 1:

Table 1		
Layer Name	Mean (us)	Standard Deviation (us)
aten::conv2d	16.37	6.46
aten::convolution	14.19	7.67
aten::_convolution	104.57	82.89
aten::mkldnn_convolution	10130000	4330000
aten::conv2d	14.37	10.15
aten::convolution	9.625	5.73
aten::_convolution	59.67	21.99
aten::mkldnn_convolution	7330000	1920000
aten::conv2d	18.35	12.48
aten::convolution	14.51	9.56
aten::_convolution	75.45	40.65
aten::mkldnn_convolution	6920000	1680000
aten::conv2d	10.99	5.02
aten::convolution	10.52	4.47
aten::_convolution	51.58	16.25

DOI: 10.36838/v6i11.4

layers. In the standard scale graph, only the mkldnn_convolution layers took longer than 1 second. In Table 1, each layer has a mean amount of time it took to process each layer and a standard deviation. The means and standard deviations were calculated by averaging 5 test runs.

Analysis of Results:

We repeated the same graph twice, in Figure 1 and Figure 2, but chose to represent the first graph in a log scale and the second graph in a standard scale so the scale of how much larger the mkldnn_convolution layers are is clear and to represent all the values on the plot. In both Figure 1 and Figure 2, there is a clear pattern. A significant part of the time was taken in the mkldnn_convolution layers, around 1 to 7 seconds, while the rest of the layers are only around 9 to 75 microseconds. There are many reasons why such a jump in time has occurred. For example, the model did more processes in the mkldnn_convolution layers, which took longer to complete. Another factor is that convolutional layers are the most vital parts of a neural network that does image processing, making it the most resource expensive. Compared to the mkldnn_convolution layers, _convolution layers take much less time. However, a pattern emerges when compared to the other layers that process in microseconds. The _convolution layers take 10 to 35 microseconds longer to process than the conv2d and convolution layers. This is because _convolution layers are general and handle many convolution operations. Layers like conv2d are only for specific use cases.

A reason current models fail to work well on complicated images is that the models only learn the overall features of an image. Instead of learning the concept of each object, the model does not understand the image but remembers where to put some shapes and colors.[9] According to the author, the model does not know the whole object but its colors and overall shape. Since the model needs multiple iterations on each image to learn the features, the model must run for several epochs or full passes over the entire dataset. This leads to increased runtime and processing power compared to just running one epoch. Multiple epochs of running computationally expensive convolutions lead to overall exhaustive training processes. As we can see, convolutions are the most expensive operation, so finding a way to speed these up would allow for overall faster training and inference.

Discussion

Our literature survey about FFCs used in LaMa revealed that FFCs are highly memory-bound operations. Our results show that the Self CPU corresponds to the actual computation done by FFCs while the remaining time is spent in data transfer from memory. One possible solution to achieve higher performance in CPUs would be to choose compute-bound convolution types. FFCs can be improved by reducing the number of computations through combined operations. This would speed up the process and reduce the load on the CPU. The LaMa model uses the Intel Math Kernel Library (MKL) in the form of the layers called “mkldnn_convolution.” The benefit of using MKL is that it does various mathematical operations, including FFCs, and works well on intel-based CPUs. This is because MKL employs multi-versioning, creating multiple implementations of one function. Hedjazi states, "Reducing the model size without affecting the quality of the generated images is desirable. We present a GAN-based inpainting method that reduces the inference time and generates plausible results”. One of the main observations with the LaMa model was that the mkldnn_convolution layers took too much time to process. With observations similar to ours regarding their image inpainting model, their solution to the problem was to reduce the model size.[10] Similarly, the same results can be reproduced in LaMa by removing some overly complex convolutional layers. Furthermore, in "Generative Image Inpainting with Contextual Attention," the authors use memory-based attention mechanisms.[11] This can be implemented into LaMa to efficiently use context in an image, removing the burden on convolutions and increasing its performance. Since FFCs are memory-bound, and one layer is the most expensive, another way to reduce CPU usage is to modify the use of FFCs in future works.

Conclusion

AI has greatly influenced image synthesis applications. Within this broad domain, we specifically focus on in-painting applications and profile a well-known model named LaMa, which has limitations that affect its performance and user experience. In this paper, we gathered observations to create improvements and make this a better model. In the experiments, we used PyTorch to run metrics and figured out how long the model took to run each layer and how much processing power it took. We discovered that completing the mkldnn_layers takes more time to process. Since these layers had more math to compute, processing would take significant time. We hope that people in the future can use our results to improve the results of future image-generation models.

Acknowledgments

I want to thank my project manager, Kabir Hiranandani, for guiding submissions. Additionally, I want to express my gratitude to my writing coach, Rory Wakeford, for her insightful improvements to this project. Finally, I want to acknowledge Lumiere for their resources and support in this paper.

References

1. Suvorov, R.; Logacheva, E.; Mashikhin, A.; Remizova, A.; Ashukha, A.; Silvestrov, A.; Kong, N.; Goka, H.; Park, K.; Lempitsky, V. Resolution-Robust Large Mask Inpainting with Fourier Convolutions. https://arxiv.org/pdf/2109.07161.pdf. (accessed 2023-08-22).
2. Andrus, C.; Jeong Joon Ahn; Alessi, M. E.; Dib, A.; Gosselin, P.-H.; Cédric Thébault; Chevallier, L.; Romeo, M. FaceLab: Scalable Facial Performance Capture for Visual Effects. The Digital Production Symposium 2020. https://doi.org/10.1145/3403938. (accessed 2023-08-22).
3. Bau, D.; Peebles, W.; Csail, M.; Wulff, J.; Zhou, B.; Strobelt, H. Semantic Photo Manipulation with a Generative Image Prior ANTONIO TORRALBA, MIT CSAIL and MIT-IBM Watson AI Lab Input Photo Change Rooftops Output Result Input Photo Remove Chairs Output Result Input Photo Add Windows Output Result Input Photo Restyle Trees for Spring Restyle Trees for Autumn. https://arxiv.org/pdf/2005.07727.pdf. (accessed 2023-08-22)..
4. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Far

ley, D., Ozair, S., Courville, A., & Bengio, Y.. Generative Adversarial Networks. arXiv.org. https://arxiv.org/abs/1406.266. (accessed 2024-08-27).
5. PyTorch. PyTorch. Pytorch.org. https://pytorch.org/. (accessed 2023-08-22).
6. torchvision — Torchvision master documentation. pytorch.org. https://pytorch.org/vision/stable/index.html. (accessed 2023-08-22
7. psutil documentation — psutil 5.7.0 documentation. psutil.readthedocs.io. https://psutil.readthedocs.io/en/latest/. (accessed 2023-08-22).
8. (1)*Matplotlib. Matplotlib: Python plotting — Matplotlib 3.3.4 documentation*. matplotlib.org. https://matplotlib.org/stable/index.html. (accessed 2024-8-27)
9. Huang, Z.; Wu, J.; Van Gool, L. Manifold-Valued Image Generation with Wasserstein Generative Adversarial Nets. *Proceedings of the AAAI Conference on Artificial Intelligence* 2019, 33, 3886–3893. https://doi.org/10.1609/aaai.v33i01.33013886. (accessed 2023-8-22).
10. Hedjazi, M. A., & Genc, Y. (2021, January 27). *Efficient texture-aware multi-gan for image inpainting*. Knowledge-Based Systems. https://www.sciencedirect.com/science/article/abs/pii/S0950751 21000526. (accessed 2024-10-09).
11. Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., & Huang, T. S. (1970, January 1). Generative image inpainting with contextual attention . CVF Open Access. https://openaccess.thecvf.com/content_cvpr_2018/html/Yu_Generative_Image_Inpainting_CVPR_2018_paper.html. (accessed 2024-10-09).

Author

Arin is a high school senior who is interested in artificial intelligence and applied mathematics. He loves playing the guitar and reading science fiction books. He spends his free time drinking boba and building a semi-automated greenhouse. He is also part of the Youth Leadership Council at Climate Roots, an organization dedicated to solving environmental and health challenges in his local community.

DOI: 10.36838/v6i11.4