Universidade Federal do Piauí – UFPI
Centro de Ciências da Natureza – CCN
Departamento de Computação – DC
Disciplina: Arquitetura de Computadores (2025.2)
Prof.: Dr. Ivan Saraiva Silva
Discentes: Ana Beatriz dos Santos Sousa Costa, Diego Bruno Sousa Silva e José Nelson Fernandes Da Silva

RELATÓRIO TERCEIRA AVALIAÇÃO – TRABALHO PRÁTICO

SUMÁRIO

Introdução

Neste trabalho prático da disciplina de Arquitetura de Computadores, nosso objetivo foi a geração de mapas de características a partir de uma imagem fornecida pelo professor, processando os três canais (R, G, B) em Python e em Assembly RISC-V (simulador RARS). Para realização de tal objetivo, na programação em Python, utilizamos as bibliotecas Pillow e NumPy. Ao final de cada programa, tanto em Python quanto em Assembly, os resultados dos mapas de características estão em formato ".asm". A comparação entre eles foi programada e contabilizada em Python.

Desenvolvimento

A princípio, executamos o código "extrai_Imagem.py", fornecido pelos monitores via SIGAA. Esse programa lê a imagem "11.jpeg", extrai os três canais RGB e organiza cada um deles na forma de buffers lineares, armazenados em .byte com 16 valores por linha e cria o arquivo "imagem_dados.asm".

Em seguida, programamos o arquivo "mapaDeCaracteristicas.py", ele possui as seguintes funções: convolucao_3x3, ativacaoLeakyRelu, avgPooling, salvarMapaASM, gerarMapaTuplas, salvarMapaTuplasASM e main. Ao ser executado, ele cria dois arquivos: "mapa_caracteristicas_tuplas.asm" e "mapa_caracteristicas_python_G11.asm", ambos possuem os dados de cada canal após a convolução com os Kernels (R=[2, 1 , 2], [1 , 4, 1], [2, 1, 2]; G=[1, 2, 1], [2, 4, 2], [1, 2, 1]; B=[2, 2, 2], [1, 4, 1], [2, 2, 2]), ativação (LeakyReLU(x) = x, se $x \geq 0$; $\alpha \cdot x$, se $x < 0$ (com α=1/16)) e Pooling (Média 2×2). No entanto, a formatação é diferente, pois o arquivo "mapa_caracteristicas_python_G11.asm" organiza cada canal individualmente em blocos de .byte, enquanto o arquivo de tuplas apresenta cada pixel como um trio (R, G, B), facilitando a leitura e depuração dos valores gerados no processamento.

Após isso, implementamos a solução em Assembly, utilizando o arquivo "imagem_dados.asm" como base, no arquivo "processamento.asm". Nesse desenvolvimento, cada canal é processado individualmente, reproduzindo exatamente as três etapas realizadas

no Python: convolução 3×3, ativação LeakyReLU e média 2×2 (avg pooling). O programa carrega os buffers lineares imagem_R, imagem_G e imagem_B a partir da seção .data, aplica o kernel correspondente a cada canal e armazena o resultado intermediário em BUFFER_TEMP. Após a convolução, a função ativacao_leaky_relu percorre todos os valores gerados e aplica a ativação com fator $\alpha = 1/16$ para elementos negativos. Em seguida, a rotina avg_pooling_byte reduz as dimensões do mapa calculando a média dos blocos 2×2, realizando tratamento de arredondamento e clamp (0–255) para garantir que os valores caibam em um byte. Os resultados finais são armazenados em MAPA_R, MAPA_G e MAPA_B e, ao término do processamento, a função imprimir_mapa_formatado imprime cada mapa. Dessa forma, a solução em Assembly replica a lógica implementada na versão em alto nível, permitindo comparar pixel a pixel os resultados entre as duas abordagens.

Também desenvolvemos dois programas auxiliares para a saída do RARS. O primeiro, "extrator_assembly.py", converte a saída textual que foi gerada pelo Assembly, copiada e colada manualmente em um novo arquivo "processamento_assembly", em arquivos .asm estruturados. O script lê o arquivo copiado, localiza os rótulos mapa_carac_R:, mapa_carac_G: e mapa_carac_B:, extrai todos os valores numéricos e reorganiza os dados em dois formatos: (1) o formato com 16 valores por linha em diretivas .byte (arquivo "mapa_caracteristicas_assembly_G11.asm"), e (2) o formato por tuplas RGB, onde cada linha contém os três canais de um pixel (.byte R, G, B) (arquivo "mapa_caracteristicas_tuplas_assembly.asm".

O segundo programa, "comparador_de_arquivos.py", foi utilizado para verificar se os arquivos produzidos no Assembly são idênticos aos gerados em Python. Ele lê ambos os arquivos, extrai os números presentes e compara cada valor individualmente, registrando divergências de conteúdo ou de tamanho. O comparador foi aplicado tanto ao formato por tuplas RGB quanto ao outro formato, para confirmar a equivalência entre as duas implementações e identificar eventuais discrepâncias durante os testes.

Figura 1. Imagem fornecida pelo professor para a realização do trabalho

Resultados

Após gerar os mapas de características em Python e em Assembly, utilizamos o programa auxiliar para padronizar a saída do RARS. Com os arquivos finais, aplicamos o programa de comparação, que avaliou cada valor numérico dos três canais. O resultado final mostrou equivalência total entre as duas implementações: nenhuma discrepância foi identificada e todos os arquivos apresentaram o mesmo tamanho e os mesmos valores, confirmando que a lógica implementada no Assembly reproduziu exatamente o comportamento da versão implementada em Python.