CONSTRUÇÃO DE UMA REDE NEURAL EM FPGA PARA INFERÊNCIA DE BAIXA LATÊNCIA

Lucas Gomes de Araújo[1] Silvio Roberto Fernandes de Araujo[2]
[1]Graduando – araujolucas3005@gmail.com
[2]Orientador – silvio@ufersa.edu.br

Resumo: As doenças em plantas representam um dos principais fatores que contribuem para a perda de colheitas na produção agrícola. Logo, diagnosticar uma planta rapidamente pode ser essencial para a produção. Contudo, um diagnóstico manual pode ser custoso, lento e subjetivo. Dessa forma, a implementação de um sistema de baixo consumo de energia capaz de avaliar automaticamente o estado das plantas de maneira contínua, rápida e precisa é crucial no contexto. Dito isto, esse artigo apresenta uma solução baseada em aprendizado de máquina possível de ser implantada em FPGAs para realizar inferências acerca de imagens do domínio tratado. Para isso, foram utilizados recursos como redes neurais convolucionais, transferência de aprendizado e quantização com o *framework* Vitis AI. Além disso, testes com os diferentes tipos de quantização foram realizados em GPU, CPU e FPGA com o propósito de comparar as métricas de desempenho em cada *hardware*. Ao fim, com uma acurácia de 97.74%, o modelo classificador obtido pela técnica *quantization aware training* (QAT) conseguiu melhores resultados frente àquele gerado pelo método *post training quantization* (PTQ). Ademais, o modelo em FPGA conseguiu obter uma vazão de 27.6 inferências por segundo nos testes, suficiente para diagnósticos em tempo real.
Palavras-chave: doenças em plantas; FPGA; redes neurais; transferência de aprendizado; quantização.

1. INTRODUÇÃO

Doenças em plantas são, segundo [1], um grande desafio à produção agrícola. As colheitas, em sua totalidade, estão suscetíveis a doenças causadas por uma grande variedade de patógenos. Além disso, doenças em plantas correspondem a 25% das causas de perdas de colheitas da produção agrícola mundial por ano [1], agravando ainda mais o déficit de abastecimento alimentar, em que 800 milhões de pessoas, pelo menos, são alimentadas inadequadamente [2]. Portanto, um diagnóstico correto e rápido do agente causador é fundamental para o manejo do problema.

Contudo, ultimamente, de acordo com [3], a existência de um grande número de espécies de plantas e doenças torna desafiador e complexo uma avaliação a olho nu. Ainda, a examinação visual, apesar de eficiente em alguns casos, está sujeita a erros humanos graças ao monitoramento contínuo cansativo [3]. Além disso, em grandes fazendas, o uso da técnica pode ter um custo muito elevado e a precisão das classificações, nesses casos, pode ser reduzida. Dessa forma, tratamentos inadequados decorrentes de classificações errôneas podem prejudicar severamente uma colheita.

Devido a esse contexto problemático, de acordo com [4], técnicas de processamento de imagem para classificação de doenças em plantas se tornaram um assunto bastante pesquisado. Um sistema computacional automatizado de baixo consumo, capaz de monitorar e diagnosticar de maneira precisa, contínua e rápida, doenças em plantas, seria de grande valia para os agricultores.

Entretanto, desenvolver aplicações de alto desempenho capazes de processar imagens requer, segundo [5], um balanço entre tempo de execução e restrições de energia. Um monitoramento contínuo de plantas a fim de classificar suas doenças, caso realizado por *hardwares* de alto consumo energético, pode ser algo custoso. Ademais, como a tarefa de classificação deve ser rápida, o dispositivo ainda precisa ter um bom poder de processamento capaz de suprir uma determinada demanda. No grupo de pesquisa que o autor deste texto faz parte está sendo desenvolvido um trabalho de dissertação de mestrado, cujo objetivo é realizar uma classificação de contexto de fitopatógenos. Para isso, está sendo desenvolvida uma ferramenta baseada em aprendizado de máquina, a qual recebe como entrada uma imagem de uma planta que pode conter várias folhas. O sistema é composto por 2 redes neurais, a primeira deve detectar cada folha e recortá-las, transformando-as em imagens individuais, as quais serão entregues para a segunda que fará a classificação individual das folhas, conforme a Figura 1. A posteriori o sistema deve compilar as classificações individuais em uma única classificação do contexto daquela imagem inicial. Resultados intermediários demonstram que a acurácia da classificação das imagens das folhas recortadas melhorou de 84% para 93% em comparação com classificação das imagens originais.

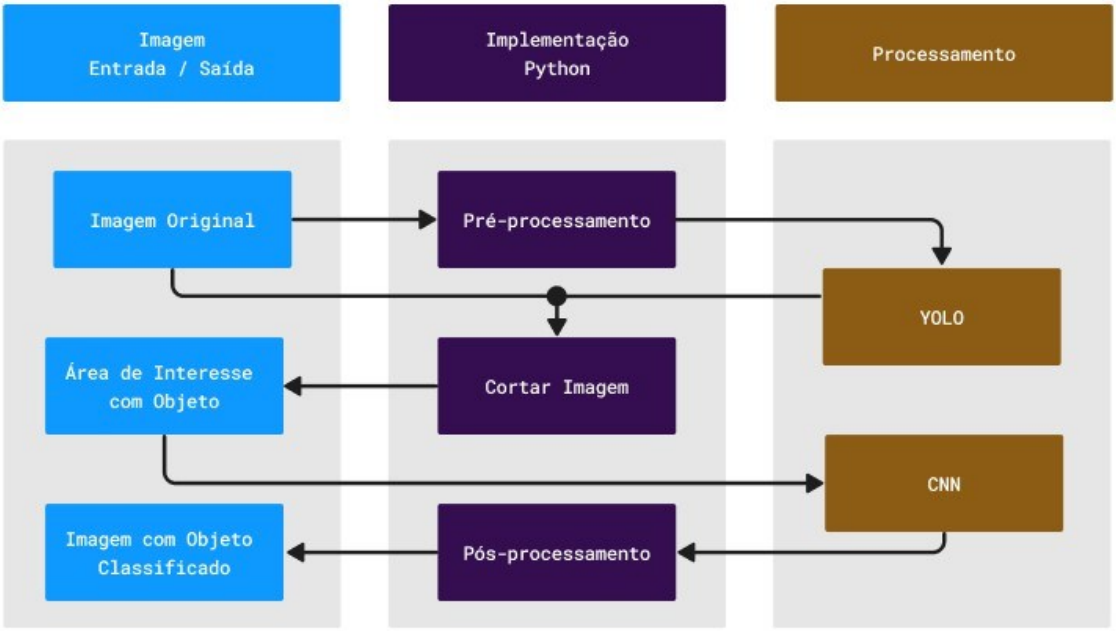


Figura 1. Sistema de classificação de contexto. Fonte: Autoria própria

Logo, tal sistema deve melhorar o diagnóstico de doenças de plantas, contudo, todas essas etapas implicam em aumento do tempo de processamento para as classificações. Assim, o trabalho proposto aqui tem o objetivo de acelerar a etapa de classificação para diminuir o tempo de inferência do sistema de classificação de contexto como um todo.

Por isso, a utilização do *hardware* programável *Field Programmable Gate Array* (FPGA) pode ser algo promissor na área, já que o equipamento consegue, com o seu paralelismo, acelerar aplicações com poucos recursos e baixo consumo energético. Dito isto, o presente trabalho propõe a criação de um acelerador em FPGA capaz de classificar imagens de plantas doentes com precisão e mais rápido que uma unidade central de processamento (CPU). O restante do texto está organizado da seguinte forma: na seção 2 é apresentado o referencial teórico e trabalhos relacionados, na seção 3 se encontra o trabalho proposto (metodologia, *dataset* e implementação), na seção 4 são apresentados os resultados e discussões e, por fim, na seção 5 é apresentada a conclusão do estudo.

2. REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS

Nesta seção será apresentado o referencial teórico dos conceitos básicos para o entendimento deste trabalho. Entre estes conceitos estão CPU, GPU, FPGA e DPU, relacionados aos dispositivos de processamento. Em relação aos assuntos de inteligência artificial, destacamos aprendizado de máquina e redes neurais, em conjunto com as técnicas de transferência de aprendizado e quantização. Ainda é apresentado o ambiente Vitis AI, utilizado para o desenvolvimento deste trabalho. E em seguida são apresentados outros trabalhos que se relacionam com o aqui proposto.

2.1 CPU

A unidade central de processamento (CPU), de acordo com [6], é a porção do computador que recupera e executa instruções. Ainda, [7] diz que a CPU é frequentemente chamada como o “cérebro do computador” dado que é o local onde todas as atividades de processamento do computador ocorrem. A arquitetura da CPU, conforme [6], é composta por: unidade de lógica e aritmética (ULA), cuja função é realizar operações de aritmética, lógica e relacionadas de acordo com as instruções do programa; unidade de controle, que controla todos as operações da CPU, como a troca de dados e controle de sinais entre interface externas; registradores, os quais são memórias de armazenamento de alta velocidade em que uma parte é disponibilizada para o programador utilizar em seus programas e outra para uso estrito da CPU para questões de controle. Além disso, a CPU possui um relógio (do inglês *clock*) interno que sincroniza todos os componentes da CPU. Existe uma velocidade, denominada velocidade de *clock* (número de pulsos de clock por segundo), que é medida em hertz (Hz) e corresponde a agilidade com a qual a CPU processa uma instrução [6].

2.2 GPU

Unidades de processamento gráfico (GPU), segundo [8], são *hardwares* especializados projetados para realizar uma única instrução em múltiplos dados ao mesmo tempo. Além disso, de acordo com [9], a GPU é especializada no cálculo de matrizes cujas aplicações visuais necessitam. Originalmente, as GPUs foram projetadas para renderizar efeitos visuais em uma superfície 2D, como o monitor [10]. Contudo, agora o *hardware* é também utilizado para aceleração computacional de propósito geral devido a sua massiva capacidade de paralelismo disponível pelo seu grande número de núcleos [11]. Com isso, as GPUs começaram a ser utilizadas para treinamento e inferências rápidas de modelos de aprendizado de máquina. Contudo, devido ao alto poder de processamento, as GPUs possuem um alto consumo energético quando comparadas às CPUs [12].

2.3 FPGA

Os FPGAs são, segundo [13], módulos manufaturados a partir de silício que contém blocos de lógica programável e interconexões configuráveis entre eles. Além disso, [14] diz que FPGAs podem ser programados com linguagem de descrição de *hardware* (HDL) para implementar qualquer circuito de *hardware* digital. FPGAs, quando comparados com aplicativos personalizados de circuitos integrados, possuem custos de engenharia e tempo de lançamento no mercado muito menores [14]. A estrutura geral de um FPGA é composta por portas lógicas, RAMs embarcadas, multiplicadores e blocos de escrita e leitura (I/O).

De acordo com [14], FPGAs se tornaram uma solução atrativa para realização de inferências *deep learning*, dado suas habilidades de implementar caminhos de dados de precisão customizados, eficiência energética e seu baixo custo de desenvolvimento. Ainda, [15] afirma que FPGAs consomem muito menos energia quando comparados a GPUs. Contudo, apesar de seus benefícios, o desenvolvimento de suas aplicações requer um conhecimento de *design* de *hardware*, o que comumente é um obstáculo para programadores [16]. Dito isto, os fabricantes de FPGA têm desenvolvido ferramentas para implementação de redes neurais artificiais (RNA) em FPGA, que aumentam a abstração e reduzem muito o tempo e necessidade de *expertise* em *design* de *hardware*. Entre elas podemos destacar a FINN [17] e Vitis AI [18], ambas da AMD/Xilinx [19].

2.4 DPU

A DPU da Xilinx (*Deep-Learning Processor Unit*), segundo [20], é um motor programável para CNNs que pode ser integrado nos blocos de lógica programável dos FPGAs da Xilinx, cuja arquitetura é apresentada na Figura 2. O módulo é composto por: motores de processamento (MP), que são responsáveis por realizar cálculos como multiplicação de matrizes e convoluções; *pool* de memória global, a qual é um espaço de armazenamento compartilhado entre os motores e é utilizada para guardar resultados intermediários obtidos nos cálculos; escalonador de alta performance, componente que gerencia os processos referentes aos motores de processamento; unidade de busca de instrução, cuja função é buscar as instruções da memória e enviá-las ao escalonador para que ele as coordene entre os motores de processamento; tudo de dados de alta velocidade, que intermedeia a comunicação entre a DPU e a memória RAM; unidade de processamento da aplicação, que é a aplicação responsável por fornecer entradas à DPU e também receber os resultados das inferências obtidas pela última.

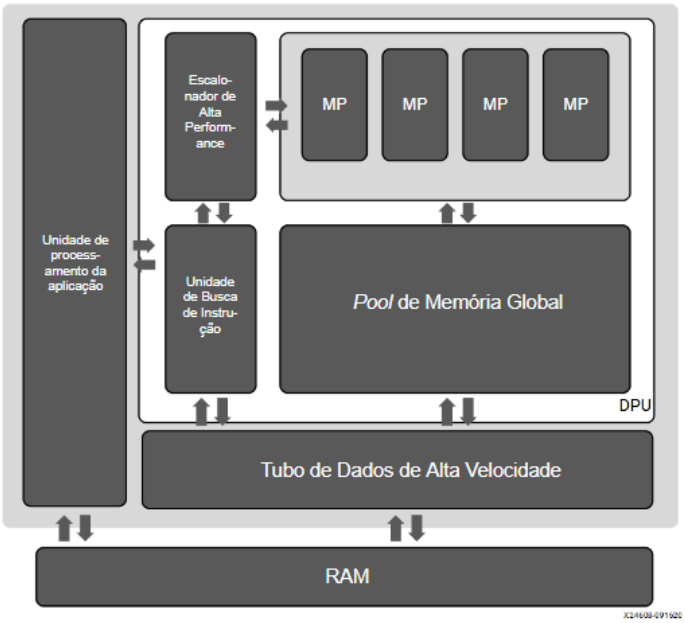


Figura 2. Arquitetura da Xilinx DPU. Fonte: Adaptado de [21]

Ainda, de acordo com [22], a DPU utiliza os recursos dos FPGAs para acelerar o processo de inferência das redes neurais através de otimizações de operações de matrizes e *hardware*. Para isso, o módulo utiliza motores de processamento que realizam cálculos referentes a multiplicação de matrizes e convoluções. Os resultados intermediários são armazenados na memória *pool* global, que é um espaço compartilhado entre os motores.

2.2 Aprendizado de máquina

Aprendizado de máquina, segundo [23], é um conjunto de métodos utilizados por computadores para melhorar predições ou comportamentos baseados em dados. Ainda, [24] diz que o aprendizado de máquina descreve a capacidade dos sistemas em aprender através de dados acerca de um problema específico, a fim de automatizar o processo de criação analítica de modelos e de solução tarefas relevantes ao domínio. A técnica pode ser dividida entre os tipos: aprendizado supervisionado, onde já é conhecido o resultado de interesse no conjunto de dados de treinamento e deseja-se conhecê-lo para dados desconhecidos; não-supervisionado, cujo objetivo é agrupar dados semelhantes, sem o uso de rótulos; aprendizado por reforço, no qual o agente aprende a otimizar uma recompensa através da sua atuação no ambiente [23]. Neste trabalho, estamos interessados nos métodos supervisionados.

Segundo [23], o aprendizado supervisionado, no caso de saídas categóricas (ex: predizer a raça de um cachorro em uma foto), denomina-se classificação. Contudo, quando a saída é numérica (ex: predição do valor de uma casa a partir de suas características), trata-se de um problema de regressão. Existem, por exemplo, na área da saúde, diversos problemas de classificação que podem ser resolvidos com aprendizado de máquina. Uma das principais técnicas de aprendizado de máquina supervisionado é por meio de redes neurais.

2.3 Redes Neurais

Redes neurais, de acordo [25], são modelos computacionais que possuem simples unidades de processamento (como neurônios), arranjados em camadas interconectadas. Ainda, [26] diz que redes neurais são algoritmos capazes de aprender e generalizar padrões que podem ser utilizados em classificações e predições. As redes neurais mais simples possuem apenas duas camadas (entrada e saída). Agora, quando diversas camadas são sobrepostas, cria-se uma rede neural profunda (do inglês *deep neural network*). Podemos dizer que existem 2 estágios bem definidos no processo de desenvolvimento de redes neurais: o primeiro é o treinamento para gerar um modelo; e o segundo, é a avaliação do modelo. A partir daí, o modelo pode ser incluído em uma aplicação para fazer inferências. Para realização dos dois estágios são usados conjuntos de dados (do inglês *dataset*), o qual é dividido com uma proporção maior para treino (os pesos se ajustam para se aproximar dos exemplos) e outra para teste (avaliação do modelo treinado para exemplos inéditos para a rede).

Durante o processo de treinamento, os exemplos são passados pela rede e ao final de cada iteração (chamada de época), os resultados preditos pela rede são comparados com os resultados esperados. A diferença entre os valores preditos e os esperados é calculada por meio de uma função perda (do inglês *loss*) que servirá de base para ajuste dos parâmetros de conexão entre os neurônios (*weight* e *bias*) para uma nova época [27]. Existem diversas métricas para medir a qualidade de uma rede neural a depender da classe do problema. Para problemas de classificação de imagens é muito comum utilizar-se da acurácia, precisão, *recall* e F1-score. Acurácia é uma medida simples e intuitiva que representa a proporção de previsões corretas feitas pelo modelo em relação ao total de previsões. Ela mede a proporção de exemplos positivos previstos corretamente em relação ao total de exemplos positivos previstos pelo modelo, ou verdadeiros positivos (do inglês *true positives*). Em outras palavras, a precisão indica entre todas as previsões positivas feitas pelo modelo, quantas estão corretas. O *recall* mede a proporção de exemplos positivos previstos corretamente em relação ao total de exemplos positivos reais. Enquanto o F1-score combina precisão e *recall* em uma única medida para avaliar o desempenho de modelos de classificação. É especialmente útil quando se deseja encontrar um equilíbrio entre a capacidade de identificar corretamente exemplos positivos (recall) e evitar falsos positivos (precisão) [28]. Depois de treinada uma rede neural, é gerado um modelo, o qual é utilizado para fazer inferência sem ajustes dos seus parâmetros. No entanto, as métricas de avaliação da qualidade são as mesmas.

Uma parte dos algoritmos de aprendizado de máquina lidam apenas com dados numéricos. Portanto, dados textuais precisam de um processamento prévio para que possam ser representados em números. Para isso, existe, dentre outras técnicas de pré-processamento, uma técnica denominada *One-hot Encoding*. Ela é comumente utilizada para representar palavras que possuem um conjunto finito de valores, como em problemas de classificação. Com a técnica, as palavras são representadas por um vetor binário onde um dos elementos é 1 e os outros 0. Dessa forma, é eliminada qualquer influência que a diferença do valor numérico possa conter durante o treinamento (ex. valor 1 ter menos peso que o valor 100 em uma rede neural) [29].

As técnicas de desenvolvimento de redes neurais criam tipo com melhor eficiência para determinados domínios. Assim, as redes neurais convolucionais (CNN, do inglês *Convolutional Neural Network*) é subtipo de rede neural profunda, as quais são amplamente utilizadas para classificar imagens (e.g. reconhecimento de objetos, monitoramento) [30]. Essas redes são compostas, além das camadas de entrada e saída, por múltiplas camadas convolucionais, cujo papel é detectar características (*features*) com um nível semântico maior quanto mais profunda a camada [31]. Além disso, [32] diz que a utilização de camadas *Pooling*, que têm como função substituir a saída de uma camada convolucional pela estatística sumária das saídas ao redor, contribuem positivamente na

eficiência desse tipo de rede. No entanto, para ter alta performance de classificação, as CNN necessitam de uma grande quantidade de dados de treinamento [33]. Uma forma de reduzir o tempo de treinamento e aproveitar a extração de características primárias de qualquer imagem é por meio da técnica de transferência de aprendizado.

2.3.1 Transferência de Aprendizado

A transferência de aprendizado (do inglês *Transfer Learning*) é uma técnica utilizada para melhorar uma rede neural em um domínio a partir da transferência de informações de um domínio relacionado [33]**.** Em aprendizado de máquina, o método se refere ao uso de modelos pré-construídos desenvolvidos por uma coleção de dados específica que servem como ponto de partida para um novo modelo que deseja aprender a partir de diferentes tipos de dados e atributos [34]**.** O método, de acordo com [35], tem alta aplicabilidade em classificação de imagens, classificação de sentimentos, sistemas de diálogo e computação urbana. Além disso, *Transfer Learning,* dentre outras técnicas (ex: *data augmentation*), pode ser considerado quando a quantidade de dados para um domínio não é suficiente para treinar um modelo eficaz.

A combinação de diversos tipos de camadas em redes neurais profundas permitiu o desenvolvimento de arquiteturas as quais são amplamente utilizadas no processo de transferência de aprendizado. Exemplos de arquiteturas muito utilizadas em *transfer learning* são EfficientNet [36], Resnet [37], MobileNet [38], entre outras.

2.3.2 Quantização

CNNs altamente profundas podem ocupar muito espaço de armazenamento devido às suas grandes quantidades de camadas e parâmetros (milhões) [39]. Por essa razão, diversas técnicas de compressão de redes neurais têm sido propostas. Isso permite, por exemplo, o uso dessas redes em *hardwares* de recursos limitados, como sistemas embarcados, que antes era proibitivo. Dito isto, segundo [40], a quantização é um processo que converte uma rede neural de alta precisão (ponto flutuante) em uma versão com baixa largura de bits (inteiro ou binário). No entanto, após o processo, o desempenho da rede pode ser reduzido. Contudo, [41] diz que a técnica é uma ótima solução para reduzir o tamanho do modelo e seu consumo de energia durante a execução de inferências.

A quantização pode ser feita de duas formas: pós-treinamento (PTQ), onde um modelo já treinado é quantizado; quantização durante o treinamento (QAT), cujo treinamento é feito após a quantização. Ainda, com QAT, pode-se quantizar um modelo pré-treinado. Dessa forma ocorre, primeiramente, uma quantização inicial para transformar seus pesos em inteiros e, depois, realiza-se um novo treino, mas agora a partir do modelo quantizado, realizando assim um ajuste fino dos pesos da rede. Normalmente, QAT resulta em uma menor perda de acurácia quando comparada com PTQ [42]. De acordo com [42], QAT é preferível no cenário onde o modelo quantizado deve ser implantado por um longo período tempo e também quando questões de eficiência e acurácia são os principais objetivos. Agora, PTQ é a melhor opção quando existem dados insuficientes de treinamento e quantização rápida e simples é necessária. As técnicas de quantização também são grandes aliadas no desenvolvimento de redes neurais em *hardware* para alto desempenho, como em FPGA.

2.4 Vitis AI

Vitis AI é uma ferramenta composta por uma avançada biblioteca de aceleração e ferramentas de design para desenvolvedores conduzirem trabalhos relacionados a *deep learning* em FPGAs [43]. Ela permite a conversão de modelos redes neurais profundas, construídos a partir de *frameworks* especializados, em um formato possível de ser executado nos FPGAs. Para isso, Vitis AI utiliza as ferramentas *AI Quantizer*, que converte, com o uso da quantização, modelos DNN 32-bit em modelos 8-bit, e AI Compiler, cuja função é transformar a rede quantizada em dados e instruções executáveis em uma DPU. A Figura 3 demonstra todos os passos necessários para a implantação de uma rede neural em FPGA utilizando a ferramenta Vitis AI.

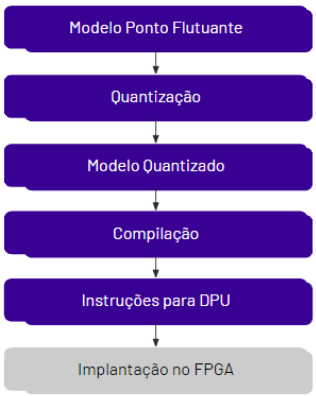


Figura 3. Etapas do Vitis AI para implantação da rede neural em FPGA. Fonte: Autoria própria

A entrada da ferramenta Vitis AI é um modelo de uma CNN desenvolvido por meio dos principais *frameworks* de redes neurais em software (PyTorch, Tensorflow, ONNX), utilizando representação dos pesos, vieses e funções de ativação em ponto flutuante. Tais modelos passam pelo processo de quantização para então ser compilados e gerar instruções para a DPU. Em seguida, a DPU e o respectivo modelo treinado da CNN podem ser carregados para o FPGA para fazer inferências. A Vitis AI também permite a configuração de diversos parâmetros para exploração do espaço de projeto de CNN em FPGA, como quantidade de bits no processo de quantização, otimização para redução de latência ou aumento da acurácia, entre outros.

2.5 Trabalhos Relacionados

Vários outros trabalhos propuseram o uso dos FPGAs na classificação de doenças em plantas através de redes neurais. No trabalho [44] foi implementada uma CNN em um FPGA para classificar doenças de plantas em tempo real. Nele, foi utilizado um *dataset* que contém imagens de diversos tipos de plantas saudáveis e doentes para treinar e validar o modelo classificador. No estudo, o autor preferiu, ao invés de aplicar um processo de quantização de uma rede possivelmente grande, criar um modelo 16-bit raso com apenas 176 mil parâmetros. Para isso, eles tiveram que utilizar um método de destilação de conhecimento durante o processo de treinamento. No final, foram alcançados acurácia de 95.24%, velocidade de inferência de 0.071s por *frame* e 2.41W de consumo de energia que, segundo os autores, são resultados suficientes para justificar a aplicação do sistema na identificação de doenças de plantas em tempo real.

No artigo [45], os autores utilizaram a ferramenta Vitis AI para gerar um modelo capaz de ser executado em um FPGA. Para tal, eles primeiramente obtiveram um modelo CNN pré-treinado com uma base de dados pública de doenças de plantas. Depois, utilizando Vitis AI, foi feita uma quantização na rede para reduzir o seu tamanho. Contudo, a quantidade de bits utilizada não foi revelada, assim como a completude do processo de implantação. A aplicação, no final, obteve uma acurácia de 98.04%, tempo de inferência de 0.025s por frame. O consumo de energia não foi informado. Apesar das métricas mostradas serem melhores que aquelas do estudo anterior, a rede utilizada possui 51 milhões de parâmetros que, quando comparada à outra, são 29 vezes mais. Ainda, quanto ao tempo de inferência, como o modelo de FPGA utilizado no segundo estudo é mais poderoso computacionalmente do que o primeiro, resultados melhores são previstos.

No trabalho [46], foi proposta a utilização da arquitetura ResNet50, que possui 50 camadas convolucionais, com *transfer learning*, para classificar doenças em plantas. FPGAs não foram utilizados neste estudo. O *dataset* utilizado consiste de 38 diferentes categorias de plantas saudáveis e com doenças. O modelo obteve uma acurácia de 99.32%. Nota-se que, apesar de obter melhor resultado, aqueles obtidos nos FPGAs dos estudos anteriores estão próximos. Portanto, a implementação de modelos CNN classificadores de doenças de plantas em FPGAs são altamente viáveis.

3. TRABALHO PROPOSTO

3.1. Metodologia de Desenvolvimento

No estudo, primeiramente, foi realizado o treinamento de uma CNN utilizando um conjunto de dados que contém imagens acerca de plantas saudáveis e doentes de diferentes tipos. Com isso, obtêve-se, ao fim do treinamento, um modelo classificador com pesos pontos flutuantes de 32 bits. Para reduzir os bits de precisão da rede, foram gerados, a partir do modelo original, dois modelos de pesos inteiros de 8 bits (com quantizações durante e pós-treino) e um de 4 bits com a técnica de quantização pós-treino, conforme a Figura 4. Com o intuito de executar os modelos quantizados no FPGA, eles foram compilados para instruções DPU para que possam ser lidos na placa pela biblioteca disponível pelo *framework* Vitis AI e analisar o impacto da quantidade de bits na acurácia, tempo de inferência e tamanho do modelo. Para efeito de comparação, os testes de inferência com imagens de plantas diferentes daquelas do conjunto de treinamento foram realizados para cada modelo em cada *hardware* (GPU, CPU e FPGA), sendo que, para o FPGA, foram executados apenas os modelos quantizados.

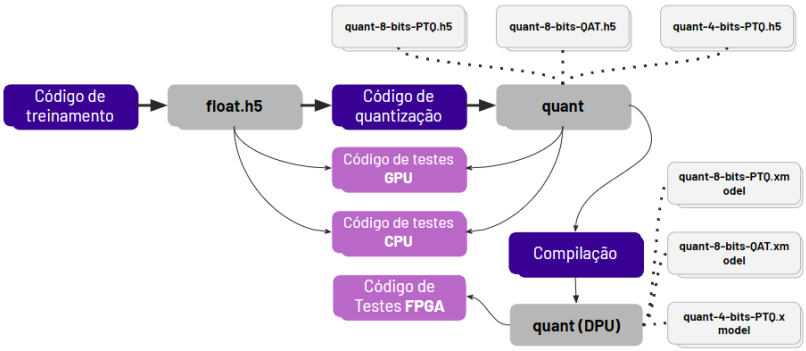


Figura 4. Fluxo da geração dos modelos e testes. Fonte: Autoria própria

No final, foram comparadas as métricas: acurácia, F1-Score, vazão (inferências por segundo) e tamanho dos modelos obtidas nos testes a fim analisar se a implantação da rede do FPGA é favorável frente a diferentes tipos de *hardware*. A CPU utilizada possui as seguintes configurações: Intel Core I5 10500H 4.5GHz; a GPU Nvidia Geforce GTX 1650 4GB; e o FPGA Xilinx Zynq UltraScale+ MPSoC da placa Ultra96-v1 [47]. Vale salientar que a depender do modelo do FPGA, a DPU pode utilizar diversos núcleos, no entanto no modelo que temos disponível utiliza-se apenas um núcleo, conforme documentação [48].

3.2 *Dataset*

O *dataset* utilizado é constituído pela combinação das bases *Plant Village* [49] e *FieldPlant* [50]. Ele possui 55.690 imagens, todas de tamanho 224 x 244 pixels, de diferentes tipos de plantas saudáveis e doentes. O conjunto tem, ao todo, 27 classes, como apresentado na Tabela 1. Quanto à distribuição, aproximadamente 70% dos dados foram utilizados para treinamento, 20% para validação e 10% para testes.

Tabela 1: Conjunto de dados de treinamento e validação. (Autoria própria)

Rótulo	Classe	Quantidade de Imagens	Rótulo	Classe	Quantidade de Imagens
Maçã com sarna	0	2.219	Uva saudável	13	1.892
Maçã com podridão negra	1	2.187	Batata com ferrugem precoce	14	2.144
Maçã com ferrugem	2	1.964	Batata com ferrugem tardia	15	2.139
Maçã saudável	3	2.208	Batata saudável	16	2.026
Pimentão com manchas bacterianas	4	2.113	Tomate com manchas bacterianas	17	1.902
Pimentão saudável	5	2.188	Tomate com ferrugem precoce	18	2.126
Milho com mancha cinza	6	1.842	Tomate com ferrugem tardia	19	2.051
Milho com ferrugem	7	2.110	Tomate com mofo de folha	20	2.082
Milho com ferrugem da folha do norte	8	2.108	Tomate com mancha foliar de Septoria	21	1.945
Milho saudável	9	2.059	Tomate com ácaro	22	1.941
Uva com podridão negra	10	2.088	Tomate com ponto alvo	23	2.027
Uva com sarampo preto	11	2.120	Tomate com vírus do enrolamento da folha	24	2.167
Uva com ferrugem	12	1.922	Tomate com vírus do mosaico	25	1.990
			Tomate saudável	26	2.130

3.3. Implementação

Toda a implementação em código foi feita utilizando a linguagem Python juntamente com o *framework* de aprendizado de máquina TensorFlow. O treinamento do modelo original (sem quantização) foi realizado em GPU graças à sua velocidade superior em tal etapa. Para as camadas iniciais do modelo, utilizou-se a técnica de *transfer*

leraning da arquitetura ResNet50, sem alteração dos seus pesos. As camadas posteriores, aquelas que recebem, como entrada, a saída da ResNet50, e têm seus pesos atualizados durante o treinamento, foram formadas por GlobalAvaragePooling2D, Dropout e Dense. O modelo gerado (Figura 5) teve, no total, cerca de 23.643.035 de parâmetros, sendo 55.323 destes treináveis e 23.587.712 não treináveis (ResNet50). O modelo treinado, no final, foi salvo em formato ".h5" para poder ser lido posteriormente pelas bibliotecas do Tensorflow nas etapas de teste e quantização.

```
 Layer (type)                 Output Shape              Param #
=================================================================
 resnet50 (Functional)        (None, 7, 8, 2048)        23587712

 global_average_pooling2d (G  (None, 2048)              0
 lobalAveragePooling2D)

 dropout (Dropout)            (None, 2048)              0

 dense (Dense)                (None, 27)                55323

=================================================================
Total params: 23,643,035
Trainable params: 55,323
Non-trainable params: 23,587,712
```

Figura 5. Modelo CNN 32-bits ponto flutuante. Fonte: Autoria própria

Os rótulos do conjunto foram transformados utilizando o método *one-hot encoding* para otimizar o resultado da rede. Para testes, imagens não pertencentes ao *dataset* anterior foram usadas. Uma semente foi utilizada na leitura dos arquivos de teste para que todos os modelos pudessem ser comparados igualmente com uma mesma sequência. Além disso, os dados não foram embaralhados (*shuffle*) nos testes.

Para a quantização do modelo original (32 bits), utilizou-se o conjunto ferramental disponibilizado pela AMD Xilinx, Vitis AI. A Figura 6 demonstra como as camadas são renomeadas pela ferramenta Vitis AI após o processo de quantização. Posteriormente, com o auxílio do compilador Compiler AI, as redes quantizadas foram compiladas em instruções capazes de serem lidas pela DPU e salvas em arquivos de formato "xmodel".

```
quant_conv5_block3_out (Quanti  (None, 7, 7, 2048)  4          ['quant_conv5_block3_add[0][0]']
zeWrapper)

quant_global_average_pooling2d  (None, 2048)        4          ['quant_conv5_block3_out[0][0]']
_2 (QuantizeWrapper)

quant_dense (QuantizeWrapper)  (None, 27)           55330      ['quant_global_average_pooling2d_
                                                               2[0][0]']

quant_dense_softmax (QuantizeW  (None, 27)          4          ['quant_dense[0][0]']
rapper)

==================================================================================================
Total params: 23,537,463
Trainable params: 23,536,795
Non-trainable params: 668
```

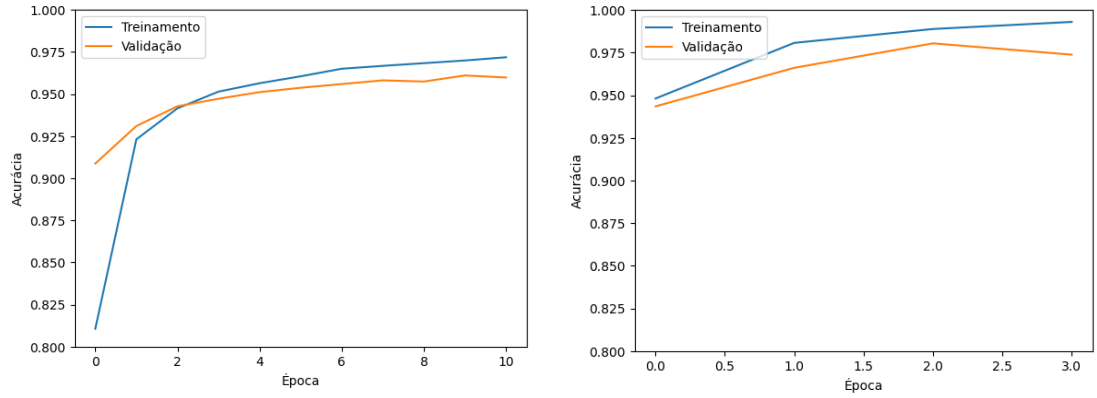
Figura 6. Modelo CNN 8-bits inteiro. Fonte: Autoria própria

Com todos os modelos criados, testes de inferência em cada um foram realizados. O modelo original (ponto flutuante) foi testado apenas em CPU e GPU, já que, para a DPU, tal modelo não pode ser compilado. As redes quantizadas foram testadas em CPU, GPU e FPGA. Ademais, para conseguir com que as bibliotecas da Vitis AI fossem executadas no FPGA disponível (Ultra96-v1) [47] e para comunicação do código Python com modelo no FPGA foi preciso utilizar a biblioteca overlay PYNQ-DPU [48], já que a placa não tem suporte nativo disponível pela AMD Xilinx. Assim, o autor deste trabalho precisou fazer alguns ajustes no tutorial Vitis para realização dos experimentos, o qual está disponibilizado em [51], assim como o código dos experimentos [52].

2.6. Resultados e Discussões

Durante o treinamento do modelo original (32 bits) foi possível, como mostra o gráfico da Figura 7(a), obter uma acurácia de 97,18% e, na validação, de 96,10% com 10 épocas. No treinamento do modelo quantizado com QAT (8 bits), que utiliza o modelo original como base, obteve-se uma acurácia de 99.74% e, na validação, de 97,74% com apenas 3 épocas, utilizando parada precoce (do inglês *early stopping*) com monitoramento automático de perda. conforme Figura 7(b). Nota-se que o resultado utilizando QAT conseguiu ser ainda melhor que o modelo original. Talvez o modelo original pudesse obter uma melhor saída caso a taxa de aprendizado fosse menor. Ou

ainda se mais épocas fossem utilizadas. Contudo, caso a última opção fosse realizada com a mesma taxa de aprendizagem, possivelmente aconteceria *overfitting*, dado que a acurácia validação tende a se distanciar da acurácia de treinamento nas épocas finais.



(a) (b)

Figura 7. Acurácias de treinamento e validação por época durante o treinamento: (a) modelo original com 32 bits; (b) modelo quantizado com QAT. Fonte: Autoria própria

Para testar tais ideias, foi realizado um novo treinamento para gerar um novo modelo 32 bits, no entanto, agora, utilizou-se uma taxa de treinamento de 0.0001 e 50 épocas. A acurácia obtida no treinamento e validação foram, respectivamente, 96,24% e 95,50%. Portanto, os resultados obtidos com o novo modelo foram inferiores que aqueles do anterior. Contudo, observa-se na Figura 8 que a chance de *overfitting* foi menor, já que ambas as acurácias de treinamento e validação estão bem próximas.

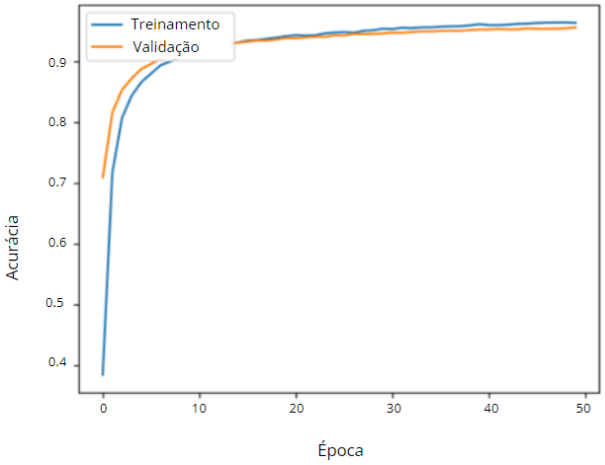


Figura 8. Acurácias de treinamento e validação por época durante o treinamento do segundo modelo original. Fonte: Autoria própria

Portanto, o resultado obtido pelo modelo quantizado com quantização durante o treino foi melhor, possivelmente devido a um segundo treinamento. Para corroborar, os autores [53] obtiveram, em seu trabalho, melhor acurácia quando treinaram sua rede 2 vezes com *transfer learning*. Na primeira fase, apenas os pesos das novas camadas foram inferidos, enquanto aqueles das camadas convolucionais do modelo pré-treinado se mantiveram. Na segunda fase, todos os parâmetros foram treinados utilizando o *dataset* alvo. Com isso, segundo os autores, pôde-se fazer um ajuste fino de todos os pesos da rede.

A Figura 9(a) exibe as métricas obtidas a partir do teste do modelo original em GPU, enquanto a Figura 9(b) apresenta as métricas resultantes do teste do modelo original em CPU. O referido modelo não foi testado em FPGA dado que a DPU não tem suporte para redes neurais com pesos de ponto flutuante. Quanto ao resultado, nota-se que as métricas obtidas não se diferenciam em ambos os *hardwares*. No domínio, *F1-score* pode ser a melhor métrica a ser avaliada, já que falsos negativos (predizer que uma planta doente está saudável) possuem custo semelhante a falsos positivos (predizer que uma planta saudável está doente). Dito isto, a rede obteve um ótimo *F1-Score*. Porém, a classe 23, quando comparada às outras, não obteve resultados tão promissores.

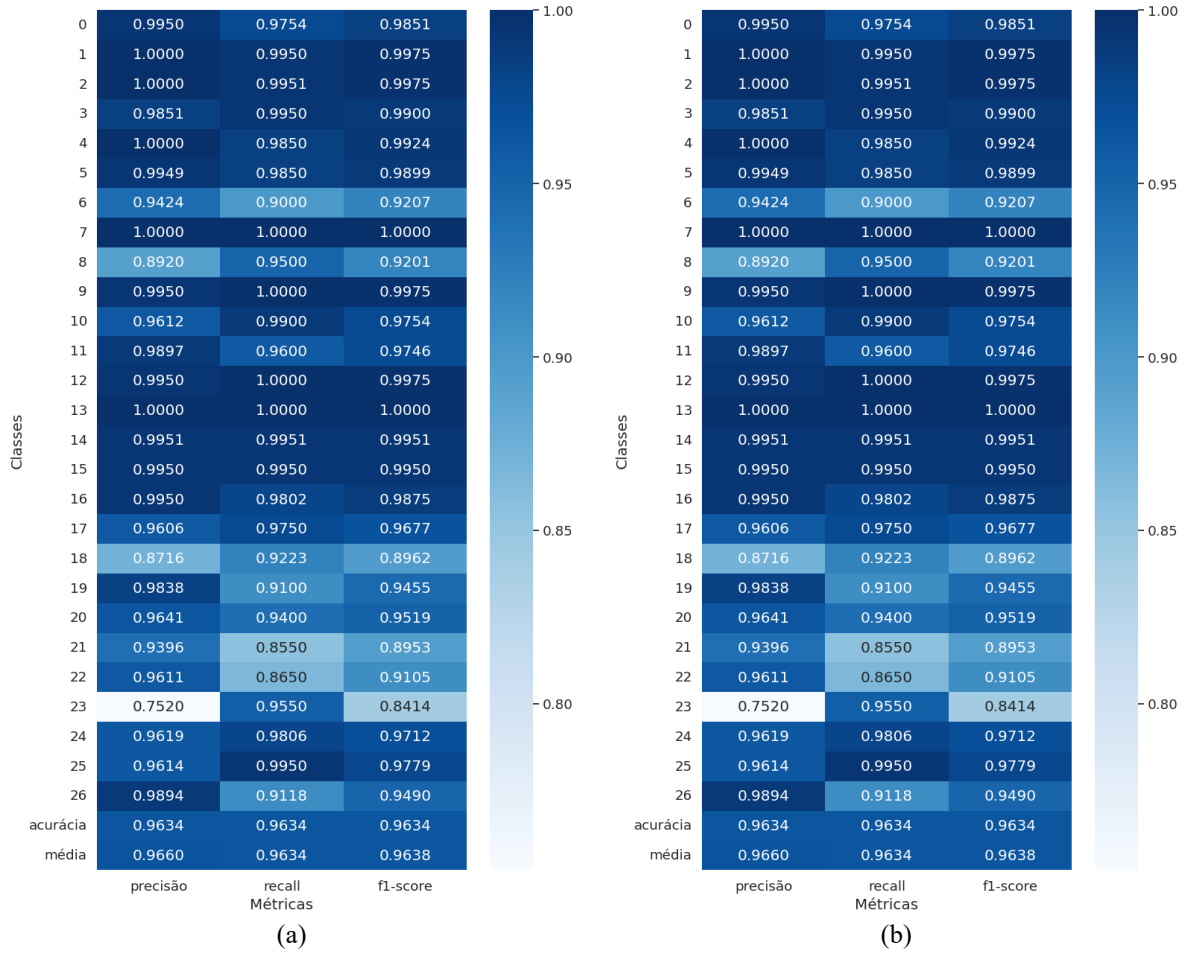


Figura 9. Métricas, por classe, obtidas na etapa de teste do modelo original em GPU (a) e CPU (b). Fonte: Autoria própria

Na Figura 10, vê-se a comparação do F1 Score, de forma individual para cada classe, para os modelos quantizados utilizando as técnicas QAT e PTQ (8 bits), em CPU. Também é possível visualizar a média de cada uma das técnicas por meio de linhas pontilhadas. Algumas das classes que não foram tão favoráveis no modelo original, tornaram-se melhores neste (ex. classe 23). É possível que, com um segundo treinamento, a rede neural tenha conseguido absorver melhor as características dessas classes. Em contrapartida, as predições de algumas classes foram reduzidas. Por exemplo, a classe 5, que tinha F1-Score de 99%, reduziu-se para 95%. Isso se deu, possivelmente, por um *overfitting* dessas classes durante o segundo treinamento, que aparentemente não era necessário para a melhoria do aprendizado tais. Entretanto, no geral, os resultados de todas as métricas foram melhores. Além disso, os resultados dos testes obtidos com o modelo 8 bits PTQ foram inferiores quando comparados aos do modelo quantizado com QAT. O F1-Score médio obtido com PTQ foi de 95,65%, com uma acurácia de 95,60%. Por outro lado, ao utilizar QAT, o F1-Score médio atingiu 97,83%, acompanhado por uma acurácia de mesmo valor. Portanto, para a rede do presente estudo, a quantização durante o treino se mostrou mais eficiente, realçando assim a afirmação dita no referencial teórico que QAT pode conseguir melhores métricas que PTQ, no entanto com a desvantagem de maior tempo de implantação.

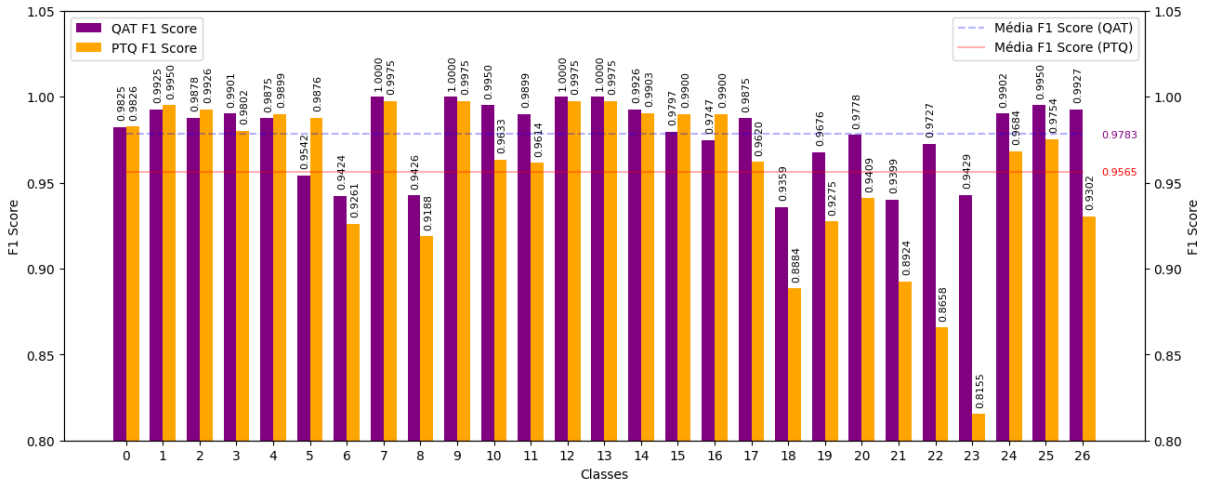


Figura 10. F1-Score, por classe e média, obtido na etapa de teste dos modelos quantizados com QAT e PTQ (8 bits) em CPU. Fonte: Autoria própria

As métricas obtidas nos testes dos modelos quantizados com QAT e PTQ (8 bits) em GPU foram semelhantes àquelas alcançadas em CPU, conforme a Figura 11. Com QAT foi obtida uma acurácia de 97,74% e F1-Score médio de mesmo valor, enquanto com PTQ foi alcançada uma acurácia 95,66% e F1-Score médio de 95,69%. Portanto, para o estudo, inferências feitas em ambos os *hardwares* não tiveram diferenças significativas quanto à classificação.

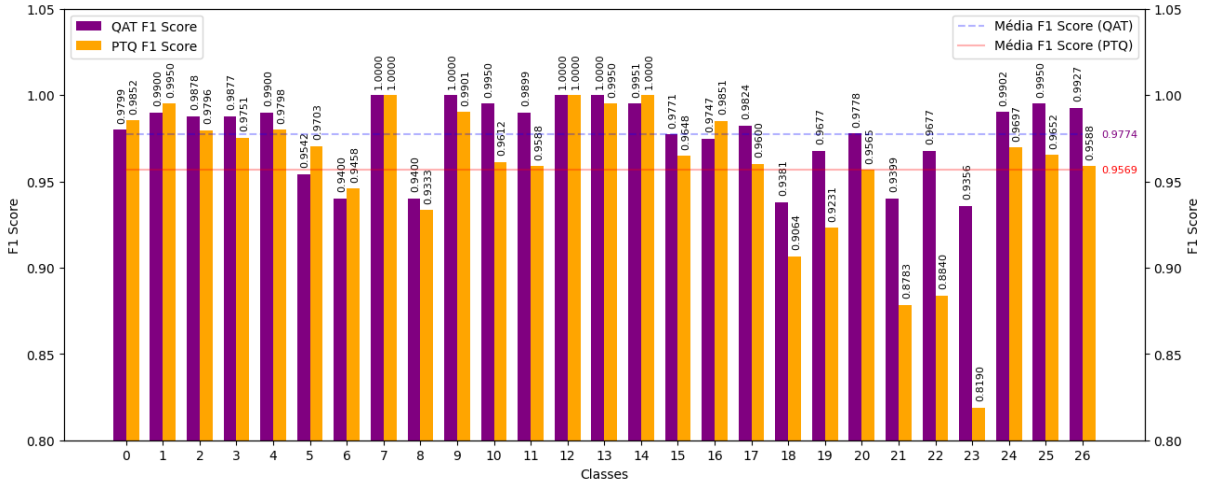


Figura 11. F1-Score, por classe e média, obtido na etapa de teste dos modelos quantizados com QAT e PTQ (8 bits) em GPU. Fonte: Autoria própria

O resultado do teste com o modelo 4 bits obtido por PTQ (Figura 12) foi significantemente menor que os alcançados anteriormente com os outros modelos. É possível que uma precisão de 4 bits não seja suficiente para inferir predições acerca do domínio abordado. Portanto, para a rede, a quantização com 4 bits é inviável. Apesar disso, nota-se que algumas classes conseguiram ser superiores às outras. Talvez com um *dataset* maior e com o uso da técnica QAT, possa-se obter um melhor resultado. No entanto, não foi possível testar um modelo QAT de 4 bits, dado que, segundo a documentação da ferramenta Vitis AI, não tem como informar a quantidade de bits desejada para tal técnica.

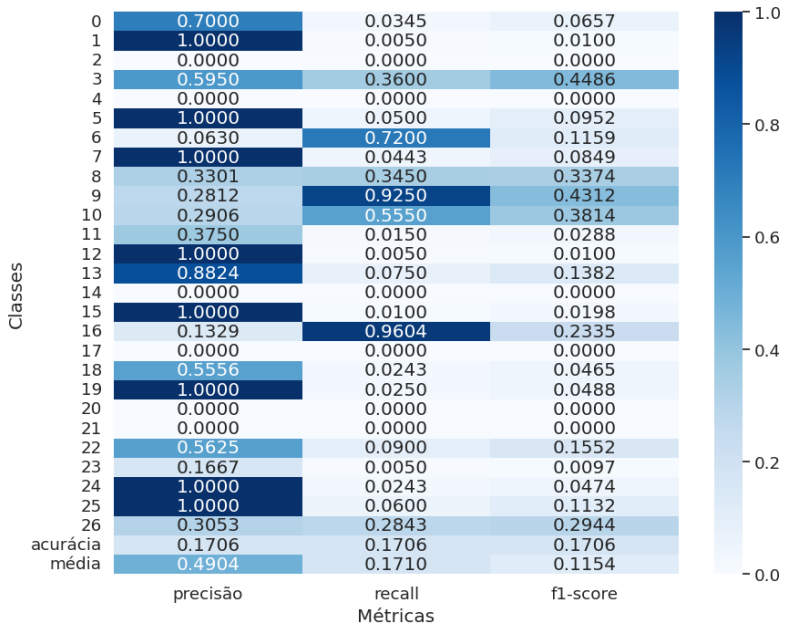


Figura 12. Métricas, por classe, obtidas na etapa de teste do modelo quantizado (PTQ 4 bits) em CPU. Fonte: Autoria própria

Inferências com ambos os modelos quantizados (8 bits) também foram realizadas no FPGA. Os resultados, quanto às métricas, se mantiveram satisfatórios quando executados na placa, como visto na Figura 13. Inclusive, o valor do F1-score médio com quantização durante o treino foi o mesmo daquele obtido em GPU. Ao fim, as quantizações com QAT e PTQ tiveram, respectivamente, F1 scores médios de 97,74% e 95,62% e acurácias de 97,74% e 95,56%.

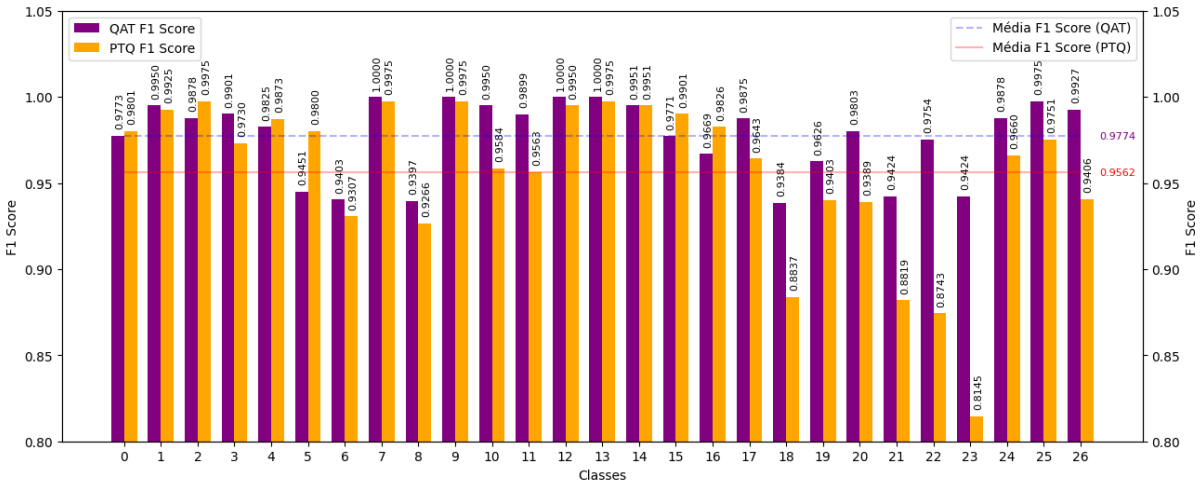


Figura 13. F1-Score, por classe e média, obtido na etapa de teste dos modelos quantizados com QAT e PTQ (8 bits) em FPGA. Fonte: Autoria própria

Quanto à vazão (inferências por segundo), como pode ser visto na Figura 14, a GPU conseguiu, como esperado, uma melhor vazão em todos os modelos. Já o FPGA conseguiu ser superior à CPU, mas não por muito. Contudo, os modelos quantizados quando executados em CPU e GPU, tiveram vazões consideravelmente inferiores quando comparadas com aquelas obtidas pelo modelo 32 bits nos mesmos *hardwares*. Tal diferença pôde se dar devido a ferramenta Vitis AI ser projetada especificamente para que seus modelos sejam executados em seus sistemas embarcados. Ademais, esperava-se uma melhor vazão do modelo 4 bits frente aos outros modelos, uma vez que cálculos computacionais são simplificados quando utilizados menores números de bits. Agora, quanto a GPU, apesar dela ser mais rápida, o gasto de energia é maior. Portanto, caso a aplicação da rede não requeira baixíssimas latências e o alto consumo de energia seja algo proibitivo, a implantação do modelo em FPGAs se torna a melhor opção. Além disso, a vazão obtida na placa (27,60 inferências/s) ainda consegue suprir a velocidade necessária para aplicações em tempo real. Vale salientar que tempos de inferência melhores poderiam ser atingidos caso o FPGA utilizado não fosse limitado a apenas um núcleo de DPU.

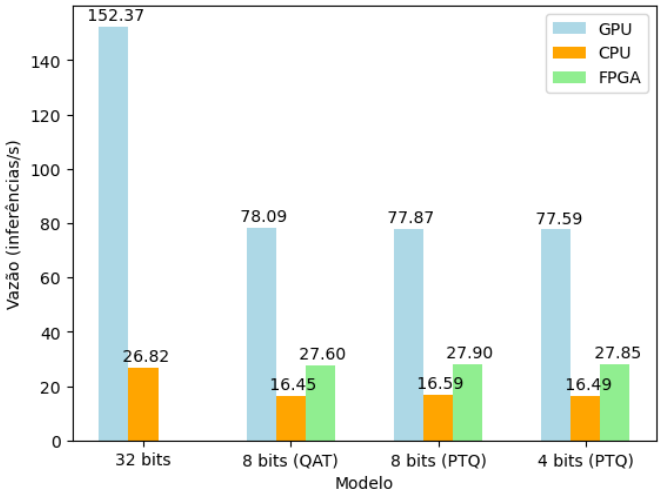


Figura 14. Vazão (inferências/s) por modelo para cada tipo de *hardware*. Fonte: Autoria própria

Para efeito de comparação do tamanho do modelo e impacto gerado pela quantização, utilizamos o tamanho do arquivo gerado (.h5) na criação de cada modelo. A diferença entre os modelos quantizados e o modelo original foi ínfima, como pode ser visto na Figura 15. Contudo, quando compilados para a DPU (.xmodel), eles se tornaram aproximadamente 5x menores, o que é bem favorável, dado que estes são os arquivos que devem ser armazenados no FPGA. Ainda, a rede quantizada de 4 bits não teve nenhum benefício de tamanho frente aos seus semelhantes.

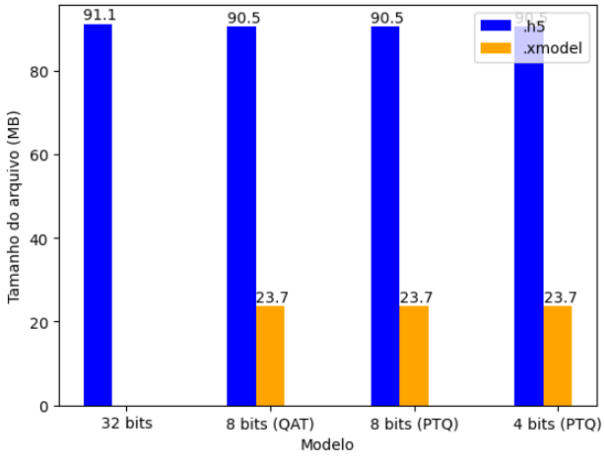


Figura 15. Tamanho do arquivo (MB) por modelo para cada tipo de arquivo. Fonte: Autoria própria

3. CONCLUSÕES

Os resultados obtidos pelas inferências no FPGA foram satisfatórios. Com a vazão e acurácia obtidas na placa, é possível classificar, a partir de uma imagem, o estado atual de uma planta em tempo real e de maneira precisa. Ainda, os modelos quantizados mostraram métricas similares quando executados em FPGA em comparação com a CPU e GPU. Quanto às vazões, àquelas obtidas em ambos CPU e FPGA foram semelhantes enquanto que a GPU se mostrou superior no quesito. Ademais, a redução significativa no tamanho dos arquivos gerados após a compilação para a DPU é benéfica, já que torna mais eficiente o armazenamento no FPGA, economizando recursos. Potanto, como os resultados das classificações com os três *hardwares* foram semelhantes, caso a vazão obtida em FPGA seja suficiente para a realização de um monitoramento contínuo de uma colheita, a implantação da rede na placa pode ser a melhor opção dado que, dentre os *hardwares* testados, ele possui o menor consumo energético. Agora, caso inferências muito rápidas sejam necessárias e um aumento do consumo energético não seja algo proibitivo, a GPU parece a melhor escolha.

Quanto a quantização utilizando 4 bits, ela se mostrou inviável para o estudo devido à sua baixa acurácia obtida nos testes. Ainda, os resultados obtidos com a quantização durante o treino se mostraram superiores quando comparados com a quantização pós-treino, inclusive, com a versão 32 bits. Portanto, um segundo treinamento pode ser benéfico em casos específicos.

Contudo, a quantização do modelo de pesos ponto flutuante 32 bits em inteiro 4 bits utilizando quantização durante o treino não foi possível de ser realizado com a ferramenta escolhida. O uso da técnica poderia ser

promissor na tentativa de aumentar a acurácia de redes com pesos de precisão reduzida, dado que sua utilização na geração de um modelo 8 bits conseguiu aumentar generosamente a métrica em questão. Ademais, apenas um núcleo da DPU pôde ser aproveitado com o modelo de FPGA utilizado. O número de inferências por segundo poderia ser maior caso mais núcleos estivessem disponíveis. A eficiência energética de cada *hardware* durante a execução do algoritmo também não foi abordada. Além disso, validações externas em condições de campo real não foram realizadas.

Como trabalho futuro, seria interessante comparar resultados obtidos com a ferramenta Vitis AI com a ferramenta FINN, também da AMD Xilinx, já que ambas possuem diferentes opções de implementações. Enquanto o framework Vitis AI é um acelerador genérico de redes neurais profundas que utiliza um motor próprio (DPU) para realizar as inferências, FINN é um framework que possibilita, além da quantização, a conversão de modelos quantizados em *bitstream*, que é um formato de arquivo nativamente legível ao FPGA e específico para o modelo que está sendo gerado. Dessa forma, com tal estudo, poderia analisar se o uso de um motor genérico traz ou não benefícios frente ao formato nativo. Além disso, como o consumo de energia durante as inferências não foi medido neste estudo, outro trabalho poderia tentar avaliar essa métrica em diferentes tipos de *hardware* e, com isso, determinar se, por exemplo, qual dispositivo tem maior eficiência de vazão por gasto de energia. E por fim, como trabalho futuro pretendido, planeja-se integrar o modelo em FPGA aos sistemas de classificação de contexto e comparar sua eficiência como acelerador, comparando com a versão puramente em software.

4. REFERÊNCIAS BIBLIOGRÁFICAS

[1] O'BRIEN, P. A. Biological control of plant diseases**. Australasian Plant Pathology**, v. 46, n. 4, p. 293–304, 23 mar. 2017.

[2] STRANGE, R. N.; SCOTT, P. R. Plant disease: a threat to global food security**. Annual Review of Phytopathology**, v. 43, p. 83–116, 2005.

[3] KC, K. et al. Depthwise separable convolution architectures for plant disease classification. **Computers and Electronics in Agriculture**, v. 165, p. 104948, out. 2019.

[4] LI, L.; ZHANG, S.; WANG, B. Plant Disease Detection and Classification by Deep Learning—A Review. **IEEE Access**, v. 9, p. 56683–56698, 2021.

[5] QASAIMEH, Murad et al. Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels. In: **2019 IEEE international conference on embedded software and systems** (ICESS). IEEE, 2019. p. 1-8.

[6] ROSATO, Dominick V. **Plastics engineered product design**. Elsevier, 2003.

[7] PATEL, Chandrakant D.; CHEN, Chun-Hsien (Ed**.). Digital Manufacturing: The Industrialization of" Art to Part" 3D Additive Printing**. Elsevier, 2022.

[8] TU, Jiyuan et al. **Computational fluid dynamics: a practical approach**. Elsevier, 2023.

[9] BANERJEE, Ananya. An in-depth look at blockchain technology: Architecture and security concerns. In: **Distributed Computing to Blockchain**. Academic Press, 2023. p. 297-325.

[10] PARKER, Michael. **Digital Signal Processing 101: Everything you need to know to get started**. Newnes, 2017.

[11] CHAN, Lau Mai; SRINIVASAN, Rajagopalan. A graphic processing unit (gpu) algorithm for improved variable selection in multivariate process monitoring. In: **Computer Aided Chemical Engineering**. Elsevier, 2012. p. 1532-1536.

[12] MEHDIPOUR, Farhad; NOORI, Hamid; JAVADI, Bahman. Energy-efficient big data analytics in datacenters. In: **Advances in Computers**. Elsevier, 2016. p. 59-101.

[13] YAZDEEN, Abdulmajeed Adil et al. FPGA implementations for data encryption and decryption via concurrent and parallel computation: A review**. Qubahan Academic Journal**, v. 1, n. 2, p. 8-16, 2021.

[14] BOUTROS, Andrew; BETZ, Vaughn. FPGA architecture: Principles and progression**. IEEE Circuits and Systems Magazine**, v. 21, n. 2, p. 4-29, 2021.

[15] ZHANG, Chen et al. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In: **Proceedings of the 2016 International Symposium on Low Power Electronics and Design**. 2016. p. 326-331.

[16] DÁVILA GUZMÁN, María Angélica et al. Cooperative CPU, GPU, and FPGA heterogeneous execution with EngineCL. **The Journal of Supercomputing**, v. 75, p. 1732-1746, 2019.

[17] FINN. [*s. d.*]. **finn**. Disponível em: https://xilinx.github.io/finn/. Acesso em: 28 set. 2023.

[18] VITIS AI. [*s. d.*]. Disponível em: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html. Acesso em: 27 set. 2023.

[19] PRODUTOS DA XILINX. [*s. d.*]. Disponível em: https://www.amd.com/pt/products/xilinx. Acesso em: 28 set. 2023.

[20] FUKUDA, Yuta; YOSHIDA, Kota; FUJINO, Takeshi. Evaluation of Model Quantization Method on Vitis-AI for Mitigating Adversarial Examples. **IEICE Technical Report; IEICE Tech. Rep**., v. 122, n. 286, p. 182-187, 2022.

[21] ZYNQ ULTRASCALE+ MPSOC: DPUCZDX8G • VITIS AI USER GUIDE (UG1414) • READER • AMD ADAPTIVE COMPUTING DOCUMENTATION PORTAL. [*s. d.*]. Disponível em: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Zynq-UltraScale-MPSoC-DPUCZDX8G. Acesso em: 28 set. 2023.
[22] LI, Chao et al. Edge Real-Time Object Detection and DPU-Based Hardware Implementation for Optical Remote Sensing Images. **Remote Sensing**, v. 15, n. 16, p. 3975, 2023.
[23] MOLNAR, Christoph. **Interpretable machine learning**. Lulu. com, 2020.
[24] JANIESCH, C.; ZSCHECH, P.; HEINRICH, K. Machine learning and deep learning. **Electronic Markets**, v. 31, p. 685–695, 8 abr. 2021.
[25] CICHY, R. M.; KAISER, D. Deep Neural Networks as Scientific Models. **Trends in Cognitive Sciences**, v. 23, n. 4, p. 305–317, abr. 2019.
[26] ZANIOL, Cristina; MORAES, Jean Carlo Pech de. Previsão e núcleo de inflação: uma abordagem baseada em wavelets e redes neurais. In: **Congresso Nacional de Matemática Aplicada e Computacional (39.: 2019: Uberlândia, MG). Proceeding Series of the Brazilian Society of Computational and Applied Mathematics. São Carlos: SBMAC, 2020**. 2020.
[27] HAYKIN, Simon. **Neural networks: a comprehensive foundation**. Prentice Hall PTR, 1998.
[28] MÜLLER, Andreas C.; GUIDO, Sarah**. Introduction to machine learning with Python: a guide for data scientists**. O'Reilly Media, Inc.", 2016.
[29] YU, Lean et al. Missing data preprocessing in credit classification: One-hot encoding or imputation?. **Emerging Markets Finance and Trade**, v. 58, n. 2, p. 472-482, 2022.
[30] UCHIDA, Kazutaka; TANAKA, Masayuki; OKUTOMI, Masatoshi. Coupled convolution layer for convolutional neural network. **Neural Networks**, v. 105, p. 197-205, 2018.
[31] HUSSAIN, Mahbub; BIRD, Jordan J.; FARIA, Diego R. A study on cnn transfer learning for image classification. In: **Advances in Computational Intelligence Systems: Contributions Presented at the 18th UK Workshop on Computational Intelligence, September 5-7, 2018, Nottingham, UK**. Springer International Publishing, 2019. p. 191-202.
[32] SUN, Manli et al. Learning pooling for convolutional neural network. **Neurocomputing**, v. 224, p. 96-104, 2017.
[33] WEISS, K.; KHOSHGOFTAAR, T. M.; WANG, D. A survey of transfer learning. **Journal of Big Data**, v. 3, n. 1, 28 maio 2016.
[34] GUPTA, Jaya; PATHAK, Sunil; KUMAR, Gireesh. Deep learning (CNN) and transfer learning: a review. In: **Journal of Physics: Conference Series**. IOP Publishing, 2022. p. 012029.
[35] YING, Wei et al. **Transfer learning via learning to transfer**. In: International Conference on Machine Learning. PMLR, 2018. p. 5085-5094.
[36] TAN, Mingxing; LE, Quoc. Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on machine learning. PMLR, 2019. p. 6105-6114.
[37] HE, Kaiming et al. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. 2016. p. 770-778.
[38] HOWARD, Andrew G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.
[39] GONG, Yunchao et al. Compressing deep convolutional networks using vector quantization. **arXiv preprint arXiv:1412.6115**, 2014.
[40] YANG, Jiwei et al. Quantization networks. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. 2019. p. 7308-7316.
[41] GUO, Yunhui. A survey on methods and theories of quantized neural networks**. arXiv preprint arXiv:1808.04752**, 2018.
[42] SHYMYRBAY, Ayan; FOUDA, Mohammed E.; ELTAWIL, Ahmed. Low Precision Quantization-aware Training in Spiking Neural Networks with Differentiable Quantization Function. **arXiv preprint arXiv:2305.19295**, 2023.
[43] WANG, Jin; GU, Shenshen. Fpga implementation of object detection accelerator based on vitis-ai. In: **2021 11th International Conference on Information Science and Technology (ICIST)**. IEEE, 2021. p. 571-577.
[44] LUO, Yuexuan et al. FPGA–accelerated CNN for real-time plant disease identification**. Computers and Electronics in Agriculture**, v. 207, p. 107715, 2023.
[45] SHAH, Parin et al. Plant Leaf Disease Classification using Convolutional Neural Network on FPGA. In: **2023 International Conference on Device Intelligence, Computing and Communication Technologies,(DICCT)**. IEEE, 2023. p. 307-311.
[46] BALAVANI, Kola et al. An Optimized Plant Disease Classification System Based on Resnet-50 Architecture and Transfer Learning. In**: 2023 4th International Conference for Emerging Technology (INCET)**. IEEE, 2023. p. 1-5.
[47] ULTRA96-V1 - ELEMENT14 COMMUNITY. 20 out. 2021. Disponível em: https://community.element14.com/products/devtools/avnetboardscommunity/w/boards/23080/ultra96-v1. Acesso em: 28 set. 2023.

[48] DPU ON PYNQ. [*S. l.*]: Xilinx, 15 ago. 2023. Disponível em: https://github.com/Xilinx/DPU-PYNQ. Acesso em: 27 set. 2023.
[49] HUGHES, David et al. An open access repository of images on plant health to enable the development of mobile disease diagnostics. **arXiv preprint arXiv:1511.08060**, 2015.
[50] MOUPOJOU, Emmanuel et al. FieldPlant: A Dataset of Field Plant Images for Plant Disease Detection and Classification With Deep Learning. **IEEE Access**, v. 11, p. 35398-35410, 2023.
[51] TCC-VITIS-AI-REPOSITORY/VITIS-AI-TUTORIAL AT MAIN · ARAUJOLUCAS3005/TCC-VITIS-AI-REPOSITORY. [*s. d.*]. **GitHub**. Acesso em: https://github.com/araujolucas3005/tcc-vitis-ai-repository/tree/main/vitis-ai-tutorial. Acedido em: 28 set. 2023.
[52] TCC-VITIS-AI-REPOSITORY/CODE AT MAIN · ARAUJOLUCAS3005/TCC-VITIS-AI-REPOSITORY. [*s. d.*]. **GitHub**. Disponível em: https://github.com/araujolucas3005/tcc-vitis-ai-repository/tree/main/code. Acesso em: 28 set. 2023.
[53] CHEN, Junde et al. Identification of plant disease images via a squeeze-and-excitation MobileNet model and twice transfer learning. **IET Image Processing**, v. 15, n. 5, p. 1115-1127, 2021.