Lucas Oliveira de Figueiredo

Estudo de viabilidade para implementação em hardware de técnica de aprendizado de máquina

Campina Grande, PB

2020

Lucas Oliveira de Figueiredo

Estudo de viabilidade para implementação em hardware de técnica de aprendizado de máquina

Trabalho de Conclusão de Curso (TCC) submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática

Departamento de Engenharia Elétrica

Orientador: Rafael Bezerra Correia Lima

Campina Grande, PB

2020

Lucas Oliveira de Figueiredo

Estudo de viabilidade para implementação em hardware de técnica de aprendizado de máquina

Trabalho de Conclusão de Curso (TCC) submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Trabalho aprovado. Campina Grande, PB, _____/_____/_____.

Rafael Bezerra Correia Lima
Orientador

George Acioli Júnior
Convidado

Campina Grande, PB
2020

Agradecimentos

Agradeço aos meus pais, Solange Oliveira e Pelágio Figueiredo, pela paciência e apoio nas melhores e piores circunstâncias. E sim, pedir os almoços ajudou bastante.

À Nikelisson por emprestar sua FPGA para os fins deste trabalho e pelos rápidos debates sobre HDLs na cozinha.

Obrigado Adeilmo Júnior pela consultoria acadêmica e a Ravi Helon pela descontração tão necessária.

Por fim, sou grato ao Prof. Rafael Bezerra por sua orientação e disponibilidade excepcional frente as restrições de tempo.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

FPGA	*Field Programmable Gate Array*
ANN	*Artificial Neural Network*
IoT	*Internet of Things*
HDL	*Hardware Description Language*
CNN	*Convolutional Neural Network*
DNN	*Deep Neural Network*
BRAM	*Block Random-Access Memory*

Resumo

A popularização de redes neurais para problemas de classificação, a disponibilidade de ferramentas para descrição e treinamento de tais redes em serviços de nuvem e softwares para cálculo numérico, tem implicado em uma redução na carga de trabalho associada ao desenvolvimento de uma solução por rede neural. Porém limitações de desempenho em processadores e placas gráficas, comparados com o disponível em FPGAs (*Field Programmable Gate Array*) no seu suporte a paralelismo e design sob demanda, tem tornado a escolha desta última plataforma como hardware aceleradora para modelos treinados de redes. As múltiplas competências necessárias para a execução de um fluxo de desenvolvimento compreendendo desde o treinamento até a descrição em hardware de uma rede neural tem seus nuances estudados neste trabalho.

Palavras chave: Redes neurais, serviço de nuvem, FPGA, descrição em hardware.

Abstract

The proliferation of the use of neural networks for classification issues, the availability of tools to describe and train such networks in cloud infrastrucutres and numerical computing softwares, results in a reduction of the work load related to the deployment of neural network solutions. Limitations in performance of processors and graphical units, in comparison to what FPGAs have to offer in its support to parallel computing and under demand design, has make these platforms as first choice in hardware acceleration for trained neural networks models. The several proficiencies required to fulfill a development flow comprising the training of a network and its description in hardware have their details studied in this undergraduate thesis.

Keywords: Neural networks, cloud infrastructure, FPGA, hardware description.

Sumário

1 Introdução

O reconhecimento ou classificação de um objeto de natureza específica em uma imagem, de maneira automatizada, é uma tarefa cumprida com relativo sucesso por técnicas de aprendizado de máquina. Redes Neurais Artificiais (ANN - *Artificial Neural Networks*) em suas computações de múltiplas camadas, necessitam de um treinamento levando em consideração a disposição do objeto na imagem fornecida ao modelo, de modo que a inferência seja eficaz frente a ângulos distintos, distâncias variadas ou pouca nitidez. A disponibilidade de capacidade de processamento tanto para a etapa de treinamento, quanto de inferência, ou seja, aplicação, do modelo de aprendizado de máquina, pode estar sujeita a fatores como flexibilidade de programação ou consumo energético da plataforma de hardware escolhida (ZHANG, 2017).

Dispositivos de borda (*edge devices*), presentes no contexto de internet das coisas (IoT - *Internet of Things*) ou de IIoT (*Industrial Internet of Things*), são equipamentos eletrônicos que possuem capacidade de coletar informações de seu ambiente por meio de sensores e de transmitir essas informações para outros dispositivos. A autonomia, seja por baterias ou geradores de energia, como também a capacidade de processamento de dados desses dispositivos de borda pode ser limitada, e a otimização do uso desses recursos é de interesse no desenvolvimento e aplicação desses produtos. Um grande volume de dados oriundos de sensores pode se mostrar como desvantajoso na realização da tarefa deste dispositivo, e portanto, uma filtragem, seleção ou tratamento dessas informações gera um impacto no tempo computacional de um modelo de aprendizado de máquina sendo executado (LIN, 2019). A compressão de imagens ou o emprego de árvores de decisões para filtrar dados de sensores antes do processamento, são técnicas para melhorar o compromisso entre recursos computacionais, de armazenamento e bateria (BUSCHJÄGER, 2018).

Os processadores (CPU - *Central Processing Unit*) encontrados em computadores pessoais, ou placas gráficas (GPU - *Graphics Processing Unit*) em alguns hardwares robustos, são plataformas utilizadas e suficientes para treinamento e inferência por modelos de aprendizado de máquina, porém as limitações em sua arquitetura computacional, pelo mal desempenho na execução de paralelismo ou simplesmente em um design não específico para o modelo em questão, abrem espaço para a escolha de FPGAs (*Field Programmable Gate Array*) como plataformas para tais tarefas (ZHANG, 2017). A flexibilidade na descrição do hardware por meio de HDLs (*Hardware Description Languages*) e na alocação dos recursos para as diversas camadas de uma DNN, bem como na reutilização da plataforma para outra tarefas, são aspectos atraentes das FPGAs. Ainda, a disponibilidade de serviços em nuvem para o treinamento de modelos, como os da Microsoft Azure, libera tempo no desenvolvimento de uma solução.

Este trabalho tem como objetivo analisar a viabilidade da implementação em uma plataforma de hardware FGPA de um modelo de aprendizado máquina para classificação de imagens. Uma abordagem para tal implementação é exposta e detalhada em suas escolhas, seguindo o modelo ilustrado na Figura 1, de modo a formar um compêndio para esse fluxo de desenvolvimento que integra diversas competências. A abordagem propõe a descrição e treinamento de uma rede neural em software, com a descrição das camadas da rede e seus valores em hardware, para por fim implementar o modelo em uma FPGA, a qual irá fazer a inferência de objetos de entrada. A rede neural alvo, consiste em uma classificadora de dígitos de zero a nove manuscritos, com o conjunto de dados a serem avaliados oriundo de um banco de dados de terceiros e treinamento do modelo feito por serviço de nuvem, como também por softwares para cálculos numéricos.

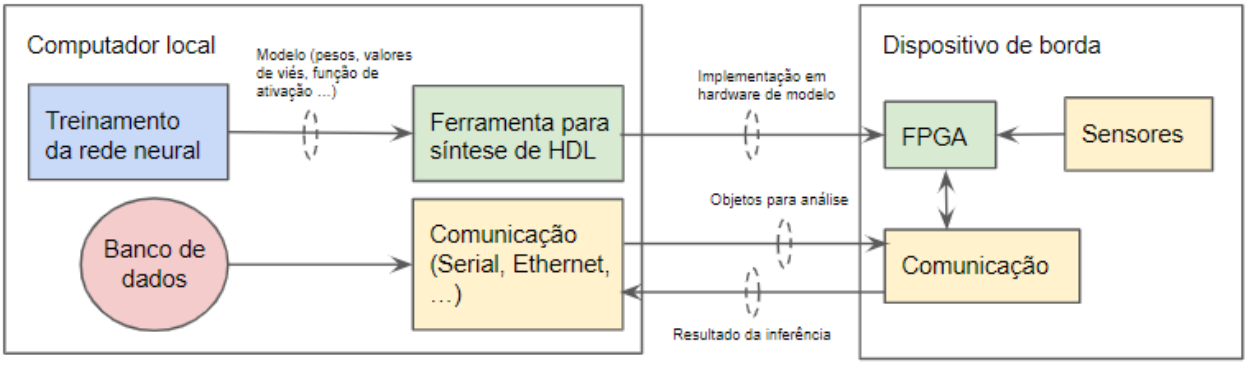

Figura 1 – Diagrama generalista para a implementação em hardware de modelo de rede neural.

2 Objetivos

2.1 Objetivo geral

O projeto de conclusão de curso tem como objetivo analisar a viabilidade da implementação em uma plataforma de hardware FGPA de um modelo de aprendizado máquina para classificação de imagens.

2.2 Objetivos específicos

Figuram os seguintes objetivos específicos:

- Estudar e compreender a técnica de aprendizado de máquina por ANN;
- Estudar e compreender a abordagem de árvores de decisão e quantização;
- Realizar o treinamento, dos modelos definidos, em serviço de nuvem da Microsoft Azure e software MATLAB;
- Buscar e entender o modelo de uma placa de desenvolvimento para FPGA em suas funcionalidades;
- Descrever as ferramentas para descrição em hardware por HDLs e seu uso em ambiente de desenvolvimento específico;
- Apresentar abordagem final para implementação em hardware de um modelo de aprendizado de máquina.

3 Referencial Teórico

3.1 Redes neurais artificiais

Em dividir para conquistar, redes neurais são estruturas matemáticas implementadas computacionalmente, porém com motivação na biologia. A segmentação de um conjunto de dados em seus atributos constituintes, a aplicação de operações e funções matemáticas sob esses valores para por fim gerar um produto que, após análise humana, recai sob uma classificação pré-definida e, portanto, classifica, identifica ou determina um padrão ou objeto, sumariza uma rede neural.

Como ilustrada na Figura 2, uma rede neural é vista como um conjunto de camadas, constituídas por elementos menores chamados de neurônios. Informações a serem analisadas são distribuídas ao longo dos neurônios da camada de entrada e cada camada subsequente possuem neurônios que realizam operações sob valores oriundos da camada imediatamente anterior. A camada de saída tem em seus produtos a representação da estimativa da rede neural sob a entrada em análise, seja um número de neurônios correspondentes ao número de classes disponíveis e o valor de cada neurônio corresponde a probabilidade dessa classe ser a correta. Esta descrição de rede neural não é universal, uma vez que a ilustração não leva em conta a possibilidade de realimentação para os cálculos em camadas escondidas na rede.

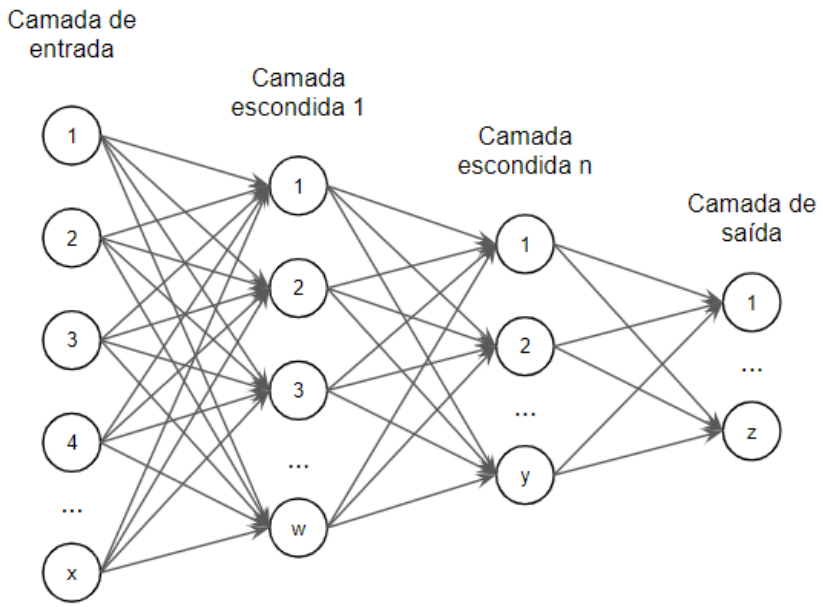

Figura 2 – Camadas em uma rede neural sem realimentação.

As camadas intermediárias na rede, chamadas de camadas escondidas, não possuem diretivas rigorosas para seu uso, uma vez, que onde a camada de entrada reflete o objeto de estudo, como por exemplo, uma imagem de 1920x1080 pixels em RGB (*Red Green Blue*) que define no mínimo um objeto com 6075 KiB de dados e portanto uma camada entrada com 6220800 de neurônios, ou no caso da camada de saída que dispõe de três objetos possíveis de serem identificados: um avião, um pássaro ou um homem, implicando portanto em uma camada de saída com três neurônios. Tais camadas intermediários são definidas pelo usuário e o treinamento da rede neural irá indicar se a escolha gera resultados satisfatórios ou não.

Uma vez situado o problema para resolução por técnica de aprendizado de máquina, ocorre o ajuste, ou treinamento, dos ganhos e demais valores que dizem respeito as operações matemáticas executadas pela rede neural. O treino culmina na geração de um modelo que atende aos limites de operação impostos pelo usuário, e ao ser alimentado com um objeto da mesma natureza que aqueles em que ele foi treinado, irá com sucesso realizar uma identificação. O emprego de funções matemáticas polinomiais, exponenciais, entre outras, para inferir resultados a partir de entradas, e a disponibilidade de capacidade computacional para executar operações cada vez mais complexas em escala, implicam na existências de vários tipos pré-definidos de camadas escondidas.

Em seu formato clássico, uma rede neural artificial (ANN) tem como saída do neurônio i de sua primeira camada escondida, a seguinte relação:

$$o^i_{HL_1} = f_{act}(\sum_{m=1}^{x}[i^m_{IL} \ . \ w^{im}_{HL_1}] + b^i_{HL_1}) \tag{3.1}$$

Onde a saída $o^i_{HL_1}$ é calculada a partir do somatório das entradas i^m_{IL} ponderadas com os pesos $w^{im}_{HL_1}$, adicionadas do valor de viés $b^i_{HL_1}$, cujo resultado é o argumento da função de ativação f_{act}. As grandezas dos pesos permitem proporcionar a relevância das entradas com o inferência final da rede. Os valores de viés (*bias*) são tais que um deslocamento em degrau da saída do neurônio é viável e a função de ativação adiciona outro grau de liberdade na operação.

A relação para o valor de um neurônio possui implicações no treinamento e capacidades do modelo final de uma rede neural. A existência de realimentação, o uso de estruturas de filtros para ponderar uma entrada e valores próximos, bem como a quantidade de camadas escondidas em uso influenciam na nomenclatura atribuída a rede, como por exemplo, as redes neurais convolucionais (CNN - *Convolutional Neural Network*), com suas camadas convolucionais como camadas escondidas. CNNs pertencem ainda ao grupo das redes neurais profundas (DNN - *Deep Neural Network*), definido pelo uso de múltiplas camadas escondidas de natureza distinta.

3.2 Descrição de hardware

Em sua configuração sob demanda, FPGAs são circuitos integrados compostos por blocos lógicos e interconexões entre estes, de modo que uma descrição por linguagem de programação permita que as mais variadas funções sejam atingidas com o mesmo *chip*. Modificações ou correções podem ser integradas, ou a mudança completa de uma lógica digital pode ocorrer em campo. Circuitos podem ser desenhados em paralelo, sem interferência uns com os outros e com uma robustez oriunda da ausência de um sistema operacional.

Como ferreamenta de desenho assistido por computador (CAD - *Computer Aided Design*), HDLs são usadas para o desenho e síntese de sistemas digitais. Por meio de uma descrição em linguagem de alto nível, pacotes de HDLs implementam simuladores e *test benches* que permitem a depuração do código antes de sua síntese e programação em uma FPGA. Dentre as mais populares na industria e na literatura acadêmica, as linguagens Verilog e VHDL implementam abstrações a nível de RTL (*Register-Transfer-Level*) e possuem distinções em tipos de dados e verbosidade. A linguagem SystemVerilog é um conjunto de extensões da Verilog para funcionalides de verificação de um desenho, e que também pode ser utilizada em descrição.

Na Figura 3 está ilustrado um módulo de meio somador descrito em Verilog onde observa-se que em sua declaração e corpo, há, respectivamente, suas conexões de entrada e saída, e relação entre elas.

```
module meio_somador (
    input a,
    input b,
    output s,
    output c);

    assign s = a ^ b;
    assign c = a & b;
endmodule

module mainCode(
```

Figura 3 – Estrutura de módulo em Verilog.

Uma verificação da funcionalidade de um módulo ou sistema, prática recomendada e comum na indústria, poderia ser feita com uma descrição em SystemVerilog de uma fonte de *clock* e uma sequência de entradas em um módulo Verilog, de modo que as mudanças nos sinais dos blocos lógicos são analisadas por formas de onda e as saídas são comparados ao esperado.

3.3 Serviços em nuvem

A computação em nuvem se dá por meio da disponibilidade de infraestrutura de processamento computacional e comunicação via internet, simplificando o fluxo de desenvolvimento de soluções no contexto de IoT.

A descrição de uma plataforma de nuvem é dependente da empresa proprietária, porém a disposição de um conjunto de serviços ou recursos para acesso escalável ao usuário, com encriptação e processamentos de carga computacional remota, são características comuns em tais plataformas.

O Azure é a plataforma de nuvem da Microsoft e é composta por um conjunto de serviços de computação em nuvem denomidaos de recursos. Os serviços disponíveis variam de armazenamento escalável a centrais de mensagens para comunicação bidirecional entre aplicativos de IoT e os dispositivos gerenciados. Como ilustrado na Figura 4, o acesso ao Azure se dá via o Portal Azure, onde a definição e interação com recursos é viável por interface com o usuário ou por meio de APIs em linguagens de programação como Python e C#.

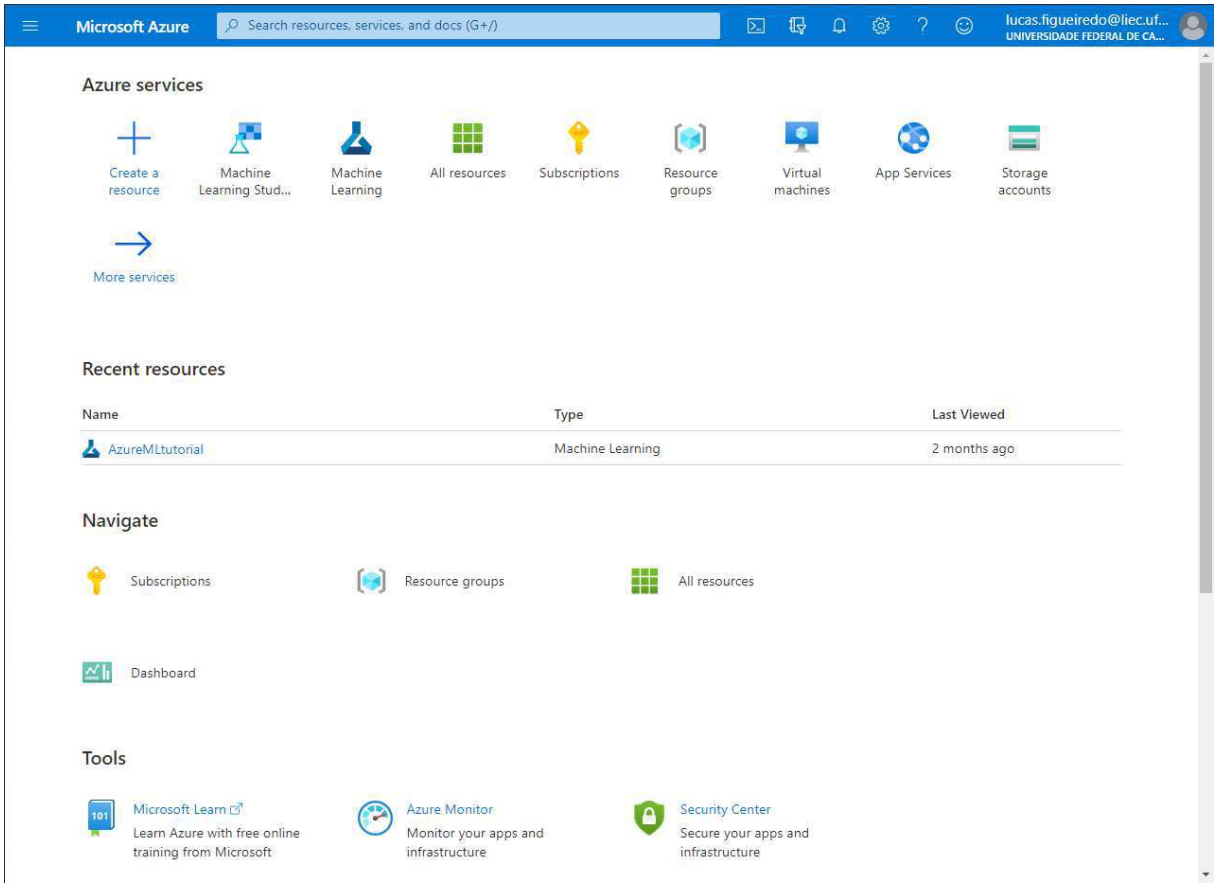

Figura 4 – Página inicial do Portal Azure.

4 Metodologia

4.1 Treinamento de rede neural por serviço de nuvem

O Azure dispõe de três recursos para desenvolvimento com aprendizado de máquina, sendo eles:

- Azure Machine Learning Studio Classic;
- Azure Machine Learning;
- Azure Machine Learning Studio.

O Azure Machine Learning compreende toda a infraestrutura para operação de aprendizado máquina na plataforma. Suporte a treinamentos, com descrição de código em Python ou R, APIs para as ferramentas de desenvolvimento do Microsft Visual Studio, Jupyter Notebooks e PyCharm, como também o suporte a SDKs (*Software Development Kit*) de terceiros e *frameworks* para aprendizado de máquina tais como o PyTorch do Facebook e TensorFlow da Google. A página inicial do recurso está ilustrada na Figura 5.

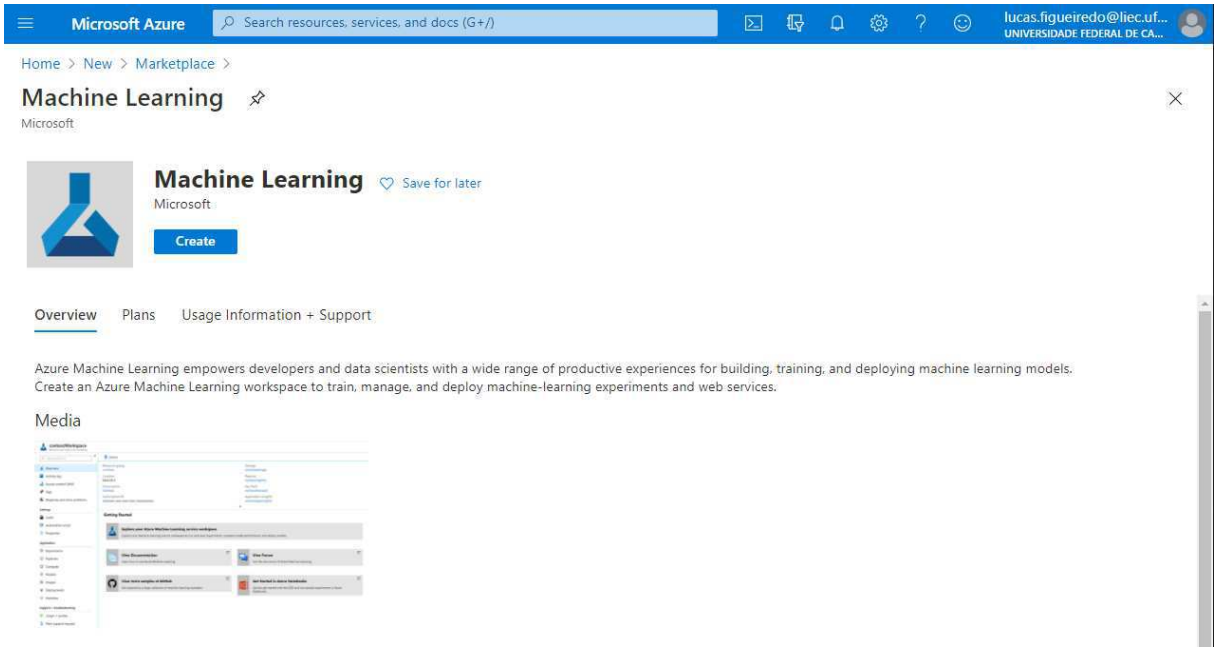

Figura 5 – Recurso Azure Machine Learning.

O Azure Machine Learning Studio possui funcionalidades para uso com pouco código ou sem código fonte para treinamento. Uma ferramenta de design é disposta

para compor um ambiente de treinamento e depuração puxe-e-arraste. O rastreio de experimentos permite uma visualização em tempo real no portal.

Por fim, o Azure Machine Learning Studio Classic foi um serviço *stand alone* que oferecia funcionalidades semelhantes ao Machine Learning Studio atual, porém sem o suporte a *frameworks* de terceiros. Esse recurso não possui compatibilidade com o Azure Machine Learning.

A criação de uma instância do recurso Azure Machine Learning é a primeira etapa para o treinamento em nuvem de uma rede neural. como ilustrado na Figura 6, a definição da rede segue a estrutura de camadas e funções de ativação. Como exemplo para este trabalho, o objetivo de classificar digitos manuscritos é com relativo sucesso atingido por uma camadas convolucionais e funções de ativação lineares.

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figura 6 – Modelo para treinamento no Azure.

A configuração do treinamento está ilustrada no código em Python da Figura 7. Tal operação é iterativa com o modelo, de modo que os pesos e valores de viés tem valores inciais ajustados ao longo de iterações para melhor refletir o objetivo de identificação. A quantidade de parâmetros a serem ajustados se traduz em um crescimento exponencialmente do tempo de treinamento e, portanto, o deslocamento da carga computacional para um processamento dedicado em nuvem tem sua importância notada aqui.

```python
from model import Net

# download CIFAR 10 data
trainset = torchvision.datasets.CIFAR10(
    root="./data",
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor(),
)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=4, shuffle=True, num_workers=2
)

if __name__ == "__main__":

    # define convolutional network
    net = Net()

    # set up pytorch loss /  optimizer
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

    # train the network
    for epoch in range(2):

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # unpack the data
            inputs, labels = data

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999:
                loss = running_loss / 2000
                print(f"epoch={epoch + 1}, batch={i + 1:5}: loss {loss:.2f}")
                running_loss = 0.0

    print("Finished Training")
```

Figura 7 – Configuração do treinamento no Azure.

O resultado do treinamento em nuvem está no conjunto de valores para pesos e demais atributos da rede neural, assim como a acurácia do modelo, notada através de uma função de perda, ou indicador de divergência da predições.

4.2 Treinamento de rede neural por software

O software MATLAB possui um conjunto de ferramentas para *deep learning*, permitindo o desenho, treinamento e aplicação de redes neurais CNN. A definição de uma rede segue o mesmo padrão do Azure Machine Learning em listar as camadas componentes da mesma, desde camadas totalmente conectadas em que pesos e valores de viés são aplicados aos dados de entrada, até a funções de ativação ou computação com filtros.

Ilustrado na Figura 8 está um código exemplo para tal definição.

```
% defines the CNN
layers = [
    imageInputLayer([28 28 1],'Normalization','none')

    fullyConnectedLayer(128)
    reluLayer

    fullyConnectedLayer(128)
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer

    classificationLayer];

% training options
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

% training the network
net = trainNetwork(imdsTrain,layers,options);

% classify validation images and compute accuracy
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation)
net = trainNetwork(imdsTrain,layers,options);
```

Figura 8 – Configuração de uma rede neural e seu treinamento no MATLAB.

Para um conjunto de dados de entrada do banco de dados MNIST ilustrado na Figura 9, a estrutura resultante de uma rede neural no MATLAB é ilustrada na variável *net* da Figura 10. Os digitios a serem identificados estão expostos em imagens com dimensão 28x28 pixels, porém em escala de preto em branco, resultando em 784 pixels a serem analisados e cada um com 8 bits de informação.

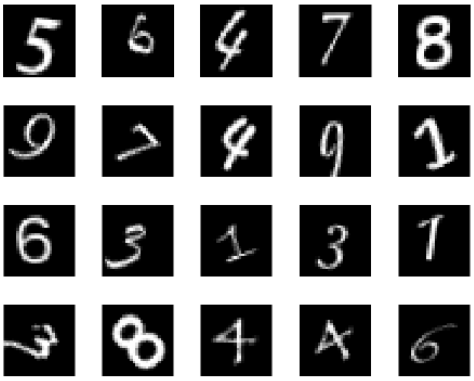

Figura 9 – Conjunto de imagens MNIST de dígitos manuscritos.

net | net.Layers | net | net.Layers | net.Layers(2, 1)

net.Layers | net.Layers(2, 1)

	1	Property	Value
1	1x1 ImageInputLayer	Name	'fc_1'
2	1x1 FullyConnectedLayer	InputSize	784
3	1x1 ReLULayer	OutputSize	128
4	1x1 FullyConnectedLayer	Weights	128x784 single
5	1x1 ReLULayer	Bias	128x1 single
6	1x1 FullyConnectedLayer	WeightLearnRateFactor	1
7	1x1 SoftmaxLayer	WeightL2Factor	1
8	1x1 ClassificationOutputLayer	BiasLearnRateFactor	1
		BiasL2Factor	0

Figura 10 – Estrutura de rede neural no MATLAB.

O MATLAB dispõe ainda da interface ilustrada na Figura 11 para visualização do progresso do treinamento de uma rede.

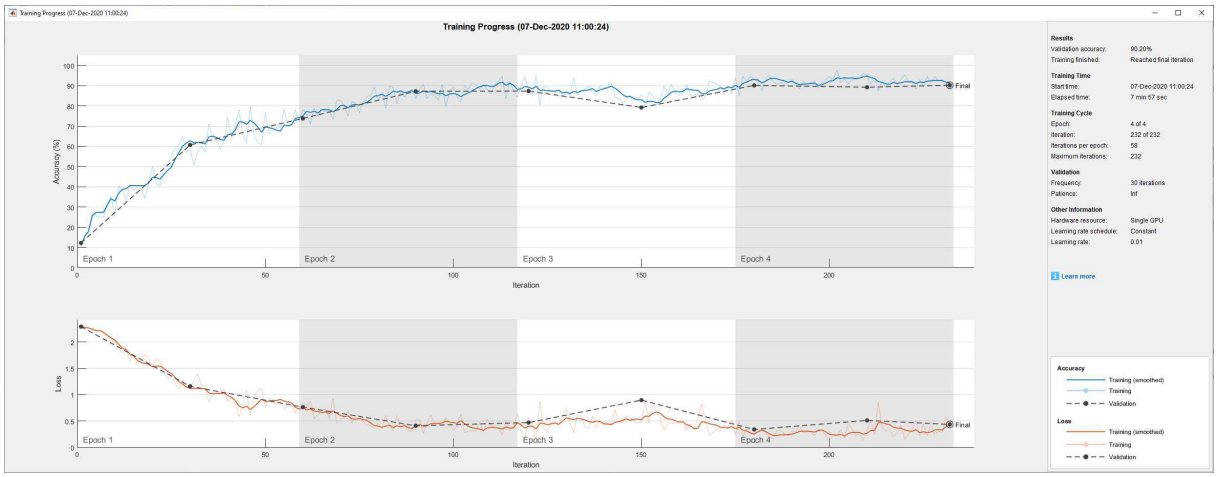

Figura 11 – Progresso do treinamento de rede neural no MATLAB.

4.3 Hardware e ambiente de desenvolvimento para FPGA

A placa de desenvolvimento escolhida para estudo foi a Basys 3 da fabricante Digilent, ilustrada na Figura 12 e que conta com um chip FPGA Artix-7 XC7A35T da Xilinx. É anunciada como uma placa para de entrada para desenvolvimento, com funcionalidades disponíveis generalistas e inferiores as demais da família Artix-7.

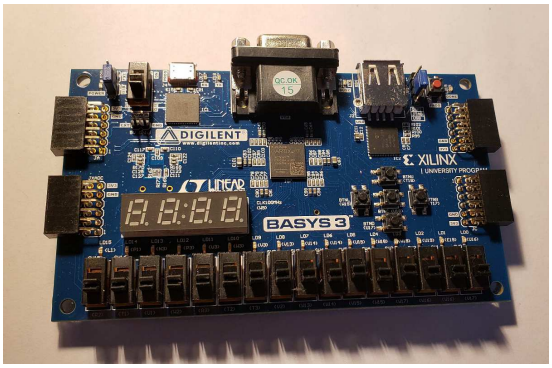

Figura 12 – Placa de desenvolvimento Basys 3 com a FPGA Artix-7.

A disponibilidade recursos de memória para armazenar um modelo treinado de rede neural é um fator fundamental na escolha de um modelo de FPGA. Na Figura 13 está um comparativo dos recursos dos chips da Família Artix-7 da Xilinx, onde é notável o recurso de BRAM (*Block* RAM), implementada utilizando circuitos dedicados na FPGA e resultando em blocos discretos de memória que podem ser inferidos por HDL.

Artix-7 FPGAs

		Transceiver Optimization at the Lowest Cost and Highest DSP Bandwidth (1.0V, 0.95V, 0.9V)							
	Part Number	XC7A12T	XC7A15T	XC7A25T	XC7A35T	XC7A50T	XC7A75T	XC7A100T	XC7A200T
Logic Resources	Logic Cells	12,800	16,640	23,360	33,280	52,160	75,520	101,440	215,360
	Slices	2,000	2,600	3,650	5,200	8,150	11,800	15,850	33,650
	CLB Flip-Flops	16,000	20,800	29,200	41,600	65,200	94,400	126,800	269,200
Memory Resources	Maximum Distributed RAM (Kb)	171	200	313	400	600	892	1,188	2,888
	Block RAM/FIFO w/ ECC (36 Kb each)	20	25	45	50	75	105	135	365
	Total Block RAM (Kb)	720	900	1,620	1,800	2,700	3,780	4,860	13,140
Clock Resources	CMTs (1 MMCM + 1 PLL)	3	5	3	5	5	6	6	10
I/O Resources	Maximum Single-Ended I/O	150	250	150	250	250	300	300	500
	Maximum Differential I/O Pairs	72	120	72	120	120	144	144	240
Embedded Hard IP Resources	DSP Slices	40	45	80	90	120	180	240	740
	PCIe® Gen2[1]	1	1	1	1	1	1	1	1
	Analog Mixed Signal (AMS) / XADC	1	1	1	1	1	1	1	1
	Configuration AES / HMAC Blocks	1	1	1	1	1	1	1	1
	GTP Transceivers (6.6 Gb/s Max Rate)[2]	2	4	4	4	4	8	8	16
Speed Grades	Commercial Temp (C)	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2
	Extended Temp (E)	-2L, -3	-2L, -3	-2L, -3	-2L, -3	-2L, -3	-2L, -3	-2L, -3	-2L, -3
	Industrial Temp (I)	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L

Figura 13 – Comparação dos recursos disponíveis na família Artix-7 de FPGAs da Xilinx (Dados da fabricante, disponíveis em <https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf>

Para problemas de classificação de imagens por ANN, como o caso de identificação

de dígitos manuscritos, os recursos de memória disponíveis crescem proporcionalmente a dimensão da imagem analisada e a eficácia desejada para a rede neural. Para imagens em cores, por exemplo, a problemática de identificação já é multiplicada por três, em comparação. Imagens em maior detalhe implicam em redes neurais mais sofisticadas para distinguir o objeto desejado, e portanto, cresce a escala de pesos, valores de viés e funções matemáticas computacionalmente complexas necessárias.

A fabricante Xilinx dispõe das seguintes soluções de software para a programação em suas FPGAs:

- Vitis Unified Software Platform: ferramenta que combina todos os aspectos de desenvolvimento de software para produtos Xilinx em um único ambiente.
  - Vitis IDE: compreende as funcionalides de desenvolvimento para aplicações de software embarcado do Vitis, trabalhando em cima de desenhos de hardware importados da Vivado Design Suite;
  - Vitis HLS: ferramenta para acelerar a descrição de hardware por meio da síntese em RTL de funcionalidades para FPGAs descritas a partir de código em C/C++.
- Vivado Design Suite: ambiente de desenvolvimento para o código, simulação e carregamento de descrições de hardware com alvo em FPGAs da Xilinx.

Com atenção na descrição de hardware, a Vivado Design Suite segue um fluxo de desenvolvimento de síntese, implementação e geração da *bitstream*, observável na aba lateral da interface com o usuário da IDE ilustrada na Figura 14. Para módulos em Verilog (.v), os código-fonte do projeto são seguidos de arquivos de *constraints*, ou restrições, em linguagem TCL (.xdc), listando as funcionalidades da placa de desenvolvimento em uso, como interruptores, botões, LEDs e entradas seriais.

Durante a síntese, arquivos de HDL são traduzidos em em uma descrição a nível de transistores baseando-se em restrições de tempo e entrada/saída. O resultado da síntese é então repassado para o estágio de implementação, que dentre suas múltiplas etapas, realiza uma otimização do desenho para caber na FPGA alvo e faz o roteamento dos sinais ao longo do tecido da FPGA, resultando na geração do arquivo de *bitstrema*, apto a ser carregado no chip alvo.

Além do suporte a descrição de hardware por código em HDL, a Vivado possui um desenhador de blocos, ou integrador de IP (*Internal Peripheral*), onde as restrições de uma placa de desenvolvimento podem ser conectadas a blocos de alto nível em uma visualização em formato de diagrama, gerando-se um bloco final que pode ser utilizado em uma aplicação. A funcionalidade de simulação, ou verificação de uma descrição de

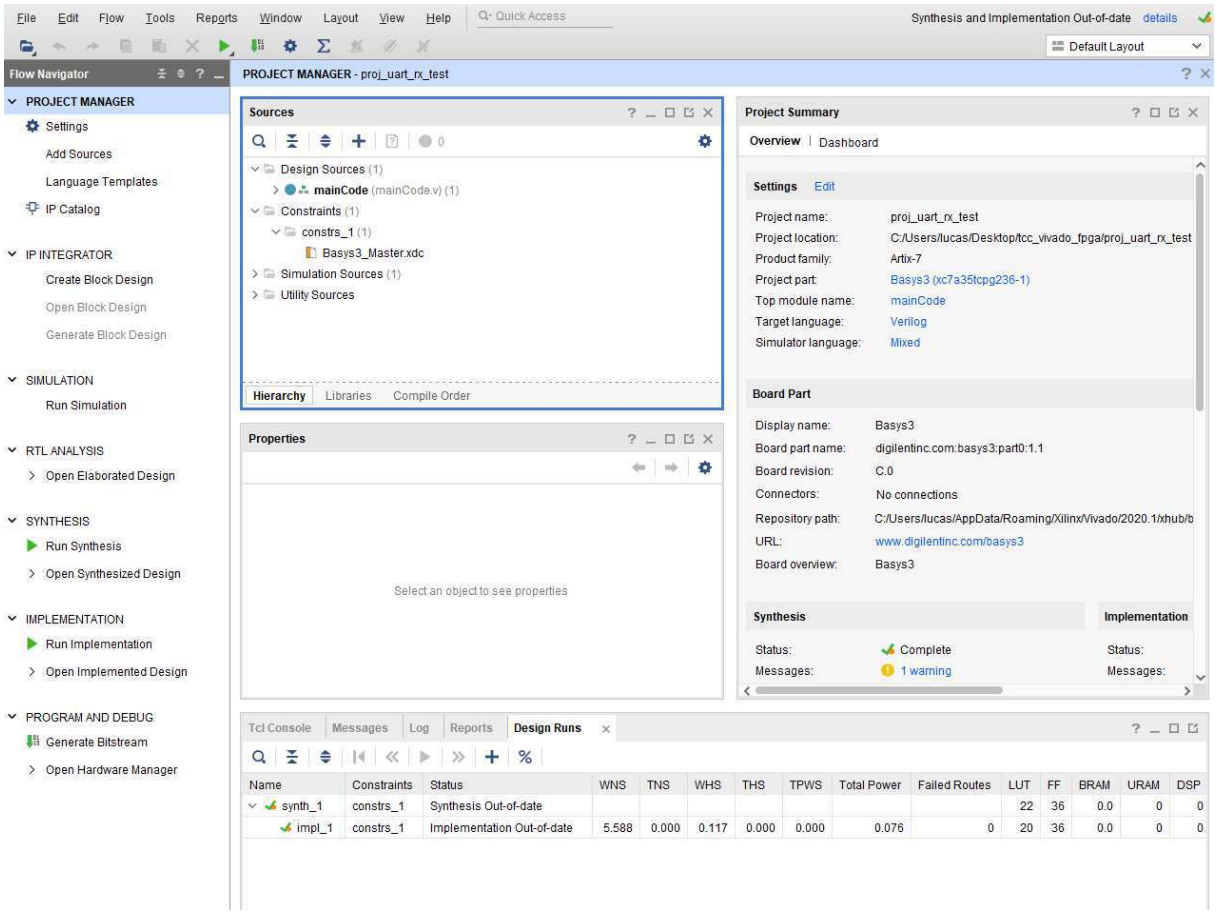

Figura 14 – Ambiente de desenvolvimento Vivado Design Suite.

hardware, está disponível no fluxo de desenvolvimento do Vivado, onde a partir de uma fonte de simulação, como por exemplo, um arquivo em SystemVerilog (.sv) estimulando entradas e impondo restrições de tempo, a mudança de sinais e saídas podem ser estudadas a nível de software em capturas de formas de onda.

5 Resultados e discussões

5.1 Funcionalidades em HDL para implementação de rede neural

Um objeto a ser analisado por uma rede neural e a divisão dos seus atributos componentes para estudo, se traduz no caso de analise de imagens nos pixels componentes desta figura. A representação, por exemplo, em escala de uma única cor, de um pixel pode ser traduzida para 8 bits, com 256 tonalidades, porém para fins de precisão em uma rede neural, e em razão de uniformizar os dados em uso, como pesos dos neurônios, o recurso de números por ponto fixo são vantajosos.

Diferindo da computação por ponto flutuante, definido pelo padrão técnico IEEE 754, uma representação em ponto fixo tem, pelo nome, um número fixo de bits para cada parte do número, a inteira e a fracionada. O formato de representação Qn.m para números binários descreve a possibilidade ilustrada na Tabela 1, em que para um número de 8 bits, os primeiros 4 representam o sinal e a parte inteira, enquanto que os últimos 4 representam a parte fracionada, com a precisão de $\frac{1}{16} = 0,0625$. Operações com números negativos são suportadas por meio do complemento de dois de um dos operandos, e o resultado final de uma operação deve ser atribuído a um registrador que o comporte. Para os dígitos manuscritos em imagens de 28x28 pixels, uma representação por com 16 bits em ponto fixo é viável, levando em consideração o treinamento da rede neural para tal entrada e o processamento prévio da imagem para ser alimentada a FPGA.

Bit	7	6	5	4	3	2	1	0
Valor decimal	Sinal	4	2	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$
Potência de 2	-	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

Tabela 1 – Representação de número binário com 8 bits em formato Q4.4.

Os pesos, valores de viés e saídas de funções de ativação, são dados estáticos para a implementação de um modelo de rede neural em FPGA. A inicialização de memória em Verilog pode ser feita pelas funções *$readmemh* e *$readmemb*, que recebem como argumento, respectivamente, os valores de interesse expressos em hexadecimal ou binário, e presentes em um arquivo de texto (.mem) separados por espaços. Ainda, a matriz em Verilog que irá conter os dados deverá estar presente no argumento, como ilustrado no código da Figura 15

```
module readWeights();
    reg [127:0] weights [15:0];
    initial begin
        $readmemh("neuralNet_weights.mem", weights);
    end
endmodule
```

Figura 15 – Código em Verilog para inicializar uma matriz de memória.

A aquisição pela FPGA de uma imagem para processamento se dá por meio das funcionalidades de entrada e saída de sua placa de avaliação. Para a Basys 3 da Digilent, a ponte USB-UART dispõe de um acesso simples para carregar dados de entrada da rede neural na memória. Em sua comunicação por dois fios, o protocolo UART requer a configuração prévia dos parâmetros de transmissão, como *baudrate* e presença de bit de paridade.

5.2 Abordagem para implementação de modelo em hardware

Como resumo das problemáticas enfrentadas para implementação em hardware de um modelo treinado de rede neural, ilustrada na Figura 16 está a abordagem proposta, em virtude dos recursos disponíveis e previamente detalhados.

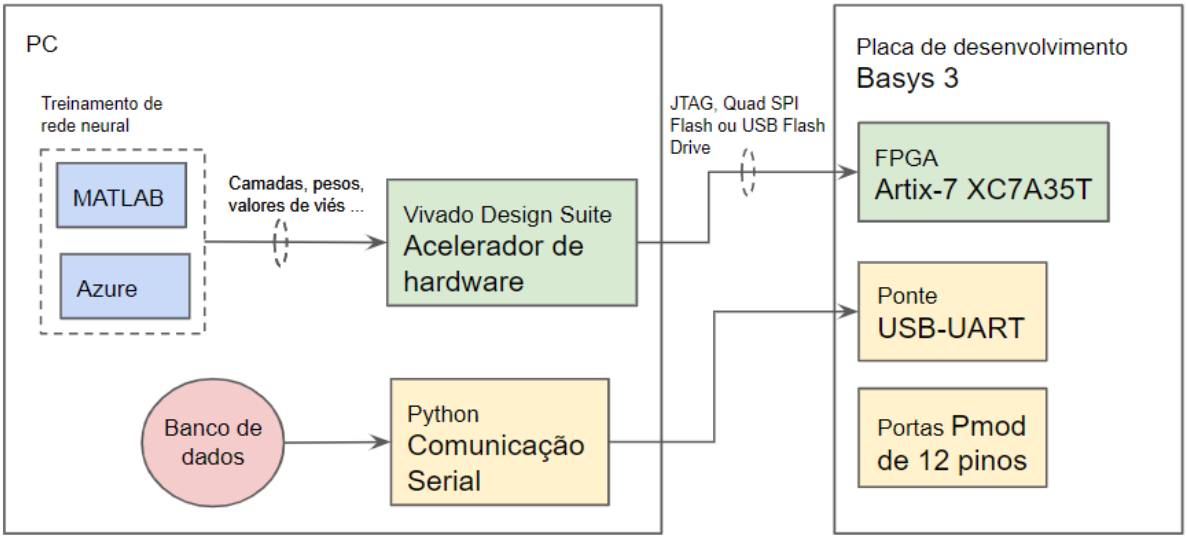

Figura 16 – Proposta de abordagem para implementação de modelo de rede neural.

A alimentação da rede neural com uma imagem para processamento, em sua forma mais simples, pode ocorrer via ponte USB-UART a partir de um código em Python

transmitindo a imagem quebrada em bytes, porém por ventura uma câmera pode ser conectada via serial para tornar independente a placa de avaliação.

A classificação realizada pela rede neural norteia em grande parte as escolhas subsequentes de arquitetura da rede e hardware para comportar o modelo final, onde mais uma vez, a escala dos recursos de memória é um fator limitante. As operações de multiplicação de pesos e adição de valores de viés, bem como da passagem por uma função de ativação, são corriqueiras para uma descrição em hardware, porém a dimensão final dessas operações, adicionam complexidade a solução. Ilustrada na Tabela 2 estão os graus de precisão para inferência de diferentes arquiteturas de redes neurais artificiais, juntamente com as quantidades de valores de pesos e de viés para os modelos das redes, e o consumo de memória resultante destes valores considerando uma representação de 16 bits em ponto fixo. As arquiteturas avaliadas variaram entre si no número de camadas escondidas totalmente conectadas e no número de neurônios por camada. O treinamento foi realizado para quatro épocas, conjunto de dados de 7500 imagens e com o algoritmo SGDM (*Stochastic Gradient Descent with Momentum*) em uma taxa de aprendizado de 0,01.

Camadas	**Neurônios**	**Precisão (%)**	**Pesos**	**Valores de viés**	**Memória (kB)**
1	64	19,08	50176	64	100,48
1	96	16,16	75264	96	150,72
1	128	21,36	100352	128	200,96
2	64	83,20	54272	64	108,672
2	96	87,76	84480	96	169,152
2	128	88,52	116736	128	233,728
3	64	85,60	58368	64	116,864
3	96	90,16	93696	96	187,584
3	128	93,32	133120	128	266,496

Tabela 2 – Comparação do desemepenho e uso de recursos por arquiteturas de redes neurais treinados com o *Deep Learning toolbox* do MATLAB.

É notável analisando-se a Tabela 2 que a precisão final do modelo treinado cresce com o número de neurônios por camada, porém o número de camadas presentes na arquitetura implica na obtenção de uma precisão superior a 80% quando duas camadas são utilizadas, em contraste a 20% quando uma camada é utilizada, ou seja, uma precisão quase que equivalente a um palpite às cegas com 10% de chance de acerto em virtude das 10 classificações possíveis. Quanto ao uso de memória, a influência da quantidade de pesos necessária é absoluta, uma vez que camadas totalmente conectadas levam em consideração todas as saídas da camada anterior em seus cálculos, porém na arquitetura de três camadas e 128 neurônios, um ganho de apenas 3,16% em precisão foi em custa de um aumento de 42,06% no uso de recursos de memória em comparação com outra arquitetura de 3

camadas, porém esta com 96 neurônios. O investimento no treinamento de arquiteturas relativamente próximas de redes neurais pode, portanto, gerar um conjunto de opções igualmente boas, frente as suas restrições, para o usuário selecionar uma solução ótima.

6 Considerações finais

A partir do desenvolvimento deste trabalho, foi possível listar um fluxo de execução para a implementação em hardware de um modelo de rede neural. As múltiplas competências necessárias para concluir tal projeto compreendem desde a programação em alto nível com *frameworks* para treinamento das redes neurais e tratamento dos dados antes da alimentação a um modelo pronto, como também a descrição em baixo nível de módulos em HDL para realizar o carregamento de memória do modelo, e por fim as operações de inferência que definem uma rede neural.

Frente a uma abrangência de conceitos em inteligência artificial e nas complexidades de descrição por HDLs, em comparação com linguagens de programação de alto nível, a listagem de softwares disponíveis e escolhas de projeto neste trabalho objetivam servir como exemplo de abordagem para tal solução de aceleração de uma rede neural via hardware. Como trabalhos futuros, está, por fim, a obtenção de resultados em FPGA do modelo treinado, uma vez que este trabalho não fez tal implementação.

Referências Bibliográficas

SHARMA, A., et al. **Communication and networking technologies for UAVs: A survey**. Journal of Network and Computer Applications (2020).

ZHANG, X., et al. **Machine Learning on FPGAs to Face the IoT Revolution**. ICCAD'17, November 2017, Irvine, CA USA. 2017.

LIN, N., et al. **When Deep Learning Meets the Edge: Auto-Masking Deep Neural Networks for Efficient Machine Learning on Edge Devices**. 2019 IEEE 37th International Conference on Computer Design (ICCD).

BUSCHJÄGER, S., et al. **Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data**. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS–I: REGULAR PAPERS, VOL. 65, NO. 1, JANUARY 2018.

PECKOL, J. K. **Embedded Systems: A Contemporary Design Tool, 2nd Edition**. Wiley, 2019.

DUBEY, R. **Introduction to Embedded System Design Using Field Programmable Gate Arrays**. Springer, 2009.

CAVANAGH, J. **Digital Design Verilog HDL and Fundamentals**. CRC Press, 2008.