Avaliação do Processo para Embarcar uma Rede Neural Baseada em YOLO Utilizando um Acelerador de Hardware Dedicado

Isabel F. Costa*, Elias T. Silva Jr†, Antônio Wendell O. Rodrigues‡, Leandro V. M. Angeloni§ and Edmilson J. Dias§
*Computer Science Department of Federal Institute of Education, Science and Technology of Ceara - IFCE
Email: isabel@lit.ifce.edu.br
†Computer Science Department of Federal Institute of Education, Science and Technology of Ceara - IFCE
Email: elias@ifce.edu.br
‡Computer Science Department of Federal Institute of Education, Science and Technology of Ceara - IFCE
Email: wendell@ifce.edu.br
§Energy Company of Minas Gerais - CEMIG

***Abstract*—Object Detection is a challenging task in computer vision, but Deep Neural Networks (DNN) have made great progress in this area. This work presents the process and the results obtained in the attempts to embed a YOLO V3 model in a Neural Compute Engine, the Movidius Stick. Experiments were carried out with a Tensorflow model that is converted to Movidius (using OpenVINO) including an evaluation of the Movidius stick connected to a Raspberry Pi3. The application uses aerial images of power distribution towers captured by a drone. Although there are some fully operational networks for Neural Compute Engines, there are some difficulties in porting new networks to the platform, with gains in performance, but with losses in accuracy.**

***Index Terms*—Sistemas Embarcados, Deep-Learning, Acelerador de Hardware.**

I. Introdução

Os sistemas de geração e de distribuição de energia demandam por inspeções constantes de seus ativos, a fim de manter a sua disponibilidade. Inspeções terrestres ou através de helicópteros ainda são as formas mais comuns para monitorar essas estruturas e seus equipamentos.

Inspeções terrestres apresentam algumas restrições, como limitações de deslocamento impostas por terrenos de difícil acesso. Inspeções com helicópteros contornam essa barreira e conseguem percorrer uma grande área. Entretanto, são financeiramente custosas e dependem completamente da experiência e visualização adequada das estruturas [1]. O que normalmente ocorre é que na inspeção são necessários um piloto e um especialista treinado na estrutura sendo inspecionada. Adicionalmente, a qualidade da inspeção depende fundamentalmente da experiência do especialista. Um agravante é que não se consegue treinar novos especialistas na mesma velocidade com que eles se aposentam dessa atividade.

Nos últimos anos, a tecnologia de inspeção por Veículos Aéreos Não Tripulados (UAV) tem se difundido na detecção de possíveis falhas nos componentes elétricos. Com os UAVs, é possível abordar estruturas de forma mais seguras e capturar problemas em potencial com mais facilidade [2]. Sistemas computacionais embarcados no UAV com modelos treinados na detecção de objetos podem auxiliar nas inspeções ao capturar imagens de estruturas e equipamentos de forma automática, auxiliar na navegação do piloto e até identificar possíveis patologias ou não-conformidades.

O problema de detecção de objetos em imagens tem sido atacado com sucesso com o uso de redes neurais profundas (DNN). O principal desafio é embarcar uma DNN em um UAV utilizando um sistema computacional de baixo consumo de energia para não comprometer o tempo de vôo da missão. As DNN são algoritmos de alta complexidade e que demandam pesadas estruturas de dados, o que leva a demandarem sistemas computacionais de alto desempenho, que costumam operar em potências elevadas [3].

Este trabalho apresenta um estudo para verificar a aplicabilidade de um Stick de Computação Neural na plataforma Raspberry Pi 3 em um problema de detecção de torres e cruzetas de madeira. O objetivo é avaliar as métricas de tempo e precisão para atender as prováveis restrições de tempo real nas arquiteturas de rede YOLO e Tiny-YOLO V3, duas DNN voltadas à detecção de objetos em imagens.

II. Background

A. Detecção de objetos

O problema de detecção de objetos pode ser definido da seguinte maneira: dada uma imagem, o algoritmo deve determinar a existência de instâncias de objetos de categorias predefinidas e, se houver, retornar a localização espacial e a extensão de cada instância.

Um dos principais desafios é desenvolver um algoritmo que atinja dois objetivos concorrentes: precisão e alta eficiência. A detecção de alta qualidade deve localizar e reconhecer com precisão e alta distinção os objetos nas imagens. Além disso

deve ser generalista o bastante para que as instâncias de objetos da mesma categoria, sujeitas a variações na aparência intra-classe, possam ser localizadas e reconhecidas (alta robustez). A alta eficiência exige que as implementações apresentem consumo mínimo de memória e tempo de computação [4].

B. Arquitetura YOLO

YOLO *(You Only Look Once)* foi desenvolvida para detecção de objetos. Ela foi projetada para encontrar e reconhecer padrões em uma imagem ou em um vídeo. A arquitetura YOLO transforma o problema de detecção em um problema de regressão, gerando probabilidades de coordenadas e caixas delimitadoras para cada classe. Essa característica aumentou consideravelmente a sua velocidade de detecção em comparação com outras redes [5]. A rede divide cada imagem do conjunto de treinamento em grades S × S. Se o centro do objeto de destino cair em uma grade, a grade será responsável por detectar o alvo. Cada grade fornece as caixas delimitadoras B e seu respectivo índice de confiança e probabilidades condicionais da classe C [6]. Portanto, a saída da rede é um conjunto de caixas delimitadoras com sua respectiva confiança. De modo geral, predições com confianças abaixo de 50% são descartadas. Existem diversas versões implementadas da YOLO. A versão utilizada neste trabalho é a YOLO-V3 [5], um aprimoramento das redes YOLO [7] e YOLO-V2 [8].

A versão do YOLO V3 com 24 camadas convolucionais é chamada Tiny-YOLO V3. Essa rede foi proposta por Joseph Redmon [5]. Por possuir menos camadas, a velocidade de execução aumenta significativamente (aproximadamente 442% mais rápida que as variantes YOLO anteriores), mas a precisão na detecção é reduzida. A Tiny-YOLO V3 possui o mesmo princípio de funcionamento da versão mais profunda da rede YOLO.

C. Stick de Computação Neural Movidius

O Stick de computação neural Intel® Movidius™ (NCS - Neural Compute Stick) é um dispositivo USB de aprendizado profundo que pode ser usado para desenvolver aplicações de Inteligência Computação. Possui compatibilidade com o sistema operacional Ubuntu 18.04 instalado em arquitetura x86-64 ou Debian executando em uma Raspberry Pi Model B. Uma DNN treinada baseada em Caffe ou TensorFlow™ é compilada em uma rede neural embarcada otimizada para rodar na VPU (Vision Processing Unit) dentro do NCS. Esse processo pode ser realizado através do Neural Compute SDK ou do OpenVINO™, softwares disponibilizados pela Intel para a conversão. Essas ferramentas oferecem suporte as linguagens C++ e Python (2.7/3.5) [9].

III. Trabalhos Relacionados

Diferentes técnicas podem ser utilizadas para detectar objetos em imagens aéreas. Em [10] são utilizados algoritmos de segmentação de imagens para encontrar postes de energia elétrica. Outros trabalhos [4] [10] também exploram as características morfológicas dos postes de energia para realizar a detecção. Todos esses trabalhos focam em um único tipo de objeto, com formato particular que dificulta a generalização. Para problemas com múltiplos objetos, algoritmos detectores de objetos são mais indicados.

Em [11] os autores utilizam a YOLO V3 para detectar isoladores de uma rede elétrica. O algoritmo treinado detecta cerca de 95% dos isoladores do conjunto de teste (100 imagens). Os autores de [12] modificaram a arquitetura da YOLO V3 para detectar objetos de imagens borradas pela movimentação do drone no ar e obtêm cerca de 96% de acurácia. No entanto, não foi realizado estudos para avaliar a portabilidade dessa rede para uma plataforma embarcada.

Há pouca literatura sobre a performance do Stick Movidius em redes treinadas com datasets customizados. O trabalho apresentado em [13] descreve o processo para embarcar um classificador de emoções no NCS1 utilizando o NCSDK. A arquitetura de rede possui apenas 6 camadas convolucionais e usa imagens de entrada de tamanho 48x48 pixels. Com essa configuração, eles obtêm cerca de 10ms de tempo de inferência. No trabalho de [14] ele avalia as arquiteturas Tiny-YOLO V3 e MobilNetSSD no NCS1 e NCSDK. A média do tempo de inferência foi de 291,3ms na Tiny-YOLO V3, utilizando a Raspberry Pi 3 como plataforma.

Com o OpenVINO e utilizando a arquitetura GoogleNet os autores de [15] apresentam o processo para embarcar um classificador de gestos. A arquitetura utilizada possui 22 camadas convolucionais e recebe como entrada imagens com 224x224 pixels. Os experimentos também foram realizados na versão 1 do NCS. O tempo de inferência medido foi torno de 100ms utilizando a Raspberry Pi 3 como plataforma alvo. O trabalho de [16] apresenta um classificador de gêneros de livros baseados na capa. Os experimentos foram realizados em 3 arquiteturas de redes distintas, OpenVINO e NCS2, mas não apresenta métricas na plataforma embarcada.

Este trabalho apresenta os resultados obtidos nas versões 1 e 2 no NCS, com OpenVINO, nas arquiteturas de rede YOLO e Tiny-YOLO V3. A solução embarcada proposta utiliza uma Raspberry Pi 3 modelo B+ com o sistema operacional Raspbian instalado, uma câmera e o Neural Compute Stick da Intel (Movidius Stick). Esta solução será conectada via interface serial RS232 ao DJI Matrice 210, um UAV de 716mm x 220mm x236mm e 3.80kg. O Matrice 210 possui um SDK que permite o envio de comandos via serial para capturar fotos e direcionar o drone. O principal objetivo da aplicação é facilitar a inspeção em redes de energia elétrica, com capturas automáticas de fotos e sugestões de direção para o piloto.

IV. Treinamento e Avaliação do Modelo

Esta seção apresenta especificidades sobre o dataset e como o modelo foi treinado e avaliado.

A. Dataset de Postes e Cruzetas de Madeira

O dataset utilizado para os experimentos é composto por imagens aéreas (obtidas via câmera de drones) de postes de madeiras. Essas imagens foram capturas em inspeções realizadas no estado de Minas Gerais, Brasil e possuem resolução

de 4000x3000 pixels. Dentro do conjunto original, foram selecionadas imagens que postes e cruzetas que satisfaziam dois requisitos:

- Plano aberto, com boa visualização de toda a estrutura (identificar torre e cruzeta);
- Plano fechado, próximo a cruzeta (identificar cruzeta) para avaliação de defeitos em etapas posteriores não inclusas neste trabalho.

As imagens foram marcadas com dois possíveis objetos: ”torre” e ”cruzetas” de madeira, utilizando a ferramenta LabelImg [17]. Ao todo, 1117 imagens foram marcadas, onde 80% foram separadas para treino e 20% para teste. A figura 1 mostra como cada objeto é delimitado no processo de marcação. A caixa em vermelho demarca o que se chama de ”cruzetas” e em azul, o objeto ”torre”. No conjunto de testes, há 222 imagens, com 430 caixas delimitadoras. A tabela I apresenta a distribuição de cada objeto no conjunto de teste. Há mais cruzetas que torres porque há imagens em plano fechado, onde não foi possível delimitar a estrutura ”torre”.

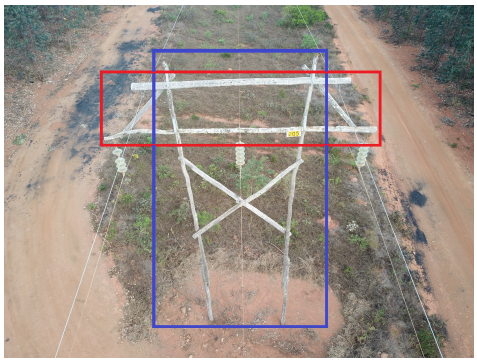

Fig. 1. Exemplo de marcação de imagens

TABLE I
DISTRIBUIÇÃO DOS OBJETOS NO CONJUNTO DE TESTE

	Quantidade de Caixas
Cruzetas	221
Torre	209
Total	430

B. Treinamento e Métricas de Avaliação

Os experimentos dessa seção foram feitos em um computador de propósito geral, com processador Intel Core i7-6500 2,5GHz e 8GB de memória RAM, rodando sistema operacional Ubuntu 18.04 LTS. Nesse momento o objetivo é avaliar as arquiteturas de rede escolhidas e suas taxas de acerto com o dataset da aplicação. Esses resultados serão usados posteriormente para comparação com a versão embarcada.

As arquiteturas YOLO V3 e Tiny-YOLO V3 foram implementadas no Framework Tensorflow [18]. A próxima etapa do processo é validar o modelo Tensorflow treinado usando um preditor independente. Para isso foi utilizada a biblioteca OpenCV 4 [19], que executa modelos treinados de diversos Frameworks de aprendizagem profunda. A biblioteca possui o pacote DNN (Deep Neural Network) e pode realizar inferência dos modelos mais populares de redes neurais de aprendizado profundo. O modelo é avaliado com base em: tempo de inferência, tempo de carregamento do modelo, FPS (quadros por segundo) e IoU (Intersection over Union), conforme detalhado a seguir.

- Tempo de inferência é o tempo para realizar o cálculo de todas as camadas da rede, desde a chegada da entrada (uma imagem) até a resposta esperada (a identificação do objeto). Permite avaliar a rapidez da plataforma onde a inferência é realizada, independentemente de outros processos de pré-processamento (tratamentos na imagem) e pós-processamento.
- Tempo de Execução é o tempo de processamento de um quadro inteiro e inclui todo o pré e pós-processamento necessário. Exemplos de pré-processamento são: redimensionar a imagem original capturada para o tamanho da entrada de rede (416x416) e aplicação de filtros na imagem. Como pós-processamento se faz remoção de possíveis caixas duplicadas e caixas com baixa confiança, por exemplo.
- O FPS é calculado baseado no Tempo de Execução.
- IoU é um padrão para definir a precisão do objeto na detecção. A IoU avalia o desempenho do modelo calculando a taxa de sobreposição entre a caixa delimitadora esperada e a caixa delimitadora entregue pelo algoritmo. A equação 1 mostra a fórmula, onde $S_{overlap}$ é a área de interseção da caixa delimitadora esperada e a caixa delimitadora real. $S_{overlap}$ é a área de união das duas caixas delimitadoras.

$$IoU = \frac{S_{overlap}}{S_{union}} \tag{1}$$

Os modelos treinados de YOLOV3 e Tiny-YOLOV3 foram avaliados a partir das métricas descritas. Os resultados obtidos e apresentados na tabela II foram gerados a partir da média de 30 execuções de todo o conjunto de teste. Não houve variação entre a quantidade de caixas preditas entre as rodadas. É notável a perda na capacidade de acerto da Tiny-YOLO em relação à versão completa. Entretanto, esse prejuízo vem acompanhado de um interessante benefício no tempo de inferência.

TABLE II
MÉTRICAS OBTIDAS NO TENSORFLOW NO CONJUNTO DE TESTE

	YOLOV3	**Tiny-YOLOV3**
IoU médio (%)	82,96	78,56
IoU, Desvio Padrão (%)	13,19	11,96
Cruzetas Preditas	212	195
Torres Preditas	203	192
Total de Predições (%)	96,51	90
Tempo de Inferência (ms) médio	1051,44	103,13
Tempo de Infer. (ms), Dv. padrão	16,58	15,85
Tempo de Execução (ms) médio	1051,44	137,02
Tempo de Exec. (ms), Dv. padrão	29,78	23,6
FPS	0,95	7,29

V. Migração do preditor para o NCS

A Intel disponibiliza duas ferramentas para migração e otimização dos modelos para o NCS: *Neural Compute SDK* e *OpenVINO*. O SDK Movidius, chamado de NCSDK (Neural Compute SDK), inclui um conjunto de ferramentas de software para compilar, obter métricas e verificar (validar) DNNs. O NCSDK possui a API de computação neural Intel Movidius (Intel Movidius NCAPI) para desenvolver aplicativos em C/C ++ ou Python. Após a obtenção do modelo treinado, ele deve ser verificado e compilado com o NCSDK. O arquivo gerado deve ser utilizado na aplicação de destino com o NCAPI. O SDK faz conversões diretas a partir dos frameworks Caffe e Tensorflow, gerando um modelo compilado para o Stick Movidius. No entanto, o modelo gerado é aceito apenas no NCS1 (versão 1 do Stick). Por conta dessa restrição, não são apresentados resultados de experimentos utilizando o NCSDK.

O kit de ferramentas OpenVINO [9] permite a integração de DNN desenvolvidas a partir de frameworks de referência, como Tensorflow, Caffe, e outros. O otimizador de modelo OpenVINO importa e prepara modelos para hardwares Intel. O principal objetivo é que o desenvolvedor tenha flexibilidade na implantação da aplicação, pois permite alterar a plataforma de destino durante o desenvolvimento de acordo com a necessidade da aplicação, evitando mudanças complexas no desenvolvimento do projeto. Para usar os Sticks Movidius é necessário utilizar o OpenVINO para converter os modelos (*Model Optimization*) para *Intermediary Representation* (IR) e utilizar o motor de inferência (*Inference Engine*) para carregar o modelo no dispositivo de destino. Um dos objetivos do OpenVINO é permitir que o mesmo código seja executado em uma ampla variedade de plataformas, o que facilita a obtenção de métricas da CPU e NCS 1 e 2.

A. Avaliação preliminar em Workstation

O Stick é conectado a uma porta USB da plataforma host. Em um primeiro momento, os resultados foram obtidos em um computador de propósito geral com o sistema operacional Ubuntu 18.04 LTS, Processador Intel Core i7-6500 de 2,5 GHz e 8 GB de RAM, com a inferência em diferentes destinos (CPU ou NCS - Myriad). Nos experimentos foi usado o OpenVINO versão 2020.1.

As tabelas III e IV exibem os resultados obtidos em cada plataforma, sempre usando o PC como host. Os tempos foram obtidos a partir de 30 execuções sobre o conjunto de teste. Todos os resultados foram obtidos com o modelo otimizado pelo OpenVINO (IR). Aqui já se pode fazer algumas comparações com os resultados da tabela II. Nota-se uma boa proximidade nos resultados que se relacionam aos acertos do preditor. Tanto na YOLO quanto na Tiny-YOLO as diferenças nos resultados (IoU e predições) estão abaixo de 2 %, com exceção do total de predições do NCS1 para Tiny-YOLO (tabela IV), que chega a 34 %.

As figuras 2 e 3 exibem os tempos obtidos em cada etapa realizada no computador. A sigla TF é o modelo Tensorflow e IR é o modelo otimizado pelo OpenVINO para *Intermediate Representation*. O *Parsing Time* é o tempo para transformar as saídas do NCS em saídas do padrão YOLO.

TABLE III
Métricas obtidas com a representação intermediária (IR) do modelo da **YOLOV3**

	CPU	**NCS 1**	**NCS 2**
IoU médio %	81,11	81,22	82,12
Iou, Desvio Padrão (%)	13,74	12,89	11,19
Cruzetas Preditas	207	208	209
Torres Preditas	200	199	197
Total de Predições (%)	94,65	94,65	94,44
Tempo de Inferência (ms) médio	424,39	1333,79	348,13
Tempo de Infer. (ms), Dv. padrão	16,01	12,56	11,25
Tempo de Execução (ms) médio	465,05	1388,33	416,45
Tempo de Exec. (ms), Dv. padrão	12,20	11,25	13,45
FPS	2,15	0,72	2,40

TABLE IV
Métricas obtidas com a representação intermediária (IR) do modelo da **Tiny-YOLOV3**

	CPU	**NCS 1**	**NCS 2**
IoU (%)	78,85	78,78	78,82
IoU, Desvio Padrão(%)	13,85	11,16	11,18
Cruzetas Preditas	189	188	189
Torres Preditas	193	193	193
Total de Predições (%)	88,83	86,6	88,83
Tempo de Inferência (ms) médio	40,91	122,81	38,46
Tempo de Infer. (ms), Dv. Padrão	17,36	11,15	12,12
Tempo de Execução (ms) médio	52,38	155,33	69,52
Tempo de Exec. (ms), Dv. Padrão	30,12	27,78	29,65
FPS	19,09	6,43	14,38

Ao analisar os resultados de tempo de execução na figura 2, verifica-se que o NCS2 realizou a inferência com mais rapidez. O mesmo ocorre na figura 3. Nota-se ainda os tempos de parsing e de transferência da amostra para o NCS como fatores de alguma importância, especialmente para a tiny-YOLO. O pior resultado do tempo de inferência está no NCS1, evidenciando a evolução da versão 1 para a versão 2. Todos os experimentos com os NCS foram realizados utilizando portas USB 2.0. Pode-se perceber a influência da transferência de dados também se reflete no tempo de carregamento do modelo. Para carga do modelo YOLO no NCS 1 e 2 são necessários, em média, cerca de 20 segundos. O modelo possui 246MB de tamanho. Para o modelo Tiny-YOLO, que possui 34,7MB, a média é de 4 segundos. O modelo é carregado apenas uma vez, impactando somente no tempo de implantação ou de *setup* do sistema.

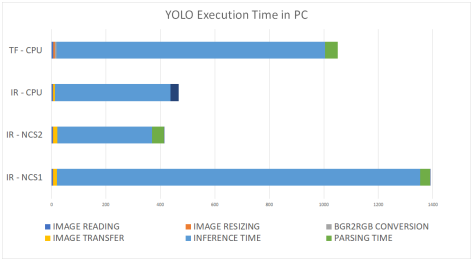

Fig. 2. Tempo de execução da YOLO

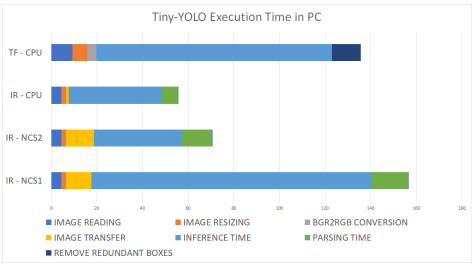

Fig. 3. Tempo de execução da Tiny-YOLO

B. Plataforma Embarcada

O processo para embarcar o modelo consiste em copiar o modelo convertido e o código desenvolvido para uma Raspberry Pi. A inferência com o IR, modelo gerado pelo OpenVINO, não pode ser realizada na CPU da Raspberry, pois é um hardware não Intel. Assim, os valores de CPU apresentados nas tabelas V-B e VI são da execução do modelo Tensorflow na plataforma Raspberry utilizando a biblioteca OpenCV.

Nas figuras 4 e 5, a sigla TF é a execução com modelo Tensorflow e IR a execução com o modelo otimizado. Na figura 4 os tempos de inferência foram omitidos, pois a execução do modelo Tensorflow na CPU da Raspberry não permitia a visualização adequada dos dados. Os tempos de inferência são exibidos na tabela III. As taxas de IoU e quantidade de predições são próximas aos valores obtidos nas execuções no computador de propósito geral apresentados nas tabelas II para YOLOV3 e Tiny-YOLOV3, tabela III para YOLOV3 e tabela IV para Tiny-YOLOV3.

O objetivo desse estudo é colocar o preditor embarcado na aeronave. Assim, usar o stick NCS conectado a um PC não é uma opção. Comparando os resultados dos sticks conectados à USB de um PC com os obtidos quando conectados à Raspberry, nota-se que o tempo de inferência é praticamente o mesmo. Entretanto, uma diferença importante surge quando se observa o tempo total de execução. A principal causa encontra-se no tempo de transferência da imagem para o NCS realizar a predição. O *Parsing Time*, que é o tempo para transformar as saídas do NCS em saídas do padrão YOLO também são maiores quando comparados com os resultados obtidos no PC, já que essa computação é feita pela Raspberry.

TABLE V
MÉTRICAS OBTIDAS NA RASPBERRY COM O MODELO **YOLOV3**

	CPU	**NCS 1**	**NCS 2**
IoU (%)	83,04	78,56	82,20
IoU, Desvio Padrão (%)	13,15	12,04	12,63
Cruzetas Preditas	212	208	209
Torres Preditas	203	199	197
Total de Predições (%)	96,51	94,41	94,41
Tempo de Inferência (ms) médio	23339,36	1334,18	399,44
Tempo de Infer. (ms), Dv. Padrão	101,45	10,84	9,37
Tempo de Execução (ms) médio	23939,19	1740,65	785,20
Tempo de Exec. (ms), Dv. Padrão	125,96	29,74	25,35
FPS	0,04	0,57	1,27

TABLE VI
MÉTRICAS OBTIDAS NA RASPBERRY COM O MODELO **TINY-YOLOV3**

	CPU	**NCS 1**	**NCS 2**
IoU (%)	78,29	78,80	78,96
IoU (%), Desvio Padrão	11,87	12,25	12,12
Cruzetas Preditas	195	189	189
Torres Preditas	192	193	193
Total de Predições (%)	90	88,83	88,83
Tempo de Inferência (ms) médio	1513,83	122,87	43,61
Tempo de Infer. (ms), Dv. Padrão	11,68	11,97	10,19
Tempo de Execução (ms) médio	1715,95	260,1	189,79
Tempo de Exec. (ms), Dv. Padrão	125,96	25,45	24,16
FPS	0,58	3,84	5,26

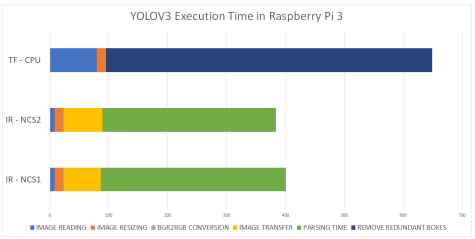

Fig. 4. Tempo de execução da YOLO conectada à Raspberry

VI. AVALIAÇÃO DE RESULTADOS

A. Caixas Preditas

A partir da análise das porcentagens de caixas preditas no modelo Tensorflow e do modelo no NCS, percebe-se mínima diferença. Isso indica que, embora haja uma redução na precisão dos pesos no processo de conversão para IR, a alteração não tem um grande impacto. A diferença na precisão é mais significativa quando se comparam as versões YOLOV3 e Tiny-YOLOv3; cerca de 6% a menos caixas.

B. IoU

A partir das caixas encontradas é calculada a IoU, que mede a precisão da caixa predita. IoU maiores que 50% são consideradas boas previsões. O modelo IR YOLO V3 obteve média de 80,38%. A média de IoU da Tiny-YOLO V3 com o NCS é de cerca de 78,88%, isso indica que, quando o objeto é encontrado o modelo o delimita de forma aproximada ao que foi marcado. A figura 6 mostra um exemplo de amostra classificada corretamente e a comparação entre as caixas marcadas e previstas. Percebe-se que, embora não coincidam completamente, a caixa delimitadora segue proporções

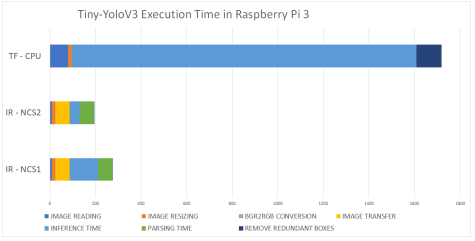

Fig. 5. Tempo de execução da Tiny-YOLO conectada à Raspberry

muito próximas às esperadas. As caixas laranjas são as caixas preditas pela rede.

Fig. 6. Marcação de objeto feita pela YOLO

C. Tempo de Inferência

O tempo de inferência da YOLOV3 na CPU é 15 vezes maior que a versão Tiny. Já no NCS2 a Tiny-YOLOV3 é 9 vezes mais rápida. O NCS2 obteve os menores tempos obtidos; ele processa quase 60 vezes mais rápido a YOLOV3 ao comparar com o processamento na CPU da Raspberry. No caso da Tiny-YOLOV3, o NCS2 é 34 vezes mais rápido do que a CPU. O trabalho de [15] obtém cerca de 100ms para a rede GoogLeNet de 22 camadas, com uma imagem de entrada de 224x224 no computador de uso geral e no NCS1. A configuração de experimento mais próximo ao realizado por este trabalho é com a versão Tiny no NCS1, onde o tempo médio obtido foi de 122ms. Como a Tiny-YOLOV3 é mais profunda que a GoogLeNet e a entrada usada foi de 416x416, esse tempo extra é justificado. No trabalho de [20] os autore obtém tempo médio de 291,3ms por imagem, mas o autor não especifica se esse tempo é somente de inferência.

VII. Conclusões

Neste artigo se apresenta o fluxo de trabalho necessário para embarcar um modelo de detecção de objeto baseado nas arquiteturas YOLO e Tiny-YOLO V3 no Neural Compute Stick (NCS) da Intel. A aplicação de destino tem como objetivo detectar postes e cruzetas em um ambiente rural usando um veículo aéreo não tripulado. Nesse cenário, é necessário um sistema de computador com baixo consumo de energia, razão pela qual se usa um Raspberry Pi com um acelerador neural.

A corretude na previsão acima de 90% é muito promissora, considerando um conjunto de dados de imagem relativamente pequeno. Pode-se observar o aumento de precisão ao utilizar a versão YOLO, a um custo mais elevado no tempo de computação. Esse é um *overread* que deve ser ponderado já que o Tiny-YOLO consegue taxas de predição acima de 80% e necessita de metade (ou menos) do tempo para inferir.

Os tempos de computação de uma DNN (Deep Neural Network) permaneceram próximos no PC e na Raspberry ao usar os Sticks, variando apenas como resultado da transferência inevitável das imagens de teste através da porta USB do Stick. Esses resultados podem ser melhorados com o uso da versão 4 da Raspberry, que utiliza USB 3.0.

Nos experimentos, o NCS2 obteve os melhores tempos no Raspberry, sendo 34 vezes mais rápido que a inferência na CPU. O tempo gasto em inferência pode ser ainda menor se as imagens entregues à rede forem de menor resolução, devido ao menor tempo de transferência para o Stick. Trabalhos futuros poderiam avaliar a influência do tamanho da imagem e analisar os custos do impacto na precisão da DNN.

References

[1] R. Jenssen, D. Roverso *et al.*, "Automatic autonomous vision-based power line inspection: A review of current status and the potential role of deep learning," *International Journal of Electrical Power & Energy Systems*, vol. 99, pp. 107–120, 2018.

[2] B. Chen and X. Miao, "Distribution line pole detection and counting based on yolo using uav inspection line video," *Journal of Electrical Engineering & Technology*, vol. 15, no. 1, pp. 441–448, 2020.

[3] B. Moons, D. Bankman, and M. Verhelst, *Embedded Deep Learning*. Springer, 2019.

[4] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[5] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[6] Y. Tian, G. Yang, Z. Wang, H. Wang, E. Li, and Z. Liang, "Apple detection during different growth stages in orchards using the improved yolo-v3 model," *Computers and electronics in agriculture*, vol. 157, pp. 417–426, 2019.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[8] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525.

[9] "Intel® distribution of openvino™ toolkit — intel® software," https://software.intel.com/en-us/openvino-toolkit, (Accessed on 12/20/2019).

[10] A. Varghese, J. Gubbi, H. Sharma, and P. Balamuralidhar, "Power infrastructure monitoring and damage detection using drone captured images," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 1681–1687.

[11] F. Guo, K. Hao, M. Xia, L. Zhao, L. Wang, and Q. Liu, "Detection of insulator defects based on yolo v3," in *International Conference on Artificial Intelligence for Communications and Networks*. Springer, 2019, pp. 291–299.

[12] H. Chen, Z. He, B. Shi, and T. Zhong, "Research on recognition method of electrical components based on yolo v3," *IEEE Access*, vol. 7, pp. 157 818–157 829, 2019.

[13] Y. Xing, P. Kirkland, G. Di Caterina, J. Soraghan, and G. Matich, "Real-time embedded intelligence system: emotion recognition on raspberry pi with intel ncs," in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 801–808.

[14] A. Pester and M. Schrittesser, "Object detection with raspberry pi3 and movidius neural network stick," in *2019 5th Experiment International Conference (exp. at'19)*. IEEE, 2019, pp. 326–330.

[15] M. F. Ab Hamid and F. H. K. Zaman, "Hand gesture recognition using movidius neural compute stick," in *2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*. IEEE, 2019, pp. 510–514.

[16] N. Tiwari and K. Mondal, "Ncs based ultra low power optimized machine learning techniques for image classification," in *2019 IEEE Region 10 Symposium (TENSYMP)*, 2019, pp. 750–753.

[17] L. Tzutalin, "Git code (2015)."

[18] "Tensorflow," https://www.tensorflow.org/, (Accessed on 08/07/2020).

[19] "Opencv 4.0," https://opencv.org/opencv-4-0/, (Accessed on 08/07/2020).

[20] A. Pester and M. Schrittesser, "Object detection with raspberry pi3 and movidius neural network stick," in *2019 5th Experiment International Conference (exp.at'19)*, 2019, pp. 326–330.