Um estudo comparativo entre as arquiteturas de redes neurais profundas *AlexNet* e YOLO aplicadas ao problema de visão computacional em Ambientes Agrícolas

Danielly O. A. de Oliveira[1], **Josué Pereira de Castro**[1]

[1]Ciência da Computação - Universidade Estadual do Oeste do Paraná (UNIOESTE)
Caixa Postal 701 – 85.819-110 – Cascavel – PR – Brazil

danielly_omori@hotmail.com, josue.castro@unioeste.br

Abstract. *The verification and control of the uniformity of irrigation water distribution is one of the most important tasks to ensure the best use of water resources. This type of activity requires the robot to be able to move in rural environments, recognizing obstacles in its path. In this article, a comparative study was developed between two convolutional neural network architectures, AlexNet and YOLO, with the objective of being embedded in a Raspberry PI processor mounted on a small robot to be used to assist in the automated adjustment of adjustable emitters for irrigation located. The networks were trained using databases containing images related to the agricultural environment and then, data regarding the performance of both architectures were collected and a comparison was made between them, using several criteria, as the execution time, the consumption of main and secondary memory, and classification accuracy. The architecture that showed the best performance was YOLO, which was embedded in the robot, undergoing the necessary adaptations so that it was able to use the images transmitted in real time by the built-in camera existing in the robot.*

Resumo. *A verificação e o controle da uniformidade de distribuição de água para irrigação é uma das tarefas de maior importância para garantir o melhor uso dos recursos hídricos. Este tipo de atividade exige que o robô seja capaz de deslocar-se em ambientes rurais, reconhecendo obstáculos em seu caminho. Neste artigo foi desenvolvido um estudo comparativo entre duas arquiteturas de redes neurais convolucionais, AlexNet e YOLO, com o objetivo de serem embarcadas em um processador Raspberry PI montado em um robô de pequeno porte a ser usado para auxiliar no ajuste automatizado de emissores ajustáveis para irrigação localizada. As redes foram treinadas, utilizando bases de dados contendo imagens relacionadas ao meio agrícola e a seguir, foram coletados dados*

referentes ao desempenho de ambas arquiteturas e realizada uma comparação entre elas, usando vários critérios, sendo os principais o tempo de execução, o consumo de memória principal e secundária, e a precisão da classificação. A arquitetura que apresentou melhor desempenho foi a YOLO, a qual foi embarcada no robô, passando pelas devidas adaptações para que a mesma fosse capaz de utilizar as imagens transmitidas em tempo real pela câmera embutida existente no robô.

1. Introdução

Desde o início da década de 1990 é possível observar um grande desenvolvimento na área de Tecnologia da Informação (TI) e como sua utilização em processos de produção poderiam aumentar a produtividade industrial [Santos et al. 2018]. Uma das áreas que mais tem recebido incentivo para entrar no mundo tecnológico é a área agrícola, que por ser uma das bases da economia brasileira, torna essencial que sejam utilizadas tecnologias para aumentar o rendimento das lavouras [Massruhá e Leite 2017, Jardim 2019].

Alguns projetos já vem sendo desenvolvidos com o intuito de estimular este mercado a investir no desenvolvimento de soluções voltadas para o agronegócio. O programa Agro 4.0 [ABDI 2019], criado pelos ministérios da Agricultura, Economia e da Ciência, Tecnologia e Inovações, juntamente com a Agência Brasileira de Desenvolvimento Industrial (ABDI) tem o objetivo de promover o aumento de eficiência e produtividade agrícolas, empregando métodos computacionais para processar grandes volumes de dados e construir sistemas de tomada de decisão.

A partir do ano 2000 tem sido dada muita importância ao uso consciente de recursos hídricos [Paz et al. 2000], com destaque para os crescentes períodos de seca, que afetam a produção agrícola. Esta situação torna imprescindível otimizar o uso de recursos hídricos, de forma a reduzir o consumo de água na produção agrícola. Segundo a UNESCO [UNESCO 2020], a agricultura é responsável por 70% da exploração global de água doce, com possibilidade de chegar a 90% em países em rápido crescimento, sendo a irrigação é responsável por 40% da produção mundial de alimentos. Disto conclui-se que há necessidade premente de desenvolver tecnologias que otimizem e reduzam o consumo de água para irrigação.

Visão computacional e robótica podem ser ferramentas úteis para diversas finalidades dentro da área de irrigação. Há vários trabalhos que utilizam visão computacional em ambiente agrícolas, por exemplo: ASTRAND [Åstrand e Baerveldt 2002] descreve um robô móvel agrícola capaz de realizar o controle de ervas daninhas em uma plantação de beterrabas; JODAS [Jodas et al. 2013]. apresenta um sistema capaz de fazer um robô móvel deslocar-se por plantações de forma autônoma, com o foco na realização de tarefas como detecção de pragas e controle da aplicação de agrotóxicos. Contudo, aplicações deste tipo voltadas para irrigação localizada ainda são poucas, por exemplo: [Lam et al. 2006] apresenta um sistema de controle de feedback de sensoriamento remoto baseado no solo para monitorar o avanço da água em sulcos e controlar automaticamente a descarga de água na entrada do sulco. [Berenstein et al. 2018] apresenta uma nova abordagem para ajustar emissores usando um manipulador robótico móvel autônomo.

Esse contexto, considerando ainda o fato da região oeste do Paraná ter vocação agrícola, motivaram o desenvolvimento deste trabalho. O principal objetivo é desenvol-

ver um sistema de visão em um robô embarcado de pequeno porte, com uma câmera embarcada, que seja capaz de detectar objetos em ambientes agrícola, localizando-os e identificando-os corretamente, para auxiliar na tarefa de ajuste automático de gotejadores. Para determinar quais seriam estes objetos, foi feita uma análise visual de uma área rural, identificando quais objetos encontravam-se dispostos neste local. Procurou-se fazer uma classificação genérica dos objetos, evitando-se detalhamentos desnecessários que não contribuíssem para a aplicação desejada, de modo que optou-se por uma classe genérica "planta"do que classes específicas como "soja"ou "feijão", por exemplo, já que este tipo de classificação não influencia na tarefa de ajuste dos gotejadores.

2. Materiais e Métodos

Esta seção tem como objetivo apresentar brevemente os recursos e a metodologia utilizados no desenvolvimento deste trabalho.

2.1. Raspberry PI

Este projeto utiliza uma placa Raspberry Pi modelo 3B integrada a um robô Wi-Fi com chassi TH *Robot Tank* produzido por *Xiao R Geek Techonology* e possui uma câmera USB (*Universal Serial Bus* - Porta Serial Universal) em um suporte com dois graus de liberdade, e um braço manipulador com 4 graus de liberdade[1]. O modelo do robô pode ser visto na Figura 1.

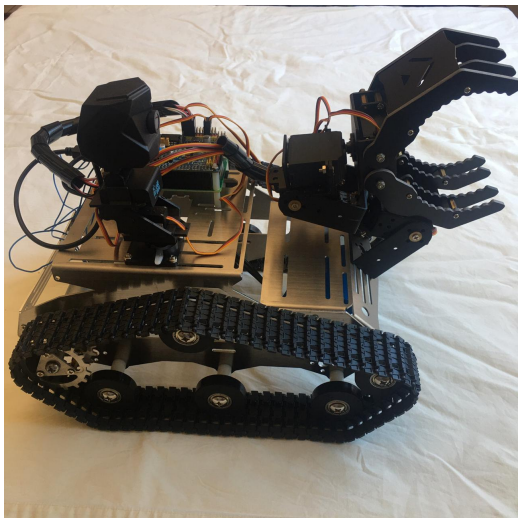

Figura 1. Modelo de robô utilizado

2.2. Linguagem Python

Python é uma linguagem de programação de alto nível, interpretada, orientada a objetos, de tipagem dinâmica e forte, de código livre, publicada no ano de 1991, por Guido van Rossum [itMídia 2019]. Neste trabalho foi utiliza esta linguagem em com suas bibliotecas OpenCV[2] e *Tensor Flow*[3].

[1]quantidade de juntas do manipulador que restringem a sua liberdade de movimento
[2]https://opencv.org
[3]https://www.tensorflow.org/learn?hl=pt-br

2.3. Estruturas de Redes Neurais Convolucionais Utilizadas

2.3.1. *AlexNet*

O trabalho desenvolvido por Krizhevsky [Krizhevsky et al. 2017] tinha como objetivo classificar a base de dados ImageNet [Yang et al. 2020]. A rede desenvolvida possuía oito camadas, com cinco delas sendo camadas convolucionais e três camadas totalmente conectadas. A saída da última camada totalmente conectada era utilizada como entrada para uma camada de ativação, que fazia uso da função *softmax* para realizar a classificação entre as 1000 classes existentes. Todo esse processo foi realizado utilizando duas GPUs (*Graphics Processing Unit* - Unidade de Processamento Gráfico), o que facilitou o processamento da rede.

2.3.2. YOLO (*You Only Look Once*)

YOLO (*You Only Look Once*) é um sistema de reconhecimento em tempo real, desenvolvido com o objetivo de detectar vários objetos em um único frame, além de gerar caixas delimitadoras (*Bounding Box Predictions*) que indicam em que região da imagem os objetos estão localizados e a qual classe eles pertencem [Orac 2020].

Seu funcionamento se difere das de diversas arquiteturas de redes neurais convolucionais pois, ele não somente realiza o treinamento da rede, como também a divisão da imagem em áreas de interesse, permitindo que o algoritmo também seja capaz de delimitar a área em que os objetos serão encontrados [Techzizou 2020].

2.4. Ambientes Utilizados

O treinamento dos protótipos criados para esse projeto foram executados no ambiente Google Colaboratory[4]. As configurações utilizadas no ambiente para a execução dos modelos foram as padrões gratuitas do Colab, consistindo em:

- Processador: Intel(R) Xeon(R), 2.30GHz, ;
- RAM: 12GB;
- Espaço Disponíve: 25GB;
- GPU: Nvidia K80, 12GB.

A máquina utilizada para a realização dos testes foi a placa Raspberry Pi, modelo 3B, embutida no robô. Suas configurações consistiam em:

- Sistema Operacional: Raspberry PI OS 32-bits. *Released*: 2021-03-04;
- Processador: Quad Core 1.2GHz Broadcom BCM2837 64bit CPU;
- RAM: 1GB.

2.5. Bases de Dados

Com objetivo de abranger objetos existentes no ambiente rural, criou-se uma base de dados contendo seis classes: animais, flores, gotejador, veículos, pessoas e plantas. Elas foram escolhidas após um estudo sobre quais objetos eram mais encontrados em ambiente agrícola local. Para construí-la, foram reunidas imagens de bases de dados disponíveis no

[4]https://colab.research.google.com

site Kaggle[5] e em alguns casos buscou-se imagens em outras fontes. Maiores informações referentes à composição das bases podem ser vistas à seguir:

- **Animais**: Foram utilizadas duas bases de imagens para o desenvolvimento dessa classe. A base *256-Objects*, utilizando as classes *dog*, *horse* e *duck* e a base *Animal-10*, utilizando as classes *cat*, *chiken*, *cow*, *dog*, *horse* e *sheep*;
- **Flores**: Para a classe flor, foi utilizada a Base *Hackathon Blossom (Flower Classification)*;
- **Gotejador**: Para a construção dessa classe, foram utilizadas imagens disponíveis no Google, utilizando a palavra chave "gotejador".
- **Veículos**: Foi utilizada a base *Vehicle Data Set*, selecionando apenas as classes *Car*, *Truck*, *Van*, *Taxi*, *Ambulance* e *Bus*. Além disso, se utilizou a classe *bulldozer* da base *256-Objects* para que fosse possível abranger veículos de maior porte;
- **Pessoas**: A base *1 Million Fake Faces* contendo imagens de rostos de pessoas falsas. Também foram utilizadas imagens disponíveis no Google (utilizando as palavras chaves "fotos paparazzi famosos") para poder completar a base de dados, pois não foi possível encontrar base de dados que representassem o corpo inteiro de pessoas;
- **Plantas**: Inicialmente utilizou-se as bases *V2 Plant Seedlings Dataset* e *Agriculture crop images* para fazerem parte da classe planta. Notou-se, porém, que utilizar essas bases apenas não seriam suficientes, pois elas não eram capazes de representar tão bem a vegetação brasileira. Por este motivo, foi realizada uma visita ao Núcleo Experimental de Engenharia Agrícola (NEEA) da UNIOESTE, para realizar a coleta de imagens de outros tipos de plantações como soja e milho.

Foram utilizadas 170 instâncias para as classes planta, veículo, animal, flor e pessoa e 150 para a classe gotejador, que por ser mais característica, não necessitava de tantas imagens. A base final, composta por um total de mil imagens, foi dividida em 70% para treinamento, 15% para validação e 15% para testes.

2.6. Experimentos Realizados

Apresentaremos aqui os protótipos desenvolvidos e os comparativos realizados com as redes escolhidas. Para cada uma das redes foi implementado um protótipo em linguagem Python e realizado o treinamento da rede neural, que ao final de cada iteração, passava por uma etapa de validação para verificar se os valores sendo obtidos durante a etapa anterior estavam se aproximando do valor esperado ou não. Ao fim do treinamento, foram colhidos os resultados referentes ao desempenho da rede e a partir de uma análise desses valores, foi escolhido o protótipo mais adequado a ser embarcado no robô.

2.6.1. Protótipo I

O primeiro protótipo criado utilizou a arquitetura *AlexNet*, utilizando o modelo desenvolvido por ALAKE [Alake 2020], redimensionando as imagens para o tamanho igual à 200×200 pixels. Para o treinamento, foram utilizadas 1.000 épocas, com 7 iterações cada, totalizando 7.000 iterações. Esse número de épocas foi escolhido após testes preliminares que verificaram que caso fossem utilizados valores maiores, a rede entraria em processo

[5]https://www.kaggle.com

de *overfitting*. Além disso, optou-se por usar uma etapa de validação após cada iteração. Os parâmetros utilizados foram taxa de aprendizado de $0,00261$[6] e *Batch Size* [7] igual à 100. Além disso, também utilizou-se o parâmetro *Sparse Categorical Crossentropy* para realizar o cálculo de perda do modelo e otimizador Adam. O código completo pode ser encontrado no Google Colaboratory[8].

2.6.2. Protótipo II

Ao analisar o primeiro modelo gerado, notou-se que ele não seria adequado para ser embarcado no robô, principamente por demandar muita memória. Portanto, para o segundo protótipo, buscou-se uma alternativa voltada para a utilização em ambiente embarcado e *mobile*, como é o caso do robô. A solução encontrada para contornar esse problema a foi o YOLO-V4 *Tiny*, versão reduzida do sistema YOLO, capaz de criar um modelo de rede neural mais simples, com número de parâmetros reduzidos, visando utilização de redes neurais em ambientes que não possuem tanto poder computacional disponível.

Conforme solicitado nas documentações do YOLO-V4 *Tiny*, foram utilizadas configurações específicas para a realização do treinamento, tais como imagens redimensionadas para o tamanho 416×416, taxa de aprendizado igual à $0,00261$, e número máximo de *batches* igual ao número de classes multiplicados por 2.000, o que totalizou em 12.000 lotes. A organização das camadas seguiu o modelo desenvolvido por [Jacob e Samrat 2020] e está disponível no Google Colaboratory[9].

3. Resultados e Discussões

3.1. Análise de Desempenho

Durante a realização dos experimentos, foram analisados os seguintes para verificação do desempenho da rede:

- Consumo máximo de memória que o treinamento da rede exigiu;
- Tempo gasto para a realização do treinamento;
- Precisão obtida durante a etapa de testes.

Além disso, ao fim do treinamento, foram exportados os pesos gerados pela rede para um arquivo que pudesse ser utilizado em um ambiente externo ao que foi utilizado nesta etapa. Os resultados podem ser vistos na Tabela 1.

3.2. Protótipo I

O protótipo I obteve uma precisão média de 72% durante a etapa de testes, atingindo um uso máximo de memória RAM de 2,56 GB e levando um total de 36 minutos e 22 segundos para finalizar seu treinamento.

[6]Parâmetro que controla o tamanho do salto na descida do gradiente e controla a velocidade e precisão do treinamento [Academy 2019]

[7]Refere-se ao número de exemplos de treinamento utilizados em cada iteração [Academy 2019]

[8]https://colab.research.google.com/drive/1vqYAJEXrIi3dtMOgSA4mqeCQwO6lpyBU?usp=sharing

[9]https://colab.research.google.com/drive/1JIHClGomsXoLw86aWzK5T05u6uoh0cxJ?usp=sharing

Apesar de ter bons resultados referentes ao desempenho, a *AlexNet* se mostrou uma rede "pesada", gerando um total de 171.557.638 parâmetros quando compilada utilizando o *TensorFlow*. Isto influenciou diretamente no tamanho do arquivo contendo os pesos finais da rede, que ao ser exportado, totalizou 670 MB.

3.3. Protótipo II

O protótipo II apresentou uma precisão média de 77,40% durante a etapa de testes, utilizando no máximo 1,28 GB de memória RAM do ambiente de treinamento, porém, levando um total de 3 horas, 56 minutos e 41 segundos para a realização dessa etapa.

Apesar da demora na etapa de treinamento, o modelo YOLO apresentou uma rede menor que a do protótipo anterior, gerando um total de 5.891.874 parâmetros em seu modelo quando convertido para o formato *Tensorflow*. O arquivo contendo os pesos finais da rede totalizou o tamanho de 23 MB.

3.4. Comparações

Como pode ser observado na Tabela 1, o protótipo I apresentou uma única vantagem em relação ao concorrente, referente ao tempo de treinamento.

Tabela 1. Comparação entre os protótipos

COMPARAÇÃO PROTÓTIPOS		
Parâmetros de Análise	**Protótipo I**	**Protótipo II**
Número de Iterações	7.000	12.000
Tempo de Execução	36m 22s	3h 56m 41s
Gasto Máximo de Memória	2,6 GB	1,8 GB
Precisão	72,00%	77,40%
Número de Parâmetros Gerados	171.557.638	5.891.874
Tamanho do Arquivo de Pesos Gerados	670MB	23MB
Possui algum marcador para a área que os objetos estão?	Não	Sim

Já o protótipo II, além de apresentar melhor precisão, obteve destaque em dois pontos. O primeiro no que diz respeito ao tamanho da rede gerada e o segundo referente ao gasto de memória RAM máximo durante o treinamento. Como visto anteriormente, a placa Raspberry Pi possui limitações referente ao poder de processamento, tendo disponível apenas 1 GB de memória RAM. Por conta disso, o modelo a ser embarcado nele deveria ser algo mais simples, que não demandasse tanto poder computacional.

Outro ponto positivo do protótipo II em relação ao seu componente é que o sistema YOLO já possui integrado em seu algoritmo de treinamento o processo de criação de caixas delimitadoras, que indicam a área em que o objeto classificado se encontra na imagem de entrada. Para que a *AlexNet* seja capaz de realizar essa tarefa, seria necessário utilizar algoritmos próprios para o tratamento e segmentação de imagem e para o desenho dessas caixas delimitadoras. Esse foi o caso do trabalho realizado por [Wang et al. 2019], que utilizou a *AlexNet* para realizar o treinamento da rede, mas também apresentou em seu projeto algoritmos de tratamento de imagens para a realização da segmentação da imagem em objeto de interesse e fundo. Técnicas como Curva ROC (Característica de

Operação do Receptor - *Receiver Operating Characteristics*), *F-Measure*, Coeficiente *Dice* [Prabha e Kumar 2016] poderiam ser utilizadas nesta etapa, porém ocasionariam em mais gasto da memória do robô.

Por esses motivos escolheu-se o protótipo II para ser utilizado no processo de desenvolvimento da visão computacional do robô. Além de apresentar um melhor uso da memória RAM que o outro protótipo e maior acurácia, o protótipo também possui integrado ao seu processo de treinamento toda a parte de segmentação da imagem em área de interesse.

As Figuras 2 e 3 apresentam alguns resultados obtidos após rodar o sistema YOLO em imagens de diferentes classes.

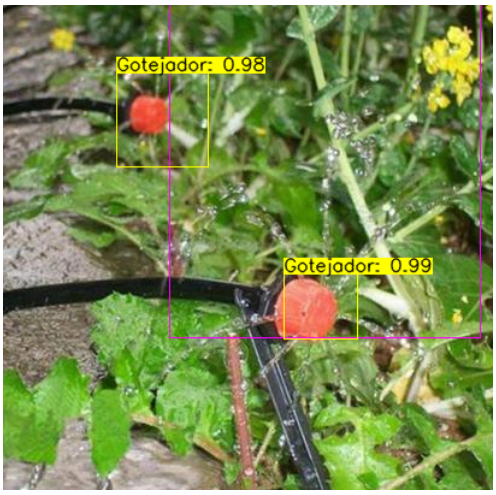

Figura 2. Exemplo Gotejador

Figura 3. Exemplo Planta

4. Considerações Finais

A arquitetura YOLO mostrou melhores resultados quando comparada à arquitetura AlexNet em relação a desempenho, número de parâmetros e arquivos gerados, gasto máximo de memória, entre outros. Além disso, ele já possui um sistema integrado que permite marcar a área que os objetos identificados estão. Por este motivo, foi escolhido o protótipo II para ser embarcado no robô. O sistema de visão desenvolvido foi então embarcado

e se mostrou capaz de detectar e identificar objetos presentes em ambiente agrícola, tais como plantas, veículos, pessoas, com uma taxa de acerto de 77.40%.

Referências

ABDI (2019). Agro 4.0.

Academy, D. S. (2019). Deep learning book.

Alake, R. (2020). Implementing alexnet cnn architecture using tensorflow 2.0+ and keras.

Åstrand, B. e Baerveldt, A.-J. (2002). An agricultural mobile robot with vision-based perception for mechanical weed control. *Autonomous robots*, 13(1):21–35.

Berenstein, R., Fox, R., McKinley, S., Carpin, S., e Goldberg, K. (2018). Robustly adjusting indoor drip irrigation emitters with the toyota hsr robot. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, Australia. IEEE, IEEE.

itMídia (2019). Python: 10 motivos para aprender a linguagem em 2019.

Jacob, S. e Samrat, S. (2020). Train yolov4-tiny on custom data - lightning fast object detection.

Jardim, A. (2019). Inovação para o agro 4.0. *AgroANALYSIS*, 38(4):48.

Jodas, D. S., Marranghello, N., Pereira, A. S., e Guido, R. C. (2013). Desenvolvimento de um sistema para navegaçao de robôs móveis por caminhos em plantaçoes. *Interciência & Sociedade*, 2(1).

Krizhevsky, A., Sutskever, I., e Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Lam, Y., Slaughter, D., Wallender, W., e Upadhyaya, S. (2006). Computer vision system for automatic control of precision furrow irrigation system. In *2006 ASAE Annual Meeting*, page 1. American Society of Agricultural and Biological Engineers.

Massruhá, S. M. F. S. e Leite, M. d. A. (2017). Agro 4.0-rumo à agricultura digital. In *Embrapa Informática Agropecuária-Artigo em anais de congresso (ALICE)*. In: MAGNONI JÚNIOR, L.; STEVENS, D.; SILVA, WTL da; VALE, JMF do; PURINI, SR ....

Orac, R. (2020). What's new in yolov4?

Paz, V. P. d. S., Teodoro, R. E. F., e Mendonça, F. C. (2000). Recursos hídricos, agricultura irrigada e meio ambiente. *Revista Brasileira de Engenharia Agrícola e Ambiental*, 4:465–473.

Prabha, D. S. e Kumar, J. S. (2016). Performance evaluation of image segmentation using objective methods. *Indian J. Sci. Technol*, 9(8):1–8.

Santos, B. P., Alberto, A., Lima, T. D. F. M., e Charrua-Santos, F. M. B. (2018). Industry 4.0: challenges and opportunities. *Revista Produção E Desenvolvimento*.

Techzizou (2020). Yolov4 vs yolov4-tiny.

UNESCO (2020). Relatório mundial das nações unidas sobre desenvolvimento dos recursos hídricos 2020: O manejo dos recursos hídricos em condições de incerteza e risco.

Wang, R., Xu, J., e Han, T. X. (2019). Object instance detection with pruned alexnet and extended training data. *Signal Processing: Image Communication*, 70:145–156.

Yang, K., Qinami, K., Fei-Fei, L., Deng, J., e Russakovsky, O. (2020). Towards fairer datasets: Filtering and balancing the distribution of the people subtree in the imagenet hierarchy. In *Conference on Fairness, Accountability, and Transparency*.