UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS DE CURITIBA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

ALEX TOREZIN MENDONÇA
GEOVANI TOREZIN MENDONÇA

AUTOMAÇÃO NO AGRONEGÓCIO DE PEQUENO PORTE: PROTÓTIPO PARA SELEÇÃO DE MORANGOS COM USO DE VISÃO COMPUTACIONAL E INTELIGÊNCIA ARTIFICIAL

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2019

ALEX TOREZIN MENDONÇA
GEOVANI TOREZIN MENDONÇA

AUTOMAÇÃO NO AGRONEGÓCIO DE PEQUENO PORTE: PROTÓTIPO PARA SELEÇÃO DE MORANGOS COM USO DE VISÃO COMPUTACIONAL E INTELIGÊNCIA ARTIFICIAL

Trabalho de conclusão de curso de Graduação em Engenharia de Controle e Automação apresentado à disciplina de Trabalho de conclusão de curso 2, do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR) como requisito para obtenção do título de Engenheiro de Controle e Automação

Orientador: Prof. Dr. Roberto Candido

CURITIBA
2019

Alex Torezin Mendonça
Geovani Torezin Mendonça

AUTOMAÇÃO NO AGRONEGÓCIO DE PEQUENO PORTE: PROTÓTIPO PARA SELEÇÃO DE MORANGOS COM USO DE VISÃO COMPUTACIONAL E INTELIGÊNCIA ARTIFICIAL

Este Trabalho de Conclusão de Curso de Graduação foi julgado e aprovado como requisito parcial para a obtenção do Título de Engenheiro de Controle e Automação, do curso de Engenharia de Controle e Automação do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR).

Curitiba, 27 de junho de 2019.

_____

Prof. Paulo Sérgio Walenia, Esp.
Coordenador de Curso
Engenharia de Controle e Automação

_____

Prof. Marcelo de Oliveira Rosa, Dr.
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia de Controle e Automação do DAELT

ORIENTAÇÃO

_____

Roberto Candido, Dr.
Universidade Tecnológica Federal do Paraná
Orientador

BANCA EXAMINADORA

_____

Roberto Candido, Dr.
Universidade Tecnológica Federal do Paraná

_____

Marco Antonio Busseti de Paula, Dr.
Universidade Tecnológica Federal do Paraná

_____

Marcelo Rodrigues, Dr.
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia de Controle e Automação

Aos nossos pais Silvanira Mendonça e Jalmir Mendonça.

Amamos muito vocês!

AGRADECIMENTOS

Aos nossos pais, por todo apoio e compreensão, nos apoiando em todos os momentos difíceis e também e também de alegria, sempre nos incentivando.

Ao nosso orientador, Prof. Dr. Roberto Candido, pela disponibilidade, por nos proporcionar poder levar nosso conhecimento acadêmico para além da universidade.

A Maria Isabel Rosa Guimarães, que por meio do SEBRAE, nos proporcionou o contato com produtores de morango da região de Curitiba e nos apoiou durante o desenvolvimento deste trabalho.

Ao SEBRAE, pelo apoio, disponibilidade e mentoria.

Aos produtores de morango que visitamos e foram essenciais para o modelamento do problema, em especial a Ozanan Martins de Oliveira.

RESUMO

MENDONÇA, Alex Torezin; MENDONÇA, Geovani Torezin. **Automação no Agronegócio de Pequeno Porte: Protótipo para Seleção de Morangos com uso de Visão Computacional e Inteligência Artificial.** 94 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Controle e Automação. Universidade Tecnológica Federal do Paraná, Curitiba, 2019.

O cultivo de alimentos por meio da agricultura familiar possui grande representatividade na produção brasileira. Porém, a disponibilidade de recursos tecnológicos aplicáveis e acessíveis para esta demanda é baixa, fazendo com que a produção seja realizada de forma manual. Devido a este fato, foi estudado o desenvolvimento de tecnologia voltada para o agricultor de pequeno porte, em específico o desenvolvimento de um seletor de protótipo para a seleção de morangos. Foi utilizada visão computacional e inteligência artificial, treinada para a seleção de morangos com base em sua maturidade, afim de atender produtores de São José dos Pinhais, região metropolitana de Curitiba no Paraná. Para a implementação da visão computacional foi utilizada a biblioteca OpenCV e para a inteligência artificial foi utilizada a biblioteca TensorFlow, ambas de código aberto. Uma acurácia de 98,72% na seleção de morangos verdes e maduros foi alcançada ao final do desenvolvimento do protótipo, bem como uma capacidade de selecionar 207,36 kg de morangos por dia.

Palavras-chave: Seleção de morangos. Agronegócio de pequeno porte. Agricultura familiar. Inteligência artificial. Visão computacional.

ABSTRACT

MENDONÇA, Alex Torezin; MENDONÇA, Geovani Torezin. **Automation in the Small Agribusiness: Prototype for Strawberries Selection with use Computational Vision and Artificial Intelligence.** 94 f. Final dissertation – Bachelor Degree Control and Automation Engineering. Federal Technological University of Parana, Curitiba, 2019.

The cultivation of food through family farming has great representativeness in Brazilian. However, the availability of technological resources that are applicable and accessible to this demand is low, making the production be manually. Due to this fact, we studied the development of technology aimed at the small farmer, in particular the development of a prototype selector for the selection of strawberries. Computational Vision and artificial intelligence was used, trained to select strawberries based on their maturity, in order to attend the producers of São José dos Pinhais, metropolitan region of Curitiba, Paraná. For the implementation of the computer vision, the OpenCV library was used and for the artificial intelligence the TensorFlow library, both open source, was used. An accuracy of 98.72% in the selection of green and ripe strawberries was reached at the end of the prototype development, as well as a capacity to select 207.36 kg of strawberries every day.

Keywords: Strawberries selection. Small Agribusiness. Family farming. Artificial intelligence. Computational vision.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CEPEA	Centro de estudos avançados em economia aplicada.
CBAP	Comissão Brasileira de Agricultura de Precisão
GPS	*Global Positioning System* (Sistema de Posicionamento Global)
IA	Inteligência artificial
IPEA	Instituto de Pesquisa Econômica Aplicada
IBGE	Instituto Brasileiro de Geografia e Estatística
MDF	*Medium Density Fiberboard* ou em tradução livre, placa de média densidade
OpenCV	*Open Source Computer Vision Library* ou em tradução livre, Biblioteca de visão computacional de código aberto
PVC	*Polyvinyl chloride* ou em tradução livre, policloreto de vinila
RNAs	Redes Neurais Artificiais
SEBRAE	Serviço Brasileiro de Apoio às Micro e Pequenas Empresas
SNA	Sociedade Nacional de Agricultura
TCC	Trabalho de Conclusão de Curso

SUMÁRIO

1. INTRODUÇÃO

O agronegócio brasileiro, em especial o de pequeno porte, em que o produtor é responsável por todas as etapas do manejo, utilizam baixo grau de automação. Já os grandes produtores, corporações produtoras de commodities, utilizam largamente a automação em seus processos, desde a plantação até o consumidor final.

O pequeno produtor fica refém de suas próprias mãos para realizar o trabalho no campo, uma vez que, grande parte dos equipamentos produzidos são para produções em larga escala, e de alto custo.

> É possível observar, no entanto, que os investimentos na agricultura estão sendo direcionados aos grandes latifúndios, para a produção de monoculturas, que em sua maioria são exportadas, gerando ganhos na balança comercial brasileira. Nessa realidade o pequeno produtor, com sua agricultura familiar, acaba esquecido e marginalizado, normalmente ficando com os "restos" dos recursos que o governo tem disponibilizado a agricultura (TROMBINI et al., 2012).

Entretanto, a agricultura de pequeno porte, necessita avançar tecnologicamente, tanto para se manter competitiva no mercado de alimentos, quanto para proporcionar a melhoria na qualidade de vida do produtor. De acordo com o Instituto Brasileiro de Geografia e Estatística (IBGE), a agricultura familiar possui uma grande representatividade no setor. Do total de cerca de 5 milhões de estabelecimentos, 4,3 milhões (84%) são de agricultura familiar e pequeno porte e 807 mil (16%) são de agricultura não familiar ou patronal. Os pequenos ocupam 12,3 milhões de pessoas (74%), e os grandes, 4,2 milhões (25%) (IPEA, 2011 apud IBGE, 2006).

De acordo com a Sociedade Nacional de Agricultura (SNA), o Brasil é o terceiro maior produtor de frutas do mundo, com uma produção anual de 41,5 milhões de toneladas, ficando atrás somente da China e Índia. Também é o terceiro maior mercado de consumo de frutas frescas do planeta, com estimativas em 2014 por volta de 18 milhões de toneladas. Em torno de 53% das frutas são destinadas ao mercado "*in natura*" e 47% à agroindústria, para a produção de sucos, polpas, compotas entre outras (Equipe SNA/RJ, 2015).

O cultivo de morangos realizado por pequenos produtores foi abordado com intuito de empodera-los. A carência de tecnologia para a produção familiar faz com

que estes produtores utilizem um baixo grau de tecnologia no campo. De modo geral, a gama de equipamentos para o cultivo e processamento do morango possui um alto custo e baixa qualidade, perante a disponibilidade de capital e qualidade de produção exigida.

1.1 TEMA

Protótipo seletor de morangos com uso de visão computacional e inteligência artificial.

1.1.1 Delimitação do Tema

Concepção de um protótipo capaz de identificar e selecionar os morangos pela sua cor. A solução desenvolvida conta com um sistema de seleção baseada na tomada de decisão de uma inteligência artificial implementada e de uma visão computacional para adquirir as informações dos frutos.

1.2 PROBLEMAS E PREMISSAS

As máquinas seletoras de morangos existentes no mercado são desenvolvidas para processos de grande porte, tornando inacessível para pequenos produtores devido ao alto custo. A falta de soluções é para todos os produtores de pequeno porte, e para o estudo buscou-se os produtores de morangos de São José dos Pinhais, na região metropolitana de Curitiba no Paraná.

O morango é um fruto delicado, e o processo de classificação pode ser destrutivo se realizado sem os devidos cuidados. O maquinário existente disponível ao produtor de frutas não leva em consideração a fragilidade dos morangos. Esta falta de cuidado torna o processo de seleção de morangos muito agressivo, gerando um produto de baixa qualidade para o consumidor final.

O processo de seleção cuidadoso tende de agregar valor ao produto, disponibilizando produtos selecionados, com maior índice de maturidade. A armazenagem destes morangos após a classificação necessita ser realizada de forma precisa e de preparo para a próxima fase que será a embalagem.

1.3 OBJETIVOS

Diante do exposto, o objetivo deste trabalho foi desenvolver um protótipo seletor de morangos com uso de visão computacional e inteligência artificial.

1.3.1 Objetivo Geral

Criar uma solução tecnológica para seleção na pós colheita, para a produção da agricultura familiar.

1.3.2 Objetivos Específicos

- Revisar a bibliografia sobre a produção tecnificada no agronegócio.
- Especificar uma linguagem de programação.
- Aplicar a visão computacional na seleção do morango.
- Usar a inteligência artificial como ferramenta no processo de seleção.
- Criar um protótipo para selecionar morangos de forma automática e não-destrutiva.
- Analisar resultados na seleção.

1.4 JUSTIFICATIVA

O Brasil possui uma grande extensão que pode ser utilizada para agricultura, para tal situação não fica atrás em produção mundial de frutas, atualmente ocupando a terceira posição segundo a Sociedade Nacional de Agricultura (SNA/RJ, 2015).

Embora um número crescente na produção de frutas, cerca de 30% da produção é desperdiçada e não chega ao consumo, este fato está estreitamente relacionado a falta de tecnologia adequada. Tais perdas se concentram na pós-colheita na qual o manuseio, transporte, conservação e uma seleção ineficiente acaba por estragar as frutas (SANCHES, LINO, 2010).

A alta exigência na qualidade da produção das frutas está ligada às mudanças culturais dos costumes alimentares dos compradores. Os consumidores buscam mais

do que nunca uma maior qualidade e isto se refere ao tamanho, cor, formato e danos da fruta (SANCHES, LINO, 2010).

Selecionar e classificar frutas é predominantemente um trabalho manual. A crescente demanda por frutos melhores e selecionados, juntamente com a ineficiência neste trabalho manual, abriu portas para a implantação de sistemas de classificação na pós colheita. Sistemas automáticos que eliminam a decisão humana e realizam um trabalho não destrutivo das frutas vem ganhando espaço, porém para pequenos produtores esta realidade é outra, pois as máquinas desenvolvidas para este tipo de trabalho são focadas em grandes produtores e possuem um valor muito elevado.

O processo autônomo de seleção de frutas é um grande passo, pois promove maior competitividade de pequenos produtores, melhoria na qualidade de vida, redução do estresse e melhora na ergonomia para os trabalhadores.

Estes dispositivos melhoram a produtividade, pois reduzem erros e operam por muito mais tempo sem interrupções, fazendo com que a produção e a qualidade não sejam afetadas em nenhum momento.

1.5 PROCEDIMENTOS METODOLÓGICOS

Os procedimentos para a construção deste TCC foram:

Fase 1: Revisão de bibliografias, afim de estudar e analisar o mercado brasileiro quanto ao agronegócio e tecnologias a utilizadas neste trabalho.

Fase 2: Especificar e estudar a aplicação da visão computacional no projeto, afim de levantar sua aplicabilidade e utilidade.

Fase 3: Especificar e estudar a linguagem computacional que foi utilizada, afim de utilizar a linguagem mais conveniente para o desenvolvimento de tecnologia utilizando inteligência artificial.

Fase 4: Fazer um estudo acerca da inteligência artificial utilizada, entender mais sobre seus conceitos, utilizações e aplicações, bem como entender os passos para a implementação desta.

Fase 5: Desenvolver um protótipo capaz de fazer a seleção dos morangos, focando no baixo custo e na utilização de componentes disponíveis no mercado.

1.6 ESTRUTURA DO TRABALHO

O trabalho é composto por 5 capítulos, sendo eles:

Capítulo 1: Introdução: conta com uma breve introdução, o tema abordado, a delimitação deste tema, os problemas e premissas, os objetivos tanto gerais quanto específicos, a justificativa, os procedimentos e o cronograma.

Capítulo 2: Revisão bibliográfica: é composto pela revisão bibliográfica a respeito do tema e estado da arte sobre visão computacional.

Capítulo 3: Estudo e desenvolvimento de uma solução tecnológica para o agronegócio.

Capítulo 4: Testes e resultados: é composto pelos testes e resultados obtidos com o protótipo.

Capítulo 5: Conclusão: é composto pelas considerações finais e conclusões do trabalho

2. REVISÃO BIBLIOGRÁFICA

A partir de estudos dirigidos, é possível analisar como o mercado do agronegócio brasileiro tem se comportado ao longo dos anos, bem como analisar as perspectivas da implementação de tecnologia no campo, sobre tudo nos pequenos produtores. Assim, estudou-se o contexto histórico e atual do agronegócio no Brasil e tecnologias que serão aplicadas neste protótipo, como visão computacional e inteligência artificial.

2.1 O AGRONEGÓCIO BRASILEIRO

O agronegócio no Brasil vem demonstrando ao decorrer dos períodos econômicos ser crucial para o desenvolvimento do país, possuindo grande influência na dinâmica econômica e social (BUAINAIN et al, 2014).

O Brasil possui umas das maiores áreas cultiváveis do planeta, garantindo o abastecimento interno e sendo um potencial exportador de alimentos. O país se encontra na terceira posição de maior exportador do planeta, porém possui potencial para ser líder nesse ranking (BRASIL PERDEU ESPAÇO, 2017).

Segundo o PIB de 2017, o agronegócio no Brasil tem uma grande representatividade na economia do país, com 21,6% do PIB. Nota-se que o país é um grande potencial e ainda segundo, ocupa o topo do ranking em exportação de carne bovina e aves, e encontra-se como líder em produção de café, açúcar, laranja e no primeiro em exportação de açúcar e etanol (CEPEA, 2018).

O Quadro 1 apresenta histórico com a participação do agronegócio no PIB brasileiro, de 1997 até 2017, na qual é possível observar a representatividade deste ramo no PIB do pais.

Quadro 1 - Participação do Agronegócio no PIB do Brasil

	Participação do Agronegócio no PIB do Brasil (em %) Agronegócio					
	PIB total_BR (a preços de mercado em R$ milhões correntes - ref 2010)	(A) Insumos	(B) Agropecuária	(C) Indústria	(D) Serviços	Agronegócio Total (A+B+C+D)
1996	854.764	0,7%	4,6%	11,8%	14,8%	31,9%
1997	952.089	0,7%	4,4%	10,7%	13,4%	29,2%
1998	1.002.351	0,7%	4,4%	10,1%	12,7%	27,9%
1999	1.087.710	0,8%	4,5%	9,9%	12,6%	27,8%
2000	1.199.092	0,9%	4,4%	10,4%	13,2%	28,9%
2001	1.315.755	0,9%	5,0%	10,1%	13,1%	29,0%
2002	1.488.787	1,1%	5,7%	9,9%	12,8%	29,5%
2003	1.717.950	1,3%	6,6%	9,7%	12,8%	30,4%
2004	1.957.751	1,4%	5,7%	8,9%	11,3%	27,4%
2005	2.170.585	1,1%	4,5%	8,4%	10,2%	24,2%
2006	2.409.450	0,9%	4,8%	7,9%	9,6%	23,3%
2007	2.720.263	1,0%	4,9%	7,4%	9,5%	22,7%
2008	3.109.803	1,2%	5,1%	7,1%	9,4%	22,8%
2009	3.333.039	1,0%	4,3%	7,1%	9,2%	21,5%
2010	3.885.847	0,9%	4,9%	6,8%	9,1%	21,6%
2011	4.376.382	1,0%	5,4%	6,2%	8,5%	21,0%
2012	4.814.760	1,0%	4,7%	5,9%	7,8%	19,4%
2013	5.331.619	1,0%	4,8%	5,7%	7,7%	19,2%
2014	5.778.953	0,9%	4,7%	5,7%	7,8%	19,1%
2015	5.995.787	1,0%	4,9%	6,1%	8,6%	20,5%
2016	6.259.228	1,0%	5,7%	6,6%	9,5%	22,8%
2017	6.559.940	0,9%	5,4%	6,3%	9,0%	21,6%

Fonte: Os autores com base CEPEA, 2018.

Segundo a Agrostat Brasil, a grande importância que o agronegócio traz ao país é visto na grande participação do produto interno bruto, gerando aproximadamente 20% de todos os empregos do país e podendo ter um grande potencial de crescimento pois apenas 7,6% das terras brasileiras são manuseadas para a agricultura. (LIMA, 2018; AGRONEGÓCIO BRASILEIRO, 2018)

Condições favoráveis como a disponibilidade de terras agricultáveis, abundância de água e solo fértil são responsáveis pelo país possuir cerca de 44,1% como representatividade do agronegócio na exportação total do país (Agrosat Brasil, 2015).

A produção rural sempre foi um grande ponto forte para a geração da riqueza do Brasil, na Quadro 2 encontra-se um histórico da balança comercial entre a exportação e importação total no Brasil e a exportação e importação na área do agronegócio durante 20 anos. É evidente que o agronegócio sempre possuiu uma participação importante para o país, participando com quase metade de tudo o que é exportado.

Quadro 2 - Balança comercial Brasileira e balança comercial do agronegócio: 1997 a 2017

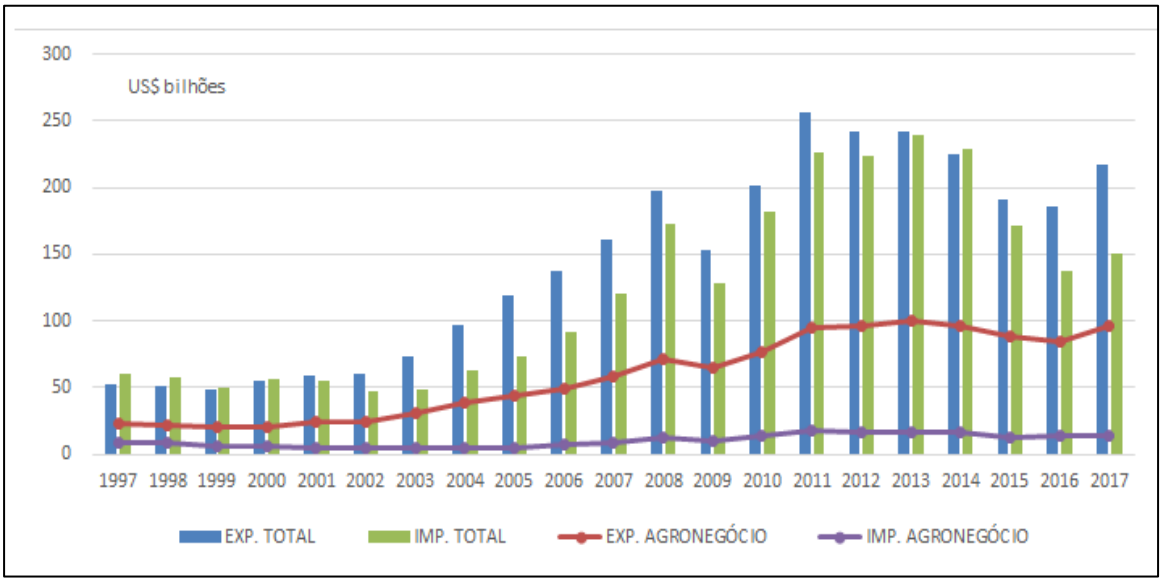
US$ bilhões

Ano	Exportações			Importações			Saldo	
	Total Brasil (A)	Agronegócio (B)	Part.% (B/A)	Total Brasil (C)	Agronegócio (D)	Part.% (D/C)	Total Brasil	Agronegócio
1997	52,98	23,37	44,1	59,75	8,20	13,7	-6,76	15,17
1998	51,14	21,56	42,1	57,76	8,04	13,9	-6,62	13,51
1999	48,01	20,50	42,7	49,30	5,70	11,6	-1,29	14,80
2000	55,12	20,60	37,4	55,85	5,76	10,3	-0,73	14,85
2001	58,29	23,87	40,9	55,60	4,81	8,6	2,68	19,06
2002	60,44	24,85	41,1	47,24	4,45	9,4	13,20	20,39
2003	73,20	30,65	41,9	48,33	4,75	9,8	24,88	25,90
2004	96,68	39,04	40,4	62,84	4,84	7,7	33,84	34,20
2005	118,53	43,62	36,8	73,60	5,11	6,9	44,93	38,51
2006	137,81	49,47	35,9	91,35	6,70	7,3	46,46	42,77
2007	160,65	58,43	36,4	120,62	8,73	7,2	40,03	49,70
2008	197,94	71,84	36,3	172,98	11,88	6,9	24,96	59,96
2009	152,99	64,79	42,3	127,72	9,90	7,8	25,27	54,89
2010	201,92	76,44	37,9	181,77	13,40	7,4	20,15	63,04
2011	256,04	94,97	37,1	226,25	17,51	7,7	29,79	77,46
2012	242,58	95,81	39,5	223,18	16,41	7,4	19,39	79,41
2013	242,03	99,97	41,3	239,75	17,06	7,1	2,29	82,91
2014	225,10	96,75	43,0	229,15	16,61	7,3	-4,05	80,13
2015	191,13	88,22	46,2	171,45	13,07	7,6	19,69	75,15
2016	185,24	84,93	45,9	137,55	13,63	9,9	47,68	71,31
2017	217,74	96,01	44,1	150,75	14,15	9,4	66,99	81,86

Fonte: Agrosat Brasil a partir de dados da SECEX/MDIC.

Para tornar os resultados da Quadro 2 mais facilmente observáveis, a Figura 1 realça o quanto o agronegócio é importante para o país e como o mesmo segue o mercado de exportações e nestes 20 anos participou com grande parte dos resultados.

Figura 1 - Evolução anual da balança comercial brasileira e do agronegócio 1997 a 2017



Fonte: Agrosat Brasil, a partir de dados da SECEX/MDIC

A tecnologia desenvolvida para o campo visa o aumento da produtividade e também da eficiência e aproveitamento de todo o alimento produzido, como resultado obtém-se um aumento do lucro.

Técnicas de manejo e uma boa tecnologia na cultivação dos alimentos ajudam em aumento na produção, mas não apenas estes impactam o objetivo final, o lucro, se um dos maiores desafios para o produtor agrícola é considerar que todo o alimento produzido passará por um longo caminho até a chegada ao consumidor, está longa cadeia danifica o produto e causa perdas que irá impactar diretamente em sua lucratividade, estas perdas podem ser reduzidas com a tecnologia, no transporte, condicionamento e seleção (ANATÂNIA, 2012, apud ASSAD E ALMEIDA, 2004)

2.1.1 Agricultura familiar no Brasil

Atualmente, cerca de 19 milhões de empregos são da área do agronegócio e o setor da agricultura familiar conta com 11,5 milhões destes empregos, assim evidenciando o quanto é grandiosa a agricultura de pequenos produtores no Brasil (AGRONEGÓCIO BRASILEIRO, 2018).

Rotineiramente surgem algumas perguntas a respeito de como a agricultura irá produzir alimentos de forma sustentável, saudável e suficiente para abastecer toda a população. E quem irá fazer esta produção de modo tão eficiente e sem prejudicar o meio ambiente.

Segundo estudos realizados nos últimos anos (IAASTD, 2009; PRETTY et.al., 2010; HLPE, 2012) é possível se prever que os pequenos produtores são a solução, pois há um senso comum em que os agricultores familiares terão que cumprir sua parte para alcançar estas necessidades, é evidente que são essenciais para o abastecimento da população e serão mais essenciais ainda no futuro em que haverá mais pessoas no mundo (SCHENEIDER, 2016).

A ideia de monoculturas que se emprega em grandes propriedades no Brasil, em que o trabalho é realizado por maquinários de grande porte em que se usa agroquímicos, pode parecer a solução, mas evidencias apontam que para abastecer o planeta até 2050 onde haverá, pelas previsões estatísticas, cerca de 10 bilhões de pessoas, será possível apenas utilizando como base a produção pela agricultura familiar (Gazzoni apud ONU, 2017). E o mais importante, sem a utilização de

agroquímicos que podem vir a poluir o meio ambiente e colaborar para o surgimento de diversos problemas e agravar um grande problema mundial, a falta de água, causando a inutilização de parte da hidrosfera por estar imprópria para consumo.

> A Organização das Nações Unidas para Alimentação e Agricultura (GLADEK et ali, 2016; Lowder, at alii, 2014; 2016) publicou que do total de 570 milhões de unidades produtivas que existem no mundo, nada menos do que 500 milhões são de pequenos agricultores, também chamados de agricultores familiares. A FAO (2014a; 2015) estima que "a agricultura familiar é de longe a forma mais prevalente de agricultura no mundo. Estimativas sugerem que ela ocupa cerca de 70-80% das terras agrícolas e produzem mais de 80% dos alimentos do mundo em termos de valor" (2014a, p.11) (SCHENEIDER, 2016).

De acordo com Leportal et. al. (2014, p.37), 84,4% das propriedades agrícolas do Brasil são pertencentes ao grupo de agricultura familiar, e que estas são responsáveis pela maioria do alimento utilizado para abastecer o país.

Apesar de a agricultura familiar representar um grande impacto na economia, há outros motivos que contribuem para que esta forma de produção continue, questões sociais e demográficas são fatores decisivos para preservação das famílias no ambiente rural. Segundo Sergio Schneider, (2016), apud Paulilo, (2013), "A agricultura familiar também é importante para as mulheres e os jovens, pois o acesso à terra e aos ativos produtivos são recursos fundamentais para garantia de seus meios de vida nos casos em que os homens migram para trabalhar fora da agricultura", pode-se concluir que a agricultura familiar contribui para que o êxodo rural diminua, assim possibilitando que o pequeno produtor possa se sustentar e sobreviver apenas com sua própria terra. Pode-se ainda verificar que as práticas de políticas de favorecimento aos pequenos produtores têm cooperado para a redução da pobreza. (SCHNEIDER, 2016, apud SILVA, et al., 2009).

2.1.2 O morango

O que torna o morango a fruta que é mais produzida entre as de pequeno porte é a sua alta rentabilidade, aceitação no mercado e pela alta variedade de processamentos, podendo ser comercializado em sucos, polpas, geleias, compotas e sorvetes (FACHINELLO et al., 2011).

São Paulo, Minas Gerais, Rio grande do Sul e cinco outros estados são destaque na produção do morango. A área ocupada pelo cultivo da fruta é cerca de 3500 hectares e em sua maioria pertencem a agricultores familiares, destacando ainda mais a importância econômica e social da fruta. (SPECHT; BLUME, 2011).

Apesar de o Rio Grande do Sul ser um dos produtores mais tradicionais do país, vem perdendo espaço para outros estados, inclusive o Paraná.

> Esse decréscimo na produção de morangos na Região Sul, segundo Amaro (2002) está vinculada com a falta de padrões de qualidade, deficiência na assistência técnica aos produtores, problemas estruturais que comprometem o fornecimento de frutas com os padrões de qualidade necessários, baixo nível tecnológico no segmento do processamento industrial, assim como deficiência na gestão de negócios e relações de mercado. Além disso, somam-se as dificuldades encontradas pelos produtores, que conforme Henz (2010) são em ordem de importância: incidência de pragas e doenças, aquisição de mudas, custo de embalagens, necessidade de mão de obra e custo de produção elevado. (PERSPECTIVAS PARA A PRODUÇÃO DE MORANGOS, 2016)

O que torna o morango a fruta que é mais produzida entre as de pequeno porte é a sua alta rentabilidade, aceitação no mercado e pela alta variedade de processamentos, podendo ser comercializado em sucos, polpas, geleias, compotas e sorvetes (FACHINELLO et al., 2011).

São Paulo, Minas Gerais, Rio grande do Sul e cinco outros estados são destaque na produção do morango. A área ocupada pelo cultivo da fruta é cerca de 3500 hectares e em sua maioria pertencem a agricultores familiares, destacando ainda mais a importância econômica e social da fruta. (SPECHT; BLUME, 2011).

Apesar de o Rio Grande do Sul ser um dos produtores mais tradicionais do país, vem perdendo espaço para outros estados, inclusive o Paraná.

> Esse decréscimo na produção de morangos na Região Sul, segundo Amaro (2002) está vinculada com a falta de padrões de qualidade, deficiência na assistência técnica aos produtores, problemas estruturais que comprometem o fornecimento de frutas com os padrões de qualidade necessários, baixo nível tecnológico no segmento do processamento industrial, assim como deficiência na gestão de negócios e relações de mercado. Além disso, somam-se as dificuldades encontradas pelos produtores, que conforme Henz (2010) são em ordem de importância: incidência de pragas e doenças, aquisição de mudas, custo de embalagens, necessidade de mão de obra e custo de produção elevado. (PERSPECTIVAS PARA A PRODUÇÃO DE MORANGOS, 2016)

Sobre esta perspectiva, tem-se a observação na necessidade da tecnificação do campo, desta forma alguns dos problemas tratados por Henz (2010) que é a necessidade de mão de obra e alto custo de produção, tendem a diminuir com a implantação de tecnologia.

2.1.3 SEBRAE

O Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) é uma instituição que teve início em 1972, porém o seu desenvolvimento data de antes. Criado pelo Banco Nacional de Desenvolvimento Econômico, mas que hoje possui o nome de Banco Nacional de Desenvolvimento Econômico e Social (BNDES), o atual SEBRAE era um programa para financiar pequenas e médias empresas (SEBRAE).

Atualmente, o SEBRAE é uma entidade privada direcionada para o auxílio a alavancar o desenvolvimento sustentável e competitivo entre empreendimentos que não ultrapassam o rendimento anual bruto de quatro milhões e oitocentos mil reais, o que se denomina para micro e pequenas empresas (SEBRAE).

Suas soluções vão desde o auxílio no planejamento de negócio de novas empresas até o posicionamento de empresas consolidadas, mas que procuram se posicionar no mercado novamente (SEBRAE).

O SEBRAE foi o principal idealizador deste trabalho, esta instituição foi a precursora da problemática abordada neste trabalho, trazendo as dificuldades dos produtores de morangos da região metropolitana de Curitiba e proporcionando o contato entre os alunos e as famílias produtoras.

2.2 TECNOLOGIA NA AGRICULTURA

O surgimento da mecanização da agricultura no Brasil se dá no começo dos anos 1960, com o surgimento do complexo agroindustrial brasileiro, com início da produção de tratores (Sorj, 2008). No Quadro 3 é possível observar o crescimento da mecanização do campo com a inserção dos tratores.

Quadro 3 - Histórico dos tratores no Brasil

Tratores na agricultura brasileira	
Ano	Unidades
1920	1706
1940	3380
1950	8372
1960	61345
1970	157340
1975	331000

Fonte: Sistema Nacional de Planejamento, apud Ministério da Agricultura, 1977, p. 94

No entanto, com o crescimento da população mundial e áreas agricultáveis restritas, a necessidade de automação e intensificação da produção de alimentos se torna cada vez mais necessária. Estimativas apontam que até 2050, a população mundial será de 10 bilhões de pessoas (Gazzoni apud ONU, 2017). Segundo a FAO, será necessário o aumento em cerca de 70%, entre 2010 e 2015, na produção de alimentos, obtendo-se a média geométrica de 1,34% ao ano para atender a demanda de alimentos. Levando em consideração a restrição de áreas agricultáveis e a também o aumento do preço da terra, é possível concluir que a expansão horizontal da produção de alimentos será muito pequena, ficando em torno de 10% do aumento da produção agrícola no período (Gazzoni, 2017). Dado esse panorama, a necessidade da automação do campo e da agricultura de precisão se faz evidente, possibilitando o aumento na produção de alimentos de acordo com o crescimento populacional.

Em 2012, o Ministério da Agricultura, Pecuária e Abastecimento (Mapa), ao instituir a Comissão Brasileira de Agricultura de Precisão (CBAP), definiu a Agricultura de Precisão como “um sistema de gerenciamento agrícola baseada na variação

espacial e temporal da unidade produtiva e visa ao aumento de retorno econômico, à sustentabilidade e à minimização do efeito ao ambiente” (INAMASU, 2014, apud BRASIL, 2012, p. 6).

2.2.1 Histórico da agricultura de precisão no Brasil

Em 1929, em um boletim do campo experimental, Illinois, Linsley e Bauer recomendaram a produtores que fizessem o desenho do mapa da produção com testes de acidez de solo amostrados em grade para aplicação de calcário. Este é o registro mais antigo, que mostra o esforço no estudo da terra e catalogação para tomada de decisão (INAMASU, 2014).

Porém, é sabido que agricultores há séculos levam em conta a variabilidade espacial do terreno para a implantação da lavoura. E de fato, considerar a variabilidade é reconhecer que o terreno, seja ele de qualquer dimensão não possui as mesmas propriedades físico químicas, e, portanto, são indicados para culturas diferentes. Agricultores com mais experiência reconhecem essa área e aproveitam estes espaços para o cultivo da cultura mais adequada e as distribuindo pela propriedade. No entanto, para áreas de cultivo extensivos, a prática de gestão da lavoura com base em conhecimentos práticos se torna pouco viável (INAMASU, 2014).

Com a necessidade de maior domínio sobre a lavoura, o conhecimento prático do agricultor se torna ineficiente diante a grandes extensões de terras, fazendo com que haja a necessidade de maior informatização do campo, coletando o conhecimento do produtor a sistemas de gerenciamento de produção em grande escala.

Já na década de 1980, o uso de eletrônica embarcada em equipamentos e veículos influenciou o desenvolvimento de máquinas agrícolas mais inteligentes. Já no chão de fábrica metal mecânico surgiram máquinas que pudessem ser programadas, veículos autoguiados e robôs industriais o que viabilizou a produção de equipamentos agrícolas mais eficientes.

Em 1978 surgiu o primeiro sistema global de geonavegação (GPS), considerado totalmente operacional em 1995. Com a disponibilização de sinal de satélites GPS, viabilizou a instalação destes dispositivos em equipamentos agrícolas,

gerando a possibilidade da coleta de dados de produção instantânea atrelado a sua coordenada no globo.

Em 1996, surge no mercado colheitadeiras com capacidade de mapeamento de produção, gerando um acentuado crescimento da agricultura de precisão no mundo, o que possibilitou o mapeamento e aplicação de insumos em taxas variáveis, de acordo com a necessidade de cada área plantada, de acordo com informações coletados (INAMASU, 2014).

Porém, o mercado brasileiro ainda muito carente de importações não acompanha o ritmo do desenvolvimento da tecnologia para com campo na década de 1980. Apenas na década de 1990, o mercado nacional de veículos para a agricultura se adequa às tecnologias que emergiram na década de 1980.

Mas, ainda no final da década de 1990, a indústria interna não tinha o mesmo ritmo da internacional. Apenas em 1999, o programa Moderfrota, levou as montadoras nacionais produzirem o que havia de mais moderno para maquinários agrícolas. Neste período, a frota de maquinário agrícola foi revitalizada, e foi possível observar a entrada de novos modelos com eletrônica embarcada, o que alavancaria a agricultura de precisão em grandes propriedades.

Observa-se ainda que a instalação de sistemas de GPS, eletrônica embarcada ou outros benefícios aos equipamentos, possuem um elevado custo para a época. Todo maquinário agrícola possuía grande porte, o sistema de processamento necessitava ser robusto. Atrelado a isto, o alto valor para o receptor de GPS levava ao questionamento da dimensão mínima da propriedade a qual o emprego dessa tecnologia traria benefícios, sugerindo que a agricultura de precisão seria viável a partir de certa dimensão de terra.

O elemento histórico e o alto custo, quando remetido para os dias atuais, explicam o motivo pelo qual a tecnologia chega apenas em grandes propriedades de agricultura extensiva, produtoras de commodities (produtos que funcionam como matéria prima, são produzidos em escala e podem ser estocados sem perda de qualidade). No entanto, a perspectiva do trabalho rural e agricultura familiar possuem destaque significativo no Brasil. O que não acontece com as soluções pensadas para o campo, que visa apenas os grandes produtores, com alto custo, maquinário pesado e necessidade de alto faturamento para custeio.

2.2.2 Análise das possibilidades e tendências do uso da tecnologia na agricultura

Como analisado no contexto evolutivo da implantação de tecnologia no campo, a tendência é para a interconexão de elementos, coleta de dados e aplicação eficiente de insumos com bases nestes. Desta forma, em larga escala, vem se utilizando em grandes lavouras a aplicação de tecnologia embarcada em equipamentos pesados, trazendo uma evolução considerável para o ramo.

Cada vez mais comum, se faz necessário grandes centros para tratamento de dados, armazenamento eficiente e acima de tudo, utilização de forma eficiente. Uma vez que, dados desconexos são apenas um grande amontoado de dados, se faz necessário a utilização de padrões de detecção, *data mining*, inteligência artificial e *deep learning*, integrados com equipamentos e sensores, gerando informações relevantes para tomada de decisão e aplicação da melhor estratégia agrícola.

Analisando essas informações é minimamente possível compreender a dimensão das estruturas necessárias. Com isso, novamente tocamos em um ponto crucial para o desenvolvimento destas tecnologias no campo, o custo elevado. Mais uma vez, os valores despendidos para implementação de soluções de agricultura de precisão ficam distantes do pequeno produtor e da agricultura familiar. Além de alta complexidade na implementação, o foco de grandes companhias, se volta apenas para grandes produtores, que possuem alto capital, e que tem interesse para investimento em equipamentos de alto valor.

2.3 INTELIGÊNCIA ARTIFICIAL

Inteligência artificial (I.A) está diretamente associada à capacidade de computadores e máquinas pensarem como os humanos. Está também atrelada a possibilidade de aprender, pensar, perceber e tomar decisões de forma eficiente. Automatização de atividades que associamos ao pensamento humano, atividades como a tomada de decisões, a resolução de problema, o aprendizado. (RUSSEL, 1962).

Para que seja possível o modelo de I.A usado atualmente, se faz necessário o uso de algumas ferramentas que possibilitem o seu desenvolvimento. Dentre elas podemos citar: *dataset,* rede neural artificial e inteligência artificial.

2.3.1 Dataset

Dataset, também conhecidos como banco de informações, são dados coletados para um determinado fim, estabelecendo um agrupamento com uma quantidade relevante de informações. É uma coleção de dados separadas que são utilizadas de forma única pelo computador (CAMBRIDGE ONLINE DICTIONARY, 2008).

Um *Dataset* possui as características necessárias para se encontrar padrões em uma repetição finita de dados. Muitas vezes são utilizados para treinamento de redes neurais, identificação de sistemas complexos e não lineares.

No contexto de inteligência artificial, *Dataset* é um conjunto de dados de qualquer formato digital, podendo ser: textos, números, fotos, áudios, vídeos ou uma combinação entre estes. O contexto e a necessidade da aplicação demandam o tipo ideal de formato de dados. Na prática, é através deste conjunto de dados que se treina o modelo de aprendizado de máquina.

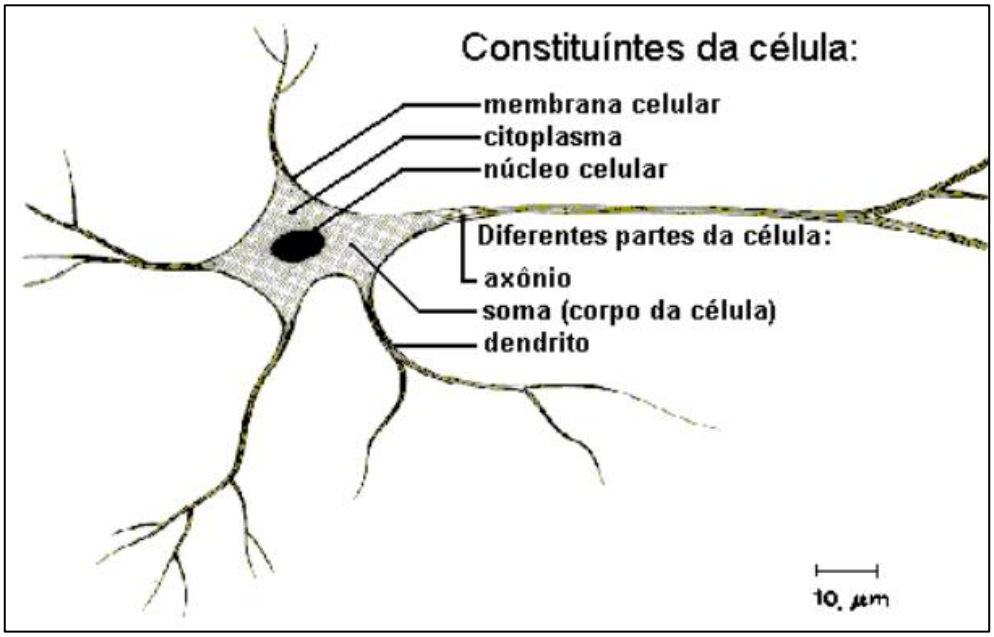
2.3.2 Rede Neural Artificial

Redes neurais artificiais (RNAs) são modelos desenvolvidos em computador a fim de simular o sistema nervoso centra de um animal (cérebro) que são capazes de realizar o aprendizado de máquina e fazer o reconhecimento de padrões.

> O sistema nervoso é formado por um conjunto extremamente complexo de células, os neurônios. Eles têm um papel essencial na determinação do funcionamento e comportamento do corpo humano e do raciocínio. Os neurônios são formados pelos dendritos, que são um conjunto de terminais de entrada, pelo corpo central, e pelos axônios que são longos terminais de saída (REDES NEURAIS ARTIFICIAIS, [201-?]).

Na imagem Figura 2 é possível observar a constituição de um neurônico e sua constituição como, dendritos, corpo central e axônios.

Figura 2 – Célula



Fonte: REDES NEURAIS ARTIFICIAIS, [201-?]

Os neurônios se comunicam através de sinapses. Sinapse é a região na qual dois neurônios entram em contato e através de onde os impulsos nervosos são transmitidos entre eles. Os impulsos recebidos por um neurônio A, em um determinado momento, são processados, e atingindo um dado limiar de ação, o neurônio A dispara, produzindo uma substância neurotransmissora que flui do corpo celular para o axônio, que pode estar conectado a um dendrito de um outro neurônio B. O neurotransmissor pode diminuir ou aumentar a polaridade da membrana pós-sináptica, inibindo ou excitando a geração dos pulsos no neurônio B. Este processo depende de vários fatores, como a geometria da sinapse e o tipo de neurotransmissor.

Em média, cada neurônio forma entre mil e dez mil sinapses. O cérebro humano possui cerca de $10x10^{11}$ neurônios, e o número de sinapses é de mais de $10x10^{14}$, possibilitando a formação de redes muito complexa (REDES NEURAIS ARTIFICIAIS, [201-?]).

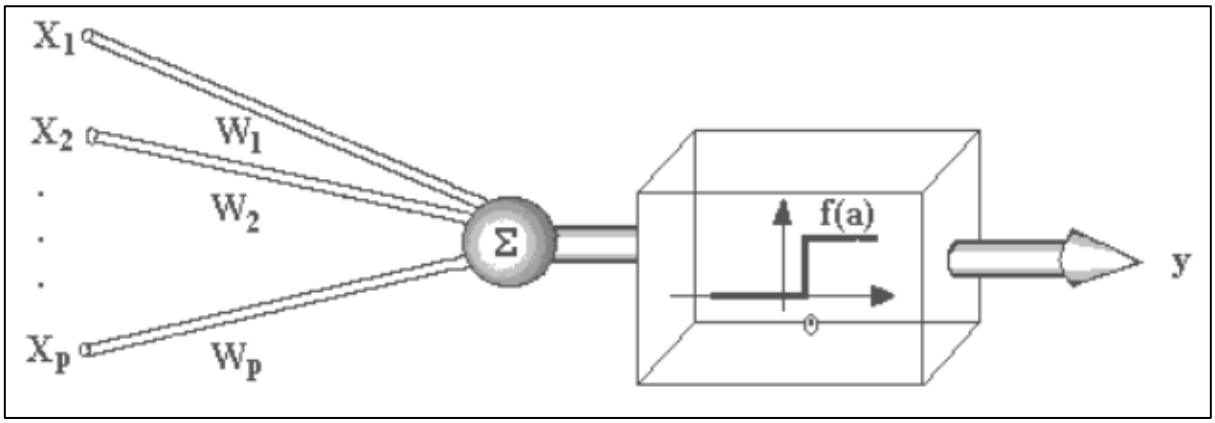
Já para as redes neurais artificiais possuem um funcionamento semelhante as redes neurais originais, cujo o funcionamento é bastante simples. As RNAs possuem diversas unidades de processamento, que se conectam entre si por canais que estão associados a pesos. As unidades fazem operações apenas sobre os dados locais, que são as entradas recebidas por aquela unidade com suas conexões.

A operação de uma unidade de processamento, proposta por McCullock e Pitts em 1943, pode ser resumida da seguinte maneira:

- Sinais são apresentados à entrada;
- Cada sinal é multiplicado por um número, ou peso, que indica a sua influência na saída da unidade;
- É feita a soma ponderada dos sinais que produz um nível de atividade;
- Se este nível de atividade exceder um certo limite (*threshold*) a unidade produz uma determinada resposta de saída.

Na Figura 3 é possível visualizar o esquema de uma unidade proposto por McCullock e Pitts em 1943. Se assemelhando com os neurônios, as unidades possuem canais de conexões que podem ser comparados aos dendritos.

Figura 3 - Esquema da unidade McCullock – Pitts



Fonte: REDES NEURAIS ARTIFICIAIS, [201-?]

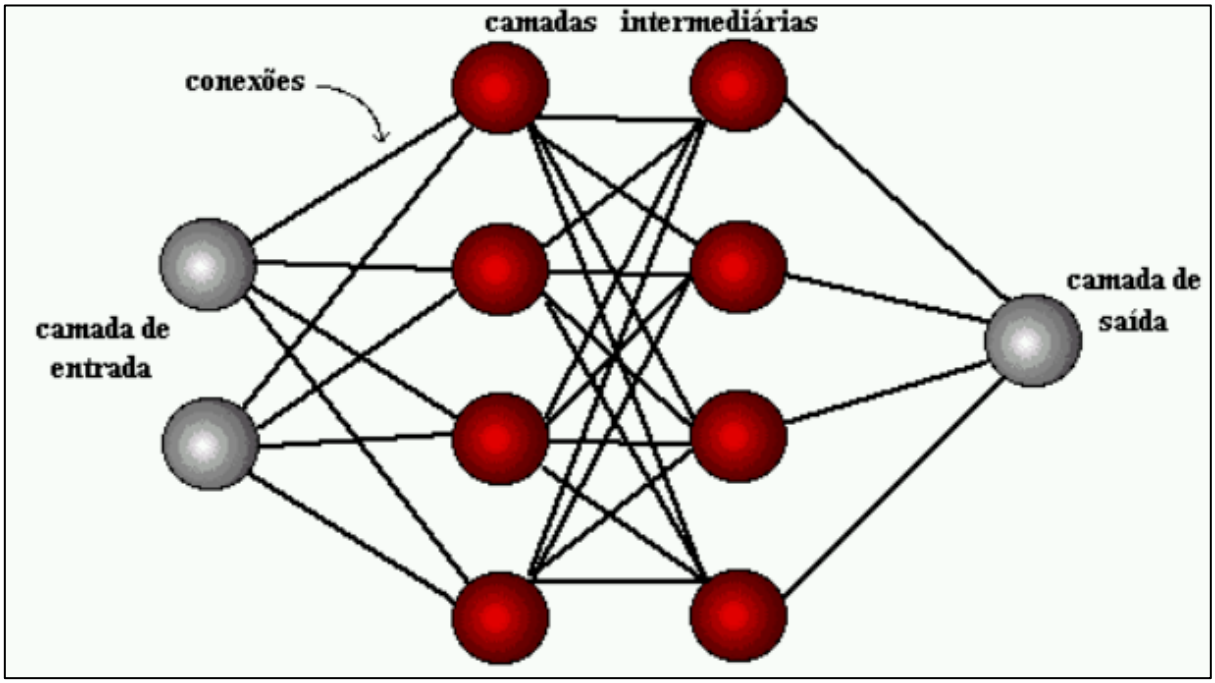
A maioria dos modelos de redes neurais possui alguma regra de treinamento, onde os pesos de suas conexões são ajustados de acordo com os padrões apresentados. Em outras palavras, elas aprendem através de exemplos. Arquiteturas neurais são tipicamente organizadas em camadas, com unidades que podem estar conectadas às unidades da camada posterior.

Na Figura 4 é possível observar como se agrupam as unidades em uma RNA, se dividindo em:

- **Camada de Entrada**: onde os padrões são apresentados à rede;

- **Camadas Intermediárias ou Escondidas**: onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características;
- **Camada de Saída**: onde o resultado final é concluído e apresentado.

Figura 4 - Organização em camadas da RNAs



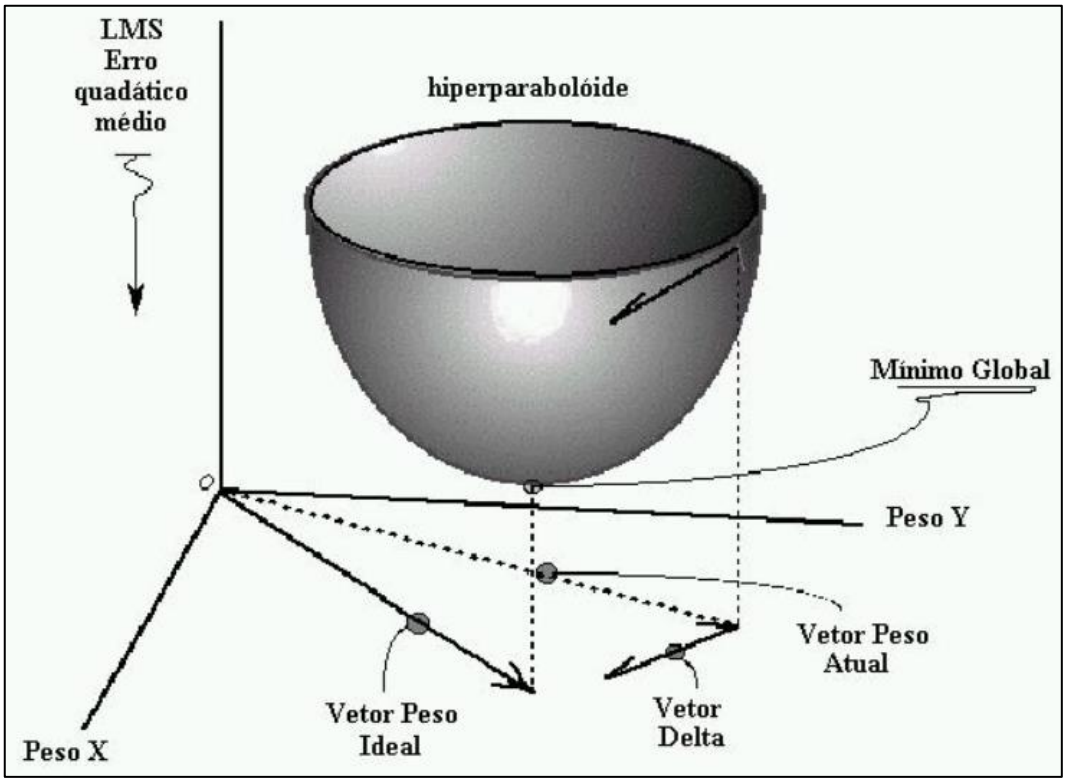
Fonte: REDES NEURAIS ARTIFICIAIS, [201-?].

Para o modelo estudado, vale ressaltar o treinamento supervisionado, que será tratado no item 2.3.3.

> O treinamento supervisionado do modelo de rede Perceptron, consiste em ajustar os pesos e os thresholds de suas unidades para que a classificação desejada seja obtida. Para a adaptação do threshold juntamente com os pesos podemos considerá-lo como sendo o peso associado a uma conexão, cuja entrada é sempre igual à -1 e adaptar o peso relativo a essa entrada. Quando um padrão é inicialmente apresentado à rede, ela produz uma saída. Após medir a distância entre a resposta atual e a desejada, são realizados os ajustes apropriados nos pesos das conexões de modo a reduzir esta distância. Este procedimento é conhecido como Regra Delta (REDES NEURAIS ARTIFICIAIS, [201-?]).

Na Figura 5 é possível observar o funcionamento, de forma genérica da Regra Delta, que relaciona o erro atual e desejado durante o treinamento supervisionado de uma RNA.

Figura 5 - Regra delta para RNA



Fonte: REDES NEURAIS ARTIFICIAIS, [201-?].

De forma geral, a rede se inicia com pesos aleatórios, e repete o processo ajustando os pesos até que o erro obtido esteja dentro de uma margem satisfatória imposta pelo desenvolvedor.

2.3.3 Machine Learning

Machine Learning é uma área de I.A que possui o objetivo do desenvolvimento de técnicas que aplicadas ao sistema de computadores são capazes de adquirir conhecimento de forma automática. O sistema computacional, com base em experiências decorridas de seu funcionamento, consegue acumular conhecimento através de soluções bem-sucedidas de problemas passados.

> A indução é a forma de inferência lógica que permite obter conclusões genéricas sobre um conjunto particular de exemplos. Ela é caracterizada como o raciocínio que se origina em um conceito específico e o generaliza, ou seja, da parte para o todo. Na indução, um conceito é aprendido efetuando-se inferência indutiva sobre os exemplos apresentados. Portanto, as hipóteses geradas através da inferência indutiva podem ou não preservar a verdade (MONARD, 2003).

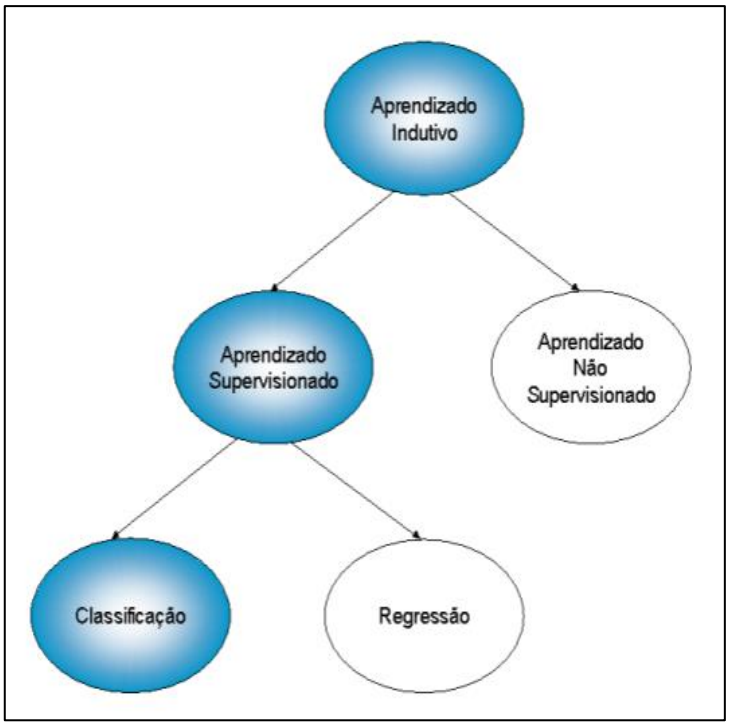
O aprendizado indutivo é efetuado a partir de raciocínio sobre exemplos fornecidos e coletados previamente os *Dataset,* como visto no item 2.3.1. O aprendizado indutivo pode ainda ser classificado como supervisionado e não supervisionado. De maneira ampla, cada exemplo é descrito por um vetor de dados, sejam eles fotos, vídeos ou textos, e etiquetados.

Para o aprendizado supervisionado objetivo do algoritmo de indução é construir um classificador de modo que novos itens ainda não etiquetados possam ser identificados. Estes ainda possuem a classificação com base no tipo de etiqueta utilizada, para *Dataset* com etiquetas com etiquetas discretas tem-se a classificação e para valores contínuos a regressão (MONARD, 2003).

Já no aprendizado não supervisionado, o indutor analisa os exemplos fornecidos e tenta determinar se alguns deles podem ser agrupados de alguma maneira, formando agrupamentos ou clusters (MONARD, 2003 apud CHEESEMAN & STUTZ 1990). Após a determinação dos agrupamentos, normalmente, é necessária uma análise para determinar o que cada agrupamento significa no contexto do problema que está sendo analisado, ou seja, etiqueta-los.

Segundo MONARD (2003), o aprendizado de máquina pode ser dividido em uma hierarquia do aprendizado, como descrito acima. Na Figura 6 é possível analisar esta hierarquia.

Figura 6 - Hierarquia do aprendizado de máquina.



Fonte: MONARD, 2003

Neste trabalho, foi dado um enfoque maior em aprendizado supervisionado, uma vez que os autores intendem que é um melhor caminho para solucionar o problema proposto: seleção de morangos.

Dentro do aprendizado supervisionado existem algumas definições que serão tratadas abaixo.

- Indutor: É o algoritmo de indução, consiste em extrair um bom classificador a partir de um *Dataset*. A saída do indutor, o classificador, pode então ser utilizada para classificar novos itens ainda não etiquetados.
- Atributo: Também conhecido como etiqueta, descrevem uma característica (por exemplo: vermelho (maduro) e verde).
- Classe: As etiquetas são pertencentes a um conjunto discreto de classes a fim de agrupamento de idênticos.
- Ruído: Inerente do mundo real é comum trabalhar com dados imperfeitos. São obtidos através do processo de geração de dados, do processo de aquisição de dados ou até mesmo da etiquetação incorreta.

- Erro e precisão: A taxa de erro é obtida quando comparado o valor estimado com o real valor, utilizando um *Dataset* de teste, em que se é conhecido a saída, a fim de testar o classificador obtido e atestar sua precisão.

Dentre os algoritmos indutores, pode-se ressaltar o TensorFlow™, desenvolvido pelo Google Brain Team, sobre a licença Apache 2.0 open source license, disponível para Linux, macOS, Windows e Android. Foi desenvolvido utilizando as linguagens de programação Python, C++ e CUDA, seu lançamento se deu em 9 de novembro de 2015 e a última versão estável foi disponibilizada em 17 de agosto de 2017.

O TensorFlow™ é uma biblioteca de software de código aberto para computação numérica que usa gráficos de fluxo de dados. Os *nodes* no gráfico representam operações matemáticas, e as arestas representam as matrizes ou tensores de dados multidimensionais que se comunicam com os *nodes*. A arquitetura flexível permite que você implante aplicações de computação a uma ou mais CPUs ou GPUs em um computador, servidor ou dispositivo móvel usando uma única API. O TensorFlow foi desenvolvido por pesquisadores e engenheiros da Google Brain Team no departamento de pesquisas de inteligência de máquina do Google com a finalidade de realizar pesquisas sobre redes neurais profundas e aprendizado de máquina. No entanto, devido à característica abrangente do sistema, ele também pode ser aplicado a vários outros domínios (TENSORFLOW™, 201-?).

2.4 VISÃO COMPUTACIONAL

Visão computacional é um campo da computação que trata de imagens e do aprimoramento de característica relevantes e exclusão de característica não pertinentes ao trabalho.

> Visão computacional é o estudo da extração de informação de uma imagem; mais especificamente, é a construção de descrições explícitas e claras dos objetos em uma imagem (RIOS, [20-?] apud Ballard and Brown, 1982). Difere do processamento de imagens porque, enquanto ele se trata apenas da transformação de imagens em outras imagens, ela trata explicitamente da obtenção e

> manipulação dos dados de uma imagem e do uso deles para diferentes propósitos (RIOS, [201-?]).

A extração de características relevantes de imagens no processo de seleção do morango se faz relevante quando se trata da necessidade do aumento da velocidade de processamento. Desta forma, pode-se aprimorar o funcionamento do protótipo afim de realizar uma seleção de morangos mais precisa e rápida.

2.4.1 OpenCV

OpenCV (*Open Source Computer Vision Library* ou em tradução livre Biblioteca de visão computacional de código aberto) é uma biblioteca de licença livre, disponível em C++, Python e Java, com suporte para Windows, Linux, Mac OS, IOS e Android. OpenCV foi desenvolvida para aplicação em tempo real, levando em consideração a alta eficiência computacional. Escrita em C/C++, a biblioteca tem uma grande vantagem de processamento quando utilizada em sistema com vários núcleos de processamentos (multi-cores) (OPENCV, [201-?]).

OpenCV tem a finalidade de tratar as imagens coletados em tempo real, destacando detalhes primordiais para as RNAs, como cores e bordas do fruto. Com os dados relevantes destacados, a RNA possui uma precisão maior quando requisitada em processo continuo de seleção.

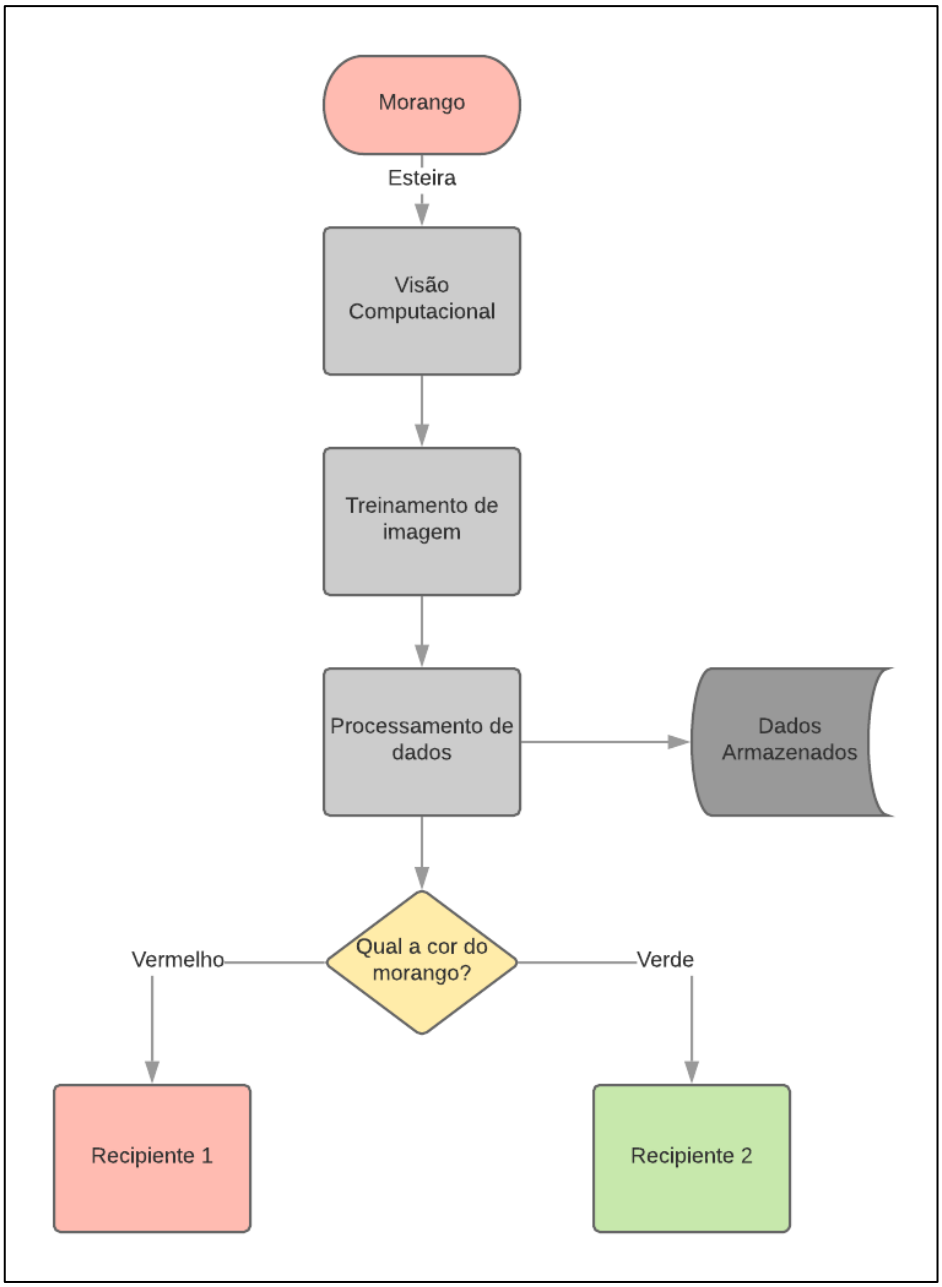
Com o openCV é possível realizar a plotagem em tempo real da imagem coletada pela câmera, identificar através de quadrados de marcação que tipo de morango estamos utilizando e ainda escrever na tela plotada a precisão daquela identificação.

2.5 SOLUÇÃO

Vistos os itens acima, os autores propõem a utilização de um sistema utilizando inteligência artificial com enfoque em *Machine Learning* e visão computacional, estabelecendo o fluxo de processos exemplificado na Figura 7.

Será utilizado um sistema composto por: computador, webcam convencional, servo motor para seleção, e esteira com motor de passo para deslocamento do fruto.

Figura 7 - Fluxo de processos solução para seleção de morangos.



Fonte: Os autores.

3. DESENVOLVIMENTO DO PROTÓTIPO

Afim de realizar o desenvolvimento do protótipo seletor de morango, foi possível a divisão do procedimento em subáreas, sendo assim apresentados neste capítulo.

3.1 PREPARAÇÃO DO *DATASET*

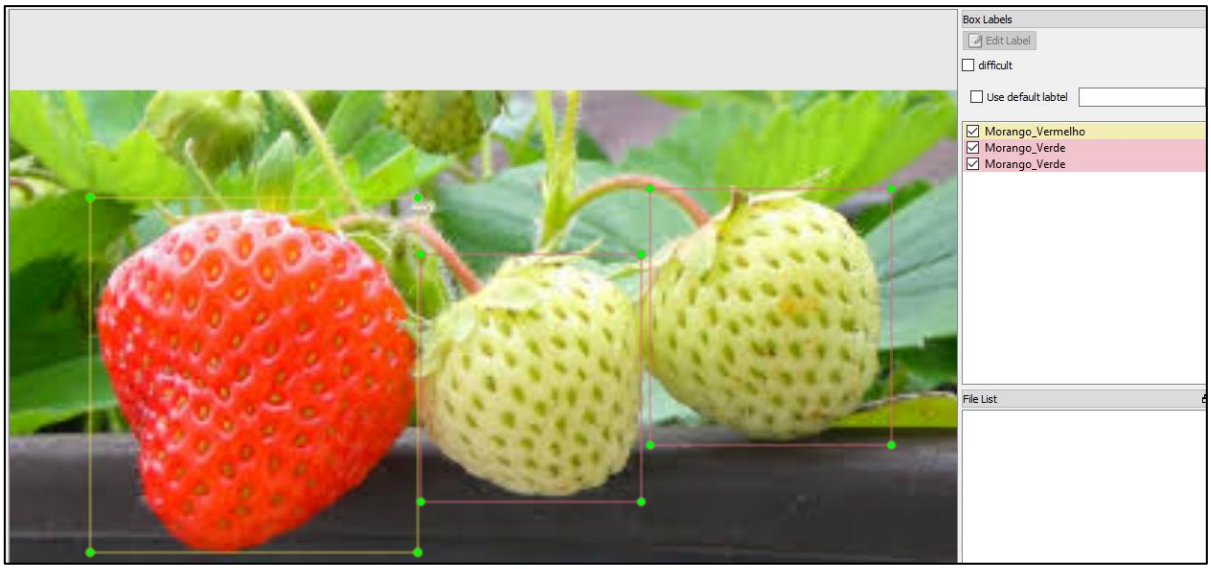
3.1.1 Escolha das imagens

Para a criação do *dataset* de imagens de morangos, os autores providenciaram fotos de morangos tiradas de vários ângulos e com vários fundos e de fotos retiradas da internet. As fotos foram escolhidas de forma a obter um balanço entre morangos vermelhos e verdes, obtendo assim uma quantidade semelhante de imagens para cada característica. Com um banco de 372 imagens, há fotos contendo morangos de cores diferentes assim como imagens com apenas um morango isolado.

3.1.2 Etiquetando as imagens

Precisa-se indicar manualmente quais morangos correspondem a qual característica, ou seja, etiqueta-los. Através do software Labellmg as fotos foram etiquetadas, este software escrito na linguagem Python é gratuito e pode ser usado para qualquer fim. O software em questão exporta um arquivo contendo as informações das fotos que posteriormente foram utilizados para o treinamento da rede. Na Figura 8 encontra-se um exemplo de aplicação do programa em etiquetar os morangos devido a suas características.

Figura 8 - Definindo as características dos morangos



Fonte: Os autores.

3.2 ELETRÔNICA

3.2.1 Arduino

Com origem em 2005 na Itália, o Arduino (Figura 9) é um microcontrolador que teve como objetivo ser um dispositivo acessível, funcional e fácil de ser programado. Seu hardware foi pensado para ser livre, onde qualquer um pode modificar, personalizar e melhorar. (WHAT IS ARDUINO?, [201-?]).

Apenas com um cabo, a placa pode ser conectada a um computador e programada sem dificuldades, podendo controlar diversos equipamentos com suas entradas e saídas digitais e analógicas.

O Arduino foi utilizado para o controle do motor de passo usado na esteira do protótipo e para controlar o servo motor utilizado para realizar a seleção dos morangos.

Figura 9 – Arduino.

Fonte: Arduino, 2019.

3.2.2 Motor de passo

O motor de passo é um motor elétrico que se desloca com ângulos discretos através de um controle digital. Possui boa repetibilidade e não necessita de realimentação para controle, porém isso pode acarretar erros em seus movimentos.

O motor é composto por um rotor onde são fixados os imãs responsáveis pelo movimento, e no estator encontras as bobinas, e essa relação é de maior número para os imãs. Esse fator faz com que quando a bobina seja energizada, esta atraia o imã mais próximo, fazendo com que o rotor gire. O movimento sincronizado de acionamento das bobinas faz com que o motor se mova constantemente.

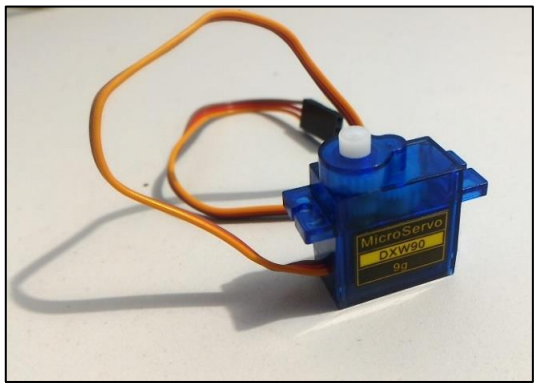
Através do drive ULN2003 o Arduino comanda o motor de passo que por sua vez é utilizado para dar movimento a esteira do protótipo.

3.2.3 Servo motor

O servo motor é parecido com um motor de passo, porém este possui um sistema de controle de realimentação, podendo assim saber exatamente sua posição.

Na Figura 10 tem-se o servo motor utilizado, do modelo Tower Pro 9g, o motor possui baixo custo sendo ideal para protótipos.

Figura 10 - Servo motor.



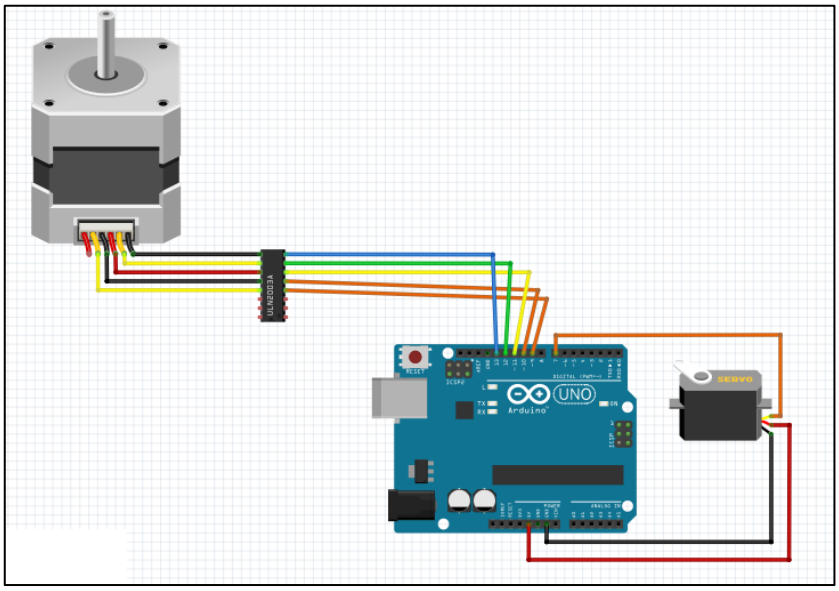
Fonte: Os autores.

Este tipo de motor foi aplicado para o controle do posicionamento do mecanismo que seleciona os morangos, podendo assim ter uma maior confiabilidade.

3.2.4 Esquemático elétrico

Na Figura 11 tem-se esquemático da montagem eletrônica, pode-se observar como foi feita as ligações dos motores ao Arduino. Para a realização das ligações foi utilizado fios de conexão rápida para protótipos (*jumpers*) coloridos, de forma a facilitar o acompanhamento das ligações e possíveis alterações.

Figura 11 – Esquemático elétrico.



Fonte: Os autores.

3.2.5 Custo do protótipo

De maneira a manter o baixo custado atrelado ao projeto, utilizou-se de componentes de prototipagem, como servo motor e motor de passo simples. Para aplicação em escala realmente utilizável se faz necessário a utilização de componentes com maior robustez, de forma que o ambiente agrícola não seja um empecilho para o funcionamento.

No Quadro 4 estão apresentados os custos por itens, sem levar em consideração o computador utilizado para o processamento da inteligência artificial.

Quadro 4 - Custo de componentes utilizados no protótipo.

Custo de componentes	
Item	Valor
Arduino	R$35,00
Esteira	R$159,75
Motor de passo + drive	R$23,00
Servo motor	R$17,00
Caixa para Arduino	R$20,00
Webcam	R$52,50
Total	**R$307,25**

Fonte: Os autores.

Para tanto, em projetos futuros e otimizados, pode-se adaptar o uso da inteligência artificial a um computador de baixo custo, como um Raspberry PI, que possui certa robustez e bom processamento de dados com um baixo custo.

Raspberry PI é um computador de pequeno porte e baixo custo, com dimensões de 9x6 cm aproximadamente e foi desenvolvido no Reino Unido pela fundação Raspberry PI com intuito de difundir o aprendizado em programação para criança e locais com baixo nível econômico.

3.3 INTELIGÊNCIA ARTIFICIAL

3.3.1 Instalação

Para que fosse possível a utilização de bibliotecas com o TensorFlow, openCV e da linguagem de programação Python, foi necessário a instalação de alguns recursos no computador utilizado para o desenvolvimento e execução da aplicação.

Foi utilizado a biblioteca TensorFlow GPU, que faz a utilização da placa gráfica do computador para o treinamento da rede neural utilizada neste trabalho. Desta forma, o tempo de treinamento é reduzido a 30%, quando comparado ao treinamento de uma rede neural, utilizando o TensorFlow CPU, que utiliza o processador convencional do computador para executar o treinamento da rede.

Foi necessário também a instalação do pacote CUDA ToolKit, software produzido e disponibilizado gratuitamente pela NVIDIA, para o gerenciamento de placa de vídeo NVIDIA, permitindo a otimização e paralelismo para que estas placas possam ser utilizadas para fins de *data Science, machine learning e deep learning.*

Também se fez necessário a instalação das bibliotecas Keras e Pandas, ambas de licença livre, utilizadas para facilitar o desenvolvimento de redes neurais com TensorFlow e manipulação de dados, respectivamente.

3.3.2 Treinamento da rede neural

Para o treinamento da rede neural, foi utilizado os dados gerados e apresentados no item 3.1 deste trabalho.

3.3.2.1 Criação do mapa de etiquetas

Na etapa apresentada no item 3.1.2, criou-se etiquetas para as imagens utilizadas no treinamento da rede neural. Nesta etapa do processo, se fez necessário passar a informação de qual é o nome e o número que cada etiqueta representara durante nosso treinamento supervisionado da rede.

Foi definido que, o número de identificação 1, corresponde aos morangos vermelhos. E número de identificação 2 corresponde aos morangos verdes, dados presentes no Quadro 5.

Quadro 5 - Mapa de Etiquetas

Mapa de Etiquetas	
Id	Cor
1	Vermelho
2	Verde

Fonte: Os autores.

3.3.2.2 Configurações de treino

Mediante biblioteca utilizada, fez-se necessário a atualização de parâmetros de treinamento no código da rede neural, afim de que está esteja apta a ser treinada para as duas classes de imagens que iremos realizar a detecção.

Parâmetros como coeficiente de aprendizado, número máximo de épocas, erro mínimo de treinamento mediante *set point* foram modificados empiricamente, para que o grupo pudesse realizar o treinamento e analisar os resultados. Mediantes isto, fazer possíveis alterações em parâmetros pertinentes afim de melhorar a precisão da detecção dos frutos.

3.3.2.3 Execução do treinamento

Para início do treinamento, as imagens previamente etiquetadas são divididas em duas categorias: Imagens para teste e imagens para treinamento. As imagens coletadas e classificadas manualmente no Item 3.1, são divididas na seguinte proporção, como apresentado no Quadro 6.

Quadro 6 - Categoria de Imagens

Divisão de Imagens	
Proporção (%)	Categoria
77	Treinamento
23	Teste

Fonte: Os autores.

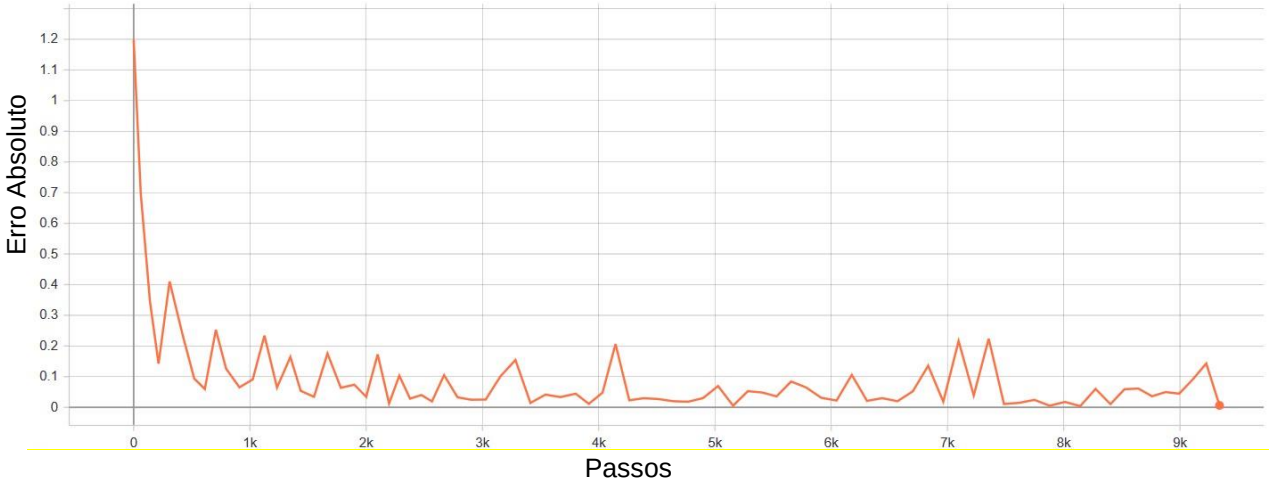
Após realizada as configurações pertinentes nos códigos de treinamento, sua execução é iniciada. Durante o treinamento a rede neural programada, a todo momento classifica, analisa o erro e pondera os pesos de forma que, o resultado seja o mais próximo do *setpoint*. O *setpoint* a cada interação são imagens de treinamento. Todas a imagens de treinamento são submetidas a rede, e os pesos de entre os neurônios são calculados, após este passo, são apresentadas a imagens de treinamento para a rede neural, e está, retorna a resposta pertinente a cada imagem.

Sendo assim, a resposta obtida é analisada com o real resultado de cada imagem de teste, e o erro é calculado. A partir deste ponto, com o erro apresentado, os pesos de ligação dos neurônios são recalculados de forma automática, de forma que na interação seguinte o erro diminua. Este procedimento segue o conceito da Regra Delta para RNA apresentado no item 2.3.2, e na Figura 5.

O treinamento como um todo, requer alto poder de processamento computacional e tempo relativo de treinamento alto, sendo assim, o grupo optou por realizar as interações até que o erro associado ao treinamento estivesse em torno de 5%. Desta forma, o processamento da seleção dos morangos não seria afetado e seria possível a otimização no tempo de treinamento da rede. Redes convencionais utilizados para os mais diversos fins, segundo estudo levantado, levam em torno de 10 horas de processamento, com utilização de computadores dedicados a este fim.

Na Figura 12 é possível observar o erro associado ao treinamento.

Figura 12 - Erro durante o treinamento.



Fonte: Os autores.

O treinamento realizado, contou com a duração de 2h40min, utilizando de um computador portátil convencional. As interações possuíram 9339 passos, ou seja, o algoritmo de inteligência artificial realizou a varredura e recalculo dos pesos associados por 9339 vez, até que o erro de detecção estivesse tolerável.

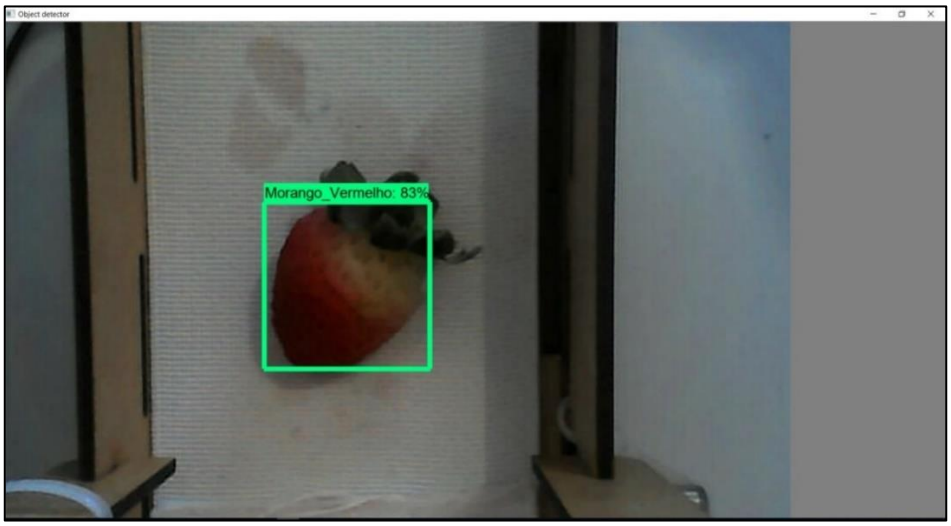
Ao final do processo, o erro obtido foi de aproximadamente 0,0630 ou seja, 6,30%.

3.3.3 Funcionamento

Para o início da classificação dos morangos executou-se o software desenvolvido para este trabalho. Ao inicializar-se a classificação, o dispositivo de detecção de imagem capta o fluxo de morangos passantes pela esteira, e envia para a rede neural previamente treinada a classificar morangos verdes e vermelhos. A partir deste ponto, a rede neural é responsável por apresentar a classe pertinente ao fruto a ser detectado e também o grau de precisão desta detecção.

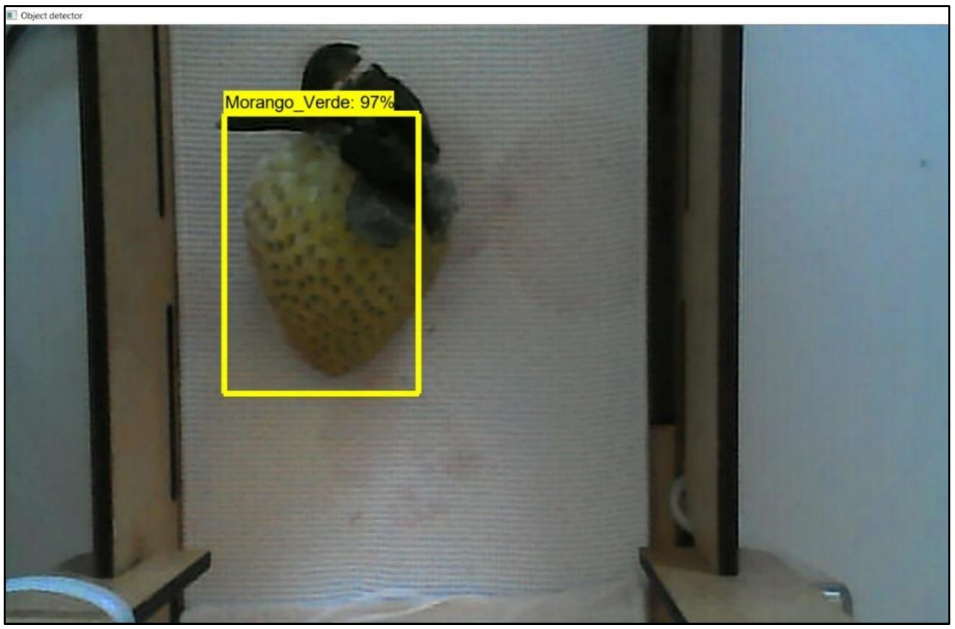
Nas Figura 13 e Figura 14 é possível observar a tela de visualização do processo, que mostra em tempo real durante a classificação, qual o tipo de morango detectado (verde ou vermelho), identifica com um retângulo a fruta presente na imagem e também apresenta a precisão daquela detecção em porcentagem.

Figura 13 - Tela de visualização com morango vermelho.



Fonte: Os autores

Figura 14 - Tela de visualização com morango verde.



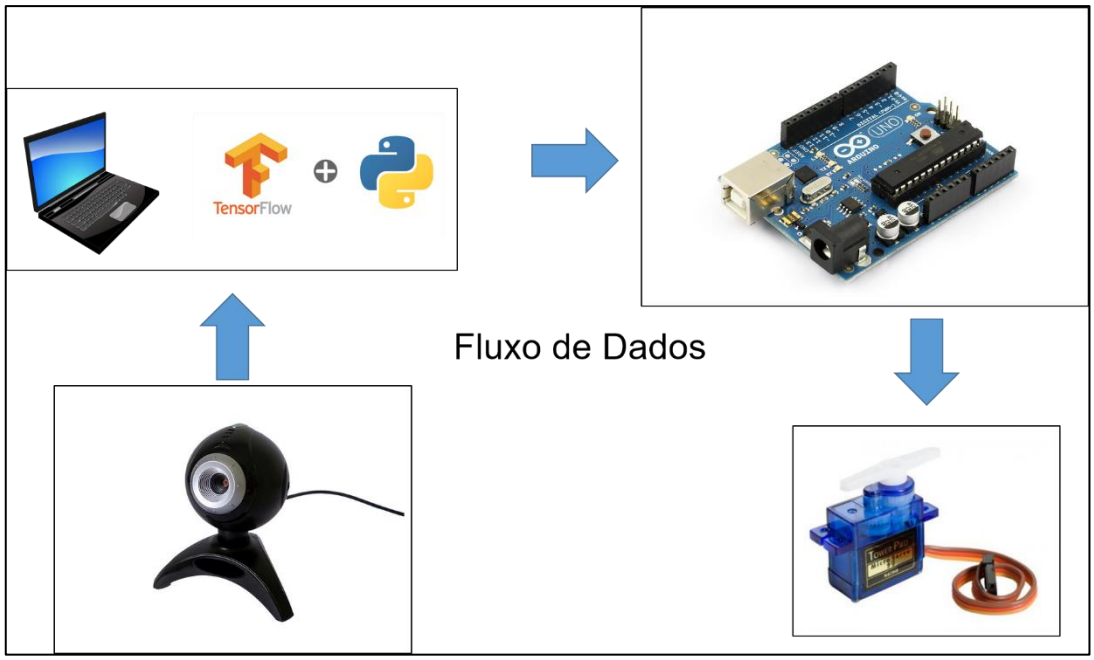
Fonte: Os autores

3.3.4 Comunicação com placa de atuação

Para realizar comandos externos, como do motor de passo e servo motor, utilizou-se os recursos presente no item 3.2. Desta forma, foi possível realizar a tomada de decisão calculada pela inteligência artificial, que foi executada no computador, e o meio físico.

Para a comunicação, utilizou-se do protocolo de comunicação serial RS232 e da biblioteca Serial, para Python e Arduino. Desta forma, foi possível o envio em tempo real dos comandos informados pela rede neural para o Arduino, e este por sua vez, executou o comando nos motores do seletor de morango. Na Figura 15 é possível observar a ilustração do fluxo de dados entre a *webcam*, computador com a inteligência artificial, o Arduino e o servo motor presente na esteira classificadora.

Figura 15 - Fluxo de Dados



Fonte: Os autores

3.4 O DISPOSITIVO

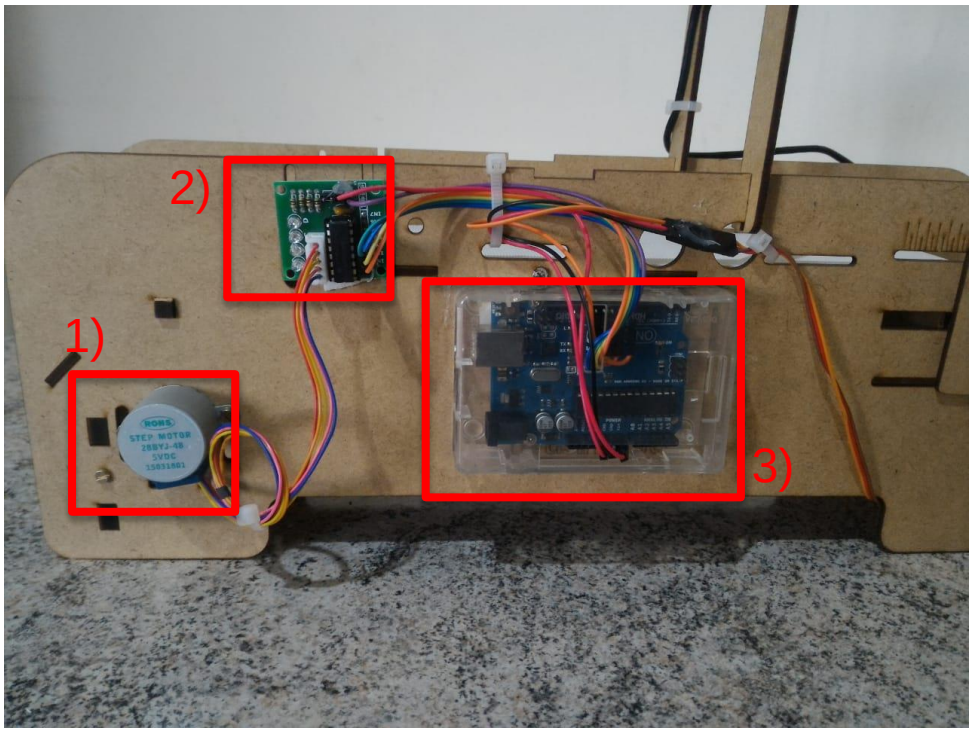
O conceito de prototipagem visa a concepção de funções que permitam a realização e execução das funções básicas proposta para um produto. Visto isto, a equipe optou pela utilização de uma esteira comercial, para que os recursos desenvolvidos até agora pudessem ser instalados, e também para que facilitasse os testes em campo, como em visitas a produtores de morango.

A esteira utilizada é de padrão comercial, vendida pela internet para realização de projeto experimentais. Sua construção é composta por peças de MDF recortada a laser. Rolos feitos de tubos de PVC com rolamentos e esteira construído com elástico.

Na Figura 16, tem-se a vista lateral do protótipo seletor de morango, sendo os componentes destacados:

1) Motor de passo, responsável pelo movimento continuo da esteira.
2) Drive do motor de passo, responsável por receber os comandos do Arduino e atuar sobre o motor de passo.
3) Arduino, responsável por receber os comandos da inteligência artificial e atuar sobre os periféricos.

Figura 16 - Vista lateral protótipo.

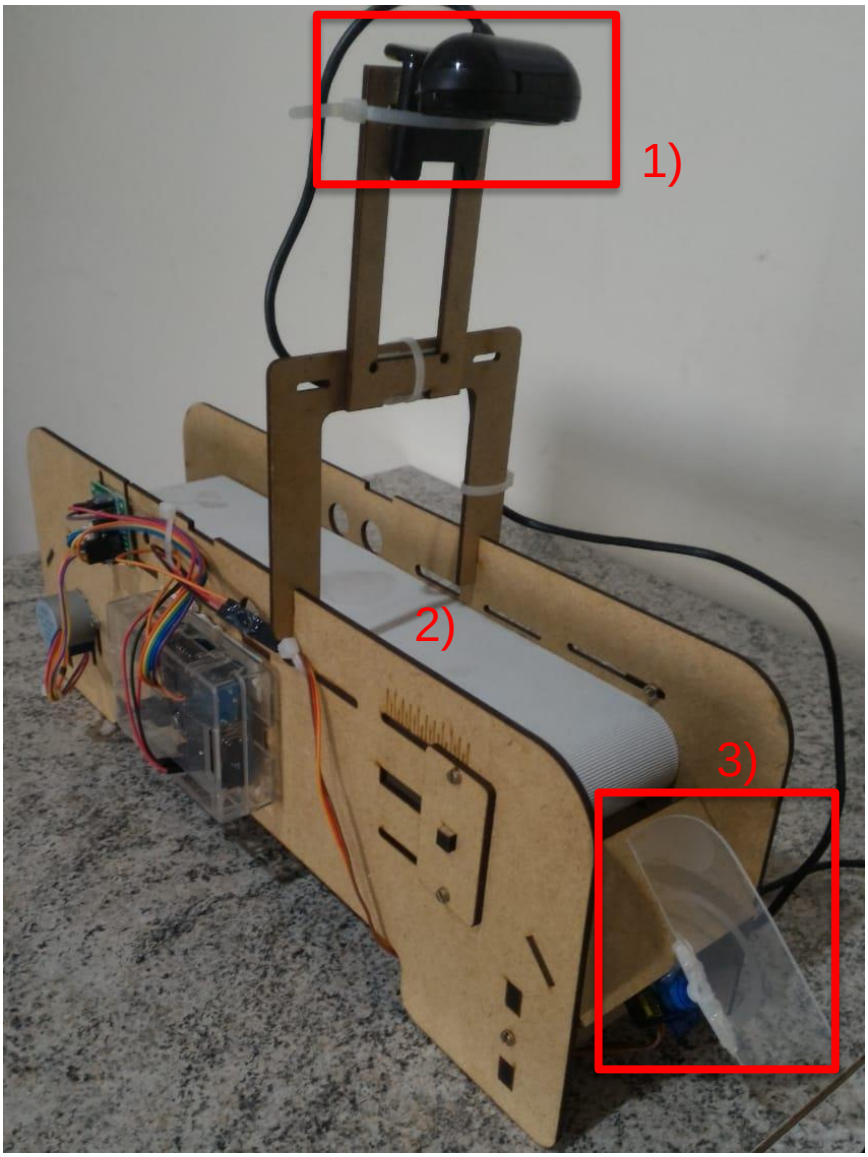


Fonte: Os autores

Na Figura 17, tem-se o protótipo completo, sendo possível observar:

1) A *webcam*, responsável pela captura da imagem dos morangos e envio para a inteligência artificial.
2) A esteira, ao centro na cor branca, que é responsável pelo movimento dos morangos ao longo do processo de seleção.
3) Servo motor com pá, responsável por direcionar os morangos para a correta posição (verde ou maduro).

Figura 17 - Protótipo.



Fonte: Os autores

4. TESTES E RESULTADOS

Para a realização da medição da eficiência do protótipo construído, dividiu-se a etapa de teste em acurácia e velocidade na seleção. Acurácia tem por finalidade medir o nível de precisão do dispositivo durante seu funcionamento e a medição da velocidade analisar a quantidade de morangos que o protótipo é capaz de selecionar.

4.1 ACURÁCIA NA SELEÇÃO

Realizou-se testes com o dispositivo para o cálculo de erros e qualidade na seleção dos morangos.

Para estes testes, foi utilizada a seguinte metodologia: Os morangos, independentemente de sua qualidade (verde ou maduro) foram colocados 1 a 1 na esteira, de forma que a *webcam* detectasse apenas 1 fruto por vez. A ordem de colocação dos morangos na esteira foi realizada em ordem aleatória, ou seja: foi disposto morangos verdes e vermelhos intercalados, verdes seguindo de outros verdes ou vermelhos seguindo de outros vermelhos.

Para os testes, foram utilizados 79 morangos, sendo estes distribuídos na seguinte proporção: 46 verdes e 33 vermelhos.

No Quadro 7 é possível observar o resultado do teste. Pode-se observar uma boa precisão na seleção de morangos, sendo que durante o teste realizado, apenas 1 morango, da cor verde foi selecionado como morango vermelho. Trazendo uma acurácia de 98,73% para o seletor de morangos.

Quadro 7 – Acurácia na seleção.

	Testes de Seleção dos Morangos
Correto	78
Incorreto	1
Total	79
Acurácia	98,73%

Fonte: Os autores

Na Figura 18 tem-se os morangos verdes utilizados para o teste de acurácia da seleção. Todos estavam congelados e com folhas, não sendo um problema para a seleção dos frutos.

Figura 18 - Morangos verdes.

Fonte: Os autores

Na Figura 19 tem-se os morangos vermelhos utilizados durante a seleção, todos congelados e alguns com folhas, outros sem folhas. Em destaque, é possível observar o morango verde, que de forma errônea foi classificado com um fruto maduro.

Figura 19 - Morangos vermelhos.

Fonte: Os autores

4.2 VELOCIDADE NA SELEÇÃO

A velocidade de seleção também é um fator importante quando se trata de seleção de morangos, seja devido a perecibilidade do fruto ou pela quantidade da produção.

Sendo assim foi realizado o levantamento do tempo mínimo que a esteira leva para a detecção dos frutos. A seguinte metodologia foi utilizada: Os frutos foram colocados na esteira até que a mesma esteja cheia, com os espaçamentos necessários entre um fruto e outro, de forma que exista sempre apenas 1 fruto na visada da *webcam*. O tempo então foi cronometrado e o resultado encontra-se disponível no Quadro 8. Os resultados para 360 e 8460 frutos foram inferidos a partir do primeiro.

Quadro 8 - Capacidade de seleção de morangos.

Capacidade de seleção		
Quantidade	Tempo	Massa (kg)
1	10s	0,024
360	1h	8,64
8640	24h	207,36

Fonte: Os autores

Em visita ao produtor de morangos Ozanan Martins, foi realizado o levantamento de sua produção diária de morangos, afim de estabelecer o tempo necessário de funcionamento do protótipo para a seleção de sua colheita.

A produção de morangos possui 2 fases distintas durante o ano, fase de alta e baixa produção. A fase de alta acontece durante a primavera e durante o restante do ano acontece a fase de baixa produção. No Quadro 9 foi realizado o levantamento do tempo necessário para seleção e da quantidade de frutos produzido ao dia em cada período.

Quadro 9 - Colheita e tempo para seleção dos morangos.

	Colheita diária	Tempo para seleção
Alta produção	500kg	58h
Baixa produção	40kg	5h40min

Fonte: Os autores

Sendo possível observar a funcionalidade do dispositivo em relação a uma produção real de morango de pequeno porte.

5. CONCLUSÕES

5.1 CONSIDERAÇÕES FINAIS

Apesar de se trabalhar com imagens e o treinamento de uma inteligência artificial, os resultados obtidos foram melhores do que os esperados pelo grupo. Com uma eficiência de 98,7%, pode ser comprovada a capacidade de o protótipo gerado poder selecionar morangos com um erro aceitável.

As dificuldades atreladas a criação do protótipo estiveram presentes no treinamento da rede neural, desde a obtenção de um bom banco de dados com imagens suficientes para um aprendizado adequado da I.A e também na complexidade do algoritmo utilizado para este aprendizado.

O protótipo elaborado possui um motor que entrega uma velocidade baixa ao processo, como visto, na velocidade atual, para uma colheita de morangos em um dia a máquina levaria horas para seleção quando em baixa produção, quando em alta produção, uma máquina apenas não seria capaz de atender a demanda. A velocidade está atrelada principalmente ao motor que comanda a esteira, porém com uma velocidade maior o sistema teria que passar por certas adequações. A câmera utilizada para a identificação do fruto não é a de melhor qualidade e é capaz de filmar apenas uma face do morango, logo para uma maior velocidade da esteira, a câmera deveria ser substituída.

Uma grande barreira é o tempo em que a fruta leva desde sua identificação até sua seleção. Caso várias frutas entrem no campo de visão da câmera, o sistema ira manipular a pá atuadora de posição levando assim a possíveis erros onde um morango vermelho poderia cair em um recipiente de morango verde. Para reduzir tais efeitos, um sistema de buffer de informação juntamente com um sensor para identificar que o morango foi realmente selecionado, pode ser adicionado ao projeto, podendo entregar maior confiabilidade no processo e também maior rapidez pois os morangos poderiam ser alocados na esteira de forma continua eliminando assim grandes lacunas.

Para aplicação deste protótipo um ponto muito importante deve ser levado em consideração, o processamento da imagem junto da inteligência artificial usada. O algoritmo exige alto processamento, tornando assim impossível a utilização de micro

controladores com baixo processamento, fazendo então necessário o uso de computador ou, como citado, Raspberry Pi.

Com todos os desafios encontrados ao longo deste trabalho, a aplicação do conhecimento para a geração de um protótipo que possui uma aplicabilidade em um ramo tão importante para o Brasil, que é o da agricultura, gerou um grande aprendizado e um olhar crítico em quais tecnologias tem-se hoje para produtores de pequeno porte, tendo então uma possibilidade de atuação em uma área diferente da usual, que seriam as indústrias.

5.2 SUGESTÕES DE TRABALHO FUTURO

Para sugestão de trabalhos posteriores, os autores indicam alguns pontos de estudos e melhorias ao longo deste trabalho:

- Otimização de algoritmo para o processamento mais rápido na seleção que exija menor processamento.
- Aumento da velocidade de seleção dos frutos, que está ligada diretamente a otimização do algoritmo de seleção e ao motor utilizado na esteira.
- A utilização de várias câmeras, identificando todas as faces do fruto, para aumentar a confiabilidade na seleção da fruta.
- E um atuador que consiga selecionar mais que um morango por vez, podendo assim alocar diversos frutos na esteira e não se preocupar em deixar todos enfileirados ou com lacunas.

REFERÊNCIAS

AGRONEGÓCIO BRASILEIRO. **Você conhece o agronegócio brasileiro?** Disponível em: <https://g1.globo.com/economia/agronegocios/agro-a-industria-riqueza-do-brasil/noticia/voce-conhece-o-agronegocio-brasileiro.ghtml>. Acessado em 19 set. 2018.

Balança Comercial Resumida. AgroStat Brasil. Setembro de 2015. Disponível em: <https://www.udop.com.br/download/estatistica/agronegocio/set15_balanca_comercial_resumida_mapa.pdf> Acessado em: 19 jun. 2019.

BRASIL PERDEU ESPAÇO no mercado agrícola mundial, afirma OMC. Estadão conteúdo, 17 jul. 2017. Disponível em: <https://g1.globo.com/economia/agronegocios/noticia/brasil-perdeu-espaco-no-mercado-agricola-mundial-afirma-omc.ghtml>. Acesso em: 17 set. 2018.

BUAINAIN, A.M.; Alves, E.; Silveira, J.M. e Navarro, Z., 2014 - **O mundo rural no Brasil do século 21. A formação de um novo padrão agrário e agrícola**. Brasília, Embrapa/Instituto de Economia da Unicamp. Disponível em: <https://www.embrapa.br/busca-de-publicacoes/-/publicacao/994073/o-mundo-rural-no-brasil-do-seculo-21-a-formacao-de-um-novo-padrao-agrario-e-agricola>. Acessado em: 16 set. 2018.

Cambridge online dictionary. Publicado em 23 de abril de 2008. Disponível em: < https://dictionary.cambridge.org/us/dictionary/english/>. Acessado em: 19 jun. 2019.

CEPEA, **PIB do agronegócio brasileiro**. Disponível em: <https://www.cepea.esalq.usp.br/br/pib-do-agronegocio-brasileiro.aspx>. Acessado em: 19 set. 2018.

DATASETS – CONJUNTO DE DADOS, 8 mai. 2017. Disponível em: <http://artificiencia.com/aprenda/dataset/> Acessado em: 16/11/2018.

DE OLIVEIRA, R. P. Apoio à Decisão na Adoção da Agricultura de Precisão: A Tecnologia da Informação em Apoio ao Conhecimento Agronômico. **RECoDAF – Revista Eletrônica Competências Digitais para Agricultura Familiar**, Tupã, v. 2, n. 1, p. 89-109, jan./jun. 2016. ISSN: 2448-0452

GAZZONI, Decio Luiz. Como alimentar 10 bilhões de cidadãos na década de 2050.**Cienc. Cult.** São Paulo , v. 69, n. 4, p. 33-38, out. 2017 . Disponível em: <http://cienciaecultura.bvs.br/scielo.php?script=sci_arttext&pid=S0009-67252017000400012&lng=en&nrm=iso>. Acessado em 16, set. 2018.

INAMASU, Ricardo Yassushi. Agricultura de Precisão. **Livro Agricultura de Precisão: Resultados de um Novo Olhar**. /Alberto Carlos de Campos Bernardi, [et al], editores técnicos. - Brasília, DF: Embrapa, 2014. 21-33 p. ISBN 978-7035-352-8. Disponível em: <https://www.macroprograma1.cnptia.embrapa.br/redeap2>. Acessado em: 16 set. 2018.

IPEA - Instituto de Pesquisa Econômica Aplicada. (2017). **Agricultura - Agricultura em família.** Disponível em: <http://www.ipea.gov.br/desafios/index.php?option=com_content&id=2512:catid=28&Itemid=23>. Acessado em: 10, mai. 2018.

LIMA. **Lavoura são apenas 7,6% do Brasil, segundo a NASA**. 28 dez. 2017. Disponível em: <http://www.agricultura.gov.br/noticias/dados-da-nasa-demonstram-que-apenas-7-6-da-area-do-brasil-e-ocupada-por-lavouras>. Acessado em: 18 set. 2018.

MAPA, SRI, DAC, CGEC. **Balança comercial brasileira e balança comercial do agronegócio: 1997 a 2017**. Disponível em: <http://www.agricultura.gov.br/assuntos/relacoes-internacionais/documentos/estatisticas-do-agronegocio/SERIEHISTORICABCARESUMIDA19972017.xls>. Acessado em: 15 set. 2018.

MONARD, Maria Carolina, BARANAUSKAS, José Augusto. Conceitos Sobre Aprendizado de Máquina. **Sistemas Inteligentes Fundamentos e Aplicações.** 1 ed. Barueri-SP: Manole Ltda, 2003. p. 89--114. ISBN 85-204-168.

OPENCV, [201-?] Disponível em <https://opencv.org/> acessado em: 19/11/2018

Serviço Brasileiro de Apoio às Micro e Pequenas Empresas. Conheça o Sebrae. Sebrae. Disponível em: <http://www.sebrae.com.br/sites/PortalSebrae/canais_adicionais/conheca_quemsomo
s.> Acessado em 16, nov. 18.

PERSPECTIVAS PARA A PRODUÇÃO DE MORANGOS no sul do Brasil: uma revisão sobre os sistemas de produção e as práticas de manejo, 2016. <http://trabalhos.congrega.urcamp.edu.br/index.php/jpgp/article/view/1143/183>, acessado em 19/11/2018

REDES NEURAIS ARTIFICIAIS, ICMC USP, [201-?]. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/> acessado em: 19/11/2018

Revista Eletrônica Competências Digitais para Agricultura Familiar, [S.l.], v. 2, n. 1, p. 89-109, nov. 2016. ISSN 2448-0452. Disponível em: <http://codaf.tupa.unesp.br:8082/index.php/recodaf/article/view/28>. Acessado em: 16 set. 2018.

ROCHA, Joaquim Eloir. **Acionamento_08_motor_de_passo**. Disponível em: < http://paginapessoal.utfpr.edu.br/joaquimrocha/acionamento-eletronico-de-maquinas-eletricas/Acionamento_08_Motor_de_Passo.pdf/view>. Acessado em: 09 jun. 2019.

Russell, Stuart J. (Stuart Jonathan), 1962 - **Inteligência artificial** / Stuart Russel, Peter Norvig; tradução Regina Célia Simille. - Rio de Janeiro: Elsevier, 2013. ISBN 978-85-352-3701-6

SANCHES, J.; LINO, A.C.L. **Uso de imagem digital para seleção e classificação de frutas e hortaliças.** 2010. Artigo em Hypertexto. Disponível em: <http://www.infobibos.com/Artigos/2010_1/imagem/index.htm>. Acessado em: 10, mai. 2018.

SNA/RJ. (2015). **Produtores de frutas precisam se aproximar mais dos consumidores.** Disponível em: <http://www.sna.agr.br/produtores-de-frutas-precisam-se-aproximar-mais-dos-consumidores/>. Acessado em: 10, mai. 2018.

SORJ, B. **Estado e classes sociais na agricultura brasileira** [online]. Rio de Janeiro: Centro Edelstein de Pesquisas Sociais, 2008. 135 p. ISBN: 978-85-9966-228-1. SciELO Books <http://books.scielo.org>.

TROMBINI, C. L. A. et al. **Mecanização da Agricultura: Oportunidade de Desenvolvimento para o Pequeno Produtor Rural ou Instrumento Reservado a Produção Capitalista.** Universitári@ - Revista Científica do Unisalesiano. Ano 3., N.7, Lins. jul./dez., 2012.

What is arduino?. Disponível em: < https://www.arduino.cc/en/Guide/Introduction>. Acessado em: 09 jun. 2019.

APÊNDICE

Os códigos disponíveis neste apêndice foram utilizados durante o desenvolvimento do protótipo e desenvolvidos, adaptados ou reutilizados pela equipe sobre a licença indica onde seja necessário.

//Código utilizado no Arduino

```
#include <Servo.h>
#include <Stepper.h>

#define SERVO 7 // Porta Digital 6 PWM

Servo s; // Variável Servo
int  graus = 90;
const int stepsPerRevolution = 500;
Stepper myStepper(stepsPerRevolution, 8, 10, 9, 11);//Definição de saídas do motor de passo.
void setup() {
        Serial.begin(9600); //inicia comunicação serial.
         s.attach(SERVO); //inicia o servo motor.
         s.write(90); //Comando o servo motor para posição inicial.
         myStepper.setSpeed(70); //Comando velocidade motor de passo.
}
void loop() {
        char leitura = Serial.read(); //Lê as informações enviadas pela inteligência artificial.
         myStepper.step(-1); // Comando o motor de passo a rodar.
  if(leitura == '1'){
        s.write(42); // Caso leitura serial receber 1, o morango é vermelho e a posição do servo motor vai para 42º.
  }
  else if (leitura == '2') {
        s.write(142); // Caso leitura serial receber 2, o morango é verde e a posição do servo motor vai para 142º.
  }
}
```

#Código inicialização de seleção de morangos em Python

```
# Copyright 2017 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Alterado por: Alex Torezin Mendonça e Geovani Torezin Mendonça, 2019.
#
================================================================================

# Import packages
import os
import cv2
import numpy as np
import tensorflow as tf
import sys
import serial

# This is needed since the notebook is stored in the object_detection folder.
# Necessário
sys.path.append("..")

# Importação de trechos de códigos utilizados e configuração
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'inference_graph'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')
```

```
# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')

# Number of classes the object detector can identify
NUM_CLASSES = 2

## Load the label map.
# Label maps map indices to category names, so that when our convolution
# network predicts `5`, we know that this corresponds to `king`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)


# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Initialize webcam feed
video = cv2.VideoCapture(0)
```

```
ret = video.set(3,1280)
ret = video.set(4,720)

#Define parametros comunicação serial
porta = "COM5"
velocidade = 9600
conexao = serial.Serial(porta,velocidade);
opcao = 0;
ii=0

while(True):

   # Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]
   # i.e. a single-column array, where each item in the column has the pixel RGB value
   ret, frame = video.read()
   frame_expanded = np.expand_dims(frame, axis=0)

   # Perform the actual detection by running the model with the image as input
   (boxes, scores, classes, num) = sess.run(
      [detection_boxes, detection_scores, detection_classes, num_detections],
      feed_dict={image_tensor: frame_expanded})

   # Draw the results of the detection (aka 'visulaize the results')
   vis_util.visualize_boxes_and_labels_on_image_array(
      frame,
      np.squeeze(boxes),
      np.squeeze(classes).astype(np.int32),
      np.squeeze(scores),
      category_index,
      use_normalized_coordinates=True,
      line_thickness=8,
      min_score_thresh=0.60)

   # All the results have been drawn on the frame, so it's time to display it.
   cv2.imshow('Object detector', frame)

   #Conta quantos itens tem em classe e score
   classes_size  = classes.size
   scores_size  = scores.size

   #print("Deteccao")
   for i in range(0, (classes_size-1)): #envia por serial ao Arduino o morango detectado (1 ou 2, para
vermelho e verde respectivamente).
      if (scores.item(i)>0.8): #Apenas envia comando caso a detecção possua acurácia de no mínimo 80%.

         print(scores.item(i)) #Faz o print das informações
         print(classes.item(i))
```

```
            x=str(classes.item(i))
            conexao.write(str.encode(x)) # Envia a informação.

    # Apertar a Tecla q para parar a detecção
    if cv2.waitKey(1) == ord('q'):
        break

#Fecha a janela de visualização da seleção.
video.release()
cv2.destroyAllWindows()
```

#Mapeamento das etiquetas utilizadas com os morangos

```
item { #Morangos vermelhos possuem código de identificação 1 e tag ‘Morango_Vermelho’
  id: 1
  name: 'Morango_Vermelho'
}

item {#Morangos verdes possuem código de identificação 2 e tag ‘Morango_Verdes’
  id: 2
  name: 'Morango_Verde'
}
```

#Geração das etiquetas apresentadas nas detecções dos morangos

```
# Copyright 2017 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Alterado por: Alex Torezin Mendonça e Geovani Torezin Mendonça, # 2019.
```

```
#
#
==========================================================================
==
"""Label map utility functions."""

import logging

import tensorflow as tf
from google.protobuf import text_format
from object_detection.protos import string_int_label_map_pb2


def _validate_label_map(label_map):
  """Checks if a label map is valid.

  Args:
    label_map: StringIntLabelMap to validate.

  Raises:
    ValueError: if label map is invalid.
  """
  for item in label_map.item:
    if item.id < 0:
      raise ValueError('Label map ids should be >= 0.')
    if (item.id == 0 and item.name != 'background' and
        item.display_name != 'background'):
      raise ValueError('Label map id 0 is reserved for the background label')


def create_category_index(categories):
  """Creates dictionary of COCO compatible categories keyed by category id.

  Args:
    categories: a list of dicts, each of which has the following keys:
      'id': (required) an integer id uniquely identifying this category.
      'name': (required) string representing category name
        e.g., 'cat', 'dog', 'pizza'.

  Returns:
    category_index: a dict containing the same entries as categories, but keyed
      by the 'id' field of each category.
  """
  category_index = {}
  for cat in categories:
    category_index[cat['id']] = cat
  return category_index
```

```
def get_max_label_map_index(label_map):
  """Get maximum index in label map.

  Args:
    label_map: a StringIntLabelMapProto

  Returns:
    an integer
  """
  return max([item.id for item in label_map.item])


def convert_label_map_to_categories(label_map,
                                    max_num_classes,
                                    use_display_name=True):
  """Given label map proto returns categories list compatible with eval.

  This function converts label map proto and returns a list of dicts, each of
  which  has the following keys:
    'id': (required) an integer id uniquely identifying this category.
    'name': (required) string representing category name
      e.g., 'cat', 'dog', 'pizza'.
  We only allow class into the list if its id-label_id_offset is
  between 0 (inclusive) and max_num_classes (exclusive).
  If there are several items mapping to the same id in the label map,
  we will only keep the first one in the categories list.

  Args:
    label_map: a StringIntLabelMapProto or None.  If None, a default categories
      list is created with max_num_classes categories.
    max_num_classes: maximum number of (consecutive) label indices to include.
    use_display_name: (boolean) choose whether to load 'display_name' field as
      category name.  If False or if the display_name field does not exist, uses
      'name' field as category names instead.

  Returns:
    categories: a list of dictionaries representing all possible categories.
  """
  categories = []
  list_of_ids_already_added = []
  if not label_map:
    label_id_offset = 1
    for class_id in range(max_num_classes):
      categories.append({
          'id': class_id + label_id_offset,
          'name': 'category_{}'.format(class_id + label_id_offset)
      })
    return categories
```

```
  for item in label_map.item:
    if not 0 < item.id <= max_num_classes:
      logging.info(
          'Ignore item %d since it falls outside of requested '
          'label range.', item.id)
      continue
    if use_display_name and item.HasField('display_name'):
      name = item.display_name
    else:
      name = item.name
    if item.id not in list_of_ids_already_added:
      list_of_ids_already_added.append(item.id)
      categories.append({'id': item.id, 'name': name})
  return categories


def load_labelmap(path):
  """Loads label map proto.

  Args:
    path: path to StringIntLabelMap proto text file.
  Returns:
    a StringIntLabelMapProto
  """
  with tf.gfile.GFile(path, 'r') as fid:
    label_map_string = fid.read()
    label_map = string_int_label_map_pb2.StringIntLabelMap()
    try:
      text_format.Merge(label_map_string, label_map)
    except text_format.ParseError:
      label_map.ParseFromString(label_map_string)
  _validate_label_map(label_map)
  return label_map


def get_label_map_dict(label_map_path,
                       use_display_name=False,
                       fill_in_gaps_and_background=False):
  """Reads a label map and returns a dictionary of label names to id.

  Args:
    label_map_path: path to StringIntLabelMap proto text file.
    use_display_name: whether to use the label map items' display names as keys.
    fill_in_gaps_and_background: whether to fill in gaps and background with
    respect to the id field in the proto. The id: 0 is reserved for the
    'background' class and will be added if it is missing. All other missing
    ids in range(1, max(id)) will be added with a dummy class name
    ("class_<id>") if they are missing.
```

```
  Returns:
    A dictionary mapping label names to id.

  Raises:
    ValueError: if fill_in_gaps_and_background and label_map has non-integer or
    negative values.
  """
  label_map = load_labelmap(label_map_path)
  label_map_dict = {}
  for item in label_map.item:
    if use_display_name:
      label_map_dict[item.display_name] = item.id
    else:
      label_map_dict[item.name] = item.id

  if fill_in_gaps_and_background:
    values = set(label_map_dict.values())

    if 0 not in values:
      label_map_dict['background'] = 0
    if not all(isinstance(value, int) for value in values):
      raise ValueError('The values in label map must be integers in order to'
                       'fill_in_gaps_and_background.')
    if not all(value >= 0 for value in values):
      raise ValueError('The values in the label map must be positive.')

    if len(values) != max(values) + 1:
      # there are gaps in the labels, fill in gaps.
      for value in range(1, max(values)):
        if value not in values:
          # TODO(rathodv): Add a prefix 'class_' here once the tool to generate
          # teacher annotation adds this prefix in the data.
          label_map_dict[str(value)] = value

  return label_map_dict


def create_categories_from_labelmap(label_map_path, use_display_name=True):
  """Reads a label map and returns categories list compatible with eval.

  This function converts label map proto and returns a list of dicts, each of
  which  has the following keys:
    'id': an integer id uniquely identifying this category.
    'name': string representing category name e.g., 'cat', 'dog'.

  Args:
    label_map_path: Path to `StringIntLabelMap` proto text file.
```

```
    use_display_name: (boolean) choose whether to load 'display_name' field
      as category name.  If False or if the display_name field does not exist,
      uses 'name' field as category names instead.

  Returns:
    categories: a list of dictionaries representing all possible categories.
  """
  label_map = load_labelmap(label_map_path)
  max_num_classes = max(item.id for item in label_map.item)
  return convert_label_map_to_categories(label_map, max_num_classes,
                                         use_display_name)


def create_category_index_from_labelmap(label_map_path, use_display_name=True):
  """Reads a label map and returns a category index.

  Args:
    label_map_path: Path to `StringIntLabelMap` proto text file.
    use_display_name: (boolean) choose whether to load 'display_name' field
      as category name.  If False or if the display_name field does not exist,
      uses 'name' field as category names instead.

  Returns:
    A category index, which is a dictionary that maps integer ids to dicts
    containing categories, e.g.
    {1: {'id': 1, 'name': 'dog'}, 2: {'id': 2, 'name': 'cat'}, ...}
  """
  categories = create_categories_from_labelmap(label_map_path, use_display_name)
  return create_category_index(categories)


def create_class_agnostic_category_index():
  """Creates a category index with a single `object` class."""
  return {1: {'id': 1, 'name': 'object'}}
```

 # Biblioteca de visualização

```
# Copyright 2017 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
```

```
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Alterado e modificado por: Alex Torezin Mendonça e Geovani Torezin Mendonça, # 2019.
#
==============================================================================

"""A set of functions that are used for visualization.

These functions often receive an image, perform some visualization on the image.
The functions do not return a value, instead they modify the image itself.

"""
import abc
import collections
import functools
# Set headless-friendly backend.
import matplotlib; matplotlib.use('Agg')  # pylint: disable=multiple-statements
import matplotlib.pyplot as plt  # pylint: disable=g-import-not-at-top
import numpy as np
import PIL.Image as Image
import PIL.ImageColor as ImageColor
import PIL.ImageDraw as ImageDraw
import PIL.ImageFont as ImageFont
import six
import tensorflow as tf

from object_detection.core import standard_fields as fields
from object_detection.utils import shape_utils

_TITLE_LEFT_MARGIN = 10
_TITLE_TOP_MARGIN = 10
STANDARD_COLORS = [
    'AliceBlue', 'Chartreuse', 'Aqua', 'Aquamarine', 'Azure', 'Beige', 'Bisque',
    'BlanchedAlmond', 'BlueViolet', 'BurlyWood', 'CadetBlue', 'AntiqueWhite',
    'Chocolate', 'Coral', 'CornflowerBlue', 'Cornsilk', 'Crimson', 'Cyan',
    'DarkCyan', 'DarkGoldenRod', 'DarkGrey', 'DarkKhaki', 'DarkOrange',
    'DarkOrchid', 'DarkSalmon', 'DarkSeaGreen', 'DarkTurquoise', 'DarkViolet',
    'DeepPink', 'DeepSkyBlue', 'DodgerBlue', 'FireBrick', 'FloralWhite',
    'ForestGreen', 'Fuchsia', 'Gainsboro', 'GhostWhite', 'Gold', 'GoldenRod',
    'Salmon', 'Tan', 'HoneyDew', 'HotPink', 'IndianRed', 'Ivory', 'Khaki',
    'Lavender', 'LavenderBlush', 'LawnGreen', 'LemonChiffon', 'LightBlue',
```

```
    'LightCoral', 'LightCyan', 'LightGoldenRodYellow', 'LightGray', 'LightGrey',
    'LightGreen', 'LightPink', 'LightSalmon', 'LightSeaGreen', 'LightSkyBlue',
    'LightSlateGray', 'LightSlateGrey', 'LightSteelBlue', 'LightYellow', 'Lime',
    'LimeGreen', 'Linen', 'Magenta', 'MediumAquaMarine', 'MediumOrchid',
    'MediumPurple', 'MediumSeaGreen', 'MediumSlateBlue', 'MediumSpringGreen',
    'MediumTurquoise', 'MediumVioletRed', 'MintCream', 'MistyRose', 'Moccasin',
    'NavajoWhite', 'OldLace', 'Olive', 'OliveDrab', 'Orange', 'OrangeRed',
    'Orchid', 'PaleGoldenRod', 'PaleGreen', 'PaleTurquoise', 'PaleVioletRed',
    'PapayaWhip', 'PeachPuff', 'Peru', 'Pink', 'Plum', 'PowderBlue', 'Purple',
    'Red', 'RosyBrown', 'RoyalBlue', 'SaddleBrown', 'Green', 'SandyBrown',
    'SeaGreen', 'SeaShell', 'Sienna', 'Silver', 'SkyBlue', 'SlateBlue',
    'SlateGray', 'SlateGrey', 'Snow', 'SpringGreen', 'SteelBlue', 'GreenYellow',
    'Teal', 'Thistle', 'Tomato', 'Turquoise', 'Violet', 'Wheat', 'White',
    'WhiteSmoke', 'Yellow', 'YellowGreen'
]


def save_image_array_as_png(image, output_path):
  """Saves an image (represented as a numpy array) to PNG.

  Args:
    image: a numpy array with shape [height, width, 3].
    output_path: path to which image should be written.
  """
  image_pil = Image.fromarray(np.uint8(image)).convert('RGB')
  with tf.gfile.Open(output_path, 'w') as fid:
    image_pil.save(fid, 'PNG')


def encode_image_array_as_png_str(image):
  """Encodes a numpy array into a PNG string.

  Args:
    image: a numpy array with shape [height, width, 3].

  Returns:
    PNG encoded image string.
  """
  image_pil = Image.fromarray(np.uint8(image))
  output = six.BytesIO()
  image_pil.save(output, format='PNG')
  png_string = output.getvalue()
  output.close()
  return png_string


def draw_bounding_box_on_image_array(image,
                                     ymin,
```

```
                               xmin,
                               ymax,
                               xmax,
                               color='red',
                               thickness=4,
                               display_str_list=(),
                               use_normalized_coordinates=True):
  """Adds a bounding box to an image (numpy array).

  Bounding box coordinates can be specified in either absolute (pixel) or
  normalized coordinates by setting the use_normalized_coordinates argument.

  Args:
    image: a numpy array with shape [height, width, 3].
    ymin: ymin of bounding box.
    xmin: xmin of bounding box.
    ymax: ymax of bounding box.
    xmax: xmax of bounding box.
    color: color to draw bounding box. Default is red.
    thickness: line thickness. Default value is 4.
    display_str_list: list of strings to display in box
                      (each to be shown on its own line).
    use_normalized_coordinates: If True (default), treat coordinates
      ymin, xmin, ymax, xmax as relative to the image.  Otherwise treat
      coordinates as absolute.
  """
  image_pil = Image.fromarray(np.uint8(image)).convert('RGB')
  draw_bounding_box_on_image(image_pil, ymin, xmin, ymax, xmax, color,
                             thickness, display_str_list,
                             use_normalized_coordinates)
  np.copyto(image, np.array(image_pil))


def draw_bounding_box_on_image(image,
                               ymin,
                               xmin,
                               ymax,
                               xmax,
                               color='red',
                               thickness=4,
                               display_str_list=(),
                               use_normalized_coordinates=True):
  """Adds a bounding box to an image.

  Bounding box coordinates can be specified in either absolute (pixel) or
  normalized coordinates by setting the use_normalized_coordinates argument.

  Each string in display_str_list is displayed on a separate line above the
```

```
bounding box in black text on a rectangle filled with the input 'color'.
If the top of the bounding box extends to the edge of the image, the strings
are displayed below the bounding box.

Args:
  image: a PIL.Image object.
  ymin: ymin of bounding box.
  xmin: xmin of bounding box.
  ymax: ymax of bounding box.
  xmax: xmax of bounding box.
  color: color to draw bounding box. Default is red.
  thickness: line thickness. Default value is 4.
  display_str_list: list of strings to display in box
                    (each to be shown on its own line).
  use_normalized_coordinates: If True (default), treat coordinates
    ymin, xmin, ymax, xmax as relative to the image.  Otherwise treat
    coordinates as absolute.
"""
draw = ImageDraw.Draw(image)
im_width, im_height = image.size
if use_normalized_coordinates:
  (left, right, top, bottom) = (xmin * im_width, xmax * im_width,
                                ymin * im_height, ymax * im_height)
else:
  (left, right, top, bottom) = (xmin, xmax, ymin, ymax)
draw.line([(left, top), (left, bottom), (right, bottom),
           (right, top), (left, top)], width=thickness, fill=color)
try:
  font = ImageFont.truetype('arial.ttf', 24)
except IOError:
  font = ImageFont.load_default()

# If the total height of the display strings added to the top of the bounding
# box exceeds the top of the image, stack the strings below the bounding box
# instead of above.
display_str_heights = [font.getsize(ds)[1] for ds in display_str_list]
# Each display_str has a top and bottom margin of 0.05x.
total_display_str_height = (1 + 2 * 0.05) * sum(display_str_heights)

if top > total_display_str_height:
  text_bottom = top
else:
  text_bottom = bottom + total_display_str_height
# Reverse list and print from bottom to top.
for display_str in display_str_list[::-1]:
  text_width, text_height = font.getsize(display_str)
  margin = np.ceil(0.05 * text_height)
  draw.rectangle(
```

```
      [(left, text_bottom - text_height - 2 * margin), (left + text_width,
                                                        text_bottom)],
      fill=color)
    draw.text(
      (left + margin, text_bottom - text_height - margin),
      display_str,
      fill='black',
      font=font)
    text_bottom -= text_height - 2 * margin


def draw_bounding_boxes_on_image_array(image,
                                       boxes,
                                       color='red',
                                       thickness=4,
                                       display_str_list_list=()):
  """Draws bounding boxes on image (numpy array).

  Args:
    image: a numpy array object.
    boxes: a 2 dimensional numpy array of [N, 4]: (ymin, xmin, ymax, xmax).
          The coordinates are in normalized format between [0, 1].
    color: color to draw bounding box. Default is red.
    thickness: line thickness. Default value is 4.
    display_str_list_list: list of list of strings.
                           a list of strings for each bounding box.
                           The reason to pass a list of strings for a
                           bounding box is that it might contain
                           multiple labels.

  Raises:
    ValueError: if boxes is not a [N, 4] array
  """
  image_pil = Image.fromarray(image)
  draw_bounding_boxes_on_image(image_pil, boxes, color, thickness,
                               display_str_list_list)
  np.copyto(image, np.array(image_pil))


def draw_bounding_boxes_on_image(image,
                                 boxes,
                                 color='red',
                                 thickness=4,
                                 display_str_list_list=()):
  """Draws bounding boxes on image.

  Args:
    image: a PIL.Image object.
```

```
    boxes: a 2 dimensional numpy array of [N, 4]: (ymin, xmin, ymax, xmax).
         The coordinates are in normalized format between [0, 1].
    color: color to draw bounding box. Default is red.
    thickness: line thickness. Default value is 4.
    display_str_list_list: list of list of strings.
                           a list of strings for each bounding box.
                           The reason to pass a list of strings for a
                           bounding box is that it might contain
                           multiple labels.

  Raises:
    ValueError: if boxes is not a [N, 4] array
  """
  boxes_shape = boxes.shape
  if not boxes_shape:
    return
  if len(boxes_shape) != 2 or boxes_shape[1] != 4:
    raise ValueError('Input must be of size [N, 4]')
  for i in range(boxes_shape[0]):
    display_str_list = ()
    if display_str_list_list:
      display_str_list = display_str_list_list[i]
    draw_bounding_box_on_image(image, boxes[i, 0], boxes[i, 1], boxes[i, 2],
                               boxes[i, 3], color, thickness, display_str_list)


def _visualize_boxes(image, boxes, classes, scores, category_index, **kwargs):
  return visualize_boxes_and_labels_on_image_array(
      image, boxes, classes, scores, category_index=category_index, **kwargs)


def _visualize_boxes_and_masks(image, boxes, classes, scores, masks,
                               category_index, **kwargs):
  return visualize_boxes_and_labels_on_image_array(
      image,
      boxes,
      classes,
      scores,
      category_index=category_index,
      instance_masks=masks,
      **kwargs)


def _visualize_boxes_and_keypoints(image, boxes, classes, scores, keypoints,
                                   category_index, **kwargs):
  return visualize_boxes_and_labels_on_image_array(
      image,
      boxes,
```

```
      classes,
      scores,
      category_index=category_index,
      keypoints=keypoints,
      **kwargs)


def _visualize_boxes_and_masks_and_keypoints(
    image, boxes, classes, scores, masks, keypoints, category_index, **kwargs):
  return visualize_boxes_and_labels_on_image_array(
      image,
      boxes,
      classes,
      scores,
      category_index=category_index,
      instance_masks=masks,
      keypoints=keypoints,
      **kwargs)


def _resize_original_image(image, image_shape):
  image = tf.expand_dims(image, 0)
  image = tf.image.resize_images(
      image,
      image_shape,
      method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
      align_corners=True)
  return tf.cast(tf.squeeze(image, 0), tf.uint8)


def draw_bounding_boxes_on_image_tensors(images,
                                         boxes,
                                         classes,
                                         scores,
                                         category_index,
                                         original_image_spatial_shape=None,
                                         true_image_shape=None,
                                         instance_masks=None,
                                         keypoints=None,
                                         max_boxes_to_draw=20,
                                         min_score_thresh=0.2,
                                         use_normalized_coordinates=True):
  """Draws bounding boxes, masks, and keypoints on batch of image tensors.

  Args:
    images: A 4D uint8 image tensor of shape [N, H, W, C]. If C > 3, additional
      channels will be ignored. If C = 1, then we convert the images to RGB
      images.
```

```
  boxes: [N, max_detections, 4] float32 tensor of detection boxes.
  classes: [N, max_detections] int tensor of detection classes. Note that
    classes are 1-indexed.
  scores: [N, max_detections] float32 tensor of detection scores.
  category_index: a dict that maps integer ids to category dicts. e.g.
    {1: {1: 'dog'}, 2: {2: 'cat'}, ...}
  original_image_spatial_shape: [N, 2] tensor containing the spatial size of
    the original image.
  true_image_shape: [N, 3] tensor containing the spatial size of unpadded
    original_image.
  instance_masks: A 4D uint8 tensor of shape [N, max_detection, H, W] with
    instance masks.
  keypoints: A 4D float32 tensor of shape [N, max_detection, num_keypoints, 2]
    with keypoints.
  max_boxes_to_draw: Maximum number of boxes to draw on an image. Default 20.
  min_score_thresh: Minimum score threshold for visualization. Default 0.2.
  use_normalized_coordinates: Whether to assume boxes and kepoints are in
    normalized coordinates (as opposed to absolute coordiantes).
    Default is True.

Returns:
  4D image tensor of type uint8, with boxes drawn on top.
"""
# Additional channels are being ignored.
if images.shape[3] > 3:
  images = images[:, :, :, 0:3]
elif images.shape[3] == 1:
  images = tf.image.grayscale_to_rgb(images)
visualization_keyword_args = {
    'use_normalized_coordinates': use_normalized_coordinates,
    'max_boxes_to_draw': max_boxes_to_draw,
    'min_score_thresh': min_score_thresh,
    'agnostic_mode': False,
    'line_thickness': 4
}
if true_image_shape is None:
  true_shapes = tf.constant(-1, shape=[images.shape.as_list()[0], 3])
else:
  true_shapes = true_image_shape
if original_image_spatial_shape is None:
  original_shapes = tf.constant(-1, shape=[images.shape.as_list()[0], 2])
else:
  original_shapes = original_image_spatial_shape

if instance_masks is not None and keypoints is None:
  visualize_boxes_fn = functools.partial(
      _visualize_boxes_and_masks,
      category_index=category_index,
```

```
      **visualization_keyword_args)
  elems = [
      true_shapes, original_shapes, images, boxes, classes, scores,
      instance_masks
  ]
elif instance_masks is None and keypoints is not None:
  visualize_boxes_fn = functools.partial(
      _visualize_boxes_and_keypoints,
      category_index=category_index,
      **visualization_keyword_args)
  elems = [
      true_shapes, original_shapes, images, boxes, classes, scores, keypoints
  ]
elif instance_masks is not None and keypoints is not None:
  visualize_boxes_fn = functools.partial(
      _visualize_boxes_and_masks_and_keypoints,
      category_index=category_index,
      **visualization_keyword_args)
  elems = [
      true_shapes, original_shapes, images, boxes, classes, scores,
      instance_masks, keypoints
  ]
else:
  visualize_boxes_fn = functools.partial(
      _visualize_boxes,
      category_index=category_index,
      **visualization_keyword_args)
  elems = [
      true_shapes, original_shapes, images, boxes, classes, scores
  ]

def draw_boxes(image_and_detections):
  """Draws boxes on image."""
  true_shape = image_and_detections[0]
  original_shape = image_and_detections[1]
  if true_image_shape is not None:
    image = shape_utils.pad_or_clip_nd(image_and_detections[2],
                                       [true_shape[0], true_shape[1], 3])
  if original_image_spatial_shape is not None:
    image_and_detections[2] = _resize_original_image(image, original_shape)

  image_with_boxes = tf.py_func(visualize_boxes_fn, image_and_detections[2:],
                                tf.uint8)
  return image_with_boxes

images = tf.map_fn(draw_boxes, elems, dtype=tf.uint8, back_prop=False)
return images
```

```
def draw_side_by_side_evaluation_image(eval_dict,
                                       category_index,
                                       max_boxes_to_draw=20,
                                       min_score_thresh=0.2,
                                       use_normalized_coordinates=True):
  """Creates a side-by-side image with detections and groundtruth.

  Bounding boxes (and instance masks, if available) are visualized on both
  subimages.

  Args:
    eval_dict: The evaluation dictionary returned by
      eval_util.result_dict_for_batched_example() or
      eval_util.result_dict_for_single_example().
    category_index: A category index (dictionary) produced from a labelmap.
    max_boxes_to_draw: The maximum number of boxes to draw for detections.
    min_score_thresh: The minimum score threshold for showing detections.
    use_normalized_coordinates: Whether to assume boxes and kepoints are in
      normalized coordinates (as opposed to absolute coordiantes).
      Default is True.

  Returns:
    A list of [1, H, 2 * W, C] uint8 tensor. The subimage on the left
      corresponds to detections, while the subimage on the right corresponds to
      groundtruth.
  """
  detection_fields = fields.DetectionResultFields()
  input_data_fields = fields.InputDataFields()

  images_with_detections_list = []

  # Add the batch dimension if the eval_dict is for single example.
  if len(eval_dict[detection_fields.detection_classes].shape) == 1:
    for key in eval_dict:
      if key != input_data_fields.original_image:
        eval_dict[key] = tf.expand_dims(eval_dict[key], 0)

  for indx in range(eval_dict[input_data_fields.original_image].shape[0]):
    instance_masks = None
    if detection_fields.detection_masks in eval_dict:
      instance_masks = tf.cast(
          tf.expand_dims(
              eval_dict[detection_fields.detection_masks][indx], axis=0),
          tf.uint8)
    keypoints = None
    if detection_fields.detection_keypoints in eval_dict:
      keypoints = tf.expand_dims(
          eval_dict[detection_fields.detection_keypoints][indx], axis=0)
```

```
groundtruth_instance_masks = None
if input_data_fields.groundtruth_instance_masks in eval_dict:
  groundtruth_instance_masks = tf.cast(
      tf.expand_dims(
          eval_dict[input_data_fields.groundtruth_instance_masks][indx],
          axis=0), tf.uint8)

images_with_detections = draw_bounding_boxes_on_image_tensors(
    tf.expand_dims(
        eval_dict[input_data_fields.original_image][indx], axis=0),
    tf.expand_dims(
        eval_dict[detection_fields.detection_boxes][indx], axis=0),
    tf.expand_dims(
        eval_dict[detection_fields.detection_classes][indx], axis=0),
    tf.expand_dims(
        eval_dict[detection_fields.detection_scores][indx], axis=0),
    category_index,
    original_image_spatial_shape=tf.expand_dims(
        eval_dict[input_data_fields.original_image_spatial_shape][indx],
        axis=0),
    true_image_shape=tf.expand_dims(
        eval_dict[input_data_fields.true_image_shape][indx], axis=0),
    instance_masks=instance_masks,
    keypoints=keypoints,
    max_boxes_to_draw=max_boxes_to_draw,
    min_score_thresh=min_score_thresh,
    use_normalized_coordinates=use_normalized_coordinates)
images_with_groundtruth = draw_bounding_boxes_on_image_tensors(
    tf.expand_dims(
        eval_dict[input_data_fields.original_image][indx], axis=0),
    tf.expand_dims(
        eval_dict[input_data_fields.groundtruth_boxes][indx], axis=0),
    tf.expand_dims(
        eval_dict[input_data_fields.groundtruth_classes][indx], axis=0),
    tf.expand_dims(
        tf.ones_like(
            eval_dict[input_data_fields.groundtruth_classes][indx],
            dtype=tf.float32),
        axis=0),
    category_index,
    original_image_spatial_shape=tf.expand_dims(
        eval_dict[input_data_fields.original_image_spatial_shape][indx],
        axis=0),
    true_image_shape=tf.expand_dims(
        eval_dict[input_data_fields.true_image_shape][indx], axis=0),
    instance_masks=groundtruth_instance_masks,
    keypoints=None,
    max_boxes_to_draw=None,
```

```
      min_score_thresh=0.0,
      use_normalized_coordinates=use_normalized_coordinates)
    images_with_detections_list.append(
      tf.concat([images_with_detections, images_with_groundtruth], axis=2))
  return images_with_detections_list


def draw_keypoints_on_image_array(image,
                                  keypoints,
                                  color='red',
                                  radius=2,
                                  use_normalized_coordinates=True):
  """Draws keypoints on an image (numpy array).

  Args:
    image: a numpy array with shape [height, width, 3].
    keypoints: a numpy array with shape [num_keypoints, 2].
    color: color to draw the keypoints with. Default is red.
    radius: keypoint radius. Default value is 2.
    use_normalized_coordinates: if True (default), treat keypoint values as
      relative to the image.  Otherwise treat them as absolute.
  """
  image_pil = Image.fromarray(np.uint8(image)).convert('RGB')
  draw_keypoints_on_image(image_pil, keypoints, color, radius,
                          use_normalized_coordinates)
  np.copyto(image, np.array(image_pil))


def draw_keypoints_on_image(image,
                            keypoints,
                            color='red',
                            radius=2,
                            use_normalized_coordinates=True):
  """Draws keypoints on an image.

  Args:
    image: a PIL.Image object.
    keypoints: a numpy array with shape [num_keypoints, 2].
    color: color to draw the keypoints with. Default is red.
    radius: keypoint radius. Default value is 2.
    use_normalized_coordinates: if True (default), treat keypoint values as
      relative to the image.  Otherwise treat them as absolute.
  """
  draw = ImageDraw.Draw(image)
  im_width, im_height = image.size
  keypoints_x = [k[1] for k in keypoints]
  keypoints_y = [k[0] for k in keypoints]
  if use_normalized_coordinates:
```

```
    keypoints_x = tuple([im_width * x for x in keypoints_x])
    keypoints_y = tuple([im_height * y for y in keypoints_y])
  for keypoint_x, keypoint_y in zip(keypoints_x, keypoints_y):
    draw.ellipse([(keypoint_x - radius, keypoint_y - radius),
                  (keypoint_x + radius, keypoint_y + radius)],
                 outline=color, fill=color)


def draw_mask_on_image_array(image, mask, color='red', alpha=0.4):
  """Draws mask on an image.

  Args:
    image: uint8 numpy array with shape (img_height, img_height, 3)
    mask: a uint8 numpy array of shape (img_height, img_height) with
      values between either 0 or 1.
    color: color to draw the keypoints with. Default is red.
    alpha: transparency value between 0 and 1. (default: 0.4)

  Raises:
    ValueError: On incorrect data type for image or masks.
  """
  if image.dtype != np.uint8:
    raise ValueError('`image` not of type np.uint8')
  if mask.dtype != np.uint8:
    raise ValueError('`mask` not of type np.uint8')
  if np.any(np.logical_and(mask != 1, mask != 0)):
    raise ValueError('`mask` elements should be in [0, 1]')
  if image.shape[:2] != mask.shape:
    raise ValueError('The image has spatial dimensions %s but the mask has '
                     'dimensions %s' % (image.shape[:2], mask.shape))
  rgb = ImageColor.getrgb(color)
  pil_image = Image.fromarray(image)

  solid_color = np.expand_dims(
      np.ones_like(mask), axis=2) * np.reshape(list(rgb), [1, 1, 3])
  pil_solid_color = Image.fromarray(np.uint8(solid_color)).convert('RGBA')
  pil_mask = Image.fromarray(np.uint8(255.0*alpha*mask)).convert('L')
  pil_image = Image.composite(pil_solid_color, pil_image, pil_mask)
  np.copyto(image, np.array(pil_image.convert('RGB')))


def visualize_boxes_and_labels_on_image_array(
    image,
    boxes,
    classes,
    scores,
    category_index,
    instance_masks=None,
```

```
  instance_boundaries=None,
  keypoints=None,
  use_normalized_coordinates=False,
  max_boxes_to_draw=20,
  min_score_thresh=.5,
  agnostic_mode=False,
  line_thickness=4,
  groundtruth_box_visualization_color='black',
  skip_scores=False,
  skip_labels=False):

"""Overlay labeled boxes on an image with formatted scores and label names.

This function groups boxes that correspond to the same location
and creates a display string for each detection and overlays these
on the image. Note that this function modifies the image in place, and returns
that same image.

Args:
  image: uint8 numpy array with shape (img_height, img_width, 3)
  boxes: a numpy array of shape [N, 4]
  classes: a numpy array of shape [N]. Note that class indices are 1-based,
    and match the keys in the label map.
  scores: a numpy array of shape [N] or None.  If scores=None, then
    this function assumes that the boxes to be plotted are groundtruth
    boxes and plot all boxes as black with no classes or scores.
  category_index: a dict containing category dictionaries (each holding
    category index `id` and category name `name`) keyed by category indices.
  instance_masks: a numpy array of shape [N, image_height, image_width] with
    values ranging between 0 and 1, can be None.
  instance_boundaries: a numpy array of shape [N, image_height, image_width]
    with values ranging between 0 and 1, can be None.
  keypoints: a numpy array of shape [N, num_keypoints, 2], can
    be None
  use_normalized_coordinates: whether boxes is to be interpreted as
    normalized coordinates or not.
  max_boxes_to_draw: maximum number of boxes to visualize.  If None, draw
    all boxes.
  min_score_thresh: minimum score threshold for a box to be visualized
  agnostic_mode: boolean (default: False) controlling whether to evaluate in
    class-agnostic mode or not.  This mode will display scores but ignore
    classes.
  line_thickness: integer (default: 4) controlling line width of the boxes.
  groundtruth_box_visualization_color: box color for visualizing groundtruth
    boxes
  skip_scores: whether to skip score when drawing a single detection
  skip_labels: whether to skip label when drawing a single detection
```

```
Returns:
  uint8 numpy array with shape (img_height, img_width, 3) with overlaid boxes.
"""
# Create a display string (and color) for every box location, group any boxes
# that correspond to the same location.
box_to_display_str_map = collections.defaultdict(list)
box_to_color_map = collections.defaultdict(str)
box_to_instance_masks_map = {}
box_to_instance_boundaries_map = {}
box_to_keypoints_map = collections.defaultdict(list)
if not max_boxes_to_draw:
  max_boxes_to_draw = boxes.shape[0]
for i in range(min(max_boxes_to_draw, boxes.shape[0])):
  if scores is None or scores[i] > min_score_thresh:

    box = tuple(boxes[i].tolist())
    if instance_masks is not None:
      box_to_instance_masks_map[box] = instance_masks[i]
    if instance_boundaries is not None:
      box_to_instance_boundaries_map[box] = instance_boundaries[i]
    if keypoints is not None:
      box_to_keypoints_map[box].extend(keypoints[i])
    if scores is None:
      box_to_color_map[box] = groundtruth_box_visualization_color
    else:
      display_str = ''
      if not skip_labels:
        if not agnostic_mode:
          if classes[i] in category_index.keys():
            class_name = category_index[classes[i]]['name']
          else:
            class_name = 'N/A'
          display_str = str(class_name)
      if not skip_scores:
        if not display_str:
          display_str = '{}%'.format(int(100*scores[i]))

          # print(scores[i]) #alex
          # print(class_name)#alex
        else:
          display_str = '{}: {}%'.format(display_str, int(100*scores[i]))

          # print(scores[i])#alex
          #print(class_name)#alex

      box_to_display_str_map[box].append(display_str)
      if agnostic_mode:
        box_to_color_map[box] = 'DarkOrange'
```

```
      else:
        box_to_color_map[box] = STANDARD_COLORS[
            classes[i] % len(STANDARD_COLORS)]

  # Draw all boxes onto image.
  for box, color in box_to_color_map.items():
    ymin, xmin, ymax, xmax = box
    if instance_masks is not None:
      draw_mask_on_image_array(
          image,
          box_to_instance_masks_map[box],
          color=color
      )
    if instance_boundaries is not None:
      draw_mask_on_image_array(
          image,
          box_to_instance_boundaries_map[box],
          color='red',
          alpha=1.0
      )
    draw_bounding_box_on_image_array(
        image,
        ymin,
        xmin,
        ymax,
        xmax,
        color=color,
        thickness=line_thickness,
        display_str_list=box_to_display_str_map[box],
        use_normalized_coordinates=use_normalized_coordinates)
    if keypoints is not None:
      draw_keypoints_on_image_array(
          image,
          box_to_keypoints_map[box],
          color=color,
          radius=line_thickness / 2,
          use_normalized_coordinates=use_normalized_coordinates)


  return image


def add_cdf_image_summary(values, name):
  """Adds a tf.summary.image for a CDF plot of the values.

  Normalizes `values` such that they sum to 1, plots the cumulative distribution
  function and creates a tf image summary.
```

```
  Args:
    values: a 1-D float32 tensor containing the values.
    name: name for the image summary.
  """
  def cdf_plot(values):
    """Numpy function to plot CDF."""
    normalized_values = values / np.sum(values)
    sorted_values = np.sort(normalized_values)
    cumulative_values = np.cumsum(sorted_values)
    fraction_of_examples = (np.arange(cumulative_values.size, dtype=np.float32)
                            / cumulative_values.size)
    fig = plt.figure(frameon=False)
    ax = fig.add_subplot('111')
    ax.plot(fraction_of_examples, cumulative_values)
    ax.set_ylabel('cumulative normalized values')
    ax.set_xlabel('fraction of examples')
    fig.canvas.draw()
    width, height = fig.get_size_inches() * fig.get_dpi()
    image = np.fromstring(fig.canvas.tostring_rgb(), dtype='uint8').reshape(
        1, int(height), int(width), 3)
    return image
  cdf_plot = tf.py_func(cdf_plot, [values], tf.uint8)
  tf.summary.image(name, cdf_plot)


def add_hist_image_summary(values, bins, name):
  """Adds a tf.summary.image for a histogram plot of the values.

  Plots the histogram of values and creates a tf image summary.

  Args:
    values: a 1-D float32 tensor containing the values.
    bins: bin edges which will be directly passed to np.histogram.
    name: name for the image summary.
  """

  def hist_plot(values, bins):
    """Numpy function to plot hist."""
    fig = plt.figure(frameon=False)
    ax = fig.add_subplot('111')
    y, x = np.histogram(values, bins=bins)
    ax.plot(x[:-1], y)
    ax.set_ylabel('count')
    ax.set_xlabel('value')
    fig.canvas.draw()
    width, height = fig.get_size_inches() * fig.get_dpi()
    image = np.fromstring(
        fig.canvas.tostring_rgb(), dtype='uint8').reshape(
```

```
        1, int(height), int(width), 3)
    return image
  hist_plot = tf.py_func(hist_plot, [values, bins], tf.uint8)
  tf.summary.image(name, hist_plot)


class EvalMetricOpsVisualization(object):
  """Abstract base class responsible for visualizations during evaluation.

  Currently, summary images are not run during evaluation. One way to produce
  evaluation images in Tensorboard is to provide tf.summary.image strings as
  `value_ops` in tf.estimator.EstimatorSpec's `eval_metric_ops`. This class is
  responsible for accruing images (with overlaid detections and groundtruth)
  and returning a dictionary that can be passed to `eval_metric_ops`.
  """
  __metaclass__ = abc.ABCMeta

  def __init__(self,
               category_index,
               max_examples_to_draw=5,
               max_boxes_to_draw=20,
               min_score_thresh=0.2,
               use_normalized_coordinates=True,
               summary_name_prefix='evaluation_image'):
    """Creates an EvalMetricOpsVisualization.

    Args:
      category_index: A category index (dictionary) produced from a labelmap.
      max_examples_to_draw: The maximum number of example summaries to produce.
      max_boxes_to_draw: The maximum number of boxes to draw for detections.
      min_score_thresh: The minimum score threshold for showing detections.
      use_normalized_coordinates: Whether to assume boxes and kepoints are in
        normalized coordinates (as opposed to absolute coordiantes).
        Default is True.
      summary_name_prefix: A string prefix for each image summary.
    """

    self._category_index = category_index
    self._max_examples_to_draw = max_examples_to_draw
    self._max_boxes_to_draw = max_boxes_to_draw
    self._min_score_thresh = min_score_thresh
    self._use_normalized_coordinates = use_normalized_coordinates
    self._summary_name_prefix = summary_name_prefix
    self._images = []

  def clear(self):
    self._images = []
```

```
def add_images(self, images):
  """Store a list of images, each with shape [1, H, W, C]."""
  if len(self._images) >= self._max_examples_to_draw:
    return

  # Store images and clip list if necessary.
  self._images.extend(images)
  if len(self._images) > self._max_examples_to_draw:
    self._images[self._max_examples_to_draw:] = []

def get_estimator_eval_metric_ops(self, eval_dict):
  """Returns metric ops for use in tf.estimator.EstimatorSpec.

  Args:
    eval_dict: A dictionary that holds an image, groundtruth, and detections
      for a batched example. Note that, we use only the first example for
      visualization. See eval_util.result_dict_for_batched_example() for a
      convenient method for constructing such a dictionary. The dictionary
      contains
      fields.InputDataFields.original_image: [batch_size, H, W, 3] image.
      fields.InputDataFields.original_image_spatial_shape: [batch_size, 2]
        tensor containing the size of the original image.
      fields.InputDataFields.true_image_shape: [batch_size, 3]
        tensor containing the spatial size of the upadded original image.
      fields.InputDataFields.groundtruth_boxes - [batch_size, num_boxes, 4]
        float32 tensor with groundtruth boxes in range [0.0, 1.0].
      fields.InputDataFields.groundtruth_classes - [batch_size, num_boxes]
        int64 tensor with 1-indexed groundtruth classes.
      fields.InputDataFields.groundtruth_instance_masks - (optional)
        [batch_size, num_boxes, H, W] int64 tensor with instance masks.
      fields.DetectionResultFields.detection_boxes - [batch_size,
        max_num_boxes, 4] float32 tensor with detection boxes in range [0.0,
        1.0].
      fields.DetectionResultFields.detection_classes - [batch_size,
        max_num_boxes] int64 tensor with 1-indexed detection classes.
      fields.DetectionResultFields.detection_scores - [batch_size,
        max_num_boxes] float32 tensor with detection scores.
      fields.DetectionResultFields.detection_masks - (optional) [batch_size,
        max_num_boxes, H, W] float32 tensor of binarized masks.
      fields.DetectionResultFields.detection_keypoints - (optional)
        [batch_size, max_num_boxes, num_keypoints, 2] float32 tensor with
        keypoints.

  Returns:
    A dictionary of image summary names to tuple of (value_op, update_op). The
    `update_op` is the same for all items in the dictionary, and is
    responsible for saving a single side-by-side image with detections and
    groundtruth. Each `value_op` holds the tf.summary.image string for a given
```

```
    image.
    """
    if self._max_examples_to_draw == 0:
      return {}
    images = self.images_from_evaluation_dict(eval_dict)

    def get_images():
      """Returns a list of images, padded to self._max_images_to_draw."""
      images = self._images
      while len(images) < self._max_examples_to_draw:
        images.append(np.array(0, dtype=np.uint8))
      self.clear()
      return images

    def image_summary_or_default_string(summary_name, image):
      """Returns image summaries for non-padded elements."""
      return tf.cond(
          tf.equal(tf.size(tf.shape(image)), 4),
          lambda: tf.summary.image(summary_name, image),
          lambda: tf.constant(''))

    update_op = tf.py_func(self.add_images, [[images[0]]], [])
    image_tensors = tf.py_func(
        get_images, [], [tf.uint8] * self._max_examples_to_draw)
    eval_metric_ops = {}
    for i, image in enumerate(image_tensors):
      summary_name = self._summary_name_prefix + '/' + str(i)
      value_op = image_summary_or_default_string(summary_name, image)
      eval_metric_ops[summary_name] = (value_op, update_op)
    return eval_metric_ops

  @abc.abstractmethod
  def images_from_evaluation_dict(self, eval_dict):
    """Converts evaluation dictionary into a list of image tensors.

    To be overridden by implementations.

    Args:
      eval_dict: A dictionary with all the necessary information for producing
        visualizations.

    Returns:
      A list of [1, H, W, C] uint8 tensors.
    """
    raise NotImplementedError


class VisualizeSingleFrameDetections(EvalMetricOpsVisualization):
```

```
  """Class responsible for single-frame object detection visualizations."""

  def __init__(self,
               category_index,
               max_examples_to_draw=5,
               max_boxes_to_draw=20,
               min_score_thresh=0.2,
               use_normalized_coordinates=True,
               summary_name_prefix='Detections_Left_Groundtruth_Right'):
    super(VisualizeSingleFrameDetections, self).__init__(
        category_index=category_index,
        max_examples_to_draw=max_examples_to_draw,
        max_boxes_to_draw=max_boxes_to_draw,
        min_score_thresh=min_score_thresh,
        use_normalized_coordinates=use_normalized_coordinates,
        summary_name_prefix=summary_name_prefix)

  def images_from_evaluation_dict(self, eval_dict):
    return draw_side_by_side_evaluation_image(
        eval_dict, self._category_index, self._max_boxes_to_draw,
        self._min_score_thresh, self._use_normalized_coordinates)
```

#Configurações de treinamento da rede neural

```
model {
  faster_rcnn {
    num_classes: 2
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
```

```
    scales: [0.25, 0.5, 1.0, 2.0]
    aspect_ratios: [0.5, 1.0, 2.0]
    height_stride: 16
    width_stride: 16
  }
}
first_stage_box_predictor_conv_hyperparams {
  op: CONV
  regularizer {
    l2_regularizer {
      weight: 0.0
    }
  }
  initializer {
    truncated_normal_initializer {
      stddev: 0.01
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
```

```
        regularizer {
          l2_regularizer {
            weight: 0.0
          }
        }
        initializer {
          variance_scaling_initializer {
            factor: 1.0
            uniform: true
            mode: FAN_AVG
          }
        }
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
```

```
    learning_rate: {
      manual_step_learning_rate {
        initial_learning_rate: 0.0002
        #schedule {
         # step: 0
         # learning_rate: .0002
        #}
        schedule {
          step: 900000
          learning_rate: .00002
        }
        schedule {
          step: 1200000
          learning_rate: .000002
        }
      }
    }
    momentum_optimizer_value: 0.9
  }
  use_moving_average: false
 }
 gradient_clipping_by_norm: 10.0
 fine_tune_checkpoint:
"C:/alex/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/mod
el.ckpt"
 from_detection_checkpoint: true
 # Note: The below line limits the training process to 200K steps, which we
 # empirically found to be sufficient enough to train the pets dataset. This
 # effectively bypasses the learning rate schedule (the learning rate will
 # never decay). Remove the below line to train indefinitely.
 num_steps: 200000
 data_augmentation_options {
  random_horizontal_flip {
```

```
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/alex/models/research/object_detection/train.record"
  }
  label_map_path: "C:/alex/models/research/object_detection/training/labelmap.pbtxt"
}

eval_config: {
  num_examples: 90
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  #max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/alex/models/research/object_detection/test.record"
  }
  label_map_path: "C:/alex/models/research/object_detection/training/labelmap.pbtxt"
  shuffle: false
  num_readers: 1
}
```