CENTRO UNIVERSITÁRIO DE ANÁPOLIS – UNIEVANGÉLICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

HUDSON MURILO LEAL PINA
MICHAEL BRYAN MIRANDA MARTINS

UTILIZAÇÃO DE BIBLIOTECAS DE VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA NA IDENTIFICAÇÃO DE RESÍDUOS SÓLIDOS (METAL E VIDRO) APLICADO A COLETA SELETIVA

ANÁPOLIS - GO
2018

HUDSON MURILO LEAL PINA
MICHAEL BRYAN MIRANDA MARTINS

UTILIZAÇÃO DE BIBLIOTECAS DE VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA NA IDENTIFICAÇÃO DE RESÍDUOS SÓLIDOS (METAL E VIDRO) APLICADO A COLETA SELETIVA

Projeto de pesquisa apresentado ao Curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA, sob a orientação do Prof. Me. William Pereira dos Santos Junior.

ANÁPOLIS - GO
2018

HUDSON MURILO LEAL PINA
MICHAEL BRYAN MIRANDA MARTINS

UTILIZAÇÃO DE BIBLIOTECAS DE VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA NA IDENTIFICAÇÃO DE RESÍDUOS SÓLIDOS (METAL E VIDRO) APLICADO A COLETA SELETIVA

Projeto de pesquisa apresentado ao Curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA, sob a orientação do Prof. Me. William Pereira dos Santos Junior.

Banca Examinadora

.....
Prof. Me. William Pereira dos Santos Junior
Orientador

.....
Prof.ª Me. Luciana Nishi
Convidada

.....
Prof. Esp. Kleber Silvestre Diogo
Convidado

Anápolis, 15 de Junho de 2018.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

3D	3 Dimensões (Tridimensional)
API	*Aplication Programming Interface*
CNN	*Convolutional Neural Network*
CSV	*Comma Separated Values*
CUDA	*Compute Unified Device Architecture*
GPU	*Graphic Processing Unit*
IA	Inteligência Artificial
OpenCV	*Open Source Computer Vision Library*
XML	*Extensible Markup Language*

SUMÁRIO

RESUMO

Este trabalho trata da utilização de bibliotecas de visão computacional e aprendizado de máquina na identificação de resíduos sólidos (metal e vidro) aplicado ao processo de Coleta Seletiva. O seu objetivo geral é desenvolver um software capaz de automatizar o processo de classificação visual realizado na coleta seletiva, por meio da aplicação de técnicas de visão computacional e aprendizado de máquina. O objetivo proposto deve ser alcançado através de pesquisa científica teórica, conhecimentos da biblioteca de visão computacional OpenCV, da biblioteca de aprendizado de máquina TensorFlow e a linguagem de programação Python. Como resultado tem-se um software de classificação que poderá ser aplicado no reaproveitamento de materiais recicláveis, de modo a colaborar para o desenvolvimento sustentável.

Palavras-chave: Aprendizado de Máquina. Coleta Seletiva. Desenvolvimento de Software. Resíduos Sólidos. Sustentabilidade. Visão Computacional.

ABSTRACT

This work deals with the use of computer vision libraries and machine learning in the identification of solid waste (metal and glass) applied to the selective waste collection process. Its general objective is to develop a software capable of automating the process of visual classification performed in the selective collection, through the application of computer vision and machine learning techniques. The proposed objective should be achieved through theoretical scientific research, knowledge of the OpenCV computer vision library, the TensorFlow machine learning library and the Python programming language. As a result there is a classification software that can be applied to the reuse of recyclable materials, in order to collaborate for sustainable development.

Keywords: *Computer vision. Machine learning. Selective Waste Collection. Software development. Solid waste. Sustainability.*

1. INTRODUÇÃO

A sociedade contemporânea está em constante evolução e em consequência disso as tecnologias estão cada dia mais avançadas. Em contrapartida o meio ambiente sofre as consequências devido à alta geração lixo, o que acarreta a criação de meios sustentáveis e efetivos para sanar esse problema.

O meio ambiente pode ser definido como um " conjunto de condições, leis, influencias e interações de ordem física, química e biológica que cerca e afeta a existência, o desenvolvimento e o bem-estar de um ser vivo ou uma comunidade. " (TASSARA, 2008, p. 128), ou seja, todos os aspectos que permitem, abrigam e reagem a vida em todas as suas formas.

A sustentabilidade de acordo com a autora Tassara (2008) é a capacidade de desenvolver processos produtivos e gerar riquezas a através dos recursos naturais sem provocar o esgotamento da natureza ou degradação socioambiental, que pode alcançar sua plenitude com a coletivização das ações individuais e comunitárias gerando sociedades sustentáveis.

Segundo Townsend (2010, p.438) "Para alcançar a sustentabilidade ou até mesmo aproximar-se dela, é necessário mais que vontade – é necessário a compreensão ecológica, cuidadosamente adquirida e mais ainda aplicada. ", essa afirmação implica que a sociedade deve buscar constantemente conhecimentos sobre meio ambiente e sustentabilidade, para que então seja possível tomar medidas eficientes em prol da preservação ambiental.

A coleta seletiva é uma das alternativas para reduzir o excesso de lixo, fazendo com que ele retorne ao consumo através de sua reciclagem. Segundo o Ministério do Meio Ambiente (2017):

> Coleta seletiva é a coleta diferenciada de resíduos que foram previamente separados segundo a sua constituição ou composição. Ou seja, resíduos com características similares são selecionados pelo gerador (que pode ser o cidadão, uma empresa ou outra instituição) e disponibilizados para a coleta separadamente.

Por ser um processo manufaturado, os trabalhadores deste setor estão sujeitos a riscos de contaminação, doenças, etc., entretanto a tecnologia pode colaborar para a automatização da coleta seletiva, reduzindo riscos e tornando o processo mais efetivo.

Dados de um estudo do Instituto de Pesquisa Econômica Aplicada (IPEA) publicado no ano de 2012 mostram que no Brasil são coletadas diariamente cerca de 183,5 mil

toneladas de resíduos sólidos, onde deste total apenas 31,9 % podem ser reaproveitados. Existem alguns processos que visam solucionar este problema de modo que o resíduo produzido possa ser reutilizado e retornar ao uso da população, a coleta seletiva é um dos mais importantes processos em prol da sustentabilidade, pois ela consiste na separação e recolhimento dos resíduos descartados todos os dias.

Há cerca de cinco décadas, a sociedade foi despertada para a consciência de que havia fatores que estavam causando a rápida e progressiva destruição do equilíbrio ecológico do planeta (MANO et al., 2010), sendo assim é necessária a criação de alternativas viáveis, inteligentes e de custo acessível que coloquem em evidência o desenvolvimento sustentável. Portanto a tecnologia pode colaborar para o desenvolvimento e melhora do processo de coleta seletiva, que atualmente é manufaturado e pode trazer sérios riscos aos trabalhadores que participam diretamente desta atividade.

Nos últimos anos a evolução da computação gráfica e inteligência artificial tem permitido que computadores possam extrair e entender informações de imagens, segundo Conci (2008, p.7) “o computador pode avaliar e representar como imagem informações que de outra forma não seriam interpretáveis”, viabilizando o reconhecimento de propriedades e características de objetos do mundo real e as transformando em dados que podem ser processados e manipulados para diversas finalidades.

O crescimento dos estudos em Visão Computacional e Aprendizado de Máquina culminou no desenvolvimento de várias bibliotecas voltadas para o desenvolvimento de sistemas, diferindo entre si por características como: linguagem de programação, funcionalidades, e facilidade de uso.

A biblioteca de visão computacional OpenCV (*Open Source Computer Vision Library*) é uma biblioteca de código aberto que possui algoritmos que podem ser utilizados em diversas situações, tais como na identificação de rostos, identificação de objetos, classificar ações, etc.

TensorFlow é uma biblioteca de aprendizado de máquina de código aberto desenvolvida pela Google, que teve seu projeto desenvolvimento baseado em redes neurais, permitindo que aplicações computacionais aprendam com ações e ampliem sua capacidade de decisão.

A linguagem de programação Python é de uso geral que detém uma rica biblioteca, permitindo a criação de aplicações de código simples e entendível, possuindo alta compatibilidade e fácil integração com outras bibliotecas, tais como: OpenCV e TensorFlow.

O tema deste projeto aborda a utilização da biblioteca de visão computacional OpenCV, a biblioteca de aprendizado de máquina TensorFlow e a linguagem de programação Python no desenvolvimento de um software que seja capaz de identificar e classificar determinados tipos de resíduos sólidos (metal e vidro). Um sistema dessa natureza pode gerar uma melhora significativa no processo de reaproveitamento destes materiais que vem sendo acumulados de modo irregular nas cidades, viabilizando seu retorno à sociedade como matéria-prima sustentável colaborando com o meio ambiente e a geração de renda.

2. OBJETIVOS DA PESQUISA

2.1. Objetivo Geral

Desenvolver software baseado em bibliotecas de Visão Computacional e Aprendizado de Máquina para a identificação e classificação de resíduos sólidos (metal e vidro).

2.2. Objetivos Específicos

- Compreender as propriedades ópticas do metal e vidro;
- Utilizar a biblioteca OpenCV e TensorFlow no desenvolvimento do software;
- Apresentar os resultados obtidos.

3. JUSTIFICATIVA

A coleta seletiva é um importante processo que permite a separação de resíduos de acordo com sua composição ou constituição, podendo ser classificados como: papéis, plásticos, metais e vidros (MINISTÉRIO DO MEIO AMBIENTE, 2017).

Por se tratar de um método de reaproveitamento do lixo a coleta seletiva tem um importante papel no que diz respeito à responsabilidade socioambiental, pois segundo Singer (2002) a coleta seletiva contribui significativamente para a sustentabilidade urbana, e vem incorporando um perfil de inclusão social e geração de renda para os setores mais carentes da população. Portanto quanto mais eficaz for o reaproveitamento do lixo maiores serão os benefícios gerados.

A fase de classificação e separação dos resíduos sólidos consiste em um procedimento manual realizado por humanos, se assemelhando com linhas de produção industrial. Por ser realizado por humanos esse procedimento é suscetível a falhas de operação, que geram perda de tempo e resultados inesperados. A otimização do processo garante também proteção às pessoas que estão sujeitas a algum tipo de contaminação por estar em contato com o lixo, ou sofrer um ferimento devido o manuseio incorreto de algum material.

Rosário (2005) afirma que caso queiramos o crescimento sustentável da sociedade, capaz de garantir o aumento da qualidade de vida sem desperdiçar em longo prazo os recursos disponíveis é necessário o estudo e implantação de sistemas de automação.

Dado este cenário, pode-se perceber que a implantação de tecnologias de automação e reconhecimento por software de visão computacional e aprendizado de máquina poderá auxiliar na otimização do processo, gerando melhor aproveitamento de tempo e maximizando os resultados obtidos, por exemplo, o ato de pegar um determinado sólido, pode durar alguns segundos sendo realizado manualmente, enquanto que em um processo assistido por computador a mesma ação precisaria apenas de milissegundos para ser realizada.

Aplicações baseadas em bibliotecas de visão computacional, aprendizado de máquina e algoritmos de processamento de imagens se fazem presentes em vários cenários, como: indústrias, sistemas de segurança e monitoramento, e até mesmo na medicina, servindo como uma importante ferramenta de automatização em processos que exigem interpretações baseadas em características e informações visuais.

Devido ao custo agregado ao desenvolvimento e implantação de sistemas automatizados, os resultados obtidos podem ser melhor aproveitados por instituições que estejam dispostas a investir em tecnologias sustentáveis.

A implantação de um sistema como o descrito acima pode ser uma solução para empresas que decidam investir na obtenção de matéria-prima a partir de materiais recicláveis, com um método automatizado, sem intervenção humana e menos vulnerável a falhas, contudo qualquer avanço gerado por estudos científicos ou mesmo o desenvolvimento de projetos com foco comercial podem ser fundamentais para permitir que a tecnologia produza aprimoramento neste cenário e colabore para o desenvolvimento sustentável.

4. REFERENCIAL TEÓRICO

As seções a seguir neste capítulo reúnem os fundamentos teóricos gerados por outros autores e que foram essenciais para o desenvolvimento deste projeto.

4.1. Resíduos sólidos - Metal e Vidro

Dentre os materiais recicláveis coletados na coleta seletiva nesta pesquisa destacam-se o metal e o vidro, materiais de alta aplicabilidade em diversas áreas tecnológicas que são descartados em grandes quantidades a cada dia e que possuem a característica de serem mais rentáveis quando são reciclados.

4.1.1. Propriedades ópticas do metal e vidro

Os metais segundo Smith e Hashemi (2012) são substâncias inorgânicas compostas de um ou mais elementos metálicos, podendo também conter alguns elementos não metálicos. Alguns exemplos de elementos metálicos são: ferro, cobre, alumínio, níquel e titânio.

Os metais são materiais altamente refletivos e opacos, ou seja, são materiais que devido sua densidade não permitem que a luz ultrapasse sua estrutura, entretanto o autor Callister Junior (2006) afirma que a maior parte da luz absorvida na superfície do metal é reemitida na forma de luz visível e que eles são transparentes para as radiações de alta frequência (raios X e γ) como demonstrado na Figura 1. Dada a opacidade dos metais e sua boa reflexividade, a cor percebida é determinada pela distribuição dos comprimentos de onda da radiação refletida.

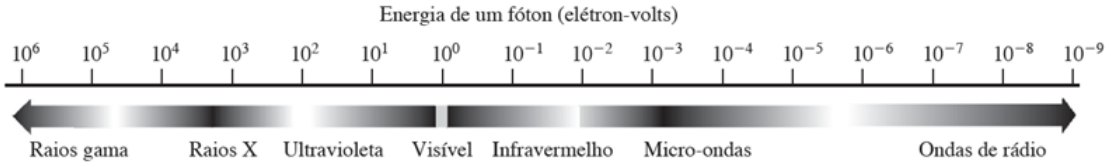
Ainda segundo Callister Junior (2006), se o metal tiver uma aparência prateada brilhante quando exposto a luz branca, isso significa uma alta reflexão ao longo de toda faixa do visível, o que significa as composições de fótons dos feixes incidentes e refletidos são similares, por exemplo, o alumínio e a prata.

Já os vidros segundo Smith e Hashemi (2012) são materiais cerâmicos, não metálicos, compostos de materiais inorgânicos a altas temperaturas, podendo ser definidos como um produto inorgânico da fusão que resfriou a uma condição rígida sem cristalização.

O autor Callister Junior (2006) define que a transparência de um vidro está diretamente relacionada à sua estrutura atômica, eles exibem cor devido a uma absorção seletiva de faixas de comprimentos de onda da luz, onde a cor observada é o resultado da combinação

dos comprimentos de onda transmitidos. Caso a absorção de luz seja uniforme em todos os comprimentos de onda, o material poderá ser incolor.

Figura 1. Espectro eletromagnético



Fonte: Adaptado de Gonzalez (2000).

4.1.2. Refletância e Absorção

Todo material tem a capacidade de absorver energia na forma de luz, até certo ponto, e a este fenômeno dá se o nome de absortividade, isso ocorre "em virtude das interações dos fótons de luz com as estruturas eletrônica e de ligações dos átomos, íons, ou moléculas que compõem o material" (SMITH, 2012, p. 584).

Em relação a absortividade nos metais pode-se afirmar que estes "refletem e/ou absorvem fortemente a radiação incidente desde comprimentos de onda longos (ondas de rádio) até a região intermediária da faixa do ultravioleta" (SMITH, 2012, p.584).

Já nos vidros, de acordo com Smith (2012), a proporção de luz que é refletida por uma das superfícies de um vidro polido é muito pequena, devido a capacidade deste material de transmitir a luz através de seu corpo. Contudo o fator de absortividade do vidro está diretamente ligado com a direção na qual a luz incide sobre o objeto, e também com a taxa de desvio sofrida pelos raios de luz enquanto permeiam o material.

Dadas as afirmações acima pode-se afirmar que a quantidade de luz que é refletida (refletividade) na superfície de um metal é maior do que a quantidade refletida na superfície de um vidro, conforme demonstrado na Tabela 1. A diferença entre os índices de refletividade dos materiais se dá devido suas capacidades de absorção de energia na forma de luz, sua composição e estruturas eletrônicas.

Tabela 1. Índice de refletividade do metal e vidro

Material	*Índice de Refletividade (R)*
Metal	R ≅ 0,90 – 0,95
Vidro	R ≅ 0,05

Fonte: Adaptado de Callister Junior (2006).

4.2. Visão Computacional

A visão computacional é uma área de estudo dentro da computação gráfica que está relacionada a análise de imagens.

Esta área de estudo tem apresentado grande desenvolvimento devido o número significativo de aplicações desta tecnologia e sua capacidade de permitir que um computador extraia e entenda informações de imagens, permitindo utilizar técnicas de inteligência artificial que possibilitam o reconhecimento de padrões e o aprendizado de máquina, podendo ser aplicados para reconhecimento de pessoas, faces e objetos, inspeção de peças em linhas de montagem, orientação de movimento de robôs etc. (CONCI et al., 2008).

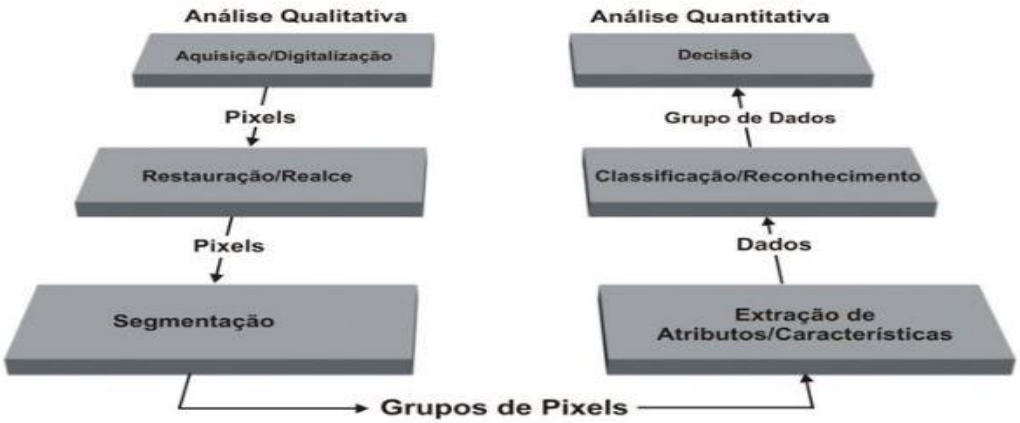
O desenvolvimento da área de estudo da ciência da computação que trata da simulação da cognição, a inteligência artificial, viabiliza a criação de softwares cada vez mais avançados, capazes de perceber visualmente o ambiente onde estão inseridos. Dado este contexto, Conci et al. (2008, p. 5) definem a visão computacional com sendo "o domínio da ciência da computação que estuda e aplica métodos que permitem aos computadores 'compreenderem' o conteúdo de uma imagem".

Pode-se perceber que agora a capacidade de visão e percepção do mundo real não é restrita apenas aos seres vivos, graças ao desenvolvimento da área de estudo da visão computacional, permitindo em muitos cenários automatizar processos de interpretação visual simples ou complexos.

4.2.1. Etapas de um Sistema de Visão Computacional

A Figura 2 mostra um diagrama de blocos que contêm as principais etapas de um sistema de visão computacional, considerando que cada amostra de uma determinada imagem é chamada de pixel.

Figura 2. Diagrama de blocos: Etapas de um sistema de visão computacional.



Fonte: Conci et al. (2008, p. 51).

A seguir serão descritas cada uma das etapas de um sistema de visão computacional genérico.

- **Aquisição de imagens**

A aquisição de imagens é o processo de obtenção de uma imagem digital, que pode ser através de scanners, câmeras digitais, sensores remotos, etc. Segundo Gonzalez (2007) a aquisição de imagens digitais é composta por dois elementos: o primeiro é um dispositivo físico que seja sensível a uma banda do espectro de energia eletromagnética, como por exemplo o infravermelho, e que seja capaz de produzir um sinal de saída proporcional a um nível de energia percebida; o segundo é um dispositivo que converte o sinal elétrico para a forma digital.

- **Restauração e realce**

A restauração e realce são dois processos importantes no processamento de imagens digitais. De acordo com Conci (2008, p. 55) "a restauração busca compensar deficiências especificas geradas no momento de aquisição, na transmissão ou em alguma etapa do processamento. ". Já no realce o objetivo é "destacar detalhes da imagem que são de interesse para analise ou que tenham sofrido alguma deterioração. " (CONCI, 2008, p.55).

Os métodos de restauração procuram modelar a distorção e aplicar o processo reverso, enquanto os métodos de realce utilizam a resposta do sistema visual humano para "melhorar" a imagem visualmente.

- **Segmentação**

A segmentação de imagem consiste em dividir a imagem em regiões que dizem respeito ao mesmo conteúdo e aplicação, procurando "isolar regiões de pontos da imagem pertencentes a objetos para posterior extração de atributos e cálculo de parâmetros descritivos." (CONCI, 2008, p. 55).

- **Extração de características**

A partir da imagem já segmentada é possível obter atributos das regiões destacadas. Os tipos de atributos ou características mais comuns segundo Conci (2008) são: número total de objetos, dimensões, geometria, luminosidade e textura.

- **Classificação e reconhecimento**

A função do processo de classificação e reconhecimento consiste em distinguir os objetos na imagem agrupando estes parâmetros de acordo com sua semelhança para cada região de pixels encontrados. A partir da classificação dos objetos novos objetos podem ser reconhecidos, sendo possível tomar decisões e relatar fatos relacionados ao mundo real (CONCI, 2008).

- **Decisão**

Os sistemas de visão computacional têm como objetivo tomar decisões a partir da extração de informações de imagens. “A tomada de decisão pode ser feita a partir de indagações simples a respeito de parâmetros extraídos dos objetos ou de algoritmos mais complexos de inteligência artificial. ” (CONCI, 2008, p. 57).

Neste projeto a tomada de decisão a respeito da classificação dos materiais será realizada através do aprendizado de máquina, que será implementado utilizando a biblioteca TensorFlow.

4.3. Inteligência Artificial

Em aplicações da natureza deste projeto são necessários conhecimentos em Inteligência Artificial (IA), que segundo Coppin (2013) pode ser definida como um estudo dos sistemas que agem de um modo que um observador qualquer possa parecer inteligente, envolvendo a utilização de métodos baseados no comportamento inteligente de humanos e outros animais para solucionar problemas complexos.

A inteligência artificial tem como objetivo capacitar o computador a executar funções que são desempenhadas pelo ser humano usando conhecimento e raciocínio (REZENDE, 2005). E para que seja possível almejar uma ação inteligente, é necessário que sejam analisados todos os aspectos relativos ao desenvolvimento da inteligência.

No campo da inteligência artificial existem duas teses: IA forte e IA fraca. Os seguidores da IA forte acreditam que, quando se possui um computador com capacidade de processamento suficiente para suportar uma inteligência, pode-se então obter um sistema capaz de, literalmente pensar e ser consciente, assim como o ser humano (COPPIN, 2013).

Entretanto muitos pesquisadores desta área consideram o conceito de IA forte como uma ideia infundada e absurda, pois a possibilidade de se criar um robô com emoções e consciência equivalentes à dos humanos é explorada no âmbito da ficção cientifica, e segundo Coppin (2013) é raramente considerado como um objetivo real da inteligência artificial.

Por outro lado, “IA fraca é simplesmente a visão de que comportamento inteligente pode ser modelado e utilizado por computadores para solucionar problemas complexos. ” (COPPIN, p.5, 2013). Este ponto de vista defende que apenas o fato de um computador agir de modo inteligente não prova que ele seja realmente inteligente no sentido humano.

Contudo, pode se concluir que a IA fraca está relacionada com a construção de máquinas ou softwares de certa forma inteligentes, porém não são capazes de raciocinar por si próprios, enquanto que a IA forte está relacionada à criação de máquinas que tenham autoconsciência e que possam pensar, e não somente simular raciocínios.

4.3.1. Principais áreas da Inteligência Artificial

A área de conhecimento da inteligência artificial é composta por subáreas, que por sua vez abrangem diversas aplicações em diferentes campos de estudo. O autor Artero (p.19, 2009) destaca as seguintes subáreas e suas aplicações:

- **Robótica**

Dispositivos robóticos têm se tornado cada vez mais importantes na indústria, sendo capazes de realizar tarefas que exigem grande força, precisão, podendo também operar em ambientes perigosos. Atualmente a robótica tem buscado introduzir algum tipo de inteligência aos dispositivos, fazendo com que sejam capazes de tomar decisões em determinados ambientes.

- **Visão por computador**

Tem o objetivo de desenvolver em um computador a habilidade de extrair informações do ambiente a partir de imagens obtidas.

- **Processamento de linguagem natural**

A habilidade linguística é uma das características mais importantes das pessoas, e dotar as maquinas desta aptidão é uma ambição antiga, o que tornaria possível a tradução online de conversas entre pessoas de diferentes etnias.

- **Sistemas especialistas**

São sistemas que a partir do processamento de uma base de conhecimentos, conseguem fornecer respostas sobre um determinado domínio a um usuário. Na prática, auxiliam a tomada de decisões em diversas áreas, por exemplo: medicina, engenharia,

matemática, etc. Devido suas características, esta área se tornou uma das mais populares e com maior número de aplicações de sucesso.

- **Reconhecimento de padrões**

Esta área abrange o desenvolvimento de aplicações tais como: reconhecimento de faces, gestos, escrita e fala, baseando-se na análise de padrões em uma determinada base de dados.

- **Base de dados inteligentes**

Consiste em adicionar às bases de conhecimento a habilidade de raciocinar, de modo que seja possível gerar resultados novos. Esta área abrange técnicas de mineração de dados.

- **Prova de teoremas**

Embora os teoremas sejam conhecidos apenas na matemática, as estratégias usadas para as suas demonstrações podem ser empregadas na solução de problemas em diversas áreas, pois, consiste na definição de sequencias lógicas de ações que levam de uma situação inicial (hipótese) até o objetivo final (tese).

- **Jogos**

A aplicação de IA em jogos envolve o uso de estratégias e raciocínio, semelhantes à de seres humanos, por este motivo a área de desenvolvimento de jogos tem sido um importante laboratório para a criação e estudo de várias técnicas de IA.

4.4. Aprendizado de Máquina

No passado a área de inteligência artificial era vista como uma área teórica, com aplicações apenas em pequenos problemas curiosos, desafiadores, mas de pouco valor prático. Boa parte dos problemas práticos que precisam de computação era resolvida pela codificação em alguma linguagem de programação dos passos necessários para a sua solução (FACELI et al., 2011).

> Um agente estará aprendendo se melhorar seu desempenho nas tarefas futuras de aprendizagem após fazer observações sobre o mundo. A aprendizagem pode variar do corriqueiro, como anotar um número de telefone, até o profundo, como mostrado por

> Albert Einstein, que inferiu uma nova teoria para o universo (RUSSEL E NORVIG, p.605, 2013).

Segundo Faceli et al. (2011) no aprendizado de máquina os computadores são programados para aprender com a experiência passada utilizando o princípio da indução, no qual se obtém conclusões genéricas a partir de um conjunto particular de exemplos. Sendo assim, a partir de um algoritmo de aprendizado de máquina um computador pode reconhecer padrões através da análise de um conjunto de dados, podendo ser capaz de tomar decisões baseadas em experiências acumuladas por meio da solução bem-sucedida de problemas anteriores.

4.4.1. Treinamento

Em grande parte dos problemas de aprendizado, o objetivo é fazer que um determinado agente inteligente aprenda a classificar dados de entrada de acordo com um conjunto classificações.

O sistema de aprendizado pode possuir um conjunto de dados de treinamento que foram classificados manualmente, possibilitando que o sistema aprenda a classificar estes dados e também novos dados ainda não observados desde que estejam inseridos no contexto de aplicação (COPPIN, 2013).

> Aprender a classificar dados desconhecidos claramente pressupõe que haja uma relação entre os dados e as classificações – em outras palavras, alguma função *f* poderá ser gerada se um grupo de dados *x* pertencer a classificação *y*. Então $f(x) = y$. (COPPIN, p.234, 2013).

4.4.2. Aprendizado de máquina supervisionado e não supervisionado

Redes de aprendizado supervisionado aprendem ao serem apresentadas a um conjunto de dados de treinamento previamente classificados, segundo Coppin (p. 249, 2013) “redes neuronais que usam aprendizado supervisionado aprendem a modificarem os pesos das conexões de suas redes, para classificar mais precisamente os dados de treinamento”.

Ainda sobre a aprendizagem supervisionada Russel e Norvig (2013) definem que o agente observa alguns exemplos de pares de entrada e saída e aprende uma função que encontra um caminho da entrada para a saída. Os autores citam como exemplo uma câmera de tráfego, onde as entradas para o sistema são as imagens dos veículos obtidas, e as saídas vem de um

agente instrutor que diz “isso é ônibus”, com o passar do tempo o sistema aprenderá a reconhecer um ônibus sozinho sempre quando o ver.

Uma outra técnica de aprendizado de máquina é o aprendizado não supervisionado, onde aprendem sem qualquer intervenção humana. Este método é útil em situações nas quais os dados devem ser classificados ou agrupados em um conjunto de classificações, onde estas não são previamente conhecidas (COPPIN, 2013).

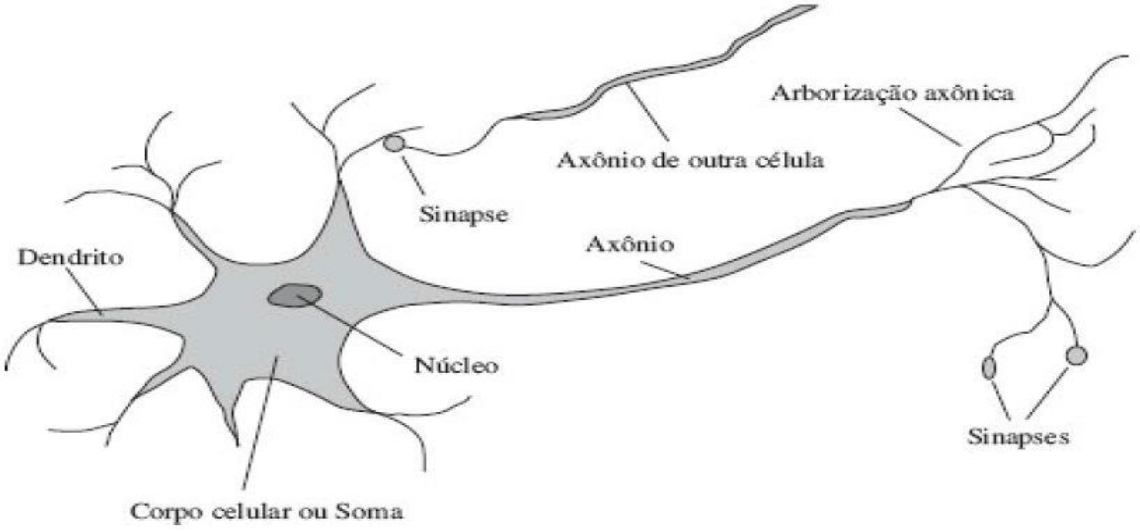
Russel e Norvig (p. 606, 2013) conceituam que “na aprendizagem não supervisionada, um agente aprende padrões na entrada embora não seja fornecido nenhum *feedback* explícito”. Os autores citam como exemplo o caso de um agente condutor de táxi que desenvolve um conceito de “dia de tráfego bom” e “dia de tráfego ruim” sem nunca ter sido rotulados exemplos de cada um deles por um instrutor.

Neste projeto a aplicação de aprendizado de máquina é fundamental para que o sistema, após o treinamento com um conjunto de imagens de cada material (metal e vidro), seja capaz de classificar corretamente o tipo do resíduo sólido apresentado na imagem analisada.

4.5. Redes Neurais

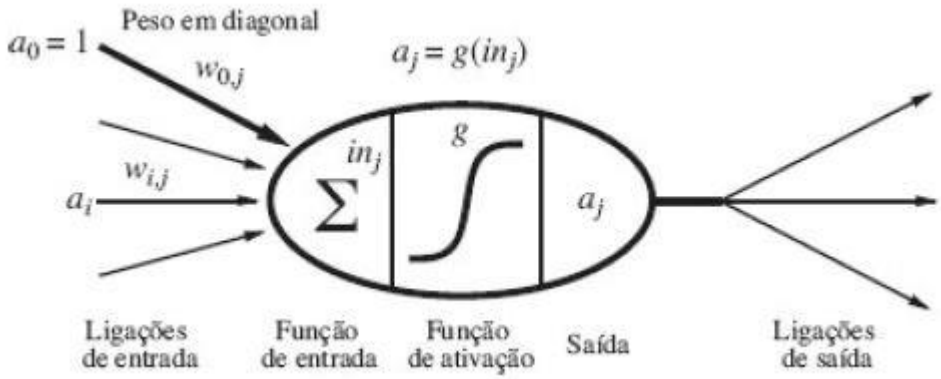
A atividade mental consiste basicamente na atividade eletroquímica em redes de células cerebrais chamadas neurônios conforme demonstrado na Figura 3. Esta hipótese inspirou vários trabalhos no campo da Inteligência Artificial com objetivo de criar redes neurais artificiais. A Figura 4 apresenta um modelo matemático simples de um neurônio artificial desenvolvido em 1943 (RUSSELL e NORVIG, 2013).

Figura 3. Representação de um neurônio.



Fonte: Russell e Norvig (2013, p.12).

Figura 4. Modelo matemático de um neurônio artificial



Fonte: Russell e Norvig (2013, p.635).

As Redes Neurais, ou redes neurais artificiais, representam uma tecnologia com raízes em muitas disciplinas, tais como: neurociência, matemática, estatística, ciência da computação e engenharia. Haykin (2001) explica que as redes neurais encontram aplicações em campos tão diversos, como modelagem, análise de séries temporais, reconhecimento de padrões, processamento de sinais e controle, em virtude de sua capacidade de aprender com ou sem um professor.

Segundo Artero (2009) apesar do fato que as redes neurais possam ser utilizadas para resolver vários problemas práticos, a sua maior utilização é na resolução de problemas que podem ser classificados como o reconhecimento de padrões, o que inclui uma ampla gama de aplicações, como o reconhecimento de imagens (faces, impressões digitais), voz, e outros objetos.

A aplicação de redes neurais neste projeto é parte importante no processo de treinamento e classificação, pois elas possibilitam o reconhecimento dos materiais através da biblioteca TensorFlow.

4.5.1. Redes Neurais Convolucionais

Rede neural convolucional (CNN) é uma das abordagens de aprendizado de máquina mais utilizadas, principalmente devido à sua eficiência na aprendizagem dos padrões extraídos diretamente dos dados bidimensionais, bem como a sua flexibilidade em termos de variação de translação ou distorções locais (LECUN et al., 1998).

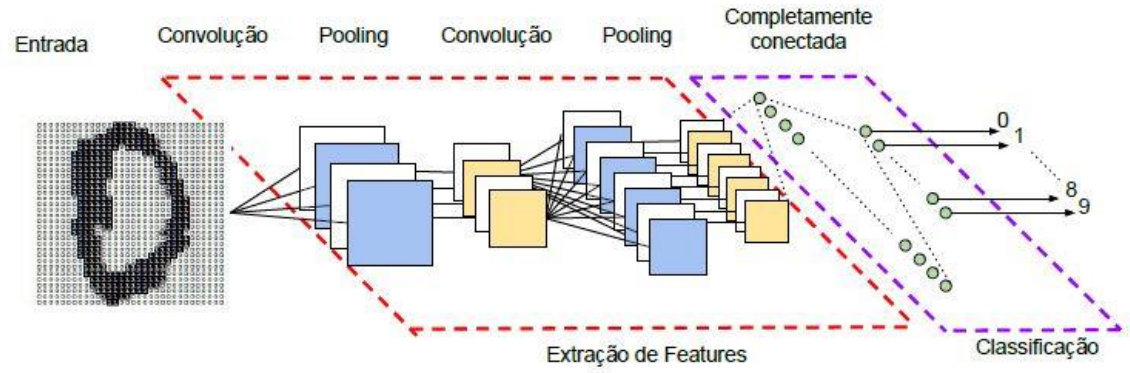
Com base no descrito acima, uma abordagem de identificação e classificação automática é necessária para obter informações detalhadas sobre diferentes tipos de resíduos sólidos de forma rápida e confiável.

A Figura 5 ilustra a estrutura de uma rede neural convulocional, onde a entrada é uma determinada região de uma imagem que passa pelos processos de convolução e *pooling*.

No processo de convolução são aplicados diferentes filtros na imagem e durante o processo de *pooling* ocorre a extração de características da imagem após a convolução.

Este tipo de rede vem sendo amplamente utilizada, principalmente nas aplicações de classificação, detecção e reconhecimento em imagens e vídeos.

Figura 5. Exemplo de uma rede neural convolucional.



Fonte: Adaptado de Lecun et al. (1998).

Os resíduos sólidos observados, embora sejam classificados em apenas duas classes, podem possuir variação significativa quanto à forma, tamanho e luminosidade, sendo este um fator que pode prejudicar a eficácia do processo de classificação do objeto. Por este motivo é fundamental a utilização de redes neurais convulacionais, que são altamente adaptáveis em um conjunto de imagens que apresentem algum tipo de distorção em relação as demais, assim como ocorre no ambiente de coleta seletiva, demonstrado na Figura 6.

Figura 6. Exemplo de deformações em latas de metal.

Fonte: Google Imagens[1]

[1] Disponível em: < https://bit.ly/2ruJv3F> Acesso em: 10 maio 2018.

4.6. Tecnologias de desenvolvimento

Para o desenvolvimento do software proposto é necessário a utilização de algumas tecnologias, tais como: linguagem de programação, bibliotecas de visão computacional e aprendizado de máquina, onde cada um deles possuem um papel de extrema importância para que o software seja criado. A seguir tem-se a fundamentação das tecnologias utilizadas: OpenCV, TensorFlow, Python e LabelImg.

4.6.1. A biblioteca de visão computacional OpenCV

As bibliotecas de software são um conjunto de funções prontas que servem de ferramenta para programadores durante o processo de desenvolvimento de um determinado sistema, garantindo que o programador não tenha necessidade de criar algumas ações do zero.

Existem diversas bibliotecas de software disponíveis para o desenvolvimento de sistemas baseados em visão computacional, que se diferem por características como: linguagem de programação, funções de tempo real ou não, e facilidade de uso.

O *OpenCV (Open Source Computer Vision Library)* é uma biblioteca de código aberto e de uso livre voltada para sistemas de visão por computador e aprendizagem de máquina. A biblioteca possui algoritmos otimizados que podem ser usados para detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de câmera, rastrear objetos em movimento, extrair modelos 3D de objetos, encontrar imagens semelhantes a partir de uma base de dados de imagens, dentre diversas outras funcionalidades.

Apesar de ser desenvolvida nativamente na linguagem C++ a biblioteca OpenCV permite o interfaceamaento com as linguagens de programação C, JAVA, Python e Matlab, além de ser totalmente portável sendo compatível com Linux, Android, Windows, e Mac OS.

Várias empresas de tecnologia ao redor do mundo utilizam o OpenCV em seus sistemas, em diversos segmentos, e com aplicações científicas ou de mercado. São algumas delas: Google, Toyota, Intel, IBM, Microsoft e Sony.

A escolha desta biblioteca para o desenvolvimento deste trabalho é baseada na grande comunidade de usuários da biblioteca, fator que facilita na busca de material sobre o assunto. Outra importante característica é sua portabilidade, que possibilita a interface com outras linguagens de programação.

4.6.2. A biblioteca de aprendizado de máquina TensorFlow

TensorFlow é uma biblioteca de código aberto desenvolvida pela Google, destinada a inteligência artificial, onde todo o seu projeto e desenvolvimento ocorreu através de estudos de redes neurais.

A biblioteca utiliza o aprendizado de máquina, permitindo que aplicações computacionais aprendam com ações e ampliem suas capacidades de decisão. É uma ferramenta de código, fator que contribui para o desenvolvimento de soluções baseadas em aprendizado de máquina de maneira acessível para os desenvolvedores.

Neste projeto o TensorFlow será utilizado no processo de treinamento e reconhecimento dos resíduos sólidos (metal e vidro) a partir de um conjunto de imagens dos mesmos.

4.6.3. A linguagem de programação Python

Python é uma linguagem de uso geral, projetada especificamente para tornar os programas bastante legíveis. Possui uma rica biblioteca, fator que torna possível a criação aplicações sofisticadas usando código de maneira simples e entendível. Por esses motivos, Python tornou-se uma linguagem de desenvolvimento de aplicações popular e também uma preferência como “primeira” linguagem de programação (PERKOVIC, 2016).

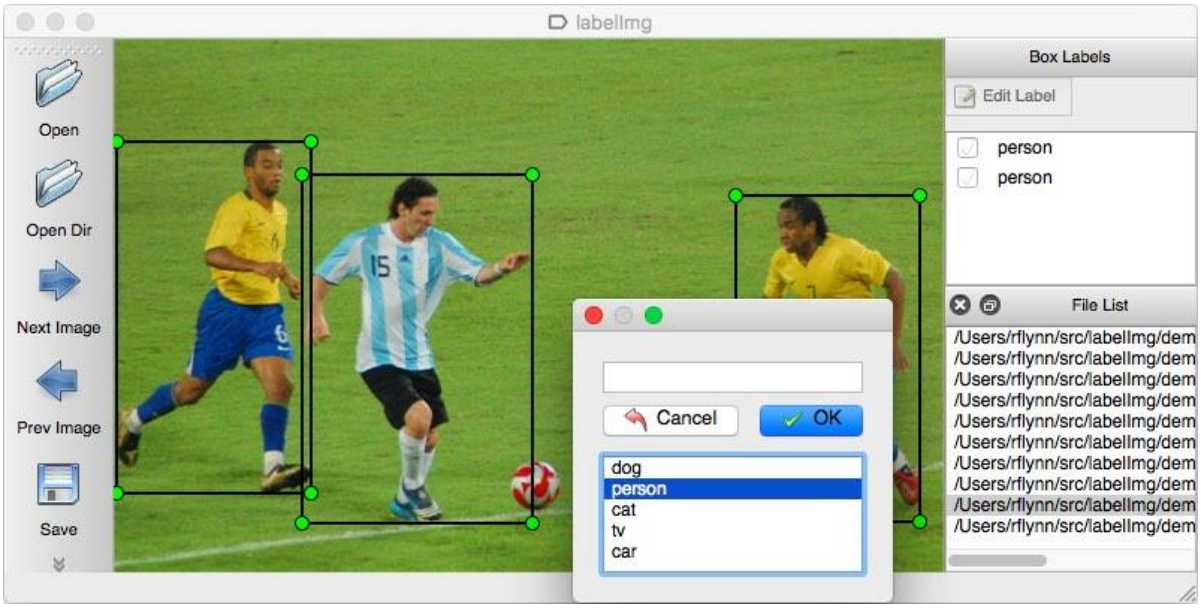
A linguagem Python foi escolhida para ser utilizada neste projeto, devido as características citadas anteriormente e também pelo fato de ser compatível e de fácil integração com as bibliotecas TensorFlow e OpenCV.

4.6.4. LabelImg

LabelImg é uma ferramenta *Open Source* de anotação de imagens gráficas e mapeamento de objetos, desenvolvido em Python e Qt (framework multiplataforma para desenvolvimento de interfaces gráficas em C++), onde as anotações produzidas são salvas como arquivos XML.

O Arquivo XML produzido contém informações importantes a respeito dos objetos mapeados nas imagens, como por exemplo: largura, altura, delimitação de pixels e classe de objetos, conforme ilustrado na Figura 7. Essas informações são utilizadas durante o treinamento de um algoritmo de aprendizado de máquina.

Figura 7. Ferramenta LabelImg.



Fonte: Página LabelImg[2]

[2] Disponível em: <https://github.com/tzutalin/labelImg> Acesso em: 09 maio 2018.

5. DESENVOLVIMENTO

Nas seguintes seções são descritas as etapas aplicadas no desenvolvimento e implementação do sistema proposto.

5.1. Ambiente de desenvolvimento

Durante a fase de desenvolvimento do projeto foi utilizado um computador *desktop* cujas configurações estão expressas na Tabela 2.

Tabela 2. Configurações do computador utilizado no desenvolvimento.

Recurso	*Quantidade / Especificação*
Memória RAM	8 GB
Disco rígido	1 TB
CPU	Intel i3, 2.5 GHz
Sistema Operacional: Linux Ubuntu 16.04 LTS	

Fonte: Autores da pesquisa.

5.2. Aquisição das imagens

Os desenvolvedores do TensorFlow[3] recomendam o uso de pelo menos 100 imagens de cada classe de itens, para que seja possível obter bons resultados na fase de treinamento e garantir a boa acurácia na fase de classificação do objeto, entretanto é recomendado que o tamanho das mesmas não sejam grandes, pois demandam mais recursos computacionais na fase de treinamento.

A variedade de resíduos de metal e vidro que podem ser encontradas no lixo é muito grande, por este motivo foram selecionados aleatoriamente imagens de objetos que são bem aproveitados na reciclagem. As imagens foram obtidas manualmente através da plataforma Google Imagens.

A Tabela 3 mostra os tipos de objetos obtidos nas imagens, sua respectiva classe, e o número de imagens por classe.

[3] Disponível em: <https://www.tensorflow.org/tutorials/image_retraining> Acesso em: 16 maio 2018.

Tabela 3. Base de dados para treinamento

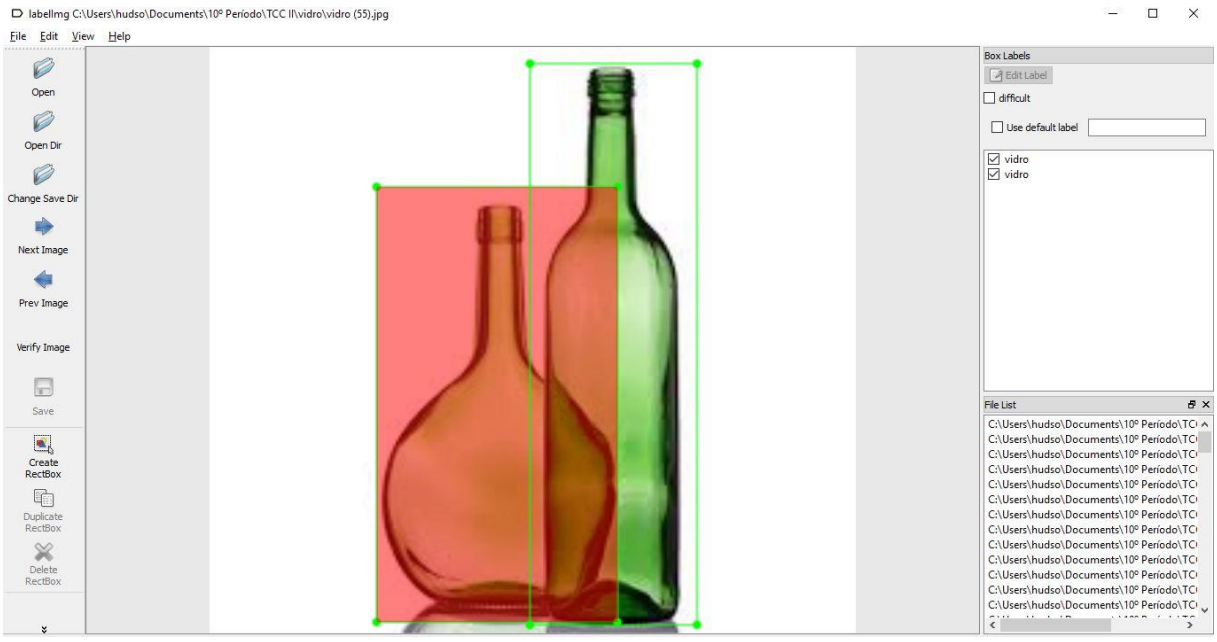
Classe do objeto	*Especificação*	*Número de Imagens por classe*
Metal	• Latinhas • Tampas • Lacres	117
Vidro	• Garrafas • Copos • Vidro quebrado	101
		Total de imagens obtidas: 218

Fonte: Autores da pesquisa.

5.3. Mapeamento de imagens

A técnica de aprendizado de máquina a ser utilizada é a supervisionada, dada a sua baixa complexidade no que diz respeito a sua implementação, por este motivo é necessário que um agente externo indique qual a classe do objeto em uma determinada imagem. Para realizar este procedimento foi necessário mapear os objetos de interesse em cada uma das imagens, conforme sua classe, este processo durou cerca de 24 horas. A Figura 8 demonstra o mapeamento feito com o software LabelImg.

Figura 8. Mapeamento de imagem com LabelImg.



Fonte: Autores da pesquisa.

Para cada imagem de entrada, o software LabelImg gera um arquivo de saída no formato XML, que contém informações sobre as delimitações em pixels dos objetos, suas dimensões e caminho da imagem. A Figura 9 representa o arquivo gerado após mapear os objetos de vidro contidos na Figura 8.

Figura 9. Arquivo XML gerado a partir do mapeamento de objetos de vidro.

```
<annotation>
    <folder>vidro</folder>
    <filename>vidro (55).jpg</filename>
    <path>C:\Users\hudso\Documents\10° Período\TCC II\vidro\vidro (55).jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>236</width>
        <height>214</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>vidro</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>61</xmin>
            <ymin>51</ymin>
            <xmax>149</xmax>
            <ymax>210</ymax>
        </bndbox>
    </object>
    <object>
        <name>vidro</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>117</xmin>
            <ymin>6</ymin>
            <xmax>178</xmax>
            <ymax>211</ymax>
        </bndbox>
    </object>
</annotation>
```

Fonte: Autores da pesquisa.

5.4. Geração dos dados de treinamento

Para que os dados das imagens fossem aproveitados pelo TensorFlow foi necessário realizar a geração e conversões de arquivos, a primeira etapa trata de converter todos os arquivos de saída do software LabelImg que estão no formato XML, para um único arquivo no

formato CSV. Este procedimento foi realizado através da execução de um conversor de uso genérico em Python disponível na plataforma *GitHub*[4].

A segunda etapa foi a criação do arquivo "label_map.pbtxt" que basicamente possui rótulos das classes de materiais (metal e vidro) e seus identificadores, este processo é necessário para a criação de um arquivo binário no formato ".record", que é entendível pela API do TensorFlow e contém informações das imagens, seus caminhos, e segmentação dos objetos.

5.5. Treinamento

Antes de rodar o treinamento á necessário realizar configurações na *pipeline* (Apêndice A – Código de configuração da *pipeline*). Este arquivo é utilizado pelo TensorFlow e nele são definidos parâmetros de treinamento e o modelo de rede neural a ser utilizado.

Foi necessário separar as imagens em dois diretórios distintos, o primeiro sendo a pasta "training" que contém as imagens a serem utilizadas no treinamento como dados de entrada para a rede neural. O segundo diretório foi a parta "test" que possui as imagens que são utilizadas no processo de comparação com os dados de saída da rede neural.

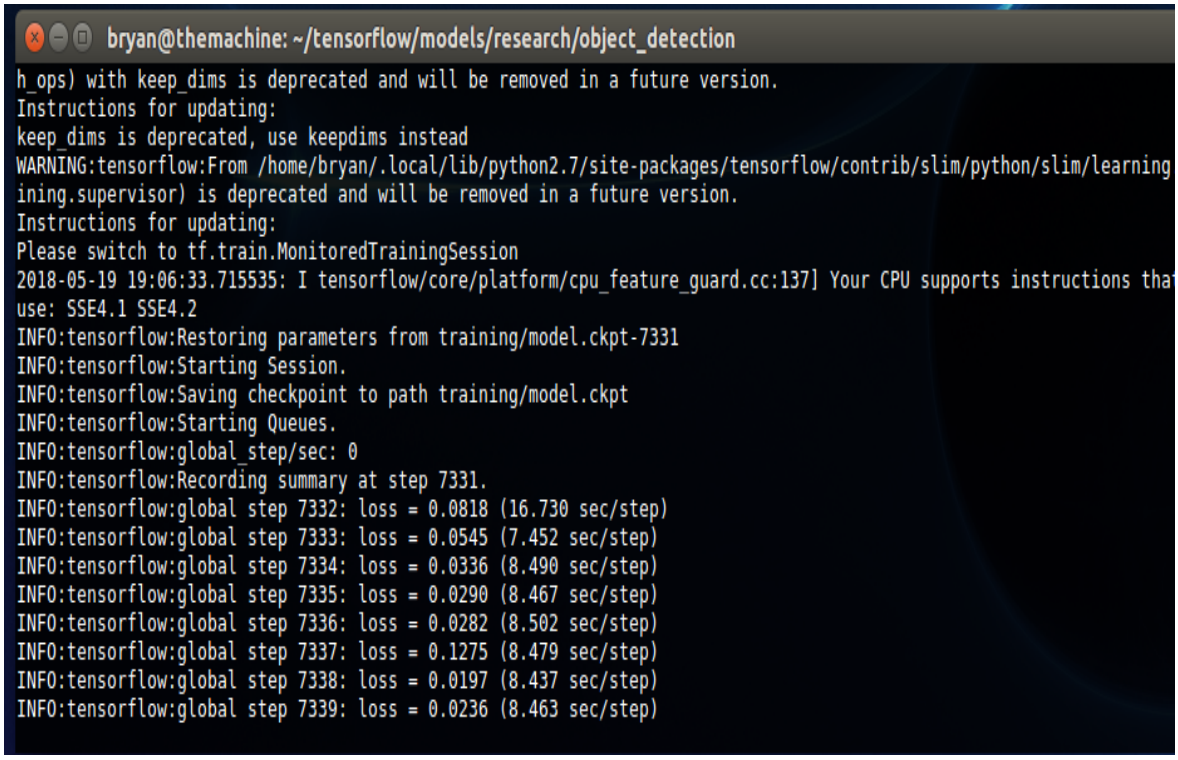
A comparação realizada entre os dois diretórios indica o índice de acerto da previsão da rede neural, que é chamado de perda. Quanto maior for o valor de perda, menor será a acurácia do classificador, e em oposto, quanto menor o valor da perda, maior será probabilidade de que a rede neural realizou a previsão correta.

Após essa etapa é possível dar início a fase de treinamento. Este processo foi realizado através da execução do arquivo "train.py" (Apêndice B – Código de treinamento), disponível na API do TensorFlow. Esta etapa foi a que demandou mais tempo de execução e recursos de hardware. A condição de parada deste procedimento é que a perda em cada *step* do treinamento esteja constantemente abaixo de 0.05, conforme recomendado na documentação da API, garantindo que se obtenha um índice de acertos aceitável.

A quantidade de perda pode ser acompanhada diretamente no terminal (Figura 10) ou através de um gráfico de perda gerado pela própria biblioteca (Figura 11).

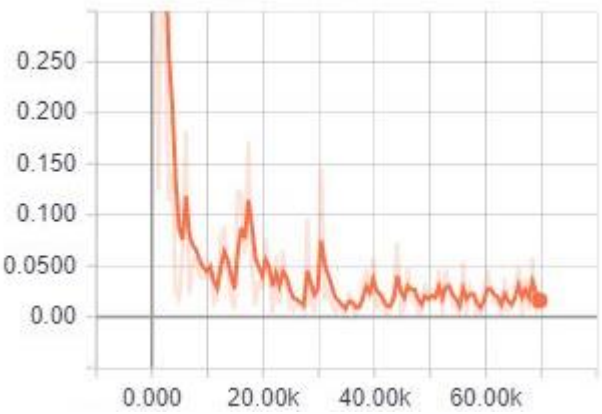
[4] Disponível em: <https://github.com/datitran/raccoon_dataset/blob/master/xml_to_csv.py> Acesso em: 15 maio 2018.

Figura 10. Perda por step vista no terminal.



Fonte: Autores da pesquisa.

Figura 11. Gráfico de perda.



Fonte: Autores da pesquisa.

Para que a quantidade de perda atingisse os níveis recomendados foram necessárias aproximadamente 48 horas de treinamento. O TensorFlow se encarrega de salvar *checkpoints* a cada 5 minutos, este procedimento é importante em caso de falhas ocasionadas por recursos insuficientes, ou fatores externos, evitando que o progresso no treinamento seja perdido.

A partir do último *checkpoint* gerado é possível exportar os dados de treinamento em um único arquivo no formato “.pb” este arquivo é o classificador e pode ser usado em aplicações variadas. A implementação deste projeto foi voltada para o processamento em vídeo recebido através de uma webcam.

5.6. Implementação

Para a codificação do sistema foi utilizada a linguagem Python na versão 2.7 que possui boa integração com bibliotecas externas, que foram necessárias para alcançar o resultado proposto. Em conjunto com o Python, foram aplicadas as bibliotecas TensorFlow na versão 1.5 para implementação da rede neural convolucional, e OpenCV para tarefas relacionadas a processamento de imagens.

A seção 6.5.1 descreve as etapas da codificação, juntamente com as respectivas amostras de código.

5.6.1. Codificação

A primeira etapa executada foi a importação das bibliotecas necessárias para tratamento de imagens, cálculos matemáticos, o TensorFlow, e OpenCV, como mostrado na Figura 12.

Figura 12. Importação das bibliotecas.

```
# Imports
import os
import cv2 #OpenCV-Python
import numpy as np
import tensorflow as tf
import sys

# Configuração do caminho para importação de outras bibliotecas
sys.path.append("..")

# Imports
from utils import label_map_util
from utils import visualization_utils as vis_util
```

Fonte: Autores da pesquisa.

Em seguida foi necessário fazer configurações que indiquem ao compilador Python o caminho dos arquivos que devem ser usados pelo classificador, sendo este o arquivo no formato “.pb”, e também indicar o arquivo que contém o rótulo das classes de objetos e seus respectivos identificadores. Este procedimento viabiliza o carregamento para a memória do computador todas as dependências do TensorFlow e do modelo utilizado (Figura 13).

Figura 13. Configurações de bibliotecas.

```
# Direcionamento do módulo object detection
MODEL_NAME = 'inference_graph'

# Armazena o diretório atual de execução
CWD_PATH = os.getcwd()

# Direcionamento do arquivo .pb que contém o classificador
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# caminho para o label_map.pbtxt que contém os rótulos das classes e seus IDs
PATH_TO_LABELS = os.path.join(CWD_PATH,'training','label_map.pbtxt')

# Número possível de classes que devem ser identificadas
NUM_CLASSES = 2

# Carregamento do arquivo label_map.
# Quando a rede neural prevê  o id '1', saberemos que este valor corresponde a 'vidro'.
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)

# Carregamento do modelo TensorFlow para memória
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)
```

Fonte: Autores da pesquisa.

Para que o classificador se comportasse como o esperado foi necessário indicar quais seriam os dados de entrada, sobre os quais o classificador deve realizar o processamento, e também definir as informações de saída, que neste caso são as caixas de segmentação do objeto, a classe do objeto, e a porcentagem de acurácia na classificação, que indica o grau de certeza com o qual a rede neural previu a classe do objeto. Estas especificações são repassadas ao TensorFlow como demonstrado na Figura 14.

Figura 14. Definição de dados de entrada e saída.

```
# Definição dos dados de entrada e saída no classificador
# Entrada -> imagem/frame de vídeo
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Saídas -> classe do objeto, acurácia, e boxes de segmentação
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
# -- Fim definição de saídas

# Identifica a quantidade de objetos identificados
num_detections = detection_graph.get_tensor_by_name('num_detections:0')
```

Fonte: Autores da pesquisa.

Para o processo de aquisição das imagens foram aplicadas funções da biblioteca OpenCV, que é a responsável por receber as imagens a partir do dispositivo de captura (neste caso trata-se de uma webcam). Os dados são recebidos em frames de vídeo, onde cada frame é uma imagem, esta imagem, portanto deve ser repassada como parâmetro de entrada para o classificador, que será responsável por identificar, segmentar e classificar os objetos. Esta ação pode ser visualizada no trecho de código na Figura 15.

Figura 15. Aquisição, processamento e classificação.

```
# Aquisição de imagens com OpenCV
# Número inteiro corresponde ao dispositivo de captura, no caso webcam -> 0
video = cv2.VideoCapture(0)
ret = video.set(3,1280)
ret = video.set(4,720)

#Loop de processamento, executado para cada frame de vídeo obtido
while(True):

    # Aquisição dos frames de vídeo
    ret, frame = video.read()

        # Expansão do frame
    frame_expanded = np.expand_dims(frame, axis=0)

    # Roda o classificador e o modelo, com a imagem de entrada, e identifica o objeto
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes, num_detections],
        feed_dict={image_tensor: frame_expanded})

    # Desenha os resultados na tela utilizando OpenCV e Numpy
    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=0.85)
```

Fonte: Autores da pesquisa.

A Figura 15 ainda demonstra a forma de processamento dos dados e apresentação dos resultados, e para garantir resultados em tempo de execução cada frame obtido foi processado em um loop infinito, onde as etapas são respectivamente as seguintes:

1. Aquisição do frame com OpenCV.
2. Expansão do frame (necessário quando o objeto na imagem tem tamanho reduzido).
3. Execução do classificador com a imagem de entrada.
4. O classificador então retorna, a caixa de segmentação, classe do objeto, e o índice de certeza de previsão da rede neural.

5. A biblioteca OpenCV desenha os resultados no frame, e em seguida os apresenta no dispositivo de saída.

O código em sua totalidade está representado no Apêndice C – Código de implementação.

6. RESULTADOS

Na fase de desenvolvimento a grande quantidade de processamento necessária para realizar o treinamento do classificador, fez com que o fossem necessárias 48 horas de treinamento. Este fator também ocasionou algumas falhas que interromperam o processo.

Apesar das interrupções ocorridas, a capacidade do TensorFlow de salvar *checkpoints* colaborou para que o processo de treinamento continuasse do ponto onde havia parado. Estes eventos aconteceram devido ao fato de o computador utilizado, cujas configuração são mostradas na Tabela 2, possuir capacidades limitadas para este tipo de aplicação.

Os desenvolvedores da biblioteca TensorFlow[5] recomendam que ao se trabalhar com redes neurais convolucionais e imagens, convém utilizar processamento paralelo em conjunto com a plataforma CUDA (*Compute Unified Device Architecture*) que permite a execução de aplicações utilizando GPU.

Após esta etapa foi possível implementar e executar o sistema, entretanto a carga de processamento exigida pela aplicação, e as limitações do hardware utilizado fizeram com que a taxa de quadros por segundo obtida através da webcam caísse drasticamente, e por este motivo o reconhecimento e classificação de um objeto demanda alguns segundos para ser concluído.

Mesmo com os impedimentos encontrados foram realizados testes afim de demostrar que os objetivos propostos foram atingidos. Os testes foram realizados obedecendo a seguinte ordem:

1. Teste com um objeto de metal.
2. Teste com um objeto de vidro.
3. Testes com múltiplos objetos.

Os testes foram executados no mesmo ambiente onde a aplicação foi desenvolvida. As configurações do computador utilizado são detalhadas na Tabela 2. Também foi necessário o uso de uma webcam como dispositivo de captura de imagens. A Tabela 4 contém um detalhamento dos testes com um único objeto por tentativa, e a Tabela 5 contém o detalhamento dos testes com múltiplos objetos simultaneamente. O Apêndice D – Testes e Resultados, contém outras imagens obtidas na fase de teste.

[5] Disponível em: < https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/tensorflow/> Acesso em: 18 maio 2018.

Tabela 4. Testes com um objeto

Teste	*Classe*	*Quantidade de objetos por tentativa*	*Quantidade de tentativas (cada tentativa utilizou um tipo de objeto diferente)*	*Acertos*	*Erros*	*Porcentagem de acertos*
1	Metal	1	6	4	2	66,66%
2	Vidro	1	4	4	0	100%

Fonte: Autores da pesquisa

Tabela 5. Testes com múltiplos objetos

Teste	*Quantidade de objetos de metal*	*Quantidade de objetos de vidro*	*Quantidade de tentativas*	*Acertos*	*Erros*	*Porcentagem de acertos*
1	1	1	1	2	0	100%
2	1	2	1	3	0	100%
3	1	2	1	2	1	75%
4	2	2	1	4	0	100%

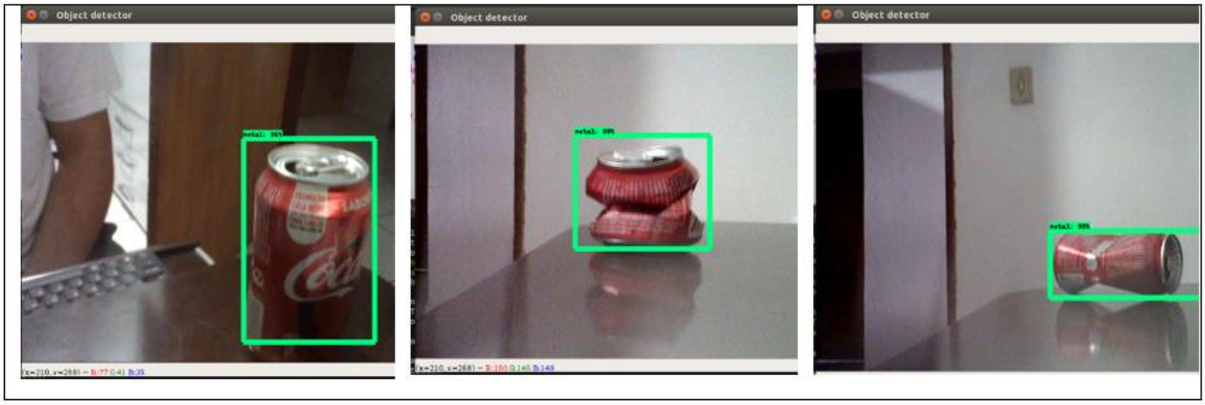
Fonte: Autores da pesquisa

As tabelas acima demonstram que os resultados obtidos foram satisfatórios, e que o software desenvolvido alcançou o objetivo proposto, pois de seis casos de testes executados apenas um obteve taxa de acerto abaixo de 75%. A Figura 17 demonstra um caso de teste com múltiplos objetos, onde houve uma taxa de acerto de 100%.

É necessário destacar que o sistema foi treinado para identificar duas classes de materiais, sendo estes metal e vidro, deste modo ao submeter a aplicação a um objeto para o qual ela não foi treinada ocorreram pequenas falhas. Estas falhas ocorrem porque ao obter uma imagem a rede neural tenta realizar uma previsão a respeito do possível objeto presente no frame.

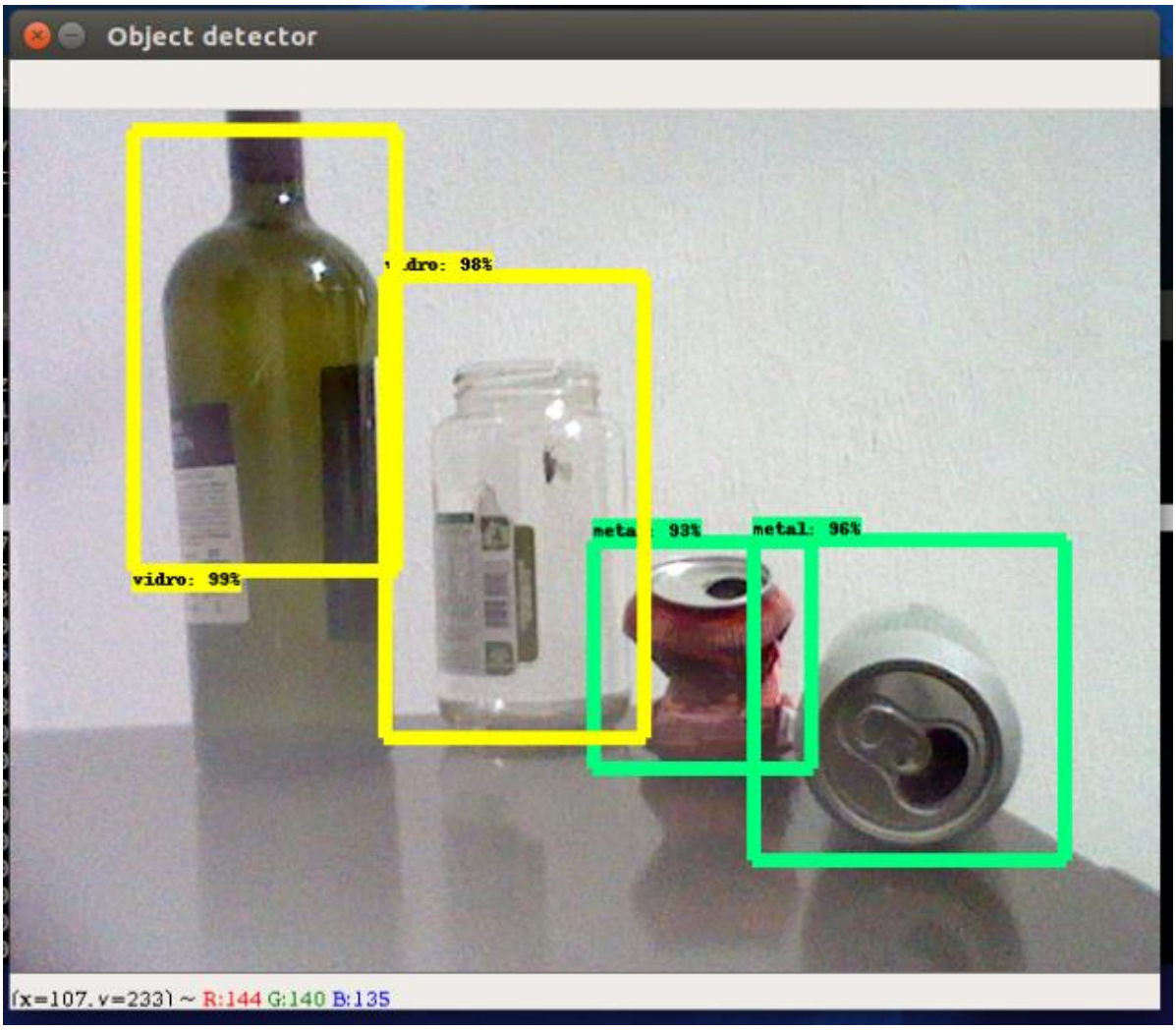
Um fator importante durante a realização dos testes foi a capacidade de identificação e classificação de um objeto mesmo em diferentes perspectivas, ou com alto grau de deformação, assim como ilustrado na Figura 16. Isto é possível devido as características de absorção de padrões de dados de uma rede neural convolucional, assim como descrito na seção 5.5.1.

Figura 16. Identificação de lata de metal com diferentes deformações.



Fonte: Autores da pesquisa

Figura 17. Teste com múltiplos objetos.



Fonte: Autores da pesquisa

7. CONSIDERAÇÕES FINAIS

O presente estudo possibilitou o desenvolvimento de um sistema de identificação e classificação de metal e vidro no âmbito da coleta seletiva, através da aplicação de visão computacional e aprendizado de máquina contidas nas bibliotecas OpenCV e TensorFlow.

Por meio das pesquisas bibliográficas, relacionadas ás áreas de Inteligência Artificial, Redes Neurais, Aprendizado de Máquina e Visão Computacional, Meio Ambiente e Sustentabilidade, foi possível alcançar os objetivos propostos.

A API TensorFlow, apesar de ser uma ferramenta recente tem sido utilizada em grande escala em diversos segmentos, não se limitando ao processamento de imagens, mas também em base de dados diversas. Entretanto é cada vez mais necessário a aplicação de novas tecnologias em prol do meio ambiente e do desenvolvimento sustentável.

A partir da implementação deste projeto foi possível demonstrar uma possível solução para a classificação de metal e vidro, porém o processo de coleta seletiva abrange outros tipos de resíduos, tais como plástico e papel, que por sua vez exigem o mapeamento de uma enorme quantidade de objetos.

Contudo apesar deste trabalho apresentar uma solução alternativa no processo de reaproveitamento de resíduos sólidos, é necessário que sejam realizados novos estudos que abranjam todo o escopo da coleta seletiva, e além disso, a integração com dispositivos eletromecânicos garantiriam, em tese a automação de todo processo. Fator este que pode culminar em uma série de avanços no setor industrial, e também colaborando para o desenvolvimento sustentável.

REFERÊNCIAS BIBLIOGRÁFICAS

ARTERO, A. O. **Inteligência Artificial:** Teórica e Prática. 1. ed. São Paulo: Editora Livraria da Física, 2009.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML.** 2. ed. Rio de Janeiro: Elsevier, 2007.

CALLISTER JUNIOR, W. D. **Fundamentos da ciência e engenharia de materiais**: uma abordagem integrada. 2. ed. Rio de Janeiro: LTC - Livros Técnicos e Científicos, 2006.

CONCI, A.; AZEVEDO, E.; LETA, F.R. **Computação Gráfica.** 2. ed. Rio de Janeiro: Elsevier, 2008.

COPPIN, B. **Inteligência Artificial.** 1. ed. Rio de Janeiro: LTC, 2013.

FACELI, K. et al. **Inteligência Artificial:** Uma Abordagem de Aprendizado de Máquina. 1. ed. Rio de Janeiro: LTC, 2011.

FELGUEIRAS, C.; GARROT, J. **Introdução ao Processamento Digital de Imagem:** Implementação em Java. Lisboa: FCA, 2008.

GONZALEZ, R. C. **Processamento de Imagens Digitais.** 1. ed. São Paulo: Editora Blucher, 2000.

HAYKIN, S. **Redes Neurais:** Princípios e prática. 2. ed. Porto Alegre: Bookman, 2001.

INSTITUTO DE PESQUISA ECONÔMICA APLICADA. **Comunicado 145 – Plano Nacional de Resíduos Sólidos:** diagnóstico dos resíduos urbanos, agrosilvopastoris e a questão dos catadores. Disponível em: <http://www.ipea.gov.br/portal/images/stories/PDFs/comunicado/120425_comunicadoipea0145.pdf >. Acesso em: 21 set. 2017.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. **Gradient-based learning applied to document recognition, in Proceedings of the IEEE**, vol. 86, no. 11, 1998, pp. 2278–2324. Disponível em: <http://ieeexplore.ieee.org/document/726791> Acesso em 12 abr. 2018.

MANO, E. B.; PACHECO, E. B. A. V.; BONELLI, C. M. C. **Meio Ambiente, Poluição e Reciclagem.** 2. ed. São Paulo: Blucher, 2010.

MINISTÉRIO DO MEIO AMBIENTE. **Coleta Seletiva.** Disponível em:< http://www.mma.gov.br/cidades-sustentaveis/residuos-solidos/catadores-de-materiais-reciclaveis/reciclagem-e-reaproveitamento> Acesso em: 1 maio 2017.

OPENCV. **OpenCV.** Disponível em:< http://opencv.org/about.html> Acesso em: 1 maio 2017.

PERKOVIC, L. **Introdução à computação usando Python:** um foco no desenvolvimento de aplicações. 1. ed. Rio de Janeiro: LTC, 2016.

REZENDE, S. O. **Sistemas Inteligentes:** Fundamentos e Aplicações. 1. ed. Barueri: Manole, 2005.

ROSÁRIO, J. M. **Princípios de Mecatrônica.** 9. ed. São Paulo: Prentice Hall, 2005.

RUSSEL, S. T. NORVIG, P. **Inteligência Artificial.** 3. ed. Rio de Janeiro: Elsevier, 2013.

SINGER, P. **A recente ressurreição da economia solidária no Brasil**. In: SANTOS, Boaventura de Souza (Org.) Produzir para viver: os caminhos da produção não capitalista. Rio de Janeiro: Civilização Brasileira; 2002. p.81-126.

SMITH, W. F.; HASHEMI, J. **Fundamentos de engenharia e ciência dos materiais**. 5. ed. Porto Alegre: McGraw-Hill, 2012.

SOMMERVILLE, I. **Engenharia de Software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011.

TASSARA, E. **Dicionário Socioambiental:** Ideias, definições e conceitos. 1. ed. São Paulo: Faarte Editora, 2008.

TENSORFLOW. **TensorFlow.** Disponível em:< https://www.tensorflow.org/> Acesso em: 05 abr. 2018.

TINOCO, J. E. P.; KRAEMER, M. E. P. **Contabilidade e gestão ambiental**. 3. ed. São Paulo: Atlas, 2004.

TOWNSEND, C. R.; BEGON, M.; HARPER, J.L. **Fundamentos em Ecologias.** 3. ed. Porto Alegre: Artmed, 2010.

WAZLAWICK, R. S. **Metodologia de Pesquisa para Ciência da Computação**. 2. ed. Rio de Janeiro: Elsevier, 2014.

APENDICE A – Código de configuração da *pipeline*

```
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:
'/home/bryan/tensorflow/models/research/object_detection/faster_rcnn_inception_v2_
coco_2018_01_28/model.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:
'/home/bryan/tensorflow/models/research/object_detection/train.record'
  }
  label_map_path:
'/home/bryan/tensorflow/models/research/object_detection/training/label_map.pbtxt'
}

eval_config: {
  num_examples: 32
  # 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:
'/home/bryan/tensorflow/models/research/object_detection/test.record'
  }
  label_map_path:
'/home/bryan/tensorflow/models/research/object_detection/training/label_map.pbtxt'
  shuffle: false
  num_readers: 1
}
```

APENDICE B – Código de treinamento

```
# 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.
# ==============================================================================

r"""Training executable for detection models.

This executable is used to train DetectionModels. There are two ways of
configuring the training job:

1) A single pipeline_pb2.TrainEvalPipelineConfig configuration file
can be specified by --pipeline_config_path.

Example usage:
    ./train \
        --logtostderr \
        --train_dir=path/to/train_dir \
        --pipeline_config_path=pipeline_config.pbtxt

2) Three configuration files can be provided: a model_pb2.DetectionModel
configuration file to define what type of DetectionModel is being trained, an
input_reader_pb2.InputReader file to specify what training data will be used and
a train_pb2.TrainConfig file to configure training parameters.

Example usage:
    ./train \
        --logtostderr \
        --train_dir=path/to/train_dir \
        --model_config_path=model_config.pbtxt \
        --train_config_path=train_config.pbtxt \
        --input_config_path=train_input_config.pbtxt
"""

import functools
import json
import os
import tensorflow as tf

from object_detection import trainer
from object_detection.builders import dataset_builder
from object_detection.builders import model_builder
from object_detection.utils import config_util
from object_detection.utils import dataset_util

tf.logging.set_verbosity(tf.logging.INFO)
```

```
flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow master to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                     'Force clones to be deployed on CPU.  Note that even if '
                     'set to False (allowing ops to run on gpu), some ops may '
                     'still be run on the CPU if they have no GPU kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of worker+trainer '
                     'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                     'Number of parameter server tasks. If None, does not use '
                     'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints and training summaries.')

flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs are ignored')

flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config file.')
flags.DEFINE_string('input_config_path', '',
                    'Path to an input_reader_pb2.InputReader config file.')
flags.DEFINE_string('model_config_path', '',
                    'Path to a model_pb2.DetectionModel config file.')

FLAGS = flags.FLAGS


def main(_):
  assert FLAGS.train_dir, '`train_dir` is missing.'
  if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
  if FLAGS.pipeline_config_path:
    configs = config_util.get_configs_from_pipeline_file(
        FLAGS.pipeline_config_path)
    if FLAGS.task == 0:
      tf.gfile.Copy(FLAGS.pipeline_config_path,
                    os.path.join(FLAGS.train_dir, 'pipeline.config'),
                    overwrite=True)
  else:
    configs = config_util.get_configs_from_multiple_files(
        model_config_path=FLAGS.model_config_path,
        train_config_path=FLAGS.train_config_path,
        train_input_config_path=FLAGS.input_config_path)
    if FLAGS.task == 0:
      for name, config in [('model.config', FLAGS.model_config_path),
                           ('train.config', FLAGS.train_config_path),
                           ('input.config', FLAGS.input_config_path)]:
        tf.gfile.Copy(config, os.path.join(FLAGS.train_dir, name),
                      overwrite=True)

  model_config = configs['model']
  train_config = configs['train_config']
  input_config = configs['train_input_config']

  model_fn = functools.partial(
      model_builder.build,
      model_config=model_config,
```

```
      is_training=True)

  def get_next(config):
    return dataset_util.make_initializable_iterator(
        dataset_builder.build(config)).get_next()

  create_input_dict_fn = functools.partial(get_next, input_config)

  env = json.loads(os.environ.get('TF_CONFIG', '{}'))
  cluster_data = env.get('cluster', None)
  cluster = tf.train.ClusterSpec(cluster_data) if cluster_data else None
  task_data = env.get('task', None) or {'type': 'master', 'index': 0}
  task_info = type('TaskSpec', (object,), task_data)

  # Parameters for a single worker.
  ps_tasks = 0
  worker_replicas = 1
  worker_job_name = 'lonely_worker'
  task = 0
  is_chief = True
  master = ''

  if cluster_data and 'worker' in cluster_data:
    # Number of total worker replicas include "worker"s and the "master".
    worker_replicas = len(cluster_data['worker']) + 1
  if cluster_data and 'ps' in cluster_data:
    ps_tasks = len(cluster_data['ps'])

  if worker_replicas > 1 and ps_tasks < 1:
    raise ValueError('At least 1 ps task is needed for distributed training.')

  if worker_replicas >= 1 and ps_tasks > 0:
    # Set up distributed training.
    server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol='grpc',
                             job_name=task_info.type,
                             task_index=task_info.index)
    if task_info.type == 'ps':
      server.join()
      return

    worker_job_name = '%s/task:%d' % (task_info.type, task_info.index)
    task = task_info.index
    is_chief = (task_info.type == 'master')
    master = server.target

  trainer.train(create_input_dict_fn, model_fn, train_config, master, task,
                FLAGS.num_clones, worker_replicas, FLAGS.clone_on_cpu, ps_tasks,
                worker_job_name, is_chief, FLAGS.train_dir)


if __name__ == '__main__':
  tf.app.run()
```

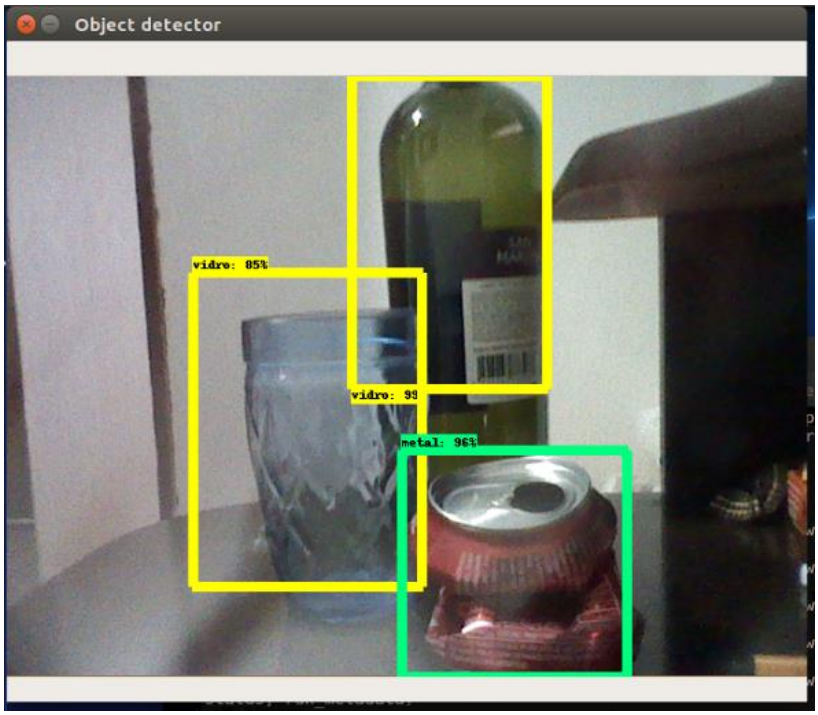
APENDICE C – Código de implementação

```
######## Detector de objeto por webcam utilizando um classificador treinado via
RNC com TensorFlow #########

# Michael Bryan e Hudson Murilo
# 10/05/18

# Imports
import os
import cv2 #OpenCV-Python
import numpy as np
import tensorflow as tf
import sys

# Configuração do caminho para importação de outras bibliotecas
sys.path.append("..")

# Imports
from utils import label_map_util
from utils import visualization_utils as vis_util

# Direcionamento do módulo object detection
MODEL_NAME = 'inference_graph'

# Armazena o diretório atual de execução
CWD_PATH = os.getcwd()

# Direcionamento do arquivo .pb que contém o classificador
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# caminho para o label_map.pbtxt que contém os rótulos das classes e seus IDs
PATH_TO_LABELS = os.path.join(CWD_PATH,'training','label_map.pbtxt')

# Número possível de classes que devem ser identificadas
NUM_CLASSES = 2

# Carregamento do arquivo label_map.
# Quando a rede neural prevê  o id '1', saberemos que este valor corresponde a
'vidro'.
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)

# Carregamento do modelo TensorFlow para memória
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)


# Definição dos dados de entrada e saída no classificador
# Entrada -> imagem/frame de vídeo
```

```
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Saídas -> classe do objeto, acurácia, e boxes de segmentação
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
# -- Fim definição de saídas

# Identifica a quantidade de objetos identificados
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Aquisição de imagens com OpenCV
# Número inteiro corresponde ao dispositivo de captura, no caso webcam -> 0
video = cv2.VideoCapture(0)
ret = video.set(3,1280)
ret = video.set(4,720)

#Loop de processamento, executado para cada frame de vídeo obtido
while(True):

    # Aquisição dos frames de vídeo
    ret, frame = video.read()

      # Expansão do frame
    frame_expanded = np.expand_dims(frame, axis=0)

    # Roda o classificador e o modelo, com a imagem de entrada, e identifica o
objeto
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes, num_detections],
        feed_dict={image_tensor: frame_expanded})

    # Desenha os resultados na tela utilizando OpenCV e Numpy
    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=0.85)

    # OpenCV se encarrega de mostrar o frame tratado
    cv2.imshow('Object detector', frame)

    # Condição de saída do programa
    if cv2.waitKey(1) == ord('q'):
        break

# Limpa o buffer de víde e encerra a janela
video.release()
cv2.destroyAllWindows()
```

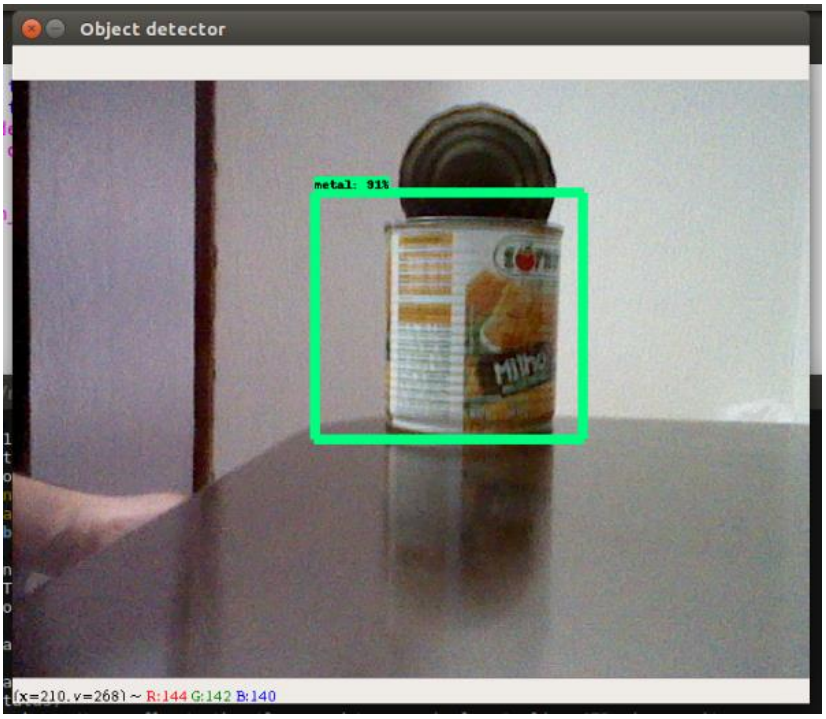
APÊNDICE D – Testes e resultados

Imagem 1. Teste com múltiplos objetos



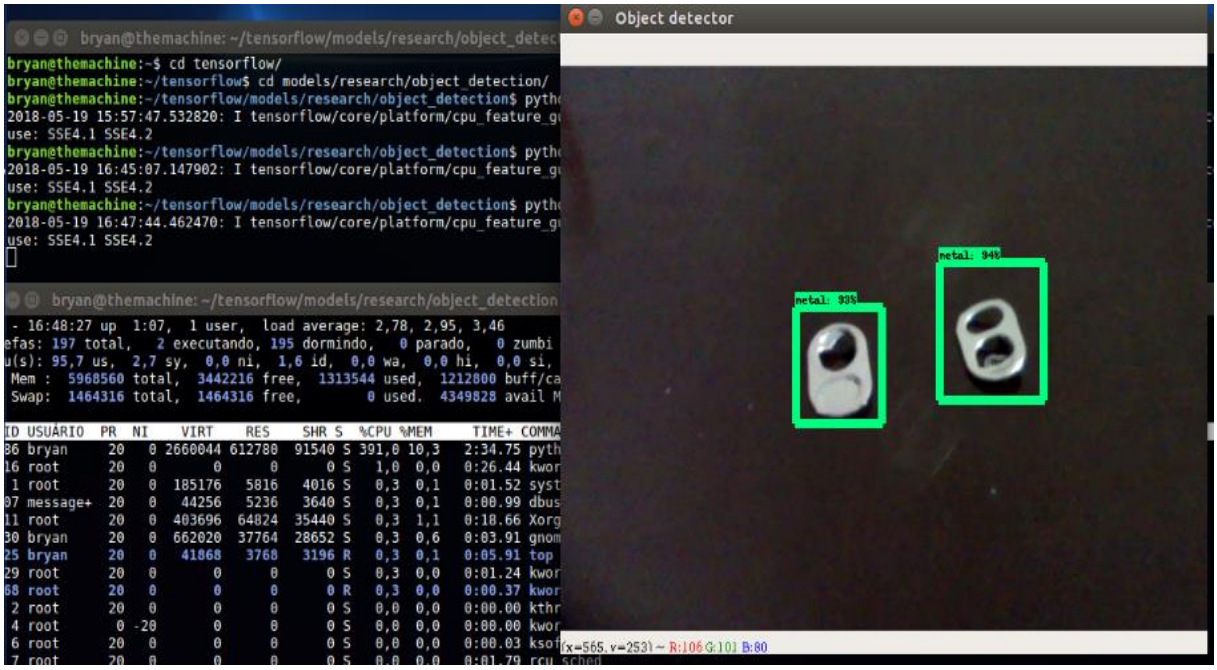
Fonte: Autores da pesquisa.

Imagem 2. Teste com metal.



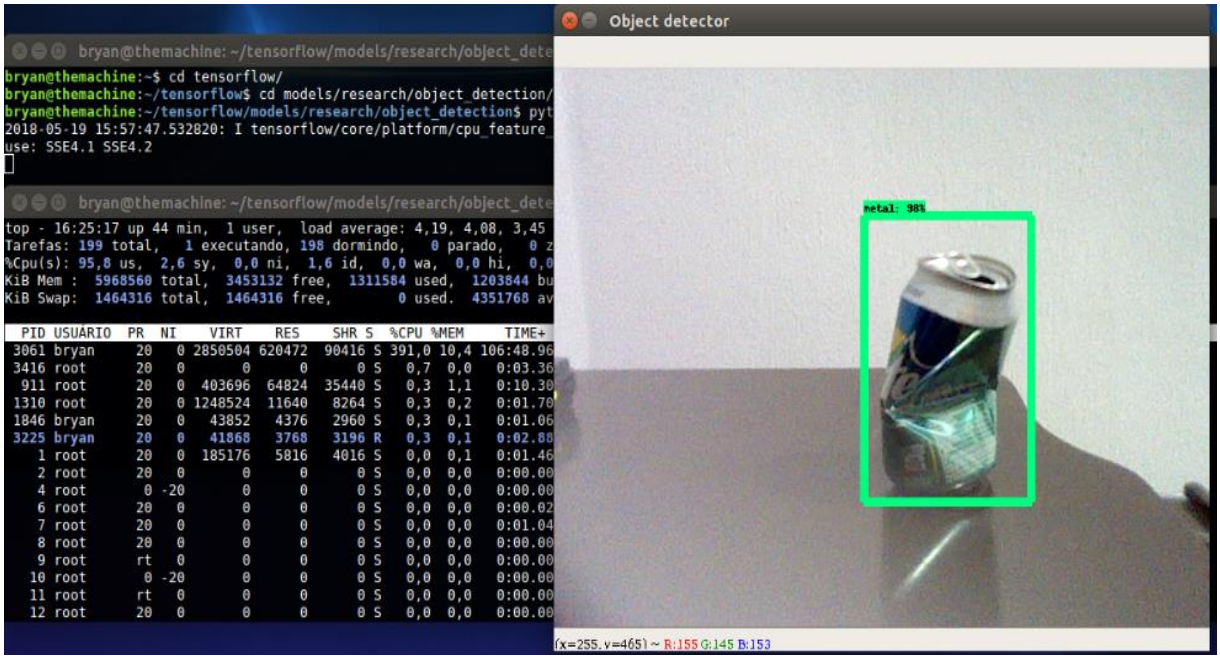
Fonte: Autores da pesquisa.

Imagem 3.Teste com metal.



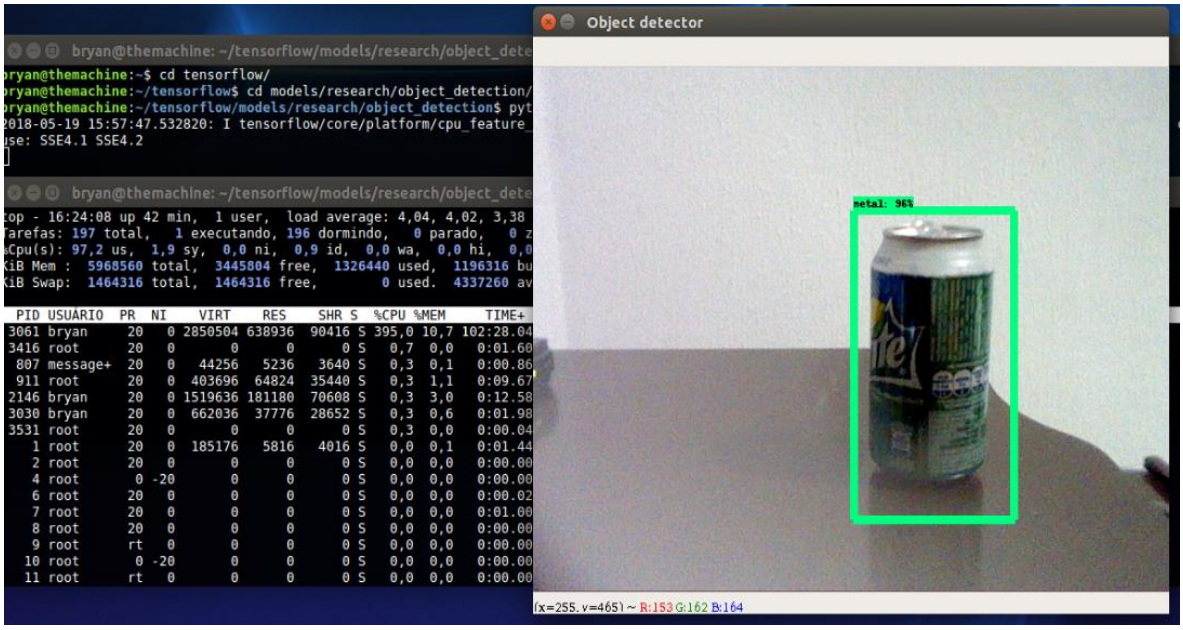
Fonte: Autores da pesquisa.

Imagem 4. Teste com metal.



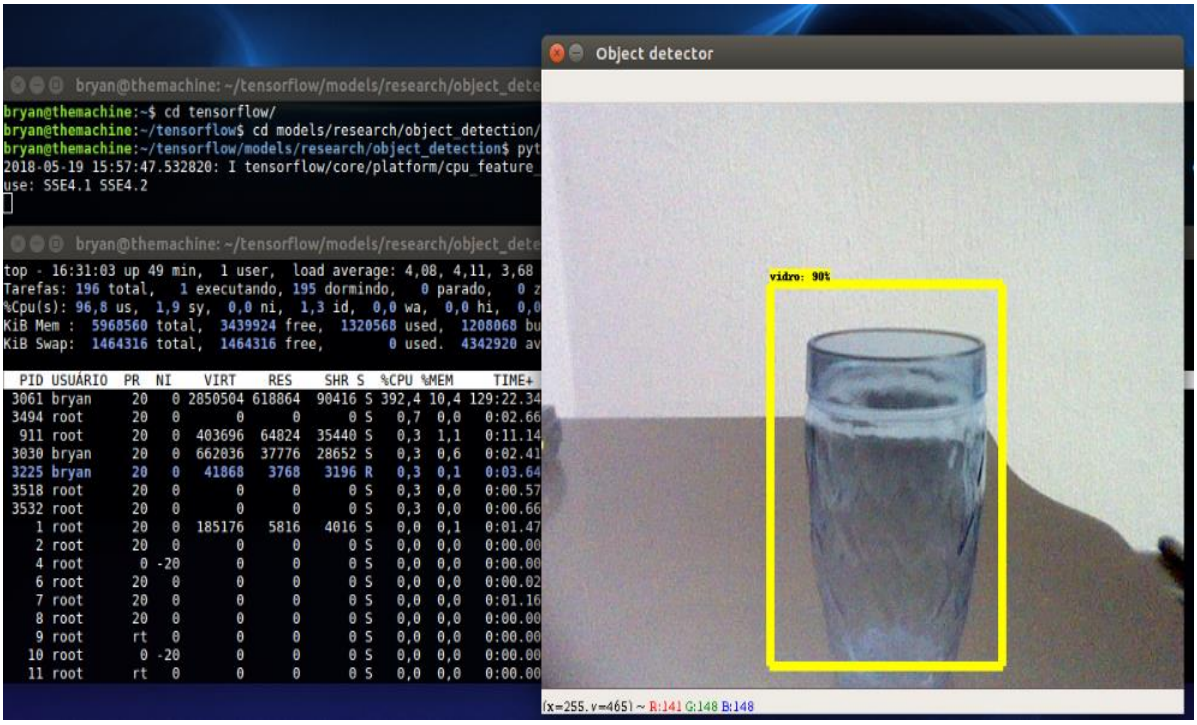
Fonte: Autores da pesquisa.

Imagem 5. Teste com metal.



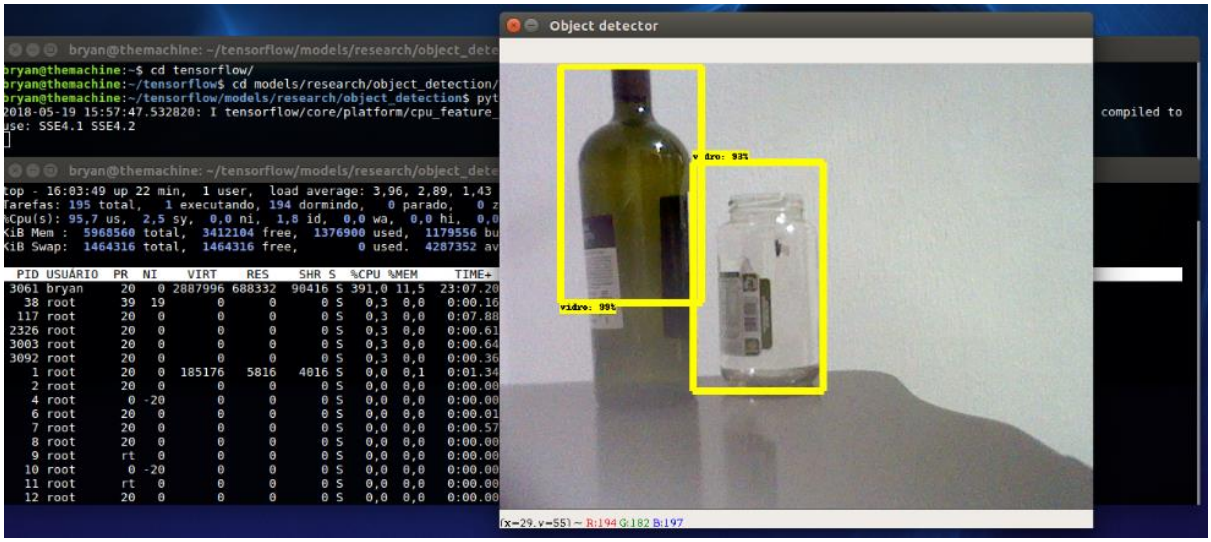
Fonte: Autores da pesquisa.

Imagem 6. Teste com vidro.



Fonte: Autores da pesquisa.

Imagem 7.Teste com vidro.



Fonte: Autores da pesquisa.