Detecção de falhas em contentores plásticos utilizando Redes Neurais Convolucionais

Cassio Girelli e Leandro Luís Corso

Resumo

A indústria 4.0 visa promover o desenvolvimento tecnológico, agilizando e automatizando processos buscando ganhos operacionais gerando diferencial competitivos nas empresas que a aplicam. As técnicas de *Deep Learning* no segmento de visão computacional estão revolucionando o processo de detecção e objetos, tornando os processos de desenvolvimento menos complexo e obtendo resultados cada vez melhores. Tradicionalmente, a análise de falhas em contentores plásticos vem sendo realizada de forma visual, cabendo ao operador a identificação das falhas. Este processo, por ser visual, demanda tempo do operador, submetendo o mesmo a executar várias tarefas de forma simultânea no seu posto de trabalho, assim, correndo o risco de levar à possíveis erros de identificação e apontamentos. Este estudo tem por objetivo demonstrar as etapas a serem seguidas pelos autores na elaboração de um modelo, o qual irá auxiliar no processo de conferência e verificação da qualidade do produto, analisando imagens dos produtos e indicando o local exato do defeito. Neste artigo, foram empregadas técnicas de *Deep Learning*, seguindo uma abordagem de aprendizado supervisionado baseado em Redes Neurais Convolucionais. Com o intuito de apresentar os passos a serem seguidos na elaboração deste modelo, abordando a tecnologia usada, iniciou-se pela coleta das imagens, rotulagem, treinamento e, por fim, a avaliação do modelo gerado. Com esse estudo, foi possível concluir que o uso das Redes Neurais Convolucionais pode auxiliar e evoluir o processo de detecção de falhas, proporcionando controles automatizados, agregando informações que possibilitam uma melhor tomada de decisão.

Palavras-chave

Redes Neurais Convolucionais, Deep Learning, Faster R-CNN, Detecção de Falhas

Fault detection in plastic containers using Convolutional Neural Network

Abstract

Industry 4.0 aims to promote technological development, streamlining and automating processes seeking operational gains generating competitive differentials in the companies that apply it. Deep Learning techniques in the computer vision segment are revolutionizing the detection process and objects, making development processes less complex and getting better and better results. Traditionally, the analysis of failures in plastic containers has been carried out visually, leaving the operator to identify the failures. This process, as it is visual, demands time from the operator, subjecting him to perform several tasks simultaneously in his workstation, thus running the risk of leading to possible identification errors and notes. This article aims to demonstrate the steps to be followed by the authors in the development of a model, which will assist in the process of checking and verifying the quality of the product, analyzing images of the products and indicating the exact location of the defect. In this article, Deep Learning techniques were used, following a supervised learning approach based on Convolutional Neural Networks. In order to present the steps followed in the elaboration of this model, addressing the technology used, it started by collecting the images, labeling, training and, finally, the evaluation of the generated model. With this article, it was possible to conclude that the use of Convolutional Neural Networks can help and evolve the failure detection process, providing automated controls, adding information that enable better decision making.

Keywords

Convolucional Neural Networks, Deep Learning, Faster R-CNN, Fault Detection

I. INTRODUÇÃO

As empresas, perante as evoluções tecnológicas, estão buscando ferramentas para agilizar e automatizar os processos produtivos. O conceito de Indústria 4.0 foi desenvolvido pelo governo alemão com o auxílio de pesquisadores e indústrias alemãs, com a finalidade de tonificar o setor industrial do país [1]. A necessidade de automatizar e otimizar processos no ambiente fabril levou

Pós graduação em Engenharia 4.0 – Universidade de Caxias do Sul (UCS)

E-mails: cassiogirelli@gmail.com; llcorso@ucs.br

Data de envio: 06/09/2020
Data de aceite: 17/10/2020

http://dx.doi.org/10.18226/23185279.v8iss2p156

ao emprego de novas tecnologias, buscando a convergência entre a área industrial e a tecnologia da informação [1].

A Indústria 4.0 fornece várias oportunidades para as indústrias moldarem seu futuro, almejando impactos econômicos, eficiência operacional, desenvolvimento de novos modelos de negócios, processos produtivos e produtos completamente novos [2]. O trabalho exposto em [3] cita como um dos inúmeros conceitos fundamentais da Indústria 4.0 que o processo produtivo seja equipado com sistemas autônomos, proporcionando a integração entre sistemas físicos e softwares, visando a obtenção de informações em tempo real.

Recentemente, técnicas de *Deep Learning* tornaram-se muito populares no segmento da visão computacional, em especial as Redes Neurais Convolucionais, que têm se destacado nos processos de classificação e detecção, como a detecção de múltiplos objetos em diferentes perspectivas em imagens e reconhecimento de dígitos escritos à mão [4] [5]. Com isso alguns trabalhos apresentaram resultados significativos, alcançando taxas de erro extremamente baixas no desafio que avalia algoritmos de localização e detecção de objeto em imagens e vídeos (chamado de *Large Scale Visual Recognition Challenge* (LSVRC) [4]), evidenciando um ganho de velocidade e eficiência no processo de detecção e classificação, comparado com os métodos utilizado anteriormente [16].

As técnicas convencionais de aprendizado de máquina eram limitadas em sua capacidade e velocidade de processar dados brutos [6] [7]. No entanto, *Deep Learning* são técnicas de aprendizado compostas de várias camadas de processamento com múltiplos níveis de representação, alcançadas por meio da composição de vários módulos, iniciando com a entrada utilizando dados brutos, e finalizando com uma saída em um nível mais elevado [6] [8].

Diante disso, este artigo tem a finalidade de apresentar as etapas de construção, treinamento e validação de um modelo, com a função de auxiliar no processo de detecção de falhas em contentores plásticos, utilizando aprendizado supervisionado empregando Redes Neurais Convolucionais.

O objetivo deste trabalho é auxiliar, de forma sistêmica, na visualização e contagem das falhas nos produtos, buscando identificar o local exato da avaria. Atualmente a verificação das avarias é realizada de forma visual pelos operadores das máquinas. Segregando as peças com defeito e registrando os devidos apontamentos. O modelo proposto busca tornar esta etapa do processo automática, vislumbrando a possibilidade de integrações futuras com sistemas de apontamentos eletrônicos.

II. Referencial Teórico

A. Aprendizado de Máquina

A capacidade de aprender deve fazer parte de qualquer sistema que demande inteligência. Sistemas inteligentes devem ser capazes de se adaptar e modificar os dados ao longo do curso de suas interações [9].

Dentre os tipos de aprendizado, pode-se citar:

a) Supervisionado: sistema necessita que os dados sejam classificados de forma manual ou por outro método externo, no qual o objetivo é suprir o sistema com informações de entrada, como informar a qual classe uma imagem pertence.

b) Não supervisionado: elimina a figura do classificador e obriga o próprio algoritmo de aprendizado avaliar os conceitos, buscando conhecimento novo por modificação e combinação, procurando similaridade entre os dados.

c) Reforço: o algoritmo evolui o aprendizado quando ocorre interação com o ambiente, realizando alterações em seus parâmetros em busca de um objetivo explícito.

B. Deep Learning

A técnica de inteligência artificial, conhecida por *Deep Learning,* possibilita que modelos computacionais elaborados utilizando várias camadas de processamento aprendam representações de dados com vários níveis de abstração [6]. Por exemplo, uma imagem usada como entrada na forma de uma matriz, na qual a primeira camada apresenta a presença ou ausência de bordas em locais específicos da imagem; na segunda camada normalmente identifica arranjos específicos de bordas e nas próximas camadas, montam-se combinações que podem corresponder à partes de objetos familiares. Por fim, nas camadas subsequentes detectam e classificam objetos com base nas partes descobertas na camada anterior [6].

Um dos métodos mais utilizado de *Deep Learning* são das as Redes Neurais Convolucionais, que serão explicadas no item C juntamente com suas camadas.

Esses modelos são amplamente desenvolvidos e aplicados em diversas áreas, como visão computacional, reconhecimento de fala, reconhecimento e classificação de objetos, classificação e processamento de linguagem natural [6].

C. Redes Neurais Convolucionais

As Redes Neurais Convolucionais (do inglês, *Convolutional Neural Networks* (CNN)) [11] é uma técnica de *Deep Learning* amplamente utilizada nas tarefas de classificação de imagens e detecção de objetos [16].

Apresentam uma série de avanços, como os resultados do desafio do *Imagnet* expõem, outra questão apresentada faz referência à profundidade da rede neural que impacta positivamente nos resultados [14] [15].

A criação da Rede Neural para a tarefa de classificação de objetos é determinada por seus pesos. O treinamento de uma Rede Neural consiste em adaptar os pesos a um problema especifico. Os pesos na primeira passagem são iniciados de forma aleatória gerando um valor de erro elevado. Então é utilizado para o treinamento o algoritmo de *Backpropagation* no qual os pesos das conexões das camadas internas são modificados conforme o erro resultante do teste realizado no processo de treinamento. Sendo executado no sentido inverso (retropropagação) a fim de reduzir o erro nas próximas execuções buscando adaptar a Rede Neural para a resolução do problema e melhorando a sua acurácia.

Um modelo de CNN gera um resultado por uma entrada após o processamento por uma pilha de camadas, incluindo camadas de:

a) Convolução: esta camada é o principal componente de uma Rede Neural Convolucional, na qual a maior parte do processo é realizado. O objetivo principal da operação de convolução é extrair características (bordas, retas e curvas) utilizando filtros aplicados a um pedaço específico da imagem. Esses filtros podem ser de tamanhos variados, por exemplo, 5x5x3, (5 pixels de altura e 5 de largura e profundidade de 3 quando utilizado imagens coloridas —o 3 em imagens coloridas indica os canais de cores RGB, vermelho, verde e azul) [6].

b) *Pooling*: a camada de *pooling* é responsável por reduzir progressivamente a dimensionalidade dos mapas de características extraídos nas camadas de convolução, mantendo as informações mais relevantes. Nesta operação, os valores de um determinado mapa de características são substituídos por alguma função aritmética. A função mais utilizada é o valor máximo (*max pooling*), onde o mapa de características é substituído pelo valor máximo encontrado na sua formulação [6][19].

c) *Fully Connected (FC)*: o resultado das camadas de convolução e *pooling* é a extração de características das imagens de entrada na rede. A função desta camada é utilizar esses recursos para classificar uma imagem com base nas classes apresentadas no conjunto de dados de treinamento. Esta camada é exatamente igual a uma *Multi Layer Perceptron* (MPL) [18].

Cada camada é responsável por diferentes funções e usa o resultado das camadas anteriores como entrada. Comparando com abordagens convencionais, as CNNs exigem menos pré-processamento de imagens e os recursos são extraídos através de aprendizado [4].

No decorrer do processo de treinamento, a rede neural pode apresentar um desempenho muito bom nos dados de treinamento, porém resultados muito ruins nos dados de validação, sendo esse comportamento chamado de *Overfiting*. Uma das técnicas utilizadas para evitar esse problema é o *Dropout* [19]. Esta técnica consiste em descartar aleatoriamente neurônios da rede a cada iteração de treinamento, adicionando-os na iteração seguinte garantindo uma melhor generalização do modelo treinado.

A *Faster R-CNN* descrita no artigo [10] é uma rede neural amplamente utilizada na detecção de objetos, principalmente pela sua velocidade de execução. Essa rede é a evolução das redes *R-CNN* [12] e *Fast R-CNN* [5].

D. R-CNN, Fast R-CNN e Faster R-CNN

A *R-CNN* [12] [17] utiliza o algoritmo *Selective Search* [13] para descobrir regiões de interesse. A rede buscar descobrir áreas possíveis de localizar um objeto combinando pixel e texturas semelhantes, na qual são separadas cerca de 2.000 áreas propostas. Após esse processo, essas áreas são enviadas para um modelo pré-treinando da CNN e suas saídas encaminhadas para classificadores SVM, que têm a função de classificar os objetos e, por fim, é realizado um cálculo de regressão para a demarcação das caixas delimitadoras. A estrutura de uma *R-CNN* pode ser vista na Figura 1, onde estão expostas as etapas de processamento da rede. A *R-CNN* apresentou grande avanço no processo de detecção de objetos, porém sua estrutura demandava um grande poder computacional [16].

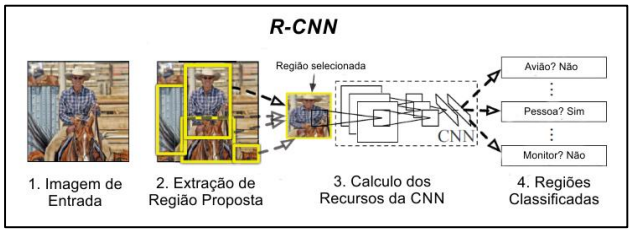

Figura 1 – Representação da arquitetura da R-CNN. Adaptado do autor (2020) [17].

O algoritmo *Fast R-CNN* [5] foi desenvolvido para resolver limitações da R-CNN (velocidade computacional), passando a imagem original para um modelo pré-treinando da CNN somente uma vez, independente de cada proposta de região. O algoritmo *Selective Search* [13] é calculado com base no mapa de recursos da saída da camada anterior. Em seguida a camada *Roi Pooling (*RoI*)* é utilizada para garantir o tamanho da saída padrão. Essas saídas são transformadas em dois vetores de saída, onde são empregados para prever o objeto utilizando um classificador *Softmax* e para calcular a dimensão das caixas delimitadoras (*Bounding Box*) utilizando regressão linear, como exposto na Figura 2.

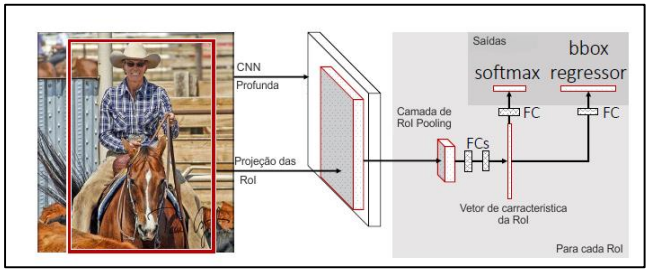

Figura 2 – Representação da arquitetura da Fast R-CNN. Adaptado do autor (2020) [5].

O desenvolvimento da *Faster R-CNN* [10] resolveu problemas de performance que não haviam sido sanados na *Fast R-CNN*. Uma das principais evoluções foi a introdução do algoritmo de *Region Propostal Network* (RPN) [10] no lugar do *Selective Search,* melhorando consideravelmente a velocidade de detecção.

O RPN (Figura 3) é uma rede convolucional utilizada para propor regiões e, para gerar as propostas de região, uma janela é deslizada sobre a saída do mapa de características. Junto a um classificador *cls layer,* um regressor *reg layer*, e uma camada intermediária *intermediate layer*, no RPN foi introduzido o conceito de âncora. Âncora (*Anchor*) é o ponto central da janela deslizante utilizada para criá-la com base nas coordenadas da *reg layer*. A quantidade de âncoras criadas varia de acordo com a altura e largura em pixel da imagem.

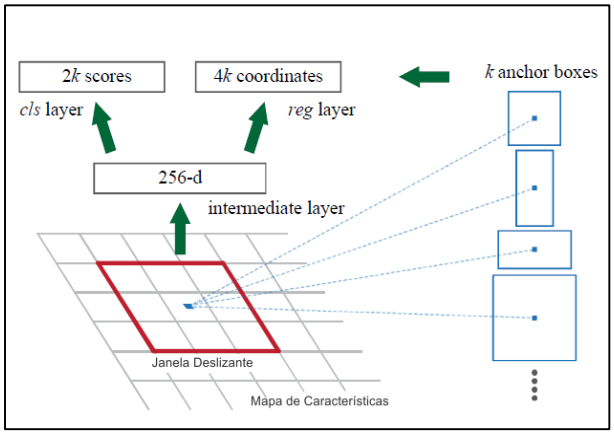

Figura 3 – Representação da arquitetura da RPN. Adaptado do autor (2020) [10].

A *Faster R-CNN* possui dois estágios, no primeiro, a rede RPN propõem os possíveis locais dos objetos a serem detectados. O segundo estágio é um classificador *Fast R-CNN* que utiliza as regiões propostas pela RPN para detectar e desenhar as caixas delimitadoras. Depois de obter o mapa de características oriundo da RPN, são geradas as propostas de região onde possíveis objeto estão localizados. Após essa etapa, o mapa de regiões passa pela camada *roi pooling* e sua saída encaminhada para classificação, como pode ser visto na Figura 4.

Todo o sistema é uma rede única e unificada para detecção de objetos, no qual o modulo RPN diz para o modulo *Fast R-CNN* onde procurar os objetos a serem detectados.

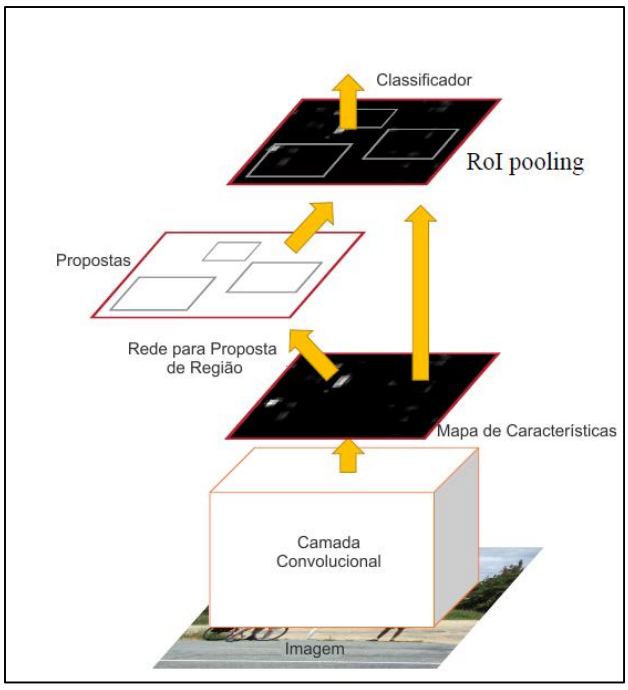

Figura 4 – Representação da arquitetura da Faster R-CNN. Adaptado do autor (2020) [10].

III. Materiail e Métodos

O objetivo deste trabalho é apresentar uma solução baseada em Redes Neurais Convolucionais que detecte e classifique problemas em embalagens plásticas, identificando o local onde a avaria se localiza.

Foi utilizado para este trabalho a *Application Programming Interface* (API) de detecção de objetos do *Tensorflow* [20] juntamente com a linguagem de programação *Python* e o sistema operacional Linux Ubuntu.

O *Tensorflow* [21] é uma biblioteca de código aberto para aprendizado de máquina desenvolvida pela empresa Google, originada do *DistBelief*. O *Tensorflow* foi desenvolvido para utilização de grande volume de dados, paralelização, processamento em *Central Process Unit* (CPUs), *Graphics Processing Unit (GPUs)* e *Tensor Processing Unit* (TPUs), sendo capaz de rodar em várias plataformas.

A API de detecção de objetos do *Tensorflow* disponibiliza uma série de modelos pré-treinados [22] em *dataset* com grande quantidade de imagens, possibilitando a detecção de objetos utilizando suas classes já treinadas, ou servindo de base para criação de modelos com conjuntos de dados específicos, como foi o caso deste artigo.

Para esse trabalho foi utilizado o modelo *Faster RCNN Inception v2 Coco* [23]. Esse modelo emprega uma Rede Neural Convolucional *Faster R-CNN,* utilizando *Inception v2* [26] como extrator de características desenvolvido pelo Google. Este modelo é treinado com imagens do *dataset* da Coco [24][25] que foi criado em 2015 e contem 91 tipos de objetos, num total de 328.000 imagens com 2,5 milhões de instâncias rotuladas. Dentre as imagens, constam objetos pequenos e com partes ocultas refletindo cenas do cotidiano [25].

A Figura 5 apresentada a seguir, mostra os passos realizados para a construção do modelo e, em seguida, as etapas do processo serão detalhadas.

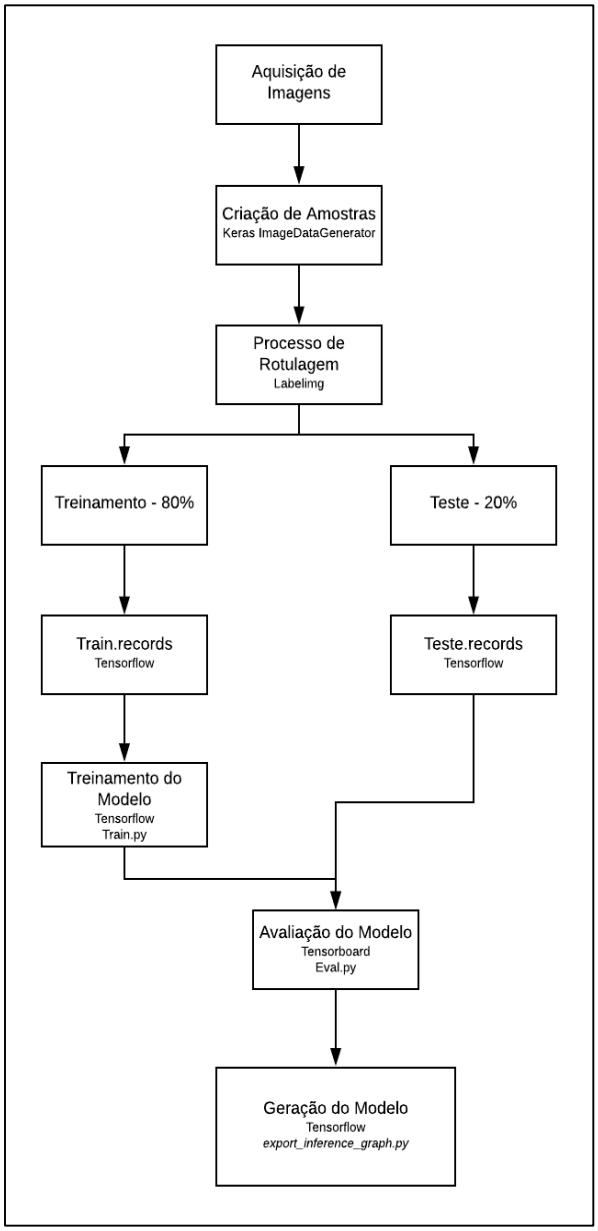

Figura 5 – Detalhamento do processo

Os produtos utilizados para o artigo foram resultados de peças com defeito segregadas em lotes de produção de um produto específico. Essas imagens foram captadas nas dependências da empresa, utilizando um smartphone, levando em conta que o escopo dos problemas detectados no produto é pequeno, pois os defeitos se concentram nas laterais. O número de imagens selecionadas foi de 30 amostras, sendo que este pode ser considerado pequeno e não apresentar variabilidade suficiente para a execução do processo de treinamento. Por esse motivo, foi utilizado um script em linguagem *Python* empregando o método *ImageDataGenerator* da API de *Deep Learning* do *Keras* [27] que executa alterações no rotação, altura, largura, zoom e escala da imagem para aumentar artificialmente o número de amostras para o processo de treinamento e teste, elevando para 215 amostras.

Destas 215 amostras foram segregadas 190 onde 80% (152 amostras) formam destinadas para o processo de treinamento e 20% (38 amostras) para teste. Ainda foram separadas 25 imagens para utilização na validação final do modelo sendo, 22 imagens com falhas e 3 imagens do produto original sem nenhuma avaria. Estas imagens não formam em nenhum momento apresentadas para o modelo a fim de obter um resultado mais aproximado do que o esperado quando inserido em produção.

Para o processo de treinamento foi definido somente uma classe contendo todos os erros mapeados devido à pouca variabilidade de amostras.

Para executar o processo de treinamento de um detector de objetos é necessário que as imagens de treinamento e teste sejam rotuladas, processo que identifica nas imagens o local onde está localizado da avaria no caso deste artigo identifica o local onde a avaria está localizada como pode ser visto na Figura 6. Foi utilizado o software *Labelimg* [28] para o processo de criação dos rótulos, gerando arquivos XML no formato *Pascal VOC* para cada imagem contendo as coordenadas dos rótulos e o nome da classe.

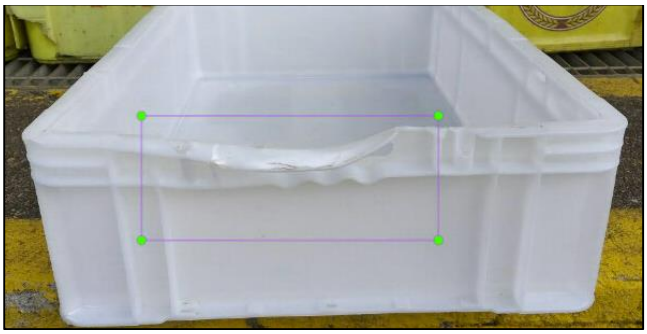

Figura 6 – Rotulo criado usando software Labelimg.

As imagens selecionadas para o processo de treinamento correspondem a 80% do total de imagens e os 20% restante são destinadas para teste do modelo. Após essa separação, foi utilizado um script em linguagem *Python* nativo do *Tensorflow* para a geração dos arquivos *Test.Records* e *Train.Records* próprios do *Tensorflow* com as informações das imagens, classes e coordenadas dos rótulos de cada marcação, sendo criados a partir dos arquivos XML gerado pelo software *Labelimg*.

O processo de treinamento é iniciado no *Tensorflow* e para o acompanhamento da evolução do processo foi utilizado a ferramenta nativa *Tensorboard*, a qual é possível acompanhar a *Mean Average Precision* (mAP). Essa métrica apresenta um valor de precisão das detecções geradas durante a execução do treinamento, tendo como base as imagens de teste. O *Tensorflow* utiliza o modelo gerado até o determinado passo e o aplica nas imagens de teste, este resultado dá origem ao valor da mAP. No decorrer da execução do treinamento, o modelo é avaliado e apresenta algumas métricas, como o mAP e a perda do modelo que retrata o erro do modelo naquele passo específico, sempre utilizando as imagens de teste da mesma forma que na mAP. Essas informações são acompanhadas em tempo real e mostram a evolução do modelo, possibilitando a análise e verificação de possíveis erros, agilizando ações de correção.

Por fim, foi gerado o gráfico de inferência do modelo, utilizando o script *export_inference_graph.py* para ser utilizado em outras plataformas e avaliado utilizando as imagens de validação.

As figuras 7 e 8 (adaptada de [29]) apresentam respectivamente os diagramas dos dois principais scripts

utilizados no processo de concepção do modelo, treinamento e validação, ambos expõem resumidamente os principais passos, possibilitado a reprodução dos mesmos.

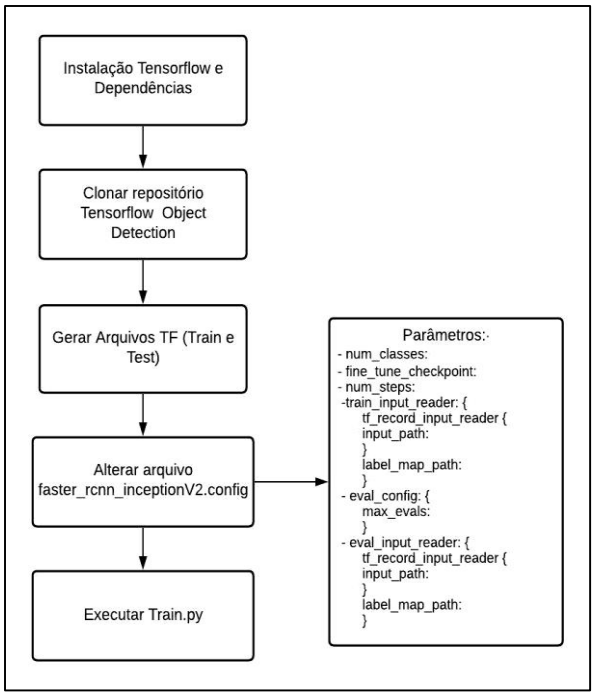

Figura 7 – Diagrama de Treinamento

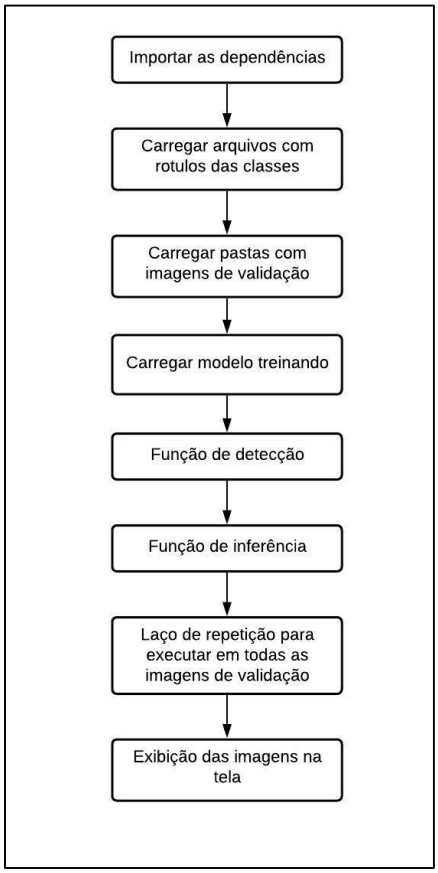

Figura 8 – Diagrama de Validação.

IV. Resultados

Com base no método proposto na seção III, o processo de desenvolvimento foi realizado em duas etapas, sendo a primeira a mais complexa que abrange todo o processo de geração do modelo exposto na Figura 6, a qual apresenta de forma detalhada as ações a serem executadas. A segunda etapa é a verificação do modelo utilizando *script* desenvolvido em Python para o carregamento do modelo e avaliação.

A. Geração do Modelo

O processo de geração do modelo foi baseado nas etapas expostas na seção III, estas etapas formam seguidas pelos autores e executadas em uma máquina virtual utilizando Sistema Operacional Linux Ubuntu. Foi criada uma classe de problemas onde foram alocadas todas as imagens devido à pouca variabilidade de amostras disponíveis. Esta classe foi chamada de "NOK" e as imagens onde não foram encontradas falhas não apresentam nenhuma marcação.

O modelo foi parametrizado para executar por 100.000 passos atingindo, conforme a Figura 9, uma perda de 0,04739. Esta figura exibe um gráfico que apresenta a redução da perda no eixo Y conforme os passos do treinamento são executados no eixo X, pode-se notar que ocorre diminuição da perda conforme a execução dos passos chegando próximo à zero. A Figura 10 apresenta uma *Mean Average Precision* (mAP) de 0,9494 no eixo Y sobre a evolução dos passos de treinamento expostos no eixo X, observou-se que ocorreu um aumento considerável da mAP em relação ao início do treinamento, porém no final do treinamento, constatou-se uma estagnação no aprendizado do modelo. Por esse motivo o treinamento foi parametrizado para 100.000 passos a fim de evitar a saturação do modelo e utilização de processamento desnecessário.

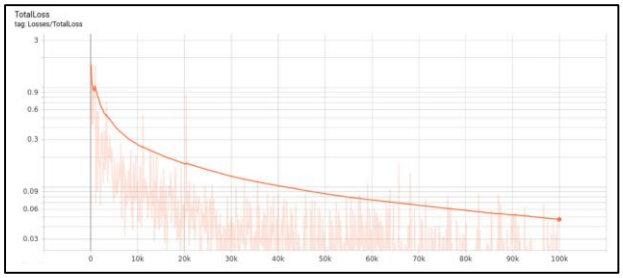

Figura 9 – Gráfico de perda

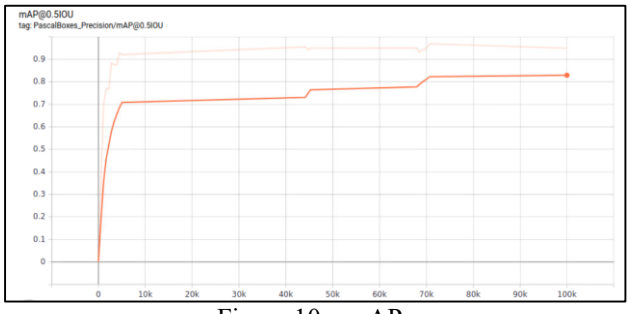

Figura 10 – mAP

A geração do modelo passou por um teste antes da geração do modelo final. O teste realizado foi o treinamento do modelo utilizando somente uma classe, concatenando nela todos os tipos de falhas. Nesta classe estavam disponíveis as imagens de todos os defeitos encontrados, aumentando o número de amostras na classe. Na primeira tentativa o modelo foi parametrizado para executar por 50.000 passos, porém apresentou falsos positivos e sobreposição de caixas

delimitadoras.

O resultado do teste anterior indicou que a utilização de somente uma classe para todos os defeitos era a melhor opção, por esse motivo o treinamento foi repetido, porém parametrizado para executar por 100.000 passos. Esta execução aumentou a acurácia do modelo dando origem ao modelo final utilizado neste artigo.

B. Avaliação do Modelo

O processo de avaliação do modelo foi realizado utilizando as imagens segregadas do processo de treinamento e teste, conforme exposto na seção III, sendo que estas imagens nunca formam apresentadas para o modelo até este momento, estas imagens foram designadas como imagens de validação.

Foi desenvolvido script em linguagem *Python* onde foi realizado o processo de carregamento do modelo, gerado no item A da seção IV, e das 25 imagens de validação para avaliação final do modelo. O modelo foi executado em dois conjuntos de dados. O conjunto de teste contendo 38 imagens com defeito e o conjunto de validação com 25, sendo que 22 imagens são de produtos com falhas e 3 sem nenhuma avaria.

A execução deste modelo nos conjuntos de dados citados acima resultou na matriz de confusão exposta na Tabela 1, onde apresenta os dados dos dois modelos gerados.

Nos dados de teste o modelo apresentou o mesmo resultado em ambas as execuções (50.000 e 100.000). Entretanto, quando o modelo foi executado utilizando o conjunto de dados de validação onde as imagens não haviam sido apresentadas ao modelo até este momento, o resultado foi melhor no modelo de 100.000 passos.

Tabela 1 – Matriz de confusão

Validação		
50.000	**Falha**	**Sem Falha**
Falha	20	2
Sem Falha	1	2

Teste		
50.000	**Falha**	**Sem Falha**
Falha	37	1
Sem Falha	0	0

100.000	**Falha**	**Sem Falha**
Falha	20	2
Sem Falha	0	3

100.000	**Falha**	**Sem Falha**
Falha	37	1
Sem Falha	0	0

Para análise do desempenho do modelo foram utilizadas as métricas de precisão, recall e acurácia apresentadas na Tabela 2, essas métricas mostram que o modelo de 100.000 passos teve um desempenho melhor no conjunto de dados de validação obtendo uma acurácia de 0,92. No conjunto de dados de teste o resultado seguiu o da matriz de confusão sendo o mesmo para ambos modelos.

Tabela 2 – Métricas do modelo

Validação		
Modelo	**50.000**	**100.000**
Precisão	0,9091	0,9091
Recall	0,9524	1,0000
Acurácia	0,8800	0,9200

Teste		
Modelo	**50.000**	**100.000**
Precisão	0,9737	0,9737
Recall	1,0000	1,0000
Acurácia	0,9737	0,9737

Esses dados mostram que o modelo treinando por mais passos teve um desempenho melhor nos dados de validação mesmo que os dados de testes indiquem que os modelos são iguais, apresentando as mesmas métricas. Isso reforça a necessidade de utilizar dados de validação para evitar que o modelo tenho um alto desempenho nos dados de teste e performance ruim quando colocado em produção. O modelo utilizando no processo de inferência das imagens foi o modelo de 100.000 passos pois apresentou métricas melhores. Este modelo apresentou uma média de acerto de 99%, sendo que o menor valor de acerto foi de 97% (um dos exemplos de inferência pode ser visto na Figura 11). Ocorreu em duas imagens a sobreposição de caixas delimitadoras no local do defeito, gerando 3 marcadores no mesmo local, ambas marcando corretamente a falha, porém com proporções diferentes.

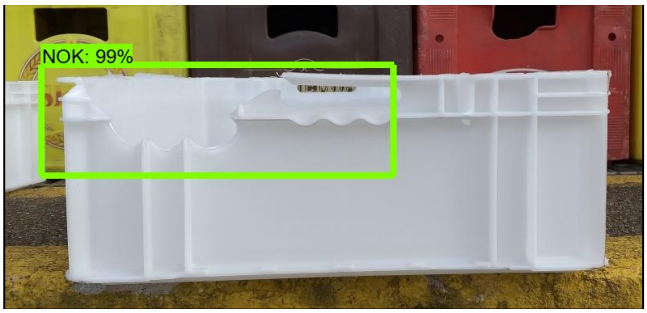

Figura 11 – Exemplo de detecção de falha.

V. CONCLUSÕES

Neste trabalho foi apresentada uma introdução sobre as Redes Neurais Convolucionais para a detecção de avarias em objetos. O resultado desse artigo foi o desenvolvimento de um modelo para identificação de falhas em embalagens plásticas.

O processo de geração de um modelo de detecção de objetos não é uma tarefa simples, requer análise e vários testes, conforme exposto a seção IV item B, para chegar a um resultado satisfatório. Como visto nesse trabalho, o primeiro modelo gerado não atendeu às expectativas, apresentando problemas e se mostrando ineficiente para o resultado esperado. Isso forçou os autores a executar os processos várias vezes a fim de encontrar um modelo que atendesse às necessidades propostas.

Outro ponto levantando no artigo é a quantidade de amostras necessárias para o desenvolvimento, pois, como as falhas são concentradas em lugares específicos, a quantidade de amostras disponível permitiu obter um modelo capaz de encontrar falhas.

O modelo proposto no artigo não foi colocado em produção, sendo que todas as validações foram realizadas fora do processo produtivo devido à falta de recursos operacionais para este fim. Além disso, a empresa trabalha com uma gama grande de produtos, onde se faz necessário a criação de um modelo para cada produto para a obtenção de ganhos de performance e análise.

É necessário, após a inserção do modelo em produção, a criação de uma sistemática de comparação dos resultados a fim de identificar quando será necessário treinar o modelo novamente, se ocorrer alterações significativas no processo produtivo e no local onde as falhas se concentram, o modelo pode começar a diminuir a acurácia das verificações.

Foi constatado neste artigo que o modelo teve uma acurácia elevada tanto nos conjuntos de dados de teste como dos dados de validação, isso sustenta a afirmação que não

ocorreu *Overfiting* neste modelo, mostrando que o modelo está ajustado e preparado para inserção em ambiente produtivo.

Uma análise necessária é o hardware a ser utilizado, pois impacta fortemente na performance da execução do modelo, o ideal é utilização de hardware específico como as placas Nvidia Jetson ou computadores com placa de vídeo dedicada Nvidia. Todo o processamento para a elaboração deste artigo foi realizado em CPU onde apresenta baixa performance tanto para validação do modelo como para treinamento mesmo assim atendeu as necessidades pois se de um processo de homologação.

O processo de treinamento foi finalizado em 100.000 passos, pois neste ponto foi alcançado acurácia a um nível elevado. Se o treinamento prosseguisse mais, a evolução perante ao modelo gerado não teria ganho significativo e, provavelmente, levaria à saturação da aprendizagem do modelo.

O resultado deste artigo foi considerado satisfatório, pois apresentou um modelo com acurácia elevada e expôs outra opção para análise de falhas sem a necessidade de interversão humana.

Aplicações de detecção de objeto utilizando *Tensorflow* pode ser usada nas mais diversas aplicações dentro da empresa, não somente no processo de detecção de falhas. O uso de Redes Neurais Convolucionais pode abranger diversas aplicações, como contagem de produtos para inventário, controles de estoque, movimentação de produtos e apontamentos de produção.

Para pesquisa futuras, recomenda-se a criação do mesmo modelo utilizando outras tecnologias de detecção de objeto a fim de comparar os resultados e desenvolvimento de modelos voltados para execução em ambiente mobile.

VI. BIBLIOGRAFIA

[1] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster, "Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry," Final report of the Industrie 4.0 Working Group, 2013.

[2] M. Hermann, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios," *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pp. 3928-3937, 2016.

[3] H. Lasi, H. Kemper, P. Fettke, and T. Feld, "Industry 4.0," *Bus Inf Syst Eng 6*, pp. 239–242, 2014.

[4] A. Krizhevsky, I. Sutskever, and H. Geoffrey, "ImageNet Classification with Deep Convolutional Neural Networks," 2012.

[5] R. Girshick, "Fast R-CNN," 2015.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[7] J. Schmidhuber "Deep Learning in neural networks: an overview," *Neural Networks*, vol. 61, pp. 65–117, 2014.

[8] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, pp. 1798–1828, 2013.

[9] G. Luger, "Inteligência Artificial 6. ed," *Person Education do Brasil*, p. 323, 2013

[10] S. Ren, K. He, R. Girshick, and J. Sun "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *IEEE Transactions on Pattern Analysis and Mahcine Intelligence*, vol. 39, pp. 137- 149, 2017

[11] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[12] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142-158, 2016.

[13] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. "Selective Search for Object Recognition," *Int J Comput Vis*, vol. 104, pp. 154–171, 2013.

[14] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vison and Pattern Recognition*, pp. 770-778, 2016

[15] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional neural networks," *ECCV*, pp. 818-833,2014.

[16] Z. Zou, Z. Shi, Y. Guo, and Ye, J. "Object detection in 20 years: A survey," 2019.

[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014.

[18] S. S. Haykin, "Neural networks and learning machines," *Pearson Upper Saddle River*, vol. 3, 2009

[19]G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[20] Application Programming Interface Objetct Detection Tensorflow. Available: https://github.com/tensorflow/models/tree/master/research/object_detection Accessed: 01-Set-2020.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: a system for large-scale machine learning," *12th USENIX conference on operating systems design and implementation, OSDI'16.*, pp 265–283, 2016.

[22] Tensorflow Model Zoo. Avaliable: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md Accessed: 01-Set-2020.

[23] Faster R-CNN Inception V2 Coco. Avaliable: http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_v2_coco_2018_01_28.tar.gz Accessed: 01-Set-2020.

[24] Coco Dataset. Avaliable: http://cocodataset.org/ Accessed: 01-Set-2020.

[25] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft coco: Common objects in context," *European conference on computer vision*, pp. 740–755, 2014.

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, "Rethinking the inception architecture for computer vision," 2015.

[27] Keras. Avaliable: https://keras.io/ Accessed: 01-Set-2020.

[28] Labelimg. Avaliable: https://github.com/tzutalin/labelImg Accessed: 01-Set-2020.

[29] Object Detection Tutorial. Avaliable: https://github.com/tensorflow/models/blob/master/research/object_detection/colab_tutorials/object_detection_tutorial.ipynb Accessed: 01-Set-2020.