XVIII CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

DESENVOLVIMENTO DE UM SISTEMA PARA INSPEÇÃO AUTOMATIZADA DE ITENS

Matheus Cardoso Araujo[1], Vanessa Batista Schramm[2]

RESUMO

A inspeção manual de itens pode ser uma atividade bastante pedante e suscetível a erros. Dessa forma, de modo a contornar isso, surge a necessidade de automatizar esse processo, reduzindo custos, erros e acelerando a sua velocidade de execução. Assim, por meio da construção de um protótipo de linha de produção com inspeção de item automatizada, foi possível analisar a qualidade de um determinado item. Primeiro, o produto é disposto em uma esteira que simula a linha de produção. Então, por meio de sensores, a esteira para com o item embaixo de uma câmera que irá fazer uma imagem do produto e verificar por meio de um modelo de predição se ele está defeituoso ou não. O modelo de predição foi gerado por meio de uma inteligência artificial, mais especificamente por redes neurais convolucionais, que apresentaram uma precisão de 90% na determinação do item como conforme ou defeituoso. Por fim, se ele é considerado como defeituoso, um braço robótico irá removê-lo da esteira. Se não, ele continuará normalmente até o fim do seu percurso, quando outro produto será colocado na esteira para análise.

Palavras-chave: Inspeção Automatizada de Itens, Redes Neurais, Visão Computacional.

[1] Aluno de Engenharia Elétrica, Departamento de Engenharia Elétrica, UFCG, Campina Grande, PB, e-mail: matheus.araujo@ee.ufcg.edu.br
[2] Doutora, Professora, Departamento de Engenharia de Produção, UFCG, Campina Grande, PB, e-mail: vanessa@labdesides.ufcg.edu.br

DEVELOPMENT OF A SYSTEM FOR AUTOMATED ITEM INSPECTION

ABSTRACT

Manual inspection of items can be a very pedantic and error-prone activity. Thus, in order to get around this, there is a need to automate this process, reducing costs, errors and accelerating its execution speed. Thus, by building a prototype of a production line with automated item inspection, it was possible to analyze the quality of a particular item. First, the product is laid out on a production belt that simulates the production line. Then, through sensors, the production belt stops the item under a camera that will take an image of the product and verify through a prediction model whether it is defective or not. The prediction model was generated by means of artificial intelligence, more specifically by convolutional neural networks, which showed an accuracy of 90% in determining the item as compliant or defective. Finally, if it is found to be defective, a robotic arm will remove it from the production belt. If not, it will continue normally until the end of its run, when another product will be placed on the production belt for analysis.

Keywords: Automated Item Inspection, Neural Networks, Computer Vision.

1 INTRODUÇÃO

A inspeção é um processo caracterizado pela avaliação minuciosa de determinada peça, amostra ou lote de modo a determinar se o produto em questão corresponde a certas especificações de qualidade (MONTGOMERY, 2016). Na indústria, a inspeção é uma técnica utilizada para separar itens defeituosos dos demais e, usualmente, é realizada por humanos. Por ser uma atividade bastante repetitiva e pedante, é muito suscetível à falha humana (PRIETO et al., 2002).

Nesse quesito, a inspeção automatizada tem sido uma alternativa bastante viável e promissora. Ela consiste na substituição da mão de obra humana por uma inteligência artificial no processo de inspeção, em que as práticas de visualização, tomada de decisão e atuação são repassadas à uma máquina (COSTA et al., 2018). Assim, as habilidades de visualização e de interpretação da informação recebida são passam a ser realizadas por uma máquina, a qual se torna capaz de extrair automaticamente características e indicar anormalidades em produtos.

Um grande avanço na área de visão computacional foi observado recentemente após a sua associação com o aprendizado de máquina (do termo em inglês *Machine Learning*). O aprendizado de máquina é um tipo de inteligência artificial, que tem como maior objetivo fazer com que os computadores se tornem capazes de aprender a partir de dados, de observações do mundo real, sem ser explicitamente programados pelas regras e lógica dos humanos (KHAN et al., 2018). Isso tem promovido a criação de algoritmos de visão computacional mais flexíveis e robustos, consequentemente, melhorando a performance de sistemas de visão práticos (KHAN et al., 2018).

Uma das subáreas do aprendizado de máquina é o aprendizado profundo (do termo em inglês *Deep Learning*). O aprendizado profundo é constituído por um conjunto de métodos que visam aprender hierarquias de características com características de alto nível a partir de uma hierarquia formada pela composição de características de baixo nível (GLOROT; BENGIO, 2010). O aprendizado profundo possibilitou grandes avanços devido à sua alta performance e capacidade de trabalhar com grandes volumes de dados. Dentre as principais características da aprendizado profundo, tem-se (KHAN et al., 2018): simplicidade – faz uso de blocos de arquiteturas básicas conectados em diversas camadas para gerar grandes redes neurais; escalabilidade – capacidade de trabalhar com grandes volumes de dados; e transferência de domínio – permite utilizar um mesmo modelo para diversas tarefas relacionadas.

Dessa forma, aliando o poder de análise oferecido pelas técnicas de aprendizado profundo, em especial das redes neurais convolucionais, e de visão computacional com a perspectiva de inspeção automatizada, o presente trabalho teve como objetivo associar termos teóricos e práticos para o desenvolvimento de um sistema artificialmente inteligente capaz de distinguir itens defeituosos de itens conformes, sendo também capaz de removê-los de uma linha de produção de forma completamente autônoma.

2 OBJETIVOS

2.1 Objetivo Geral

- O objetivo deste projeto é desenvolver um sistema para inspeção automatizada de itens para controle de qualidade.

2.2 Objetivos Específicos

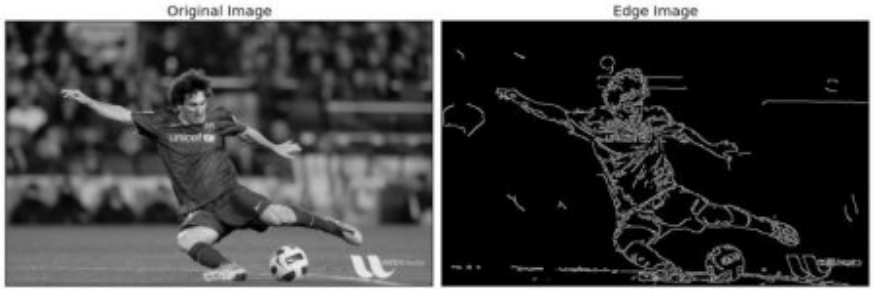
- Construir um protótipo de um processo industrial com inspeção automatizada de itens;
- Estudar e aplicar as tecnologias envolvidas nas tarefas de inspeção automatizada (visualizar, decidir e atuar);
- Desenvolver uma metodologia de ensino baseada em projeto em Indústria 4.0 para o curso de Engenharia de Produção/CCT da UFCG.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Visão Computacional

No início da década de 1970, a visão computacional era tida como a componente de percepção visual a ser replicada na tentativa de mimetizar a inteligência humana e endossar o comportamento de robôs inteligentes (SZELISKI, 2010). O que diferenciava a visão computacional do já existente campo de processamento de imagens era o anseio por recuperar a estrutura tridimensional do mundo a partir de imagens (KHAN et al., 2018). Dessa forma, os primeiros passos nesse caminho foram dados por meio da detecção de bordas, implementadas por meio da aplicação de filtros sensíveis às variações de intensidade de luz (SZELISKI, 2010), como é possível observar na Figura 1.

Figura 1 - Exemplo de aplicação de filtro de detecção de bordas



Fonte: OpenCV

Todas as atividades de detecção de aspectos da imagem evoluíram para se tornar o que hoje denominamos de extração de características. Semelhante à visão humana, são justamente essas características que irão definir o que foi visto pelo sistema por meio de um classificador. Assim, a tarefa do classificador é usar as características coletadas para relacionar uma imagem ou uma região de interesse à determinada categoria (KHAN et al., 2018). Ou seja, pode-se inferir que a combinação entre a extração de características e a classificação da informação coletada serão as componentes essenciais para um sistema básico de visão computacional.

3.2 Aprendizado de Máquina

O aprendizado de máquina pode ser amplamente definido como um conjunto de métodos computacionais que utilizam da experiência para melhorar a performance ou para fazer melhores predições (MOHRI et al., 2018). Ou seja, seguindo o fluxo contrário da programação usual de computadores, em que as regras são colocadas diretamente em forma de código, as técnicas de aprendizado de máquina envolvem o uso de grandes volumes de dados que serão analisados e interpretados pelo computador de modo a predizer quais características são compatíveis com cada uma das classificações colocadas e criar um modelo matemático correspondente que seja o mais aproximado possível da realidade (KHAN et al., 2018).

As técnicas de aprendizado de máquina evoluíram o suficiente para que grandes conquistas tenham sido alcançadas nos tempos atuais. Yao et al. (2018) citam a vitória da inteligência artificial Alpha GO contra um campeão mundial humano no jogo chinês de Go; e o fato de que o sistema de reconhecimento de fala da Microsoft é extremamente próximo do nível humano de transcrição de fala.

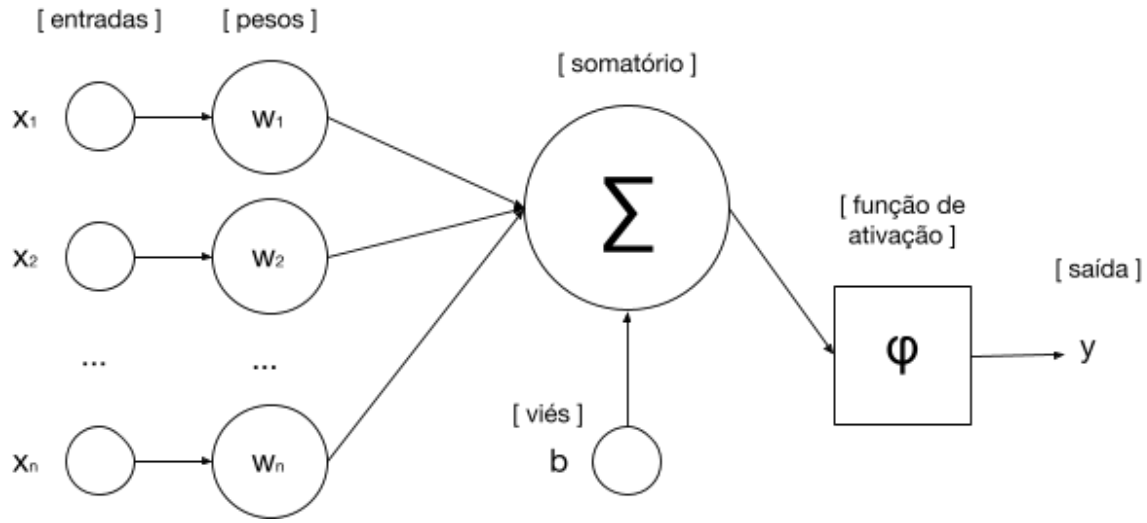
Ainda assim, as técnicas mais convencionais de aprendizado de máquina eram bastante limitadas em relação ao processamento de dados brutos e grandes, tais como conjuntos de imagens e vídeos, o que gerava a necessidade de engenheiros bastante cuidadosos e um considerável domínio de expertise no assunto para transformar esses dados de entrada em características interpretáveis pelo sistema (LECUN et al., 2015). É então que surgem as redes neurais artificiais de aprendizado profundo para contornar isso.

3.3 Redes Neurais Artificiais

Foi somente ao fim da década de 1980, com os avanços tecnológicos de integração em larga escala e com a possibilidade de construir hardwares paralelos com milhares de processadores, que o ramo de redes neurais artificiais foi reinventado como uma possível teoria para distribuir a computação de dados em um grande número de unidades de processamento (ALPAYDIN, 2020).

Inspirado no cérebro humano, em especial no córtex visual, as redes neurais artificiais apresentam como unidade básica o neurônio artificial, também conhecido como perceptron (TEUWEN; MORIAKOV, 2020). O perceptron, que consiste em uma representação matemática baseada no neurônio humano, foi proposto em 1958 pelo psicólogo americano Frank Rosenblatt no trabalho *The perceptron: a probabilistic model for information storage and organization in the brain* (TEUWEN; MORIAKOV, 2020). Na Figura 2, é possível observar a representação esquemática de um perceptron.

Figura 2 - Representação de um perceptron



Fonte: Elaborado pelo autor

Em resumo, o perceptron receberá a informação **x** (entrada), que é multiplicada por um peso **w**, e o resultado dessas operações é concatenado por meio de um somador, que reunirá todas as características atribuídas naquela rede neural, incluindo o viés **b**.

Então, tudo é repassado para a função de ativação **φ**, que é responsável por decidir se o conjunto de características atribuídas a esse perceptron irá disparar ou não (KHAN et al., 2018), ou seja, é a função de ativação que decide se os dados recebidos serão passados para a camada seguinte por meio de **y** (saída). As funções de ativação são não lineares e as mais populares são a função ReLu (*Rectified Linear Unit*) (Equação 1) e a função Sigmóide (Equação 2). A primeira mapeia o valor da entrada para zero se receber valores negativos ou o mantém inalterado caso a entrada seja positiva. Já a função Sigmóide é caracterizada por receber um valor real na entrada e emitir um valor entre 0 e 1 na saída.

Equação 1 - Função de ativação ReLu

$$f_{relu}(x) = max\,(0, x)$$

Equação 2 - Função de ativação Sigmóide

$$f_{sigm}(x) = \frac{1}{1+e^{-x}}$$

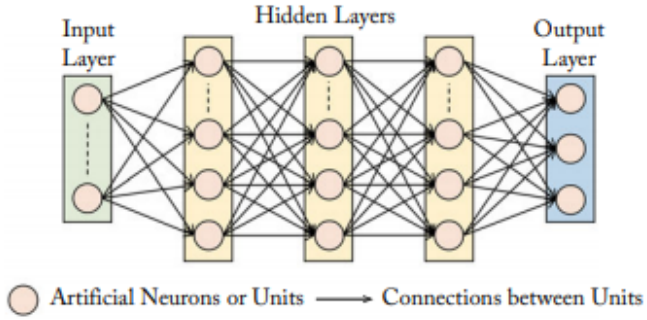
Por meio do ajuste dos pesos **w** e dos vieses **b**, na etapa de treinamento, a rede será calibrada a fim de se aproximar o máximo possível do modelo real. Esse ajuste de parâmetros é feito principalmente por um algoritmo de monitoramento de aprendizado denominado de *backpropagation* (TEUWEN; MORIAKOV, 2020). Para isso, ele faz uso dos métodos de erro médio quadrado e de gradiente descendente. Assim, os pesos das conexões da rede **w** e os vieses **b** serão alterados na tentativa de diminuir o somatório dos erros quadrados, ao mesmo tempo que são feitos cálculos do gradiente do erro relativo das amostras de treinamento (LI et al., 2012). Dessa forma, a fim de minimizar as diferenças entre o modelo de predição e do

modelo real, os parâmetros são ajustados e se tornam mais precisos a cada treinamento de amostras realizado.

Fazendo uso da concatenação de vários perceptrons, são criadas as camadas das redes neurais. Por sua vez, a associação de várias camadas de perceptrons permite que o nível de abstração e de complexidade cresça e, com isso, as camadas mais altas são capazes de amplificar os aspectos da entrada, que são importantes para a discriminação do modelo real. Esta prática é definida como técnica de aprendizado profundo (do inglês *deep learning*) em redes neurais (LECUN et al., 2015).

A arquitetura básica de uma rede neural artificial de múltiplas camadas está representada na Figura 3. Como é possível observar, sua organização é dividida entre camada da entrada, camadas ocultas e camada de saída (TEUWEN; MORIAKOV, 2020). A camada de entrada é responsável por receber os dados que alimentarão a rede durante o treinamento; se a análise em questão for a de uma imagem, por exemplo, os dados repassados à entrada serão os valores RGB dos pixels que a compõem. Já as camadas ocultas são caracterizadas por refinar o processamento da informação. Dessa maneira, a análise de características aliada ao peso conferido a cada uma das conexões entre as camadas irá direcionar os parâmetros do modelo de predição. Por fim, a camada de saída é responsável por fornecer um resultado de acordo com as classificações pré determinadas.

Figura 3 - Representação de uma rede neural artificial com multi camadas



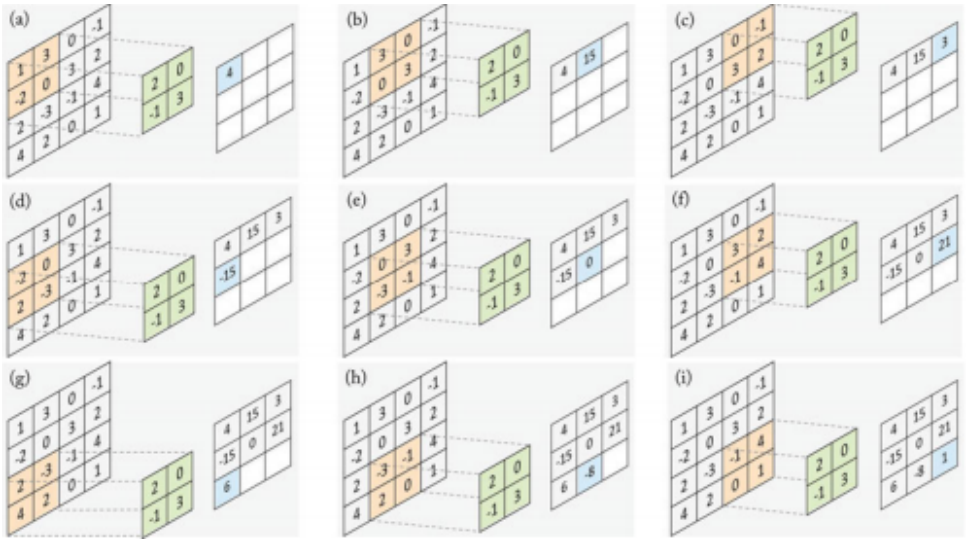
Fonte: Khan, 2018

3.4 Redes Neurais Convolucionais

Redes neurais convolucionais operam de forma bastante semelhante a redes neurais convencionais, entretanto, a diferença chave é que cada perceptron em uma camada de convolução é um filtro de duas ou mais dimensões que é convoluído com a entrada da camada (KHAN et al., 2018). Dessa maneira, essas camadas aprendem a reconhecer padrões visuais, por meio da extração de características locais, que, em seguida, são combinadas para obter representações de nível mais alto.

A Figura 4 representa bem como esse processo acontece. O filtro caracterizado pela cor verde será multiplicado por cada região da matriz de entrada, de forma a percorrê-la por completo. Então, o somatório das multiplicações de cada região irá resultar em uma saída correspondente, preenchendo assim todas as posições da matriz de saída (processo de convolução). É justamente por isso que as redes neurais convolucionais são bastante utilizadas para extração de características de imagens e de vídeos.

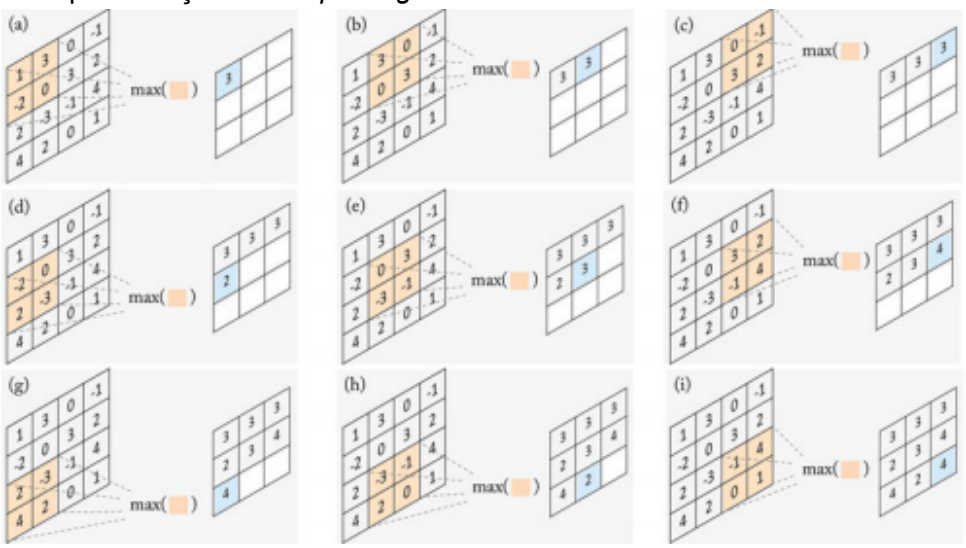
Figura 4 - Representação da saída (matriz azul) de uma convolução entre a entrada (matriz à esquerda) e um filtro (matriz verde)



Fonte: Khan, 2018

Uma técnica que é bastante associada com as redes neurais convolucionais é a aplicação de camadas de *pooling*. Em essência, a técnica de *pooling* consiste em aplicar um filtro que irá combinar as funções de ativação dos perceptrons da camada anterior de modo a diminuir a quantidade de dados na saída (KHAN et al., 2018). Essa combinação pode ser feita por meio de uma média dos valores de entrada ou por meio do valor máximo da região onde o filtro está sendo aplicado. Esse último, o *pooling* máximo, é exemplificado na Figura 5. Dessa maneira, ao mesmo tempo que a camada de convolução extrai características da entrada, a camada de *pooling* irá refinar essas características, diminuindo a quantidade de informações que o computador irá processar e realçando os padrões que serão observados na próxima camada.

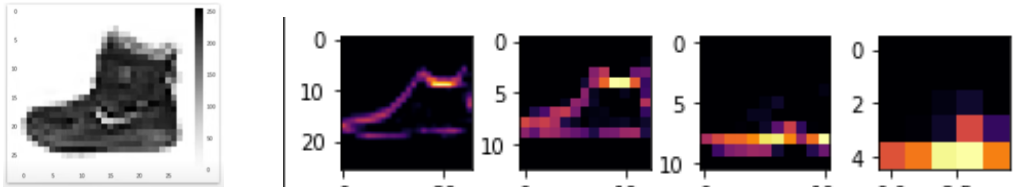
Figura 5 - Representação de um *pooling* máximo em uma matriz 4x4



Fonte: Khan, 2018

Um exemplo prático da combinação entre convolução e *pooling* pode ser observado na Figura 6.

Figura 6 - Efeito visual de uma convolução associada à *pooling*
a) Imagem original **b)** Imagem após processos de convolução e *pooling consecutivos*



Fonte: Tensor Flow

O processo de aprendizado de redes neurais convolucionais é um pouco diferente do convencional. Nas redes neurais convolucionais, o processo de treinamento é conduzido pela otimização dos parâmetros da rede a fim de minimizar a função de custo. Dessa forma, há a **função de otimização**, que é responsável por aprimorar os parâmetros observados, e a **função de custo**, que fornece a diferença entre a saída estimada do modelo e a saída real. A cada época de treinamento concluída, o processo de aprendizado aplica essas funções nas camadas da rede, aprimorando os valores dos pesos **w**, dos vieses **b** e também da taxa de aprendizado **η**, que quantifica o melhoramento nos parâmetros do modelo de predição.

A Equação 3 e a Equação 4 apresentam, respectivamente, um exemplo de função de custo e um exemplo de função de otimização. A primeira corresponde à função de custo *binary cross entropy*, em que $H(q)$ seria o custo da predição (diferença entre o modelo de predição e o valor real), N seria número total de amostras do treinamento, y corresponderia à saída real e $p\,(y)$ seria a saída prevista (por ser uma análise binária, esses dois últimos seriam os valores 0 ou 1). Já a segunda, é o otimizador RMSprop, em que w_t denota o peso atualizado, w_{t-1} seria o peso anterior, η é a taxa de aprendizado, $\frac{\delta C}{\delta w}$ seria o gradiente da função de custo em relação ao peso w_t, $E[g^2]_t$ seria a média móvel do quadrados dos gradientes e β seria o parâmetro de média móvel, cujo valor padrão gira em torno de 0.9 (KHAN, 2018).

Equação 3 - Função de *binary cross entropy*

$$H(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(p\,(y_i)) + (1 - y_i) * log(1 - p(y_i))$$

Equação 4 - Função de média móvel do quadrados dos gradientes e RMSprop

$$E[g^2]_t = \beta * E[g^2]_{t-1} + (1 - \beta) * \left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} * \frac{\delta C}{\delta w}$$

3.5 Hiperparâmetros

Todo sistema de aprendizado de máquina tem hiperparâmetros e a tarefa mais básica no aprendizado de máquina automatizado é definir automaticamente esses hiperparâmetros para otimizar o desempenho (FEURER; HUTTER, 2019). Hiperparâmetros nada mais são que as variáveis de um treinamento de aprendizado de máquina que, quando comparados com os parâmetros gerais do processo, podem ser vistos como pequenos detalhes, mas que certamente influem no resultado final do modelo de predição.

A quantidade de neurônios que vai em cada camada da rede neural ou até mesmo o tamanho dos filtros que serão aplicados nas camadas de convolução são exemplos de hiperparâmetros.

4 MATERIAIS E MÉTODOS (OU METODOLOGIA)

O projeto foi executado no Laboratório de Desenvolvimento de Sistemas de Apoio à Decisões Sustentáveis (DeSiDes) e foi executado de acordo com os cronogramas da Tabela 1 e da Tabela 2.

Tabela 1 - Cronograma de atividades da pesquisa no primeiro semestre

Fase	**SET**	**OUT**	**NOV**	**DEZ**	**JAN**	**FEV**
Fundamentação da Base Teórica						
Desenvolvimento da Rede Neural						
Testes da Rede Neural						
Elaboração do Relatório Parcial						

Tabela 2 - Cronograma de atividades da pesquisa no segundo semestre

Fase	**MAR**	**ABR**	**MAI**	**JUN**	**JUL**	**AGO**
Construção do *DataSet*						
Desenvolvimento do *hardware*						
Implementação do Braço Robótico						
Elaboração do Relatório Final						

Ainda que, de início, o desenvolvimento das atividades tenha sido planejado de uma maneira que *software* e *hardware* seriam implementados de forma simultânea, algumas modificações foram necessárias para se adequar ao contexto de distanciamento social imposto pela pandemia do COVID-19.

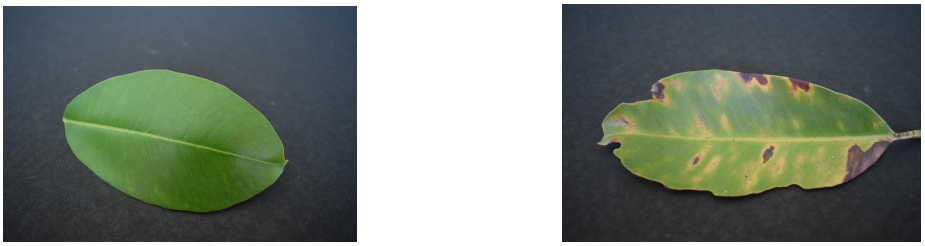
Ao longo dos três primeiros meses, foi realizada a revisão bibliográfica de toda a temática que envolve o projeto, abordando principalmente os tópicos de visão computacional, aprendizado de máquina e redes neurais convolucionais.

Do quarto ao quinto mês, foi realizada a segunda etapa do projeto, que consistiu no desenvolvimento da rede neural que iria gerir a inspeção de itens do sistema. A priori, a rede neural desenvolvida deveria ser alimentada já com o *dataset* final do projeto, ou seja, com imagens do item a ser analisado como conforme ou defeituoso. Entretanto, devido às restrições de *lockdown* e de distanciamento, toda a parte de desenvolvimento de *hardware* do projeto foi adiada para o segundo semestre de sua realização.

Assim, como forma de contornar isso, foi utilizado um banco de dados online e gratuito que fosse capaz de atender às especificações propostas. O *dataset* escolhido para isso foi o *Plant Leaves* do banco de dados do Tensor Flow (framework de aprendizado de máquina), que consiste em um conjunto de 22 espécies de folhas de plantas que foram categorizadas como saudáveis ou não saudáveis (correspondente a conforme e defeituoso, respectivamente) em um arranjo total de 4502 imagens em alta resolução. Exemplos de amostras deste *dataset* podem ser observados na Figura 7.

Figura 7 - Amostras de folhas do *dataset Plant Leaves*

a) folha saudável **b)** folha não saudável



Fonte: Tensor Flow

Inclusive, o Tensor Flow também foi escolhido para ser o framework principal que iria gerir a parte de *software* do sistema. Devido a sua simplicidade e rápida curva de aprendizado, ele permite criar redes neurais de grande complexidade com simples comandos escritos em Python, além de possuir uma vasta gama de bibliotecas integradas e associadas à popular API de aprendizado de máquina denominada de Keras.

Para a escrita do código, foi utilizado o Google Colab, que é uma plataforma de serviço em nuvem organizada em forma de *notebooks*, isto é, blocos de texto e de código dispostos de acordo com a necessidade do usuário. A linguagem de programação principal utilizada também é o Python, e por padrão já há integração com as principais bibliotecas disponíveis. Por ser justamente dedicada ao desenvolvimento e ao incentivo do aprendizado de máquina, a plataforma oferece também o uso de unidades de processamento gráfico em nuvem, democratizando o acesso à alta performance necessária para o treinamento e o desenvolvimento de redes neurais.

No que diz respeito aos hiperparâmetros da rede, a biblioteca Keras Tuner do Tensor Flow oferece medidas que viabilizam o teste variado de hiperparâmetros dentro de um conjunto de treinamentos. Assim, em essência, essa biblioteca possibilita que, dentro de um conjunto de valores pré estabelecidos, os hiperparâmetros sejam alocados de forma aleatória em várias iterações, a fim de compará-los ao final do processo e extrair as combinações que configuram o melhor modelo possível.

Dessa maneira, no segundo semestre do projeto, com o início da vacinação e a gradual queda nos casos de covid no país, as atividades no laboratório foram aos poucos sendo retomadas, tornando possível desenvolver de forma mais efetiva a parte de *hardware* do sistema. Em essência, o *hardware* principal do projeto pode ser resumido nas duas peças identificadas na Figura 8, que são correspondentes a Esteira Didática ARR9 e ao Braço Robótico ARR7, ambos da empresa nacional Automação ARR.

Figura 8 - Hardware principal do sistema, correspondente ao protótipo de uma linha de produção com inspeção de itens

a) Esteira Didática ARR9 **b)** Braço Robótico ARR7

Fonte: Automação ARR

Além delas, também foram utilizados o conjunto de peças e componentes eletrônicos, que estão todos descritos na tabela 3. O protótipo de hardware desenvolvido foi feito de modo a replicar uma linha de produção em que a esteira é responsável por comportar o produto em análise, uma câmera com vista superior da esteira iria verificar se o item em questão está conforme ou defeituoso, e, por fim, caso o objeto esteja defeituoso, o braço robótico irá remover ele da linha de produção.

Tabela 3 - Materiais e componentes eletrônicos utilizados no protótipo do projeto

Quantidade	**Material**
1	Esteira Didática ARR9
1	Braço Robótico ARR7
1	Webcam Logitech C920s
1	Arduino Uno
1	Mini Motor DC (6v) com Caixa De Redução
1	Transistor PNP TIP122
1	Regulador de Tensão (*step down*) LM2596
1	Fonte DC 12V

3	Diodo Laser
3	Fotoresistor (LDR)
3	Resistor 10k ohms
1	Resistor 330 ohms
2	Protoboard
Diversos	Fios do tipo Jumper

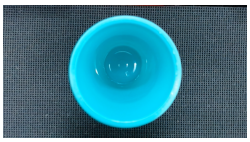
Além dos materiais já descritos, foi também utilizada a plataforma Arduino. Ela consiste em um ambiente de programação conectada com o *hardware* de modo a permitir a prototipagem de uma forma bastante acessível e simplificada. Ambos *softwares* e *hardware* da plataforma são *open source*. O modelo físico utilizado no projeto foi o Arduino Uno, que pode ser encontrado facilmente no mercado.

Por fim, considerando que o segundo semestre do projeto foi focado no desenvolvimento de todo o *hardware* do sistema, foi implementada nos primeiros 4 meses a conexão da esteira com os sistemas de visão computacional da câmera e de inteligência artificial para inspeção do item em questão. Para tal, foi utilizada uma rotina em Python no Visual Studio da Microsoft. Nos últimos dois meses, também foi feita a integração do braço robótico nessa mesma rotina em Python, que unia todos os subsistemas.

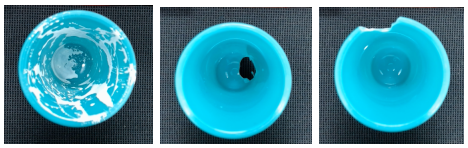
Não menos importante, o *dataset* final que iria alimentar a rede neural produzida no primeiro semestre do projeto também foi gerado, totalizando 155 imagens de itens conformes e com defeito. O item escolhido para ser objeto de estudos da inspeção em questão foram copos de plástico, devido ao fácil acesso e também ao baixo custo, que foram categorizados em 2 grupos diferentes, sendo eles: os copos conformes e os copos defeituosos. De modo a aprimorar a análise da rede neural, foram produzidos três tipos diferentes de defeitos (pintura, buracos e bordas quebradas), como é possível observar na Figura 9, ainda que a rede neural em si seja capaz de distinguir apenas itens conformes de defeituosos.

Figura 9 - Amostras do *dataset* criado para alimentar a rede neural e servir como item a ser analisado no processo de inspeção automatizada

a) Exemplo de copo conforme



b) Exemplos de copos defeituosos



Fonte: Elaborado pelo autor

5 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados e discutidos os principais resultados obtidos durante a pesquisa, os quais estão divididos em três subseções, sendo a primeira referente aos resultados de *software*, a segunda aos resultados de *hardware* e a terceira à integração do sistema.

5.1 Resultados de *software*

Como foi dito anteriormente na seção de metodologia, durante o primeiro semestre, a rede neural responsável por predizer se o item analisado é defeituoso ou não foi desenvolvida. Assim, com o *dataset Plant Leaves* do Tensor Flow, os testes começaram a ser executados. Deste dataset, foram utilizadas apenas 624 imagens, as quais correspondem à espécie da árvore de Jamun. Isso foi feito com a finalidade de promover uma maior homogeneidade entre os itens avaliados, uma vez que as folhas de diferentes espécies podem apresentar bastantes discrepâncias entre si.

Entretanto, é necessário salientar que ao se construir uma rede neural artificial, o primeiro passo a ser tomado é o pré-processamento dos dados. Logo, as imagens do *dataset* foram separadas em pastas seguindo a proporção referenciada na Tabela 4. Essa proporção segue o padrão de 80% (499 imagens) dos dados destinados ao treinamento da rede e 20% (125 imagens) destinados à validação da rede. É importante que os dados de validação sejam isolados dos dados de treinamento para que a rede seja testada com informações que ela não teve contato prévio, aumentando sua confiabilidade.

Também foi decidido que, para a finalidade do projeto, o uso de redes neurais convolucionais em conjunto com as técnicas de aprendizado profundo seriam uma alternativa mais interessante do que a proposta inicial de filtrar e categorizar as imagens de treinamento de forma manual e posteriormente classificá-las por meio de uma rede neural mais simples. Com a nova metodologia, seria possível automatizar, por fim, todo o processo de inspeção do produto, desde o treinamento da rede neural até a análise do item como conforme ou não conforme.

Em seguida, por meio da biblioteca Image Data Generator do framework Keras, as imagens foram redimensionadas e normalizadas. Ambas as técnicas são utilizadas para melhorar o desempenho do treinamento, uma vez que o redimensionamento diminui a quantidade de dados a serem analisados, bem como a quantidade de informações desnecessárias que a máquina irá processar, como também, a normalização auxilia nos métodos matemáticos, cujas variáveis passam a variar entre valores de 0 a 1, em vez de 0 a 255 (valores correspondentes às cores de cada pixel da imagem no padrão RGB).

Em seguida, a arquitetura da rede foi definida como mostra a Tabela 3. Na entrada da rede, há uma camada de *pooling* médio de modo a refinar os dados e características que serão analisados pelas camadas convolucionais. Em seguida, há três camadas de convolução intercaladas com três camadas de *pooling* máximo. Isso é feito para que as características que forem observadas pelas camadas de convolução iniciais sejam realçadas e simplificadas, consequentemente, otimizando o treinamento das camadas seguintes.

Após isso, foi inserido uma camada de *dropout* com uma taxa de rejeição de 0.5. O *dropout* atua na rede neural descartando de forma aleatória algumas das características coletadas por um conjunto de perceptrons. Com uma taxa de 0.5, isso significa que metade das características repassadas pela camada anterior será descartada, o que, mais uma vez, auxilia na otimização do treinamento em questões de processamento e de tempo, além de, por muitas vezes, remover características redundantes para a análise do modelo de predição.

Tabela 3 - Arquitetura geral da rede neural do projeto

Camada	Tipo	Função de Ativação
1	*Pooling* Médio	-
2	Convolução	ReLu
3	*Pooling* Máximo	-
4	Convolução	ReLu
5	*Pooling* Máximo	-
6	Convolução	ReLu
7	*Pooling* Máximo	-
8	*Dropout*	-
9	*Flatten*	-
10	Densa	ReLu
11	Densa	Sigmóide

Então, há uma camada *Flatten*, que transforma as matrizes com os dados da imagem em um vetor único, o qual alimentará a posterior camada Densa, que é responsável por classificar de forma isolada todas as características que foram extraídas da imagem até essa parte do percurso. Por fim, há mais uma camada Densa, responsável pela classificação final do sistema, que terá como saída o valor final de 0 ou 1, correspondentes às predições de conforme ou defeituoso do sistema, respectivamente.

Em relação às funções aplicadas em cada camada, nas camadas ocultas, a função de ativação utilizada foi a função ReLu. Já na camada de saída, a função escolhida foi a função Sigmóide. A primeira escolha se deu pelo melhor desempenho oferecido para um grande volume de dados, que acontece na extração de características dos dados de entrada pelas camadas convolucionais. Já a segunda escolha se deu porque ela oferece uma maior precisão na determinação de um valor de saída entre 0 e 1, como é o caso em questão, que consiste nas classificações: conforme (saída mais próxima de 1) ou defeituoso (saída mais próxima de 0).

Para as funções de custo e de otimização, a escolha foi pelas funções de *binary cross-entropy* e de RMSprop, respectivamente. O objetivo maior dessas escolhas foi orientado pelo desempenho do treinamento. A primeira oferece um bom desempenho para classificações binárias, oferecendo a diferença probabilística entre a saída estimada e a saída real de maneira bastante rápida. Já a segunda é um otimizador bastante efetivo devido à sua simplicidade em minimizar mudanças bruscas na taxa de aprendizado, guiando para melhores modelos de predição, além de também oferecer uma boa velocidade de processamento (REDDY; RAO; RAJU, 2018).

Com relação aos hiperparâmetros, tendo em vista que não há métricas específicas a serem seguidas, foi escolhido de forma totalmente arbitrária que os filtros das camadas de convolução, assim como os filtros das camadas de *pooling*, ambos teriam o tamanho de 3x3. Em contrapartida, os valores dos perceptrons presentes em cada camada foram acertados de forma semi aleatória pela biblioteca Keras Tuner. Isto é, foi definido um conjunto de valores (16, 32, 64, 128, 256, 512) de forma arbitrária, os quais foram combinados de maneira aleatória até que o melhor resultado de combinações fornecido pela biblioteca foi utilizado no modelo final de predição.

Todo esse conjunto de dados, funções e parâmetros resultou, após ser alimentado com o *dataset Plant Leaves* em um modelo de predição caracterizado por um treinamento de 27 minutos e 52 segundos. Em relação ao treinamento propriamente dito, a taxa de precisão de predições corretas foi de 91.79%, enquanto que nas predições realizadas com os dados de validação, a taxa de precisão foi de 88%. Isto é, na etapa de avaliação do modelo de predição proposta pela rede neural, com os dados que ele não teve contato prévio, a taxa de precisão de 88% significa que foi possível predizer de forma correta um total de 110 imagens num espaço amostral de 125 imagens do conjunto total de dados de validação.

Então, no segundo semestre, no que diz respeito ao *dataset* desenvolvido com os copos conformes e defeituosos, que são de fato os itens de análise final do processo de inspeção automatizada desse projeto, o resultado também se mostrou bastante satisfatório. Utilizando a mesma rede neural, porém, agora alimentada com o *dataset* desenvolvido, o pré processamento dos dados foi feito de acordo com a Tabela 4, totalizando 155 imagens também divididas segundo a proporção de 80% (125 imagens) destinadas ao treinamento e 20% (30 imagens) à validação.

Tabela 4 - Divisão de dados para alimentar a rede neural do sistema

	Treinamento	Validação
Conforme	62	15
Defeituoso	63	15
Total	125	30

Assim, esse conjunto de dados resultou em um modelo de predição caracterizado por um treinamento de 7 minutos e 47 segundos. No treinamento, a taxa de precisão de predições corretas foi de 88,89%, enquanto que nas predições realizadas com os dados de validação, a taxa de precisão foi de 90%. Isto é, na etapa de avaliação do modelo de predição proposta pela rede neural, com os dados que ele não teve contato prévio, a taxa de precisão de 90% significa que foi possível predizer de forma correta um total de 27 imagens num espaço amostral de 30 imagens do conjunto total de dados de validação. O código do *notebook* Google Collab da rede neural em questão se encontra no Anexo 1.

5.2 Resultados de *hardware*

Em relação ao *hardware*, o primeiro ponto a ser definido seria a alimentação de energia do sistema. O Braço Robótico ARR7 possui três servos motores do

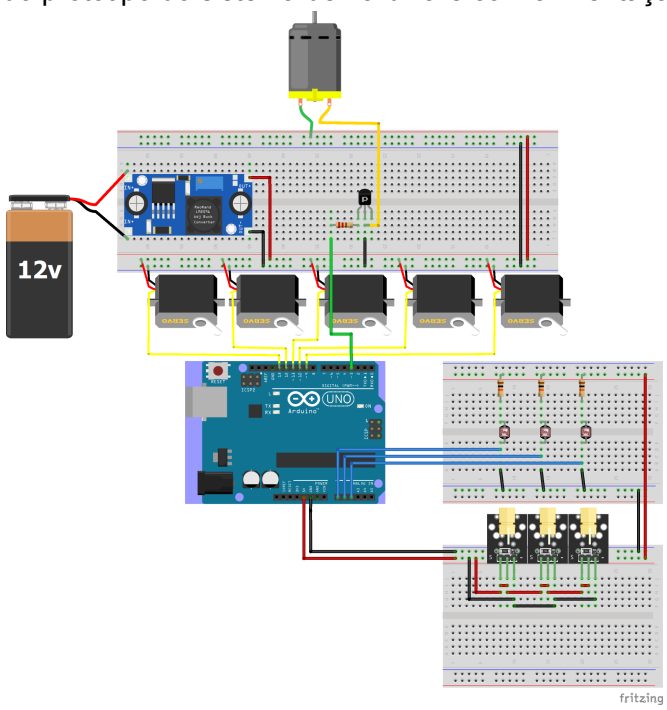
modelo MG90S e dois do modelo MG995, em que ambos são alimentados com 6V de tensão e demandam cerca de 100 a 500 mA de corrente cada. De forma semelhante, o motor DC que faz a esteira girar precisa de uma alimentação de 6V, e dependendo da sua velocidade, demanda uma corrente de aproximadamente 200 mA.

Dessa forma, como é bastante escasso no mercado fontes de baixa tensão com um fornecimento de corrente maior que 1,5 A, a solução mais viável foi utilizar uma fonte de 12V e 3A associada com o módulo regulador de tensão LM2596 que permite reduzir a tensão de entrada ao mesmo tempo em que fornece até 3A de corrente na saída. Então, o módulo foi regulado para fornecer 6V para o sistema, alimentando os servo motores do braço robótico e o motor DC da esteira. Também é válido ressaltar que o controle do motor DC foi feito com o transistor PNP TIP122, controlado pela saída PWM do Arduino com o auxílio de um resistor para diminuir a corrente que chega na base do transistor.

Os sensores de movimento da esteira foram todos alimentados pela saída de 5V do Arduino. Tanto a parte de emissão de luz, que fica por responsabilidade dos diodos lasers, como também a parte de recepção da luz, que fica a cargo de um divisor de tensão disparado por fotoresistores. Assim, os diodos laser estão sempre iluminando os fotoresistores, que mantêm uma tensão constante devido a isso. Quando algum objeto passa na frente do laser e essa iluminação é interrompida, sua resistência é alterada e consequentemente sua tensão também. Dessa forma, conectando os fotoresistores na entrada analógica do arduino, é possível ler essa variação e detectar em que lugar da esteira se encontra o objeto, disparando o motor DC quando o objeto chega no começo da esteira, parando o motor DC quando ele chega no meio da esteira para análise pela câmera, e, por fim, verificando se o objeto conforme realmente chegou até o fim da esteira.

Um esboço do protótipo do sistema pode ser observado na Figura 10.

Figura 10 - Esboço do protótipo do sistema de hardware com alimentação e sensores



Fonte: Elaborado pelo autor

Além desses componentes menores, também há a webcam e a placa do Arduino conectados no *hardware* do sistema. Ambos foram alimentados pelas portas USB de um notebook, o que também facilitou a comunicação de ambos com o sistema integrado que estava em execução para combinar todas as partes do projeto.

De modo a gravar e executar os movimentos do braço robótico que seriam capazes de remover o item defeituoso da esteira, foi utilizada a biblioteca para Arduino *VarSpeedServo*. Por meio da saída PWM, ela é capaz de controlar a velocidade e a angulação de cada um dos servo motores conectados no arduino. O código em Arduino responsável por coordenar essa movimentação e também por emitir mensagens via Monitor Serial para o computador se encontra no Anexo 2.

5.1 Resultados da integração do sistema

Para integrar o sistema, foi utilizada uma rotina em Python, contando principalmente com as bibliotecas OpenCV para realizar a captura e o gerenciamento de imagens por parte da webcam, além da biblioteca de Serial, para estabelecer uma conexão com o Arduino e disparar o braço ou a esteira de acordo com a predição feita pelo modelo salvo pelo Tensor Flow. O programa usado para executar a rotina foi o Microsoft Visual Studio.

Então, resumidamente, quando o sensor infravermelho do meio da esteira detecta o item, o Arduino envia pelo serial uma mensagem para a rotina em Python, que por sua vez envia uma mensagem para a webcam e faz com que ela tire uma foto do item parado na esteira. Essa imagem é comparada com o modelo de predição que está salvo na rotina, e de acordo com o resultado da comparação, irá enviar duas possíveis mensagens de volta para o Arduino.

Isto é, se o resultado da comparação predizer que o item em análise é defeituoso, então a mensagem enviada para o Arduino irá ativar o braço robótico e remover o item da esteira. Já se a comparação prediz que o objeto é conforme, então ele enviará uma mensagem pro Arduino que irá ativar a esteira, e o item continuará normalmente seu percurso pelo protótipo da linha de produção.

Nos testes realizados, o sistema integrado não apresentou grandes problemas visíveis. As partes funcionaram em harmonia e de forma satisfatória. Um único ponto a ser ressaltado é que após determinado tempo em execução (mais de 5 minutos ininterruptos), os servo motores do braço robótico começaram a esquentar bastante, podendo ocasionar até mesmo uma possível queima dos componentes. Ainda assim, com um tempo de uso moderado, as peças funcionaram perfeitamente.

O código da rotina em Python que integra todas as partes do sistema pode ser encontrado no Anexo 3.

6 CONCLUSÃO

O poder da análise de dados por meio de redes neurais convolucionais realmente se mostrou bastante eficiente e justificou o status de ser considerado o estado da arte no que diz respeito ao desenvolvimento de modelos de predição, principalmente no quesito relacionado à análise de imagens.

Tanto com o *dataset* do banco de dados online *Plant Leaves* quanto com o *dataset* produzido, a rede neural apresentou bons resultados, girando em torno dos 90% de precisão em suas predições com os dados de testes. Ainda assim, é válido salientar o fato de que, para evitar problemas como o *overfitting* (quando a rede

neural responde bem para um determinado escopo de dados, mas não é capaz de generalizar e interpretar bem dados mais diversos), é interessante usar técnicas como *data augmentation*, onde é feita a aplicação de filtros que, por exemplo, alteram a luz e talvez a rotação ou a coloração das imagens, multiplicando a quantidade de dados e melhorando a capacidade de generalização da rede.

Sobre o *hardware* do projeto, a plataforma Arduino se mostrou bastante confiável e simples de ser utilizada. Todavia, fazer uma melhoria de *hardware* do Arduino Uno para o Arduino Mega talvez seja uma boa escolha para o futuro, uma vez que este possui mais portas de entrada e de saída, principalmente no que diz respeito a portas de entrada analógica, que permitiriam a inserção de mais sensores e outros periféricos no sistema.

De forma geral, as ferramentas do Tensor Flow, OpenCV e a comunicação Serial se mostraram bastante eficientes, oferecendo um resultado satisfatório e que devem ser mantidos em futuras versões do projeto.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (PIBIC/CNPq-UFCG) do Brasil.

Os autores agradecem ao Laboratório DeSiDeS da UFCG/Departamento de Engenharia de Produção pela estrutura disponibilizada, além de todo o amparo e compreensão dada a situação extraordinária provocada pela pandemia.

Também é cabível um agradecimento especial ao grupo PET Elétrica UFCG e ao Capítulo Estudantil IEEE RAS UFCG, que foram de grande ajuda na resolução de imprevistos técnicos e no suporte para a finalização do projeto.

REFERÊNCIAS

1 MONTGOMERY, D. C. ***Introdução ao Controle Estatístico da Qualidade*.** 7a. ed. São Paulo: LTC, 2016. ISBN 978-8521630241.

2 PRIETO, Flavio et al. **An automated inspection system.** The International Journal of Advanced Manufacturing Technology, v. 19, n. 12, p. 917-925, 2002.

3 COSTA, I. M. V.; ALCALÁ, S. G. S.; MENDONÇA, L. C. **Um Estudo de Caso de Redes Neurais Artificiais para a Detecção de Objetos Defeituosos Numa Esteira de Produção Automatizada.** In: XXXVIII Encontro Nac. Eng. Produção. Maceió: ABEPRO, 2018.

4 SZELISKI, Richard. **Computer vision: algorithms and applications.** Springer Science & Business Media, 2010.

5 KHAN, Salman et al. **A guide to convolutional neural networks for computer vision.** Synthesis Lectures on Computer Vision, v. 8, n. 1, p. 1-207, 2018.

6 GLOROT, Xavier; BENGIO, Yoshua. **Understanding the difficulty of training deep feedforward neural networks.** In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010. p. 249-256.

7 MOHRI, Mehryar; ROSTAMIZADEH, Afshin; TALWALKAR, Ameet. **Foundations of machine learning.** MIT press, 2018.

8 YAO, Quanming et al. **Taking human out of learning applications: A survey on automated machine learning.** arXiv preprint arXiv:1810.13306, 2018.

9 LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. **Deep learning.** nature, v. 521, n. 7553, p. 436-444, 2015.

10 ALPAYDIN, Ethem. **Introduction to machine learning.** MIT press, 2020.

11 TEUWEN, Jonas; MORIAKOV, Nikita. **Convolutional neural networks.** In: Handbook of Medical Image Computing and Computer Assisted Intervention. Academic Press, 2020. p. 481-501.

12 ROSENBLATT, Frank. **The perceptron: a probabilistic model for information storage and organization in the brain.** Psychological review, v. 65, n. 6, p. 386, 1958.

13 LI, Jing et al. **Brief introduction of back propagation (BP) neural network algorithm and its improvement.** In: Advances in computer science and information engineering. Springer, Berlin, Heidelberg, 2012. p. 553-558.

14 FEURER, Matthias; HUTTER, Frank. **Hyperparameter optimization. In: Automated Machine Learning.** Springer, Cham, 2019. p. 3-33.

15 REDDY, R. Vijava Kumar; RAO, B. Srinivasa; RAJU, K. Prudvi. **Handwritten Hindi digits recognition using convolutional neural network with RMSprop optimization.** In: 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS). IEEE, 2018. p. 45-51.