UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

JOÃO VITOR RODRIGUES - 00243705

CLASSIFICAÇÃO DE PEÇAS MECÂNICAS A PARTIR DE VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA UTILIZANDO IMAGENS SINTÉTICAS

Porto Alegre
2020

JOÃO VITOR RODRIGUES - 00243705

CLASSIFICAÇÃO DE PEÇAS MECÂNICAS A PARTIR DE VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA UTILIZANDO IMAGENS SINTÉTICAS

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

ORIENTADOR(A):
Prof. Dr. Renato Ventura Bayan Henriques

CO-ORIENTADOR(A):
Dr. Carlos Solon Soares Guimarães Junior

Porto Alegre
2020

JOÃO VITOR RODRIGUES - 00243705

CLASSIFICAÇÃO DE PEÇAS MECÂNICAS A PARTIR DE VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA UTILIZANDO IMAGENS SINTÉTICAS

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo(a) Orientador(a) e pela Banca Examinadora.

Orientador(a): _____
Prof. Dr. Renato Ventura Bayan Henriques, UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

Banca Examinadora:

Prof. Dr. Renato Ventura Bayan Henriques, UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

Prof. Dr. Alcy Rodolfo dos Santos Carrara, UFGRS
Doutor pela McMaster University – Hamilton, Canadá

Prof. Dr. Heraldo José Amorim, UFGRS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

_____
Marcelo Götz
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, novembro de 2020.

DEDICATÓRIA

Pai, essa é pra ti. Se eu cheguei até aqui foi porque tu batalhou muito para que eu chegasse. Tu foi o melhor amigo que tive e que eu jamais vou esquecer. Está sendo muito difícil seguir sem ti, sem a tua motivação, me xingando e falando que eu deixo tudo pra última hora. Bem, cá estamos, digo, estou, sem você.

É veio, quando tivemos o nosso momento de maior felicidade, que pudemos realizar o sonho de cada volta de Porto Alegre, quando passávamos em frente a UFRGS e tu me dizia "um dia tu vai estudar aí e ainda vai fazer intercâmbio", e quando finalmente aconteceu, a vida nos pregou uma peça. Mesmo assim tu batalhou! Tu batalhou por mim! Tu batalhou para me esperar! Tu batalhou para ver eu me formar... Não deu. Espero que tu consigas ver agora.

Ah Pai, como eu queria que tu estivesses na minha apresentação, assim como eu fui na do teu mestrado. Queria ver tu rindo de mim, assim como eu ri de ti a cada gaguejada e "né"pronunciado.

Esse trabalho foi diferente do que imaginei, esse momento foi diferente do que imaginei, essa dor, eu nunca tinha imaginado. A vida segue e, quando toda essa "função"do TCC acabar, eu espero poder refletir e entender o que aconteceu.

É difícil pensar que tu não vai me dar aquele parabéns sarcástico, e que não tenho tu para corrigir os erros que devem existir nesse documento, assim como fizestes para Morgs.

Pai, meu tempo tá acabando para escrever aqui. Mas o tempo é relativo né? Ele é curto para escrever algumas linhas de texto que compravam que estou apto a seguir adiante em minha vida profissional, mas é longo de mais para esperar o dia em que nos reencontraremos.

Pai, Obrigado!

Pai, Te Amo!

Eu vou fazer de tudo que eu puder para ser uma pessoa tão boa quanto tu!

O MEU CARRETEL VAI SER O MAIS COLORIDO POSSÍVEL!

AGRADECIMENTOS

Eu poderia ficar horas e horas agradecendo a todas as pessoas que, de alguma forma, estiveram presentes na minha vida ao longo desses 7 anos:

Minha noiva.

Minha família.

Meu(s) cachorro(s), mesmo que por apenas 2 semanas, Duke, você é tão importante quanto o Luke. Sem cíumes, amo os dois.

Cada um dos amigos que estiveram comigo, sejam da faculdade ou da escola, rindo, chorando, comemorando ou lamentando.

Cada pessoa que eu conheci nesse mundo durante o Duplo Diploma.

Aos colegas do ISISIM.

Enfim, são muitas pessoas e, são quase meia noite, e eu tenho que encaminhar esse documento aos professores. Ah! Agradeço a eles também, pois são parte essencial dessa jornada.

Como eu dizia, o tempo está passando e eu não poderia esquecer ninguém, mas vou, certamente.

Só não posso esquecer de uma pessoa: **Fábio**, muito obrigado.

...

...

...

...

...

...

...

...

...

...

...

Fernanda, meu amor. É lógico que tu merece um agradecimento também. Agradecimento por ter me aguentado todos esses anos e, mesmo assim, ter ido até o outro lado do mundo atrás de mim. Eu te amo, Zuxinha!

RESUMO

Aplicações de aprendizado de máquina para reconhecimento de padrões têm crescido nos últimos anos e, atualmente, uma gama de estruturas e ferramentas permitem um rápido desenvolvimento em diversos domínios. Dentre as principais evoluções destaca-se a utilização de classificação e detecção de objetos em imagens a partir de redes neurais convolucionais profundas (*Deep Convolutional Neural Networks*). Um dos problemas observados ao desenvolver aplicações de aprendizagem de máquina é a necessidade de um grande volume de dados para treinamento das redes, que necessitam na ordem de centenas, por vezes milhares, de imagens com seus respectivos rótulos (*labels*). Consequentemente, a parte de processamento de dados é responsável pelo maior consumo de tempo, esforço computacional e manual. Dentro do contexto de aprendizado supervisionado, propõe-se o desenvolvimento de uma arquitetura de sistema para reconhecimento de peças mecânicas com uma solução para a geração de imagens sintéticas a partir dos modelos 3D das peças, através de ferramenta de modelagem gráfica, de maneira automática e escalável. O sistema permite gerar imagens sintéticas, variando diversos parâmetros, como iluminação do ambiente, textura, tamanho e pose das peças geradas no *dataset*. A validação da proposta baseou-se em alimentar uma rede neural convolucional para a classificação de peças mecânicas. Os testes realizados mostraram que a solução proposta é potencialmente útil para a a problemática de falta de dados no contexto de aprendizado de máquina voltado à classificação por imagens, e pode ser desenvolvida para aplicações em ambientes reais industriais.

Palavras-chave: Redes Neurais Convolucionais, Processamento de dados, Aprendizado de Máquina, Modelos CAD 3D, Imagens sintéticas.

ABSTRACT

Machine learning applications for pattern recognition have grown in recent years, and today, a range of frameworks allow for rapid development in several domains. Among the main developments, the use of classification and detection of objects in images from deep convolutional neural networks stands out. One of the problems observed when developing machine learning applications is the need for a large volume of data for training these networks, which require hundreds, sometimes thousands, of images with their respective labels. Consequently, the data processing part is responsible for the largest consumption of time, computational and manual effort. Within the context of supervised learning, it is proposed to develop a system architecture for the recognition of mechanical parts with a solution for the automatic and scalable generation of synthetic images from the 3D models of the parts, through a graphic modeling tool. The system allows to generate synthetic images, varying several parameters, such as ambient lighting, texture, size and pose of the parts generated in the dataset. The validation of the proposal was based on feeding a convolutional neural network for the classification of mechanical parts with these images. The tests carried out showed that the proposed solution is potentially useful for the problem of lack of data in the context of machine learning aimed at classification by images, and can be developed for applications in real industrial environments.

Keywords: Convolutional Neural Networks, Data processing, Machine learning,3D CAD Models, Synthetic images.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	*Application Programming Interface*
CAD	*Computer Aided Design*
CV	*Computer Vision*
CNN	*Convolutional Neural Network*
CNNs	*Convolutional Neural Networks*
DL	*Deep Learning*
FC	*Fully Connected*
IA	*Inteligência Artificial*
ML	*Machine Learning*
MLP	*Multi Layer Perceptron*

1 INTRODUÇÃO

A necessidade de aumentar a produtividade e melhorar a qualidade na fabricação de produtos exige que as indústrias implementem novas tecnologias em suas linhas de produção. O monitoramento de tais linhas para o acompanhamento do processo fabril permite a detecção de falhas em produtos ou ordens de serviço, bem como a realização de manutenção preventiva. Estudos realizados em (CHIN; HARLOW, 1982) mostram que a inspeção visual automática de processos de fabricação de circuitos integrados já era amplamente estudada em meados da década de 70.

Em linhas de produção fabris muitas vezes se faz necessário, em algum ponto do processo, realizar uma verificação das peças, seja para identificar defeitos ou a peça em si. Tais tarefas dependem da conferência manual de operadores, tanto para detecção, quanto para a análise, estando sujeitas a variações com base na percepção individual do operador e dificultando a avaliação dos resultados. Para mitigar tais problemas, a implementação de algoritmos de inteligência artificial(IA) e visão computacional em processos de linhas de fabricação é uma solução que vem sendo utilizada. (KUMAR, 2008) mostra que a automação da detecção de defeitos em processos de fabricação de tecidos utilizando visão computacional visa aumentar a qualidade do produto final devido a robustez do processamento de imagens para classificar defeitos. Já em (SCIME; BEUTH, 2018), apresenta-se uma solução para o monitoramento de processos de fusão seletiva a laser utilizando técnicas de aprendizado de máquina para detecção de defeitos.

Inteligência artificial é o nome dado a algoritmos capazes de mimetizarem o comportamento humano, realizando de tarefas simples a complexas. Dentro do contexto de IA estão contidos os algoritmos de aprendizado de máquina (*machine learning*), que são, em sua essência, análises estatísticas aplicadas a uma grande quantidade de dados e que buscam encontrar correlações entre esses para realizar a inferência de resultados a partir de novas entradas.

Na últimas décadas houve um aumento no desenvolvimento de algoritmos aplicados a IA, capazes de aprender a realizar tarefas extremamente complexas em tempo reduzido com alto nível de confiabilidade. Dentre esses algoritmos, as redes neurais profundas destacam-se por sua flexibilidade para adaptar soluções em diversos domínios. A área da visão computacional foi uma que se beneficiou muito com esse crescimento, principalmente devido ao advento das redes neurais convolucionais (CNNs do inglês *Convolutional Neural Networks*). As CNNs têm como principal característica a capacidade de, através de operações matemáticas, extrair características (*features*) de imagens em diversos níveis de abstração, permitindo que essas sejam analisadas por máquinas.Entretanto, os métodos de aprendizado de máquina, sobretudo no contexto de redes convolucionais, possuem problemas relacionados a necessidade de uma alta quantidade de dados (imagens) necessários para realização de treinamento de tais algoritmos.

Nesse contexto, esse trabalho apresenta a proposta de um método que faz uso de ferramentas computacionais de modelagem 3D para facilitar a geração de *datasets* de imagens sintéticas para a realização de classificação de peças mecânicas. Estudos relacionados a utilização de modelos 3D já foram realizados em (WONG et al., 2019), (PENG et al., 2015) e (DENNINGER et al., 2019), porém com aplicações distintas e em domínios diferentes. Busca-se nesse trabalho apresentar o processo completo para realização de classificação de peças mecânicas a partir de imagens sintéticas: desde a preparação de dados (imagens) à utilização de algoritmo de aprendizado de máquina (CNN).

A revisão do estado da arte é apresentada no Capítulo 2 desse documento. As ferramentas de *software* utilizadas são apresentadas sucintamente no Capítulo e a metodologia proposta para o projeto é apresentada no Capítulo 4, onde se detalham as ferramentas utilizadas e suas funcionalidades. Também é apresentada a estrutura do sistema para a classificação de peças, desde a preparação de arquivos, geração de imagens a partir dos modelos 3D, desenvolvimento do algoritmo de rede de classificação e técnicas de *fine-tuning*.

A validação da solução proposta é apresentada no Capítulo 5, com base na análise dos resultados obtidos a partir da metodologia descrita previamente. O Capítulo 6 apresenta considerações finais e sugestões de possíveis trabalhos futuros.

2 REVISÃO DA LITERATURA

O capítulo tem como objetivo apresentar as principais tecnologias utilizadas ao longo do projeto, bem como as utilizações dessas no domínio proposto.

2.1 Aprendizado de máquina

Aprendizado de máquina, comumente chamado de IA, é na realidade uma subdivisão desse conceito e trata-se da utilização de algoritmos computacionais capazes de compreender informações de um conjunto de dados, extraindo conceitos, encontrando correlações e os interpretando de forma similar a que humanos realizam (RUSSELL; RUSSELL; NORVIG, 2019). Esses algoritmos permitem que computadores resolvam problemas que exigem um conhecimento do mundo real e que, muitas vezes, necessitam da análise subjetiva do ser humano.

Essencialmente, são algoritmos baseados em operações estatísticas que, a partir de um conjunto de observações dos dados, aprendem a correlação entre variáveis e permitem a predição de valores para novos conjuntos. Esse aprendizado é muito dependente da maneira como os dados são representados para a máquina (GOODFELLOW et al., 2016). Os dados devem ser estruturados e os elementos relevantes (*features*) apresentados para o algoritmo de forma manual, ou seja, existe uma engenharia de *features* prévia ao aprendizado. Tal abordagem é, então, utilizada quando se tem um conhecimento prévio de como a tarefa em questão funciona, como os algoritmos para jogos de tabuleiro que, apesar de complexos, possuem uma gama de ações possíveis muito bem definida.

A dependência da forma com a qual as informações são representadas exige que os algoritmos não somente aprendam a mapear as correlações entre os dados, mas a representação em si, a forma como o dado é informado para a máquina. Essa abordagem é conhecida como aprendizado de representações (*representation learning*, em inglês), e permite que o algoritmo de IA se adapte a novas tarefas com um mínimo de interferência humana.

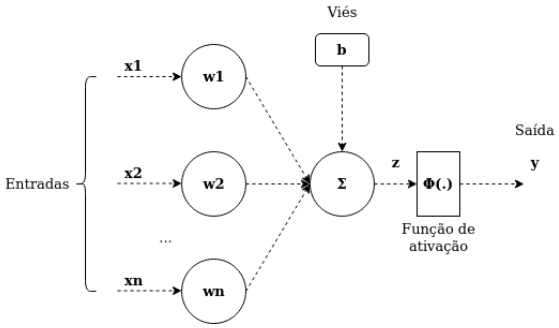
Algoritmos de aprendizado de representações servem para abstrair as informações (*features*) presentes nos dados e encontrar fatores de correlação entre cada fragmento de informação, como na análise de imagens, onde valores de pixeis podem representar diferentes objetos ou que o mesmo objeto possa ser representado por valores de pixeis diferentes conforme a cena em análise. Essas *features* são, por vezes, muito difíceis de extrair em um nível tão alto de abstração e, dessa forma, os algoritmos de aprendizado profundo (*deep learning*, em inglês) são introduzidos para a solução de problemas complexos.

2.1.1 Aprendizado profundo

O *deep learning* (DL) é a aplicação de algoritmos de *machine learning* que consistem em múltiplas camadas de representações em sequência, conectando umas às outras de forma que a máquina consiga juntar várias partes simples de informação e transformá-las em um conceito complexo dentro do contexto do problema em questão. Matematicamente falando, DL é aplicação em cascata de transformações não lineares entre os dados de entrada e um valor de saída.

A essência dos algoritmos de aprendizado profundo são os Perceptron multicamada (MLP do inglês Multilayer Perceptron). Perceptron é a formulação matemática análoga ao funcionamento aproximado do neurônio humano. Seu funcionamento se dá de maneira similar ao real, onde sua ativação é função dos sinais de entrada, transmitindo um sinal de adiante conforme a excitação recebida. Na Figura 1 fica evidenciado esse funcionamento.

Figura 1: *Modelo de um Perceptron*



Fonte: Elaborada pelo autor.

De maneira mais formal, o funcionamento do neurônio artificial é dado por:

$$\begin{cases} y = \Phi(z) \\ z = \sum_{i=1}^{N} w_i x_i + b \end{cases}$$

Os valores de w_i correspondem aos pesos associados a cada valor de entrada x_i e servem para balancear a influência de cada informação recebida pelo neurônio. Um viés é associado a cada neurônio, representando um limite(*threshold*) para que a saída z possa ser ativada.

A função de ativação $\phi(z)$ tem um comportamento não linear, limitando o valor a um intervalo e produzindo o valor de saída do neurônio. Existem diversas funções de ativação utilizadas no contexto de DL (JIANG et al., 2019), atualmente, porém, o retificador é a mais utilizada:

$$\Phi(z) = max(0, z)$$

O MLP é uma estrutura de rede neural composta pelo encadeamento de vários Perceptrons, formando uma rede profunda e densamente conectada (GOODFELLOW et al., 2016). Essa rede é capaz de representar funções complexas através da composição por funções mais simples. Assim, é possível realizar um mapeamento robusto entre os dados de entrada e de saída de algoritmos de aprendizado profundo. Cada camada do MLP encontra representações diferentes entre os valores de entrada e saída e, quanto mais profunda a rede (mais camadas de Perceptrons) maior é a abstração das informações presentes nos dados.

O aprendizado a partir de técnicas de DL pode ser categorizado de várias maneiras, sendo as principais: supervisionado, onde os dados fornecidos possuem rótulos (*labels*) e não supervisionado, onde os dados não possuem rótulos. Essas são as informações que o algoritmo utiliza para verificar o seu aprendizado, ou seja, após cada etapa de aprendizagem, ele valida o resultado obtido com base no resultado esperado. Esse aprendizado é dado através de um algoritmo chamado de *backpropagation*.

O algoritmo de *backpropagation* é baseado no cálculo do erro e_{rr} medido entre o valor de saída estimado da rede neural $\hat{y}$ e o valor real rotulado y. Desse resultado, é feita uma retro-propagação ao longo de toda a rede e são recalculados os pesos e vieses de cada elemento da rede.

Sucintamente, os pesos da rede são inicializados com valores pseudo-aleatórios, ou seja, números aparentemente aleatórios que são gerados a partir de algoritmos específicos que, em caso necessidade, podem repetir a sequência gerada múltiplas vezes (importante para a análise de resultados, de forma que a inicialização dos pesos não altere os resultados). A rede recebe então um conjunto de dados de entrada, realiza as operações ao longo das camadas e fornece um valor de predição para a sua saída. Define-se então uma função de custo $J(e_{rr})$, proporcional ao erro entre a predição e o valor esperado, a ser minimizada. Calculam-se os gradientes dessa função em relação a cada peso $\frac{\partial J}{\partial w_i}$ e, iterativamente, os pesos são ajustados. Uma vez a função custo minimizada, a rede é capaz de estimar um valor $\hat{y}$ preciso.

2.1.2 Aprendizado de máquina em visão computacional

O aprendizado de máquina voltado para problemas de visão computacional busca, através do processamento de imagens, a extração de informações relevantes para a realização de tarefas, como detecção e classificação de objetos.

Imagens, ao contrário de dados estruturados (tabelas), possuem um volume alto de parâmetros a serem passados para redes neurais profundas, visto que são matrizes tridimensionais de pontos (pixeis), onde seus valores representam a intensidade de níveis de cor em cada canal. A representação mais comum é a divisão em largura, altura e a profundidade, sendo essa composta de três canais de cores: vermelho, verde e azul (RGB, em inglês). Além disso, em aplicações reais as dimensões principais das imagens podem variar, conforme a fonte da qual são extraídas, dificultando assim o projeto de uma arquitetura de rede.

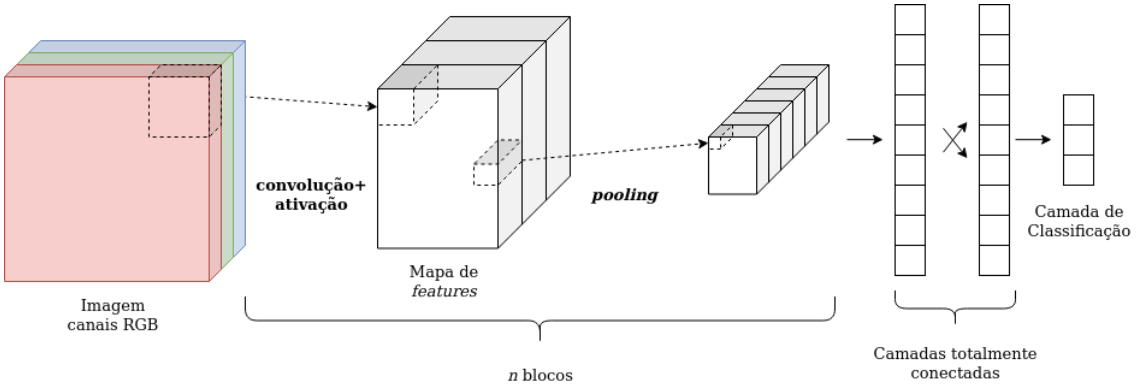
As redes neurais convolucionais são a adaptação das redes neurais clássicas para a análise de imagens. Elas baseiam-se na aplicação de filtros de convolução ao longo de toda a imagem, e buscam explorar estruturas similares em diferentes localidades dessa. A grande vantagem dessa abordagem é que o mesmo filtro é utilizado em todos os pixeis da imagem, ou seja, não são utilizados pesos individuais por valor de pixel (analogamente ao Perceptron), e assim a quantidade de parâmetros é reduzida significativamente nessas redes.

Como é visto na Figura 2, as camadas de convolução são os blocos essenciais da rede, seguidas por uma camada de ativação, camada de *pooling* e, por fim, camadas totalmente conectadas (FC, em inglês).

Em cada camada convolucional, as imagens são aplicadas a filtros (*kernels*) de convolução, Figura 3. O produto dessa operação é aplicado a uma função de ativação, geralmente o retificador, semelhante ao funcionamento do Perceptron, e os valores obtidos dessa função geram um mapa de *features*, exemplificado na Figura 4.

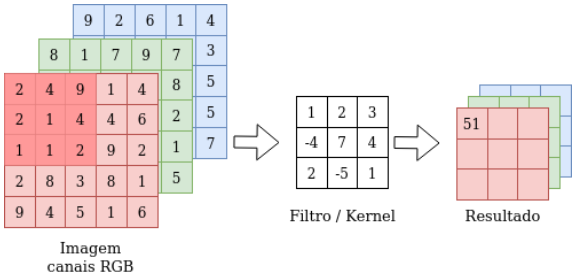
Percebe-se no mapa de *features* que cada filtro (coluna), dentro de cada camada (linha),

Figura 2: *Arquitetura genérica de uma CNN*



Fonte: Elaborada pelo autor.

Figura 3: *Exemplo de aplicação de um filtro de convolução.*



Fonte: Elaborada pelo autor.

é responsável por abstrair um tipo de informação da imagem. Os filtros de camadas iniciais da rede são responsáveis por abstrair informações em um mais baixo nível, como a detecção de bordas e contornos, enquanto os de camadas finais, apresentam uma informação mais complexa e útil de se transmitir às camadas FC para que realizem a classificação.

Após a aplicação da função de ativação no resultado da convolução, segue-se uma camada de *pooling* que serve para realizar o *down-sampling* (redução de dimensão) dos resultados da operação convolutiva. É comum considerar as operações de convolução, ativação e *pooling* como um bloco de convolução. Esse processo é repetido por um número n de vezes, representando a profundidade da rede convolucional.

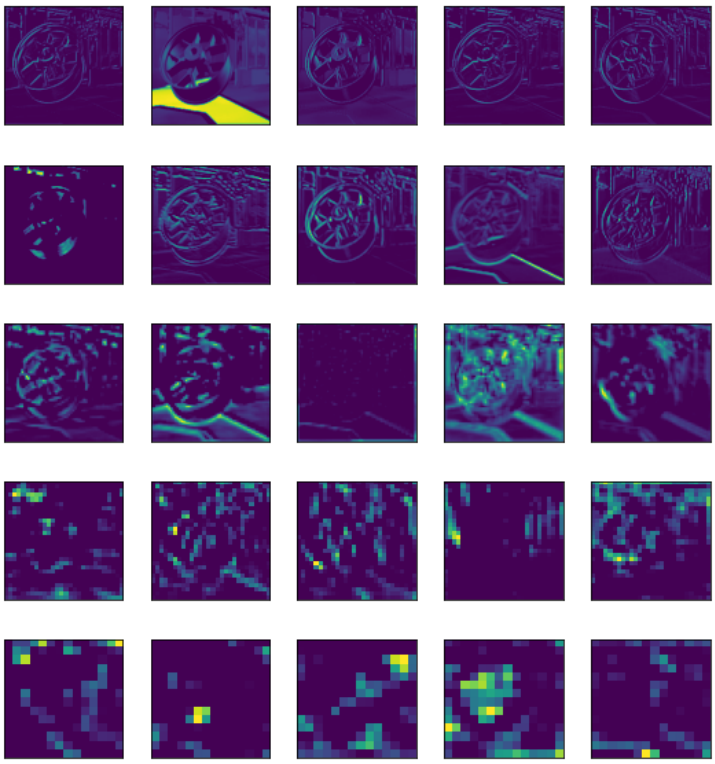
Assim, após a imagem passar por todos os blocos de convolução, as informações abstraídas ao longo das camadas são transmitidas às camadas totalmente conectadas de Perceptrons para o mapeamento das representações dos dados e, assim, correlacionar a imagem fornecida à rede a um valor de saída, conforme definido para sua arquitetura, no exemplo, a classificação de imagens.

2.2 Modelagem 3D

A modelagem tridimensional consiste em criar uma representação matemática de um objeto. Essa representação é chamada de modelo 3D. A aplicação da modelagem tridimensional é diversificada, possui inúmeras funções e está presente em diversos domínios como na engenharia, design gráfico, design de jogos, medicina, dentre outros.

No contexto de engenharia, a modelagem tridimensional é comumente utilizada na criação de modelos de desenho assistido por computador (CAD, em inglês). Tais modelos

Figura 4: *Mapa de features.*



Fonte: Elaborada pelo autor.

Nota: De cima para baixo, visualizam-se os mapas de blocos superficiais de convolução aos mais aprofundados.

são, geralmente, utilizados para a representação de peças mecânicas e componentes de sistemas complexos. O CAD permite a criação de modelos tridimensionais a partir de desenhos técnicos, esses podem ser utilizados tanto para a fabricação de peças, quanto para a simulação de seu funcionamento em um ambiente virtual simulado.

A essência da modelagem 3D é a representação matemática dos modelos através de APIs e *softwares*, cujas funcionalidades auxiliam na criação de modelos 3D. São inúmeras as possibilidades de representação digitais desses, sendo a representação de fronteiras (B-rep, em inglês) uma dentre as mais utilizadas. Esse tipo de modelagem utiliza de superfícies definidas matematicamente, como cilindros, cubos, esferas, curvas e NURBS (Non Uniform Rational Basis Spline). Modelos B-rep são a solução mais adotada no domínio da engenharia, e em muitos aplicativos de modelagem 3D para projeto, simulação e fabricação de produtos. Outra modelagem, também amplamente utilizada em design de jogos e animações, devido a sua representação mais leve, é de modelos de faceta, que aproximam as superfícies usando polígonos planares (geralmente triângulos) conectados.

Cada tipo de representação para um modelo 3D é, geralmente, associada a um tipo de formato de arquivo a ser utilizada dentro dos *softwares* de modelagem. Cada formato, mesmo que representando um mesmo modelo, armazena diferentes tipos de informações, como massa, densidade e material ou simplesmente a superfície volumétrica (modelos de faceta). Um tipo de arquivo para representação de modelos facetados é o formato STL (*stereolithography*). Essa extensão de arquivo armazena a informação volumétrica de um objeto através de uma malha de vértices e arestas que compõe a superfície externa.

Esses modelos podem ser fabricados a partir de processos de manufatura tradicional ou por meio de impressão 3D. Podem também ser convertidos em imagem através de renderização 3D, para criar representações fotorealistas dos objetos modelados.

3 MATERIAIS

3.1 Tensorflow

Tensorflow é uma plataforma com código aberto, desenvolvida pela equipe de pesquisa da Google, para o desenvolvimento de aprendizado de máquina. Possui uma vasta gama de ferramentas e bibliotecas que permite que desenvolvedores, pesquisadores e até mesmo entusiastas, implementem códigos e desenvolvam aplicativos utilizando tecnologias de *machine learning*.

A plataforma possui uma API para programação em linguagem Python, C++, Java, dentre outras. Dentro dela é possível criar ou importar modelos existentes, permitindo adaptá-los a novas tarefas e domínios. O treinamento e validação de tais modelos também pode ser realizado dentro do Tensorflow.

Neste trabalho, então, é utilizada essa ferramenta para a implementação do modelo de rede convolucional para a classificação de peças. A linguagem de programação utilizada para o desenvolvimento do modelo é Python.

3.2 Blender

O Blender é um *software* de criação e edição 3D de código aberto. Ele possui diversas ferramentas que permitem desde a modelagem, simulação, renderização até a edição de vídeos e a criação de jogos. Dentro dele existe uma API integrada para o desenvolvimento de aplicações em Python e que permitem personalizar o seu funcionamento de acordo com a tarefa desejada.

Dentro do contexto desse trabalho, utiliza-se, sobretudo, das ferramentas de modelagem e renderização e o desenvolvimento de *scripts* em Python para a automatização de tarefas.

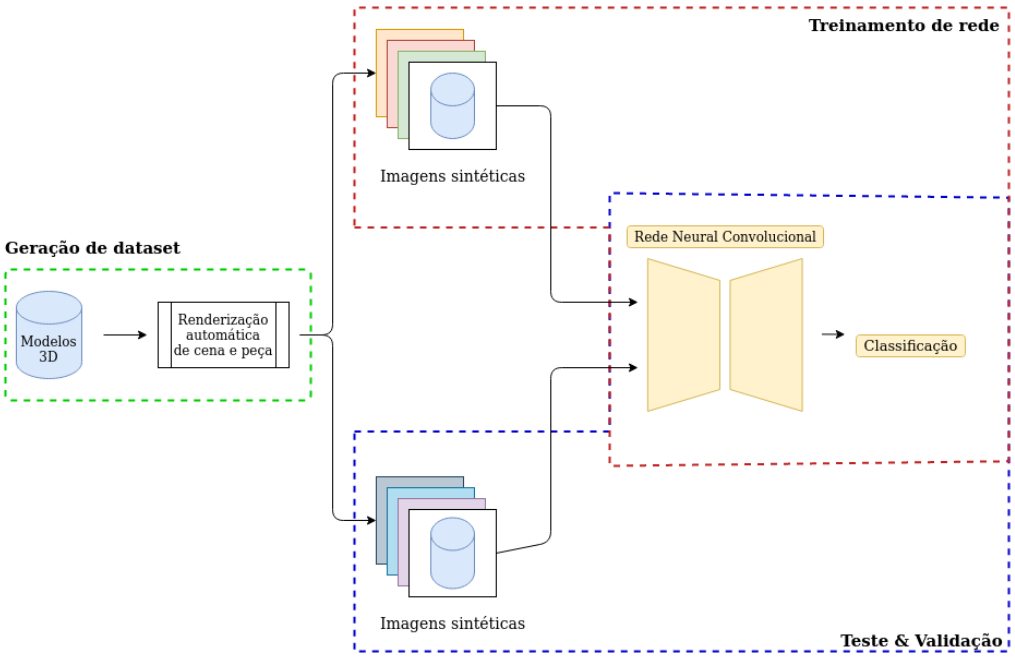
4 METODOLOGIA

Neste capítulo é apresentada a arquitetura da solução proposta, permitindo uma visão geral do projeto, e em seguida são apresentados individualmente cada subsistema, começando pela modelagem 3D de peças mecânicas, a geração das imagens sintéticas e, por fim, a determinação de modelos de redes neurais para a classificação de peças e lançamento de treinamento.

4.1 Arquitetura do sistema

O sistema tem como objetivo geral treinar um classificador de peças mecânicas a partir de imagens geradas computacionalmente. Assim, pode-se dividir a arquitetura em três subsistemas representando as etapas de geração de imagens sintéticas, o treinamento do modelo de rede classificadora e a validação do modelo treinado, como é ilustrado na Figura 5.

Figura 5: *Arquitetura geral do sistema*



Fonte: Elaborada pelo autor.

Essa subdivisão cobre desde o preparo dos dados (definição de modelos CAD 3D de peças mecânicas e a modelagem dessas em ambiente gráfico), passando pela definição do modelo da rede e o treinamento, até a validação, analisando parâmetros de precisão.

No subsistema de geração de *dataset* encontra-se o foco principal do projeto: a geração automática de um alto volume de imagens sintéticas que permitam com que a rede neural consiga abstrair informações relevantes, tais quais as presentes em imagens reais, para a classificação de peças. Para essa tarefa, faz-se necessária a utilização de modelos CAD de peças mecânicas para a renderização das imagens, contudo, com o intuito manter a concisão do trabalho, serão utilizados CADs já prontos, obtidos através de repositórios gratuitos, focando apenas na modelagem do ambiente virtual.

A rede neural de classificação recebe os dados sintéticos com suas respectivas anotações e realiza o treinamento para ajuste dos seus pesos.

4.2 Modelagem tridimensional

Conforme explicado no Capítulo 2, os *softwares* de modelagem 3D voltados para a engenharia permitem que sejam criadas representações de objetos, peças e sistemas completos de maneira precisa e detalhada. Normalmente, os modelos gerados são utilizados para processos de fabricação como usinagem, estampagem e impressão 3D. Já *softwares* voltados para a modelagem gráfica, permitem a criação não só de objetos, mas como de ambientes inteiros, bem como a adição de um alto nível de detalhe a cada objeto presente como pode ser visto na Figura 6. Para a modelagem do ambiente e pré tratamento das peças foi utilizado o *software open source* Blender.

Figura 6: *Ambiente modelado no Blender*

Fonte: Elaborada pelo autor.

Um nível alto de detalhamento das peças e do ambiente é necessário para que as imagens do *dataset* sejam o mais diversificadas possível, ou seja, apresentem diferentes componentes e informações que permitam com que os algoritmos de aprendizado consigam distinguir entre o objeto desejado a classificar e os outros elementos presentes nas imagens (JIANG et al., 2019). Assim, supõe-se que a rede de classificação consiga abstrair o maior número de características de cada peça, em contraste às características do ambiente de fundo. Tal abordagem tem o intuito de evitar que ocasione a situação de *overfitting* do treinamento do algoritmo, fenômeno que ocorre quando a rede aprende muito bem a classificar as imagens que utiliza para o treinamento e não consegue generalizar seu aprendizado para imagens fora desse conjunto. Em outras palavras, a rede memoriza cada imagem que lhe foi apresentada durante o treinamento.

4.2.1 Automatização da geração de imagens

O Blender possui uma interface de programação de aplicativos (API, em inglês) para linguagem python. Essa API permite com que todas as funções realizadas via interface gráfica possam ser realizadas via linha de comando, possibilitando assim a programação de rotinas dentro do *software*.

Para a modelagem tridimensional do ambiente, alguns parâmetros foram programados para se comportarem de forma dinâmica entre cada renderização de imagem para o dataset: iluminação, posicionamento e orientação da peça em relação a câmera e os elementos no contexto da cena (*backgrounds*). É realizada, então, uma pré-configuração do ambiente do Blender para que a automatização funcione posteriormente. Para isso, cria-se um arquivo modelo *model.blend* que serve de base para renderização das peças no ambiente. Nesse arquivo são pré-definidos objetos de câmera, pontos de luz e material.

A câmera é o elemento que permite a renderização de imagens dentro do *software*, definindo qual a área da cena a ser renderizada. Ela não é, contudo, um objeto visível dentro do ambiente de modelagem, porém possui parâmetros que podem ser regulados para o ajuste da imagem, como a perspectiva, o foco, a dimensão da imagem, a adição de *backgrounds* e suas coordenadas de posição e orientação.

A iluminação dentro do ambiente modelado pode ser ou pontual (*point*), quando a luz provém de um ponto omnidirecional irradiando de maneira uniforme os feixes de luz, ou um foco (*spot*), quando a luz é direcionada em uma projeção cônica em uma certa direção. O objeto `light` possui parâmetros reguláveis como, cor, intensidade, medida em Watts, e raio, distância até a qual luzes pontuais podem alcançar dentro da cena. Durante a renderização, o Blender calcula a incidência de raios de luz em cada objeto da cena, permitindo um efeito de reflexo fotorealista.

O material é um elemento que pode ser associado a qualquer elemento dentro de uma cena no Blender. Ele possui diversas parametrizações, entretanto, no contexto desse trabalho, foca-se na utilização de um material metálico reflexivo, tal qual peças mecânicas reais. Dentro do elemento material define-se um *shader*, que é a formulação de como é refletida a cor do material quando a luz incide nesse.

4.2.1.1 Texturização

Para a aplicação de textura na peça importada, é necessário associar um material a ela. Define-se um material metálico, com propriedades reflexivas, e então, é possível atribuir uma imagem como textura para o material e, em seguida, aplicá-lo na peça.

Uma vez renderizada, a peça fica com o aspecto mostrado na Figura 7.

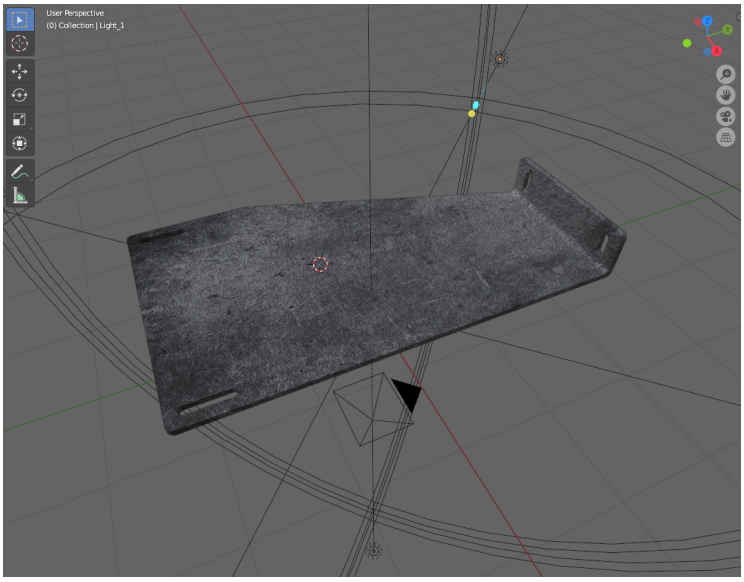
4.2.1.2 Iluminação

Para haver uma incidência de iluminação dinâmica na peça, é realizado um equacionamento para que as fontes de iluminação tenham suas posições e orientações alteradas em relação ao referencial do objeto em foco. Além desses, o outros fatores equacionados são a intensidade e a cor da iluminação, variando entre diferentes tonalidades.

As posições das fontes de iluminação são dadas por circunferências centradas na peça, como é apresentado na Figura 8. Os valores de raios e ângulos são gerados aleatoriamente dentro de uma faixa de valores especificada. A intensidade de cada *spot* também é variável dentro de uma faixa de potência, já a cor varia em um espectro do branco ao amarelo, representando iluminações naturais típicas.

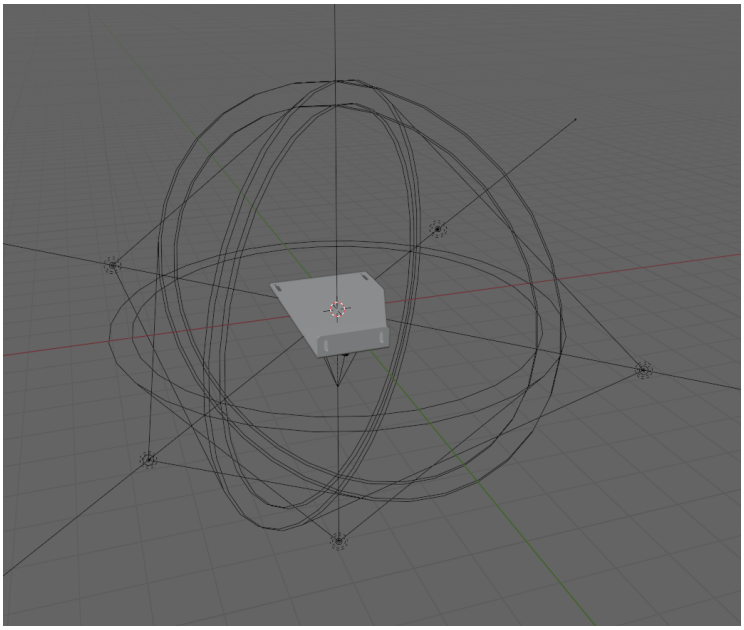
Uma vez modelada, realiza-se uma renderização prévia da iluminação para verificar

Figura 7: *Aplicação de textura metálica em peças.*



Fonte: Elaborada pelo autor.

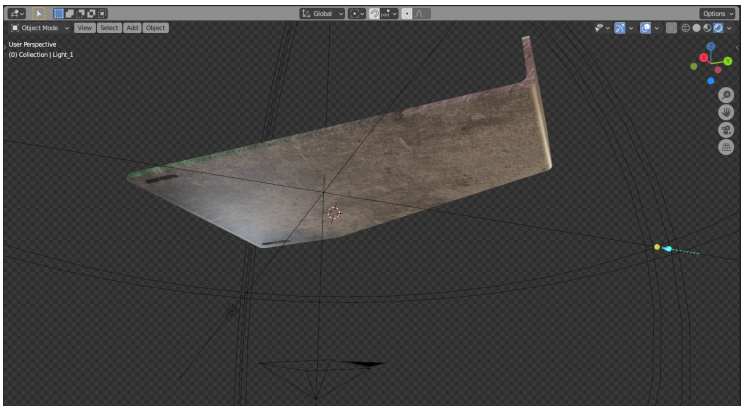
Figura 8: *Modelagem de iluminação dinâmica no Blender*



Fonte: Elaborada pelo autor.

sua incidência na peça texturizada, como mostra a Figura 9.

Figura 9: *Incidência da iluminação na peça texturizada.*



Fonte: Elaborada pelo autor.

4.2.1.3 Posicionamento e Orientação

O posicionamento e orientação da câmera em relação a peça é realizado com base na origem da peça, seu centro geométrico. Assim, a câmera mantém o foco sempre direcionado à peça definida pela função `bpy.ops.object.constraint_add(type = 'TRACK_TO)`.

Para que o conjunto de imagens da peças seja o mais robusto possível, é necessário que ele contenha imagens de diversas vistas da peça. Assim, o equacionamento para a rotação da peça é dado por:

```
face_steps = [(360/Fsteps)*i for i in range (Fsteps)]
vertical_steps = [-op_range/2] + [op_range/Vsteps]*Vsteps
horizontal_steps = [-op_range/2] + [op_range/Hsteps]*Hsteps
```

Onde `face_steps` representa a rotação em torno do eixo x, `vertical_steps` em torno do eixo z e `horizontal_steps` em torno do eixo y. Os parâmetros `Vsteps, Hsteps e Fsteps` servem para definir a quantidade de passos em que se dá a variação dos ângulos em graus dentro dos limites de `[-op_range, op_range]` para y e z, e `360 graus` para x.

4.2.1.4 Background

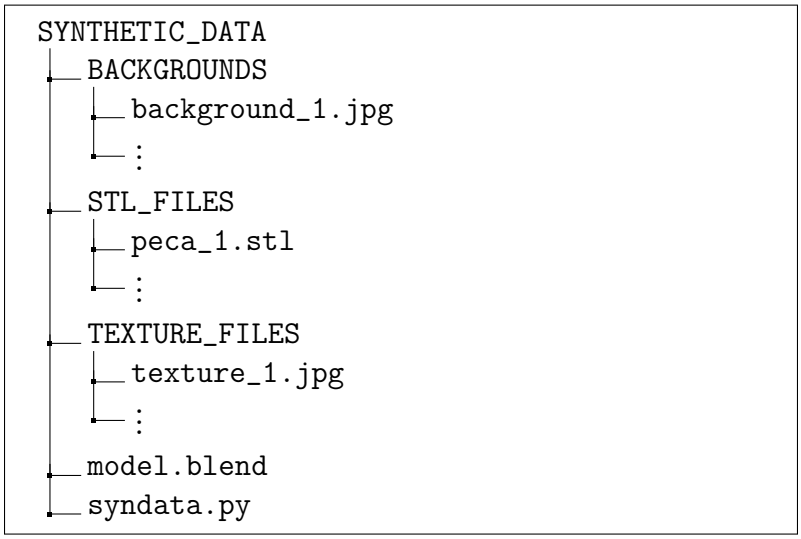
Foram modelados alguns ambientes como o visto na Figura 6, porém a abordagem foi modificada, visto que a modelagem de cada ambiente 3D, com cenários e objetos diferentes, aumenta muito a complexidade da tarefa e foge do objetivo principal da solução, a praticidade. Assim, os *backgrounds* utilizados para a renderização das peças são imagens bidimensionais, semelhante à abordagem em (PENG et al., 2015).

Para gerar diversidade, porém mantendo a contextualização, são utilizadas fotos de ambientes industriais, que apresentam elementos semanticamente semelhantes aos que as peças geralmente se encontram, e alguns fundos neutros, em que a peça fica em evidência na imagem.

4.2.1.5 Importação das peças e renderização

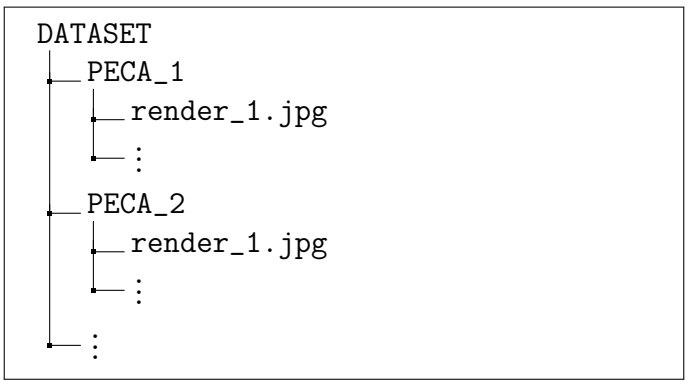
Para a renderização do conjunto de imagens é necessário que os arquivos estejam organizados em uma estrutura de diretórios como na Figura 10.

Figura 10: *Estrutura dos diretórios para renderização*



Com todas as funções já definidas, e os arquivos organizados, a renderização é feita através do *scrpit* `syndata.py` e o *dataset* é gerado com a seguinte estrutura:

Figura 11: *Estrutura de diretório do dataset gerado*

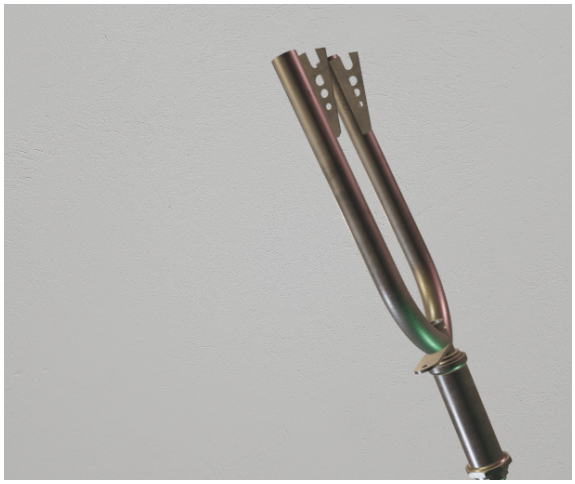


Um exemplo de resultado da renderização final é mostrado na Figura 12.

4.3 Determinação das peças mecânicas para classificação

Os modelos de peças escolhidas para a criação das imagens sintéticas e treinamento do classificador foram obtidos através de um banco de modelos 3D gratuito, desenvolvido e fornecido por (KOCH et al., 2019), com o objetivo de auxiliar no desenvolvimento de aplicações e pesquisas em *machine learning*. Os modelos utilizados são de peças que, em sua essência, são de material metálico, como parafusos, engrenagens e tubos. A utilização desses modelos 3D tem como objetivo manter a concisão do trabalho, visto que o foco está na geração das imagens sintéticas e a análise de um classificador quando alimentado por essas.

Figura 12: *Exemplo de imagem renderizada*



Fonte: Elaborada pelo autor.

4.4 Determinação de um modelo de rede para classificação

Redes neurais convolucionais, como introduzido no Capítulo 2, são redes profundas capazes de abstrair informações de imagens ao nível de pixeis. Pode-se, então, definir um modelo de rede concatenando diversos blocos de convolução, respeitando as dimensões dos dados de entrada e saída de cada um, e camadas totalmente conectadas para converter as informações obtidas a partir das convoluções para resultados de classificação. Todavia, esse processo não é otimizado e exige uma grade quantidade de testes modificando a arquitetura, adicionando ou removendo *layers*, até chegar em resultados satisfatórios.

De fato, CNNs têm sido objeto de estudo por muitos pesquisadores na área de *deep learning*. O grupo de visão computacional da universidade de Stanford, em parceria com a universidade de Princeton, mantém um banco de dados com milhões de imagens pertencentes a milhares de categorias. Esse *database* é chamado de ImageNet e, a partir do ano de 2010, a *ImageNet Large Scale Visual Recognition Competition*(ILSVRC) aconteceu anualmente até 2017. A ILSVRC (RUSSAKOVSKY et al., 2015) é uma competição que busca incentivar a descoberta de novas arquiteturas de redes para a detecção e classificação de objetos. Inúmeras arquiteturas são propostas e têm seus resultados avaliados sobre o banco de dados da ImageNet.

Um marco do desenvolvimento de CNNs se deu durante a ILSVRC de 2012, quando a arquitetura AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) conseguiu um resultado *top-5 error* (probabilidade de que a classe real não esteja entre os 5 maiores resultados de probabilidade) de 15.3% (ALOM et al., 2018). A partir desse marco, várias outras arquiteturas foram desenvolvidas baseadas nessa ao ponto que, em 2017 a maioria das soluções submetidas alcançaram uma precisão acima de 94% e, assim, a competição foi encerrada. Os estudos na área de CNNs, porém, seguem sendo realizados, sobretudo na comparação entre precisão e complexidade computacional (BIANCO et al., 2018), hoje não só a classificação de imagens, mas a detecção de objetos em aplicações em tempo real é objeto de estudo pela comunidade científica.

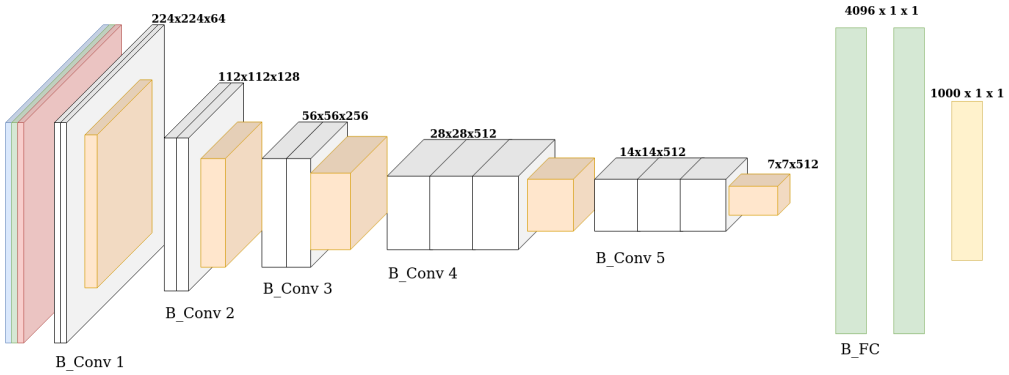
4.4.1 VGG16

Para a validação da utilização do *dataset* sintético para a classificação de peças mecânicas foi definida a arquitetura da rede VGG16 como modelo de base para a rede de

classificação. A rede VGG16 foi apresentada em 2014 pelo *Vision Geometry Group* de Oxford (SIMONYAN; ZISSERMAN, 2014), e alcançou o resultado de *top-5 error* de 6.8 % no desafio da ImageNet, vencendo a competição no quesito de classificação de imagens dentro de mil classes do *dataset* e ficando em segundo na tarefa de detecção de objetos (localização) em imagens.

Essa rede possui 5 blocos de convolução, seguidos por 2 camadas totalmente conectadas e 1 camada de predição, Figura 13. Cada bloco, como explicado no Capítulo 2, é composto por camadas de convolução, seguidas por uma função de ativação e uma camada de *pooling*. Os filtros de convolução utilizados na arquitetura são de dimensão 3x3, enquanto os filtros utilizados na camada de *pooling* são de dimensão 2x2. A função de ativação utilizada após cada camada de convolução é o retificador, também definido na Capítulo 2.

Figura 13: *Estruturas dos layers da rede VGG16.*



Fonte: Elaborada pelo autor.

A implementação desse modelo de rede é realizado utilizando a biblioteca do Tensorflow em Python.

4.4.2 Implementação do modelo

A biblioteca do Tensorflow é integrada com uma API de alto nível: Keras, onde existem diversas ferramentas úteis para o desenvolvimento de algoritmos de aprendizado de máquina. Dentro dela, é possível encontrar diversos modelos de arquiteturas de CNNs já estabelecidos pelo estado da arte. O modelo da rede VGG16 é um dos que se encontra disponível dentro da biblioteca. Ainda para aplicações com CNNs, a API possui funções de pré-tratamento de imagens para redimensioná-las, recortá-las, inverte-las e alterar a escala de cores.

Para a criação da rede de classificação, utiliza-se o módulo `keras.applications.vgg16()` para estabelecer o modelo. É possível então verificar o modelo e a totalidade dos parâmetros através do comando *model.summary()* Figura 14.

4.4.3 Tratamento das imagens

O *dataset* gerado automaticamente no Blender cria um diretório distribuído como mostrado na Figura 11. Para a realização do treinamento da rede, porém, é necessário um pré-processamento dos arquivos. As imagens são divididas em conjuntos de *treino, validação e teste*, particionados, respectivamente em 70%, 20% e 10% da totalidade do *dataset* gerado. As imagens do conjunto de *treino* são as utilizadas para o treinamento da rede e atualização dos pesos. O conjunto de *validação* é utilizado para a validação de

Figura 14: *Sumário da rede VGG16.*

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

Fonte: Elaborada pelo autor.

métricas de precisão, ou seja, ao longo do treinamento, a rede tenta validar o aprendizado até o dado instante, realizando a inferência de resultados em imagens nunca vistas. O *set* de validação não serve para a atualização dos pesos da rede. Por fim, o conjunto de *teste* é o utilizado para calcular a precisão da rede, uma vez treinada.

Para o carregamento das imagens é utilizada a função `flow_from_directory` da classe `ImageDataGenerator`. Essa função tem como papel principal realizar o carregamento das imagens em lotes (*batches*) de 10. Além disso, realiza um pré-processamento de imagens estipulado pela rede VGG16: reescala as imagens para um tamanho de 224 x 224, convertendo os valores de intensidade de cada pixel para valores entre 0 e 1, e subtrai a média dos valores dos canais RGB em cada imagem, o resultado é apresentado na Figura 15.

Figura 15: *Batch de imagens pré-processadas.*

Fonte: Elaborada pelo autor.

Uma vez que as imagens estão processadas e carregadas, realiza-se o treinamento do modelo.

4.5 Treinamento da rede

Para o treinamento da rede de classificação são utilizadas técnicas de *transfer learning*, extração de *features* e ajuste fino para a melhora de resultados.

A técnica de *transfer learning* consiste em transferir o aprendizado obtido a partir de um cenário, para outro semelhante (GOODFELLOW et al., 2016). No contexto de classificação de imagens a partir de redes convolucionais, a transferência se dá ao inicializar o treinamento de uma rede com os valores de seus pesos inicializados de uma maneira não aleatória, ou seja, utilizando-se de pesos de um modelo rede (igual) já pré-treinado em um outro conjunto de imagens.

Ao importar o modelo da rede VGG16 da biblioteca do Tensorflow, importamos também os valores dos pesos dessa rede treinada no *dataset* da ImageNet. Logo, ao inicializar o treinamento, o modelo criado já inicializa-se com esses valores de peso. Deseja-se, entretanto, adaptar o domínio ao qual estamos classificando imagens. Os pesos do modelo pré-treinado importados correspondem à classificação de imagens pertencentes a mil classes, diferentemente do proposto por esse trabalho. Esses pesos devem, então, ser alterados durante o treinamento do classificador utilizando-se de técnicas de extração de *features* e ajuste fino.

4.5.1 Extração de *features* e ajuste fino

A combinação das técnicas de *transfer learning* e extração de *features* permite a utilização de informações aprendidas pelo modelo em suas camadas de convolução e a sua adaptação para o contexto desejado. A transferência de aprendizado é possível devido às características das operações realizadas pelos filtros de convolução. Essencialmente, os filtros presentes em camadas iniciais de CNNs aprendem a sintetizar informações elementares presentes em imagens, como linhas e contornos abertos. A medida que as camadas se aprofundam, os filtros passam a contextualizar melhor as informações, e os mapa de *features*, como mostrado no Capítulo 2, apresentam informações relevantes para a máquina.

Para a extração de *features*, é necessário modificar o modelo da rede para que atenda as especificações do projeto. A rede VGG16 foi originalmente desenhada para a classificação de imagens abrangendo 1000 classes diferentes e, assim, possui um último *layer* de predição com 1000 saídas. Para a realização desse trabalho, são avaliados 2 cenários de classificação, o primeiro contendo 10 classes de peças para serem classificadas, e o segundo contendo 24 classes. O último *layer* é então substituído, em cada um dos casos, por outro com apenas o número de saídas equivalente ao de classes e é apresentado nas Figuras 16 e 17.

Figura 16: *Camada modificada da rede para 10 classes.*

```
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 10)                40970
=================================================================
Total params: 134,301,514
Trainable params: 126,666,250
Non-trainable params: 7,635,264
```

Fonte: Elaborada pelo autor.

Figura 17: *Camada modificada da rede para 24 classes.*

```
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 24)                98328
=================================================================
Total params: 134,358,872
Trainable params: 126,723,608
Non-trainable params: 7,635,264
```

Fonte: Elaborada pelo autor.

Além da modificação do *layer*, deseja-se apenas realizar a extração das *features* do modelo da rede pré-treinado, para isso congelam-se os blocos de convolução da rede, ou seja, não permite-se que esses pesos sejam atualizados durante o treinamento. Dessa maneira, apenas as camadas finais (`fc1, fc2 e predictions`) têm seus pesos modificados, de forma a mapear as informações provindas das camadas de convolução para a camada de classificação.

Uma vez finalizado o treinamento da rede com os blocos de convolução congelados, realiza-se o ajuste fino do modelo treinado, que consiste em descongelar os blocos de convolução mais próximos das camadas finais do modelo,*B_Conv 5* da Figura 13, e retreiná-lo. Dessa forma, os pesos desse bloco podem ser atualizados com base nas informações relevantes para o conjunto de peças a ser treinado. Esse tipo de prática é realizado após o treinamento de extração de *features* para evitar que os valores dos pesos mudem abruptamente e o aprendizado prévio seja perdido.

Os resultados dos treinamentos utilizando ambas as técnicas são apresentados no Capítulo 5.

5 RESULTADOS

5.1 Cenário 1

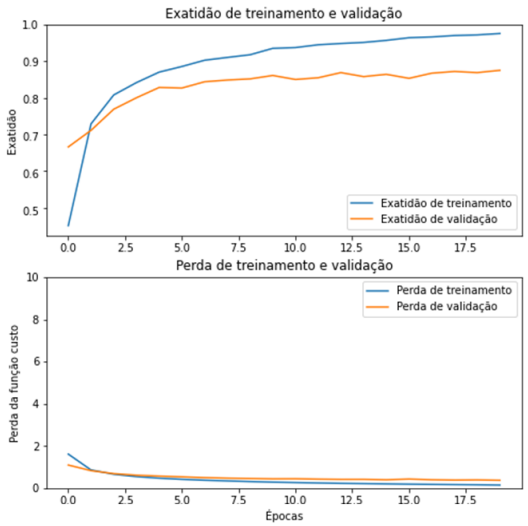
Para o primeiro cenário foram renderizadas um total de 3240 imagens, pertencentes a 10 classes distintas. Essas imagens foram divididas em:

- Treino: 2340 imagens, 234 por peça ;
- Validação: 640 imagens, 64 por peça;
- Teste: 260 imagens, 26 por peça.

A rede de classificação foi treinada em duas etapas, extração de *features* e ajuste fino, como explicado no Capítulo 4. Durante cada uma dessas etapas a rede foi treinada por 20 épocas e com uma taxa de aprendizado de 0.0001 e 0.00001, sendo épocas a quantidade de vezes que todas as imagens são passadas pela rede e a taxa de aprendizagem um fator de amortecimento aplicado na taxa em que o gradiente do erro é propagado na rede, como visto no Capítulo 2.

Após as 20 épocas iniciais do treinamento, o modelo de classificação atingiu um resultado no conjunto de imagens de validação de 87.34%, como é mostrado na Figura 18.

Figura 18: *Curvas de aprendizado do cenário 1 sem ajuste fino.*

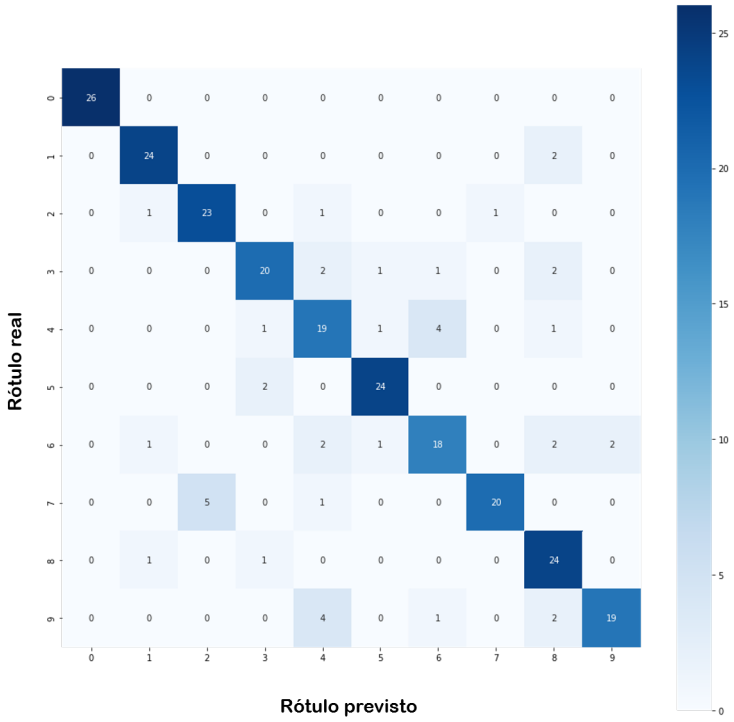


Fonte: Elaborada pelo autor.

Todavia, uma maneira mais precisa de avaliar um modelo de classificação é validando seu resultado em um conjunto de dados que ainda não tenha sido visto pelo modelo. Assim, utilizam-se as imagens separadas no conjunto de teste para verificar a precisão do modelo quanto a novos dados.

A Figura 19 mostra a matriz de confusão gerada pelo resultado das predições do modelo. Nela, em sua diagonal, percebem-se os resultados corretos de classificação (*verdadeiros positivos*), onde o rótulo previsto é igual ao real. Os valores pertencentes a uma coluna, com exceção do valor da diagonal, são chamados de *falsos positivos*, ou seja, a rede classifica como a peça *n* uma peça de outra categoria; enquanto os valores na mesma linha correspondem aos *falsos negativos*, quando na realidade trata-se da peça *n*, porém a rede a classifica como outra.

Figura 19: *Matriz de confusão do cenário 1 sem ajuste fino.*

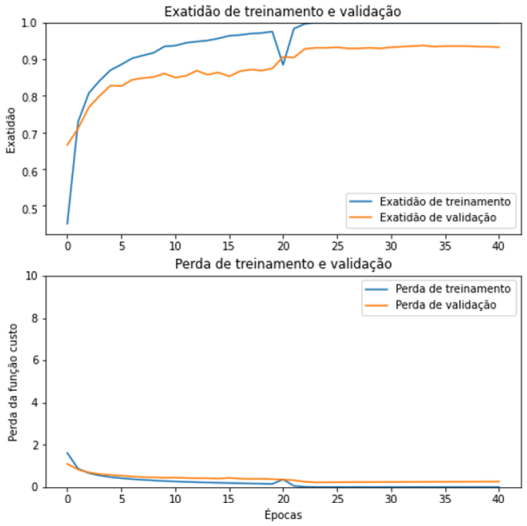


Fonte: Elaborada pelo autor.

Uma vez avaliado o modelo na primeira etapa do treinamento, realiza-se então o ajuste fino ao descongelar o último bloco de convolução da rede. Após o retreinamento do modelo, constatou-se uma melhora significativa no resultado do modelo, ficando em 93.44%, como mostrado na Figura 20. Nela, também é percebido um pico na curva de aprendizado, isso se dá pelo fato de o ajuste fino da rede continuar a partir da última etapa da extração de *features*, porém agora com os blocos convolucionais livres para terem seus pesos atualizados. Na primeira iteração do ajuste, então, o gradiente que se propaga é maior, devido a esses valores não terem ainda sido calculados para esse conjunto de imagens. Percebe-se, contudo, que a curva converge novamente em questão de 2 épocas. Esse comportamento é visto também no cenário 2.

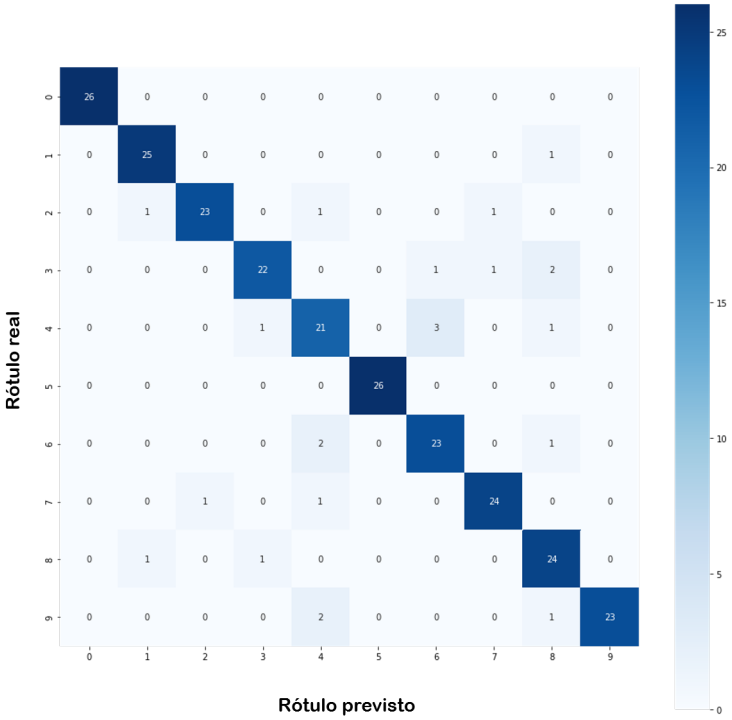
Analogamente à primeira etapa do treinamento, plota-se uma matriz de confusão, Figura 21, para verificar a precisão do modelo em relação a novas imagens. Percebe-se nela que existe uma diminuição da dispersão entre os rótulos previstos e seus respectivos valores reais.

Figura 20: *Curvas de aprendizado do cenário 1 com ajuste fino.*



Fonte: Elaborada pelo autor.

Figura 21: *Matriz de confusão do cenário 1 com ajuste fino.*



Fonte: Elaborada pelo autor.

Outra possível maneira de se analisar um modelo de classificação quanto a sua qualidade é calculando os parâmetros de *precision* e *recall*. Formalmente, *precision* representa a quantidade de vezes em que uma peça foi classificada corretamente (*verdadeiros positivos*) sobre o total de vezes em que essa peça foi classificada (*verdadeiros positivos* + *falsos positivos*), ou seja, independente da quantidade de imagens de uma dada peça *n* apresentada ao algoritmo, a precisão vai levar em consideração apenas os resultados que apontaram a imagem como sendo a peça em questão (estando certo ou errado); enquanto o *recall* é a razão entre a quantidade de vezes em que uma peça foi classificada (*verdadeiros positivos*) sobre o total de imagens dessa peça presentes no conjunto de teste, ou seja, considerando os *verdadeiros positivos* e os *falsos negativos*.

$$\begin{cases} Precision = \frac{V_{Pos}}{V_{Pos}+F_{Pos}} \\ Recall = \frac{V_{Pos}}{V_{Pos}+F_{Neg}} \end{cases}$$

A métrica *F1 score* é obtida calculando-se a média harmônica entre os valores de *precision* e *recall*. A Tabela 1 mostra os valores médios dessas três métricas para ambas partes do treinamento.

Tabela 1: *Parâmetros de precision e recall do cenário 1.*

Parâmetro	pré ajuste fino	pós ajuste fino
Precision	0.8422127680748369	0.915363247863248
Recall	0.8346153846153845	0.9115384615384616
F1 score	0.8347215487596926	0.9122768875329521

5.2 Cenário 2

Para o segundo cenário foram renderizadas um total de 9720 imagens, pertencentes a 24 classes distintas. Essas imagens foram divididas em:

- Treino: 7008 imagens, 292 por peça ;
- Validação: 1944 imagens, 81 por peça;
- Teste: 768 imagens, 32 por peça.

A rede de classificação do cenário 2 foi treinada também em duas etapas, analogamente ao que é realizado no cenário 1. Os parâmetros utilizados para o treinamento, tanto para quantidade de épocas, quanto para a taxa de aprendizagem, foram os mesmos utilizados anteriormente.

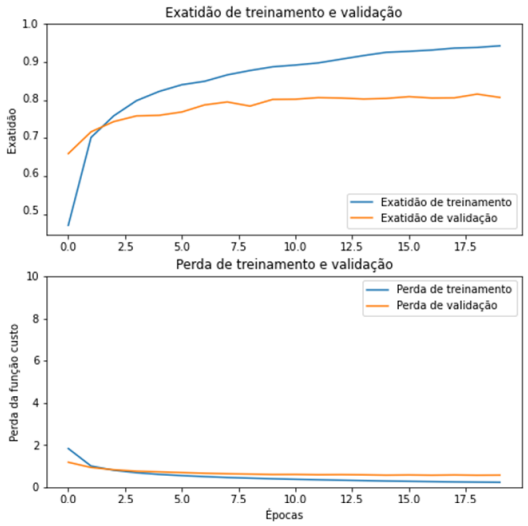
O resultado obtido pela rede no cenário 2 durante a primeira etapa do treinamento foi de 81.22%, como é visto na Figura 22.

A matriz de confusão aplicada às imagens do conjunto de teste é mostrada na Figura 23.

Seguindo o mesmo procedimento do primeiro cenário, foi realizada a segunda etapa do treinamento, referente ao ajuste fino, e o resultado obtido também mostrou uma melhora significativa, atingindo um resultado de 89.12%, visto na Figura 24.

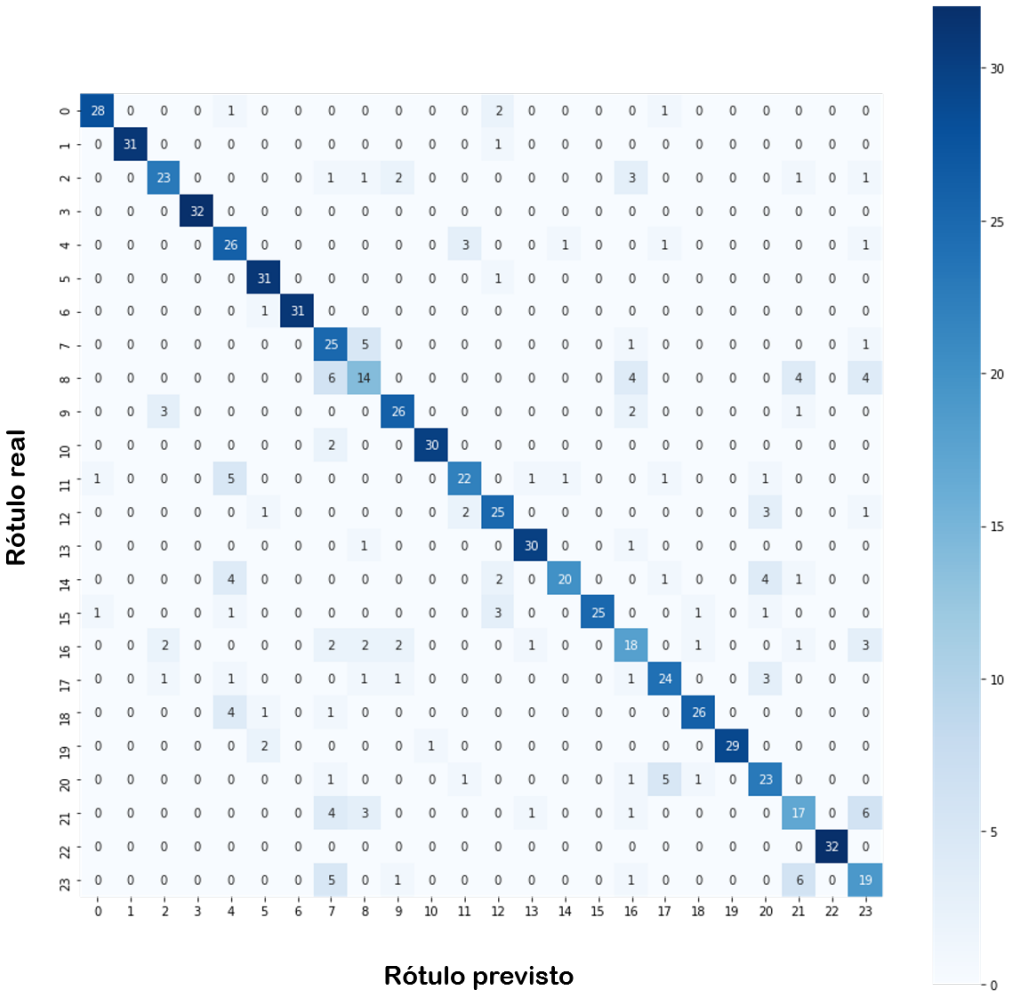
A matriz de confusão obtida pós ajuste fino para o cenário 2 é apresentada na Figura 25.

Figura 22: *Curvas de aprendizado do cenário 2 sem ajuste fino.*



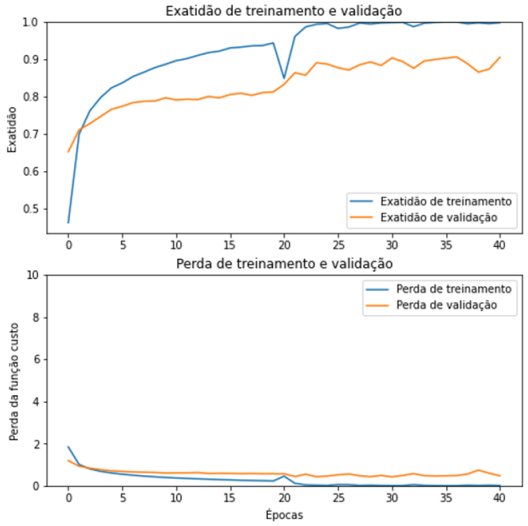
Fonte: Elaborada pelo autor.

Figura 23: *Matriz de confusão do cenário 2 sem ajuste fino.*



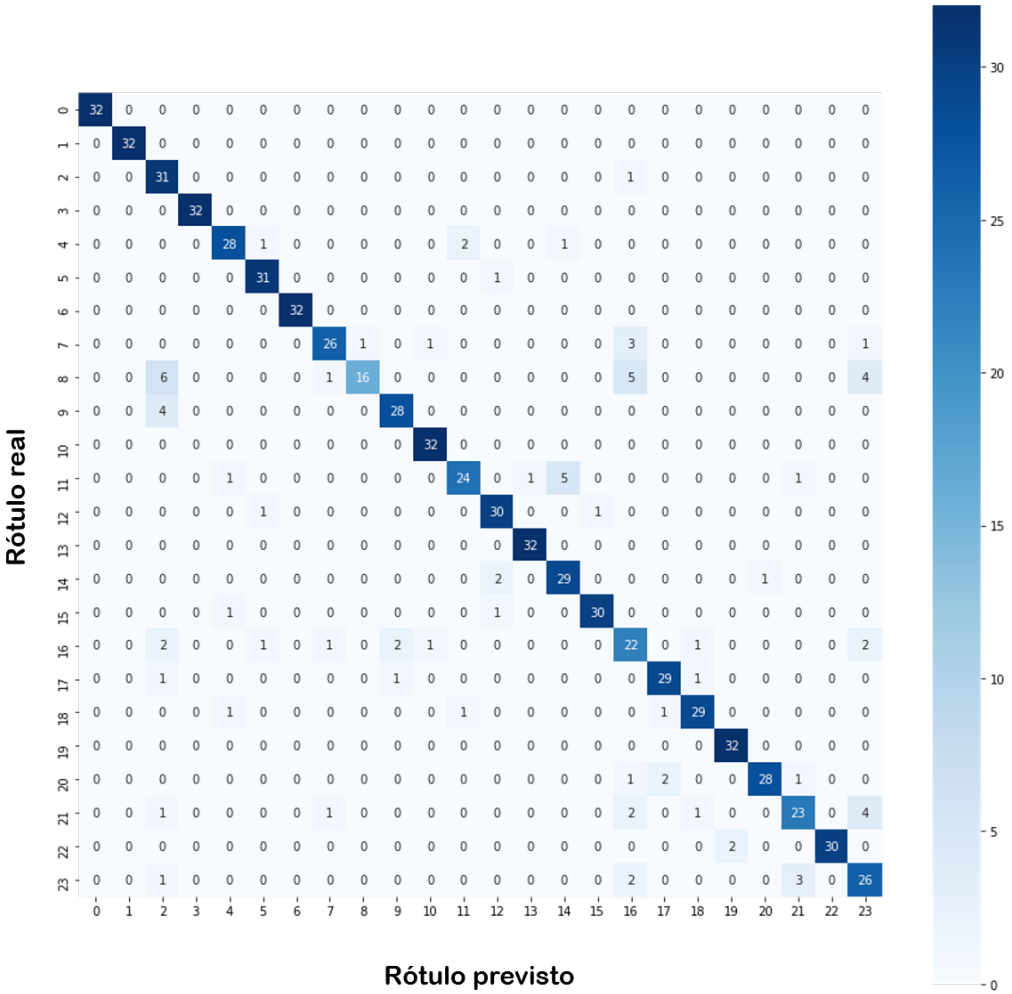
Fonte: Elaborada pelo autor.

Figura 24: *Curvas de aprendizado do cenário 2 com ajuste fino.*



Fonte: Elaborada pelo autor.

Figura 25: *Matriz de confusão do cenário 2 com ajuste fino.*



Fonte: Elaborada pelo autor.

A tabela 2 mostra os valores das métricas de avaliação obtidos para o treinamento segundo cenário.

Tabela 2: *Parâmetros de precision e recall do cenário 2.*

Parâmetro	pré ajuste fino	pós ajuste fino
Precision	0.803293658728772	0.8984467620252418
Recall	0.7903645833333334	0.890625
F1 score	0.7927080629669757	0.8894958043001613

6 CONCLUSÃO

Neste trabalho, foi proposta uma metodologia escalável e adaptável para a classificação de peças mecânicas utilizando-se de algoritmos de *deep learning* baseados em redes neurais convolucionais. Para isso, o treinamento da rede foi realizado utilizando-se de imagens sintéticas, geradas a partir de modelos CAD, e que se mostraram como uma solução efetiva para mitigar o problema relacionado ao volume de dados necessários para treinar tais algoritmos.

Utilizando-se de uma ferramenta para modelagem gráfica com uma API para desenvolvimento de algoritmos em Python, mostrou-se que é possível de se gerar uma grande quantidade de imagens sintéticas de peças mecânicas de maneira automática e robusta, renderizando tais peças de maneira fidedigna ao seus aspectos reais e gerando uma alta diversidade através da mudança de parâmetros visuais da cena renderizada.

Neste trabalho também mostrou-se que é possível utilizar as imagens geradas a partir dos modelos CAD para realizar o treinamento de uma rede neural convolucional classificadora, utilizando-se de técnicas de transferência de aprendizado, e obtendo bons resultados nas métricas de precisão, em torno de 89%, para um cenário com 24 classes diferentes e de 91% em um cenário com 10 classes diferentes. Todavia, modificações em parâmetros de treinamento, e possivelmente na estrutura da rede, podem alavancar ainda mais esses resultados.

Existem ressalvas à serem feitas acerca da solução, visto que as métricas de precisão foram medidas também a partir de imagens sintéticas. Este trabalho, portanto, sugere a implementação do sistema de classificação em ambientes reais, capturando imagens de peças em ambiente controlado e possivelmente realizando a sua implementação em ambiente industrial.

REFERÊNCIAS

ALOM, M. Z. et al. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

BIANCO, S. et al. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, IEEE, v. 6, p. 64270–64277, 2018.

CHIN, R. T.; HARLOW, C. A. Automated visual inspection: A survey. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, n. 6, p. 557–573, 1982.

DENNINGER, M. et al. BlenderProc. *arXiv preprint arXiv:1911.01911*, 2019.

GOODFELLOW, I. et al. *Deep learning*. [S.l.]: MIT press Cambridge, 2016. v. 1.

JIANG, X. et al. *Deep Learning in Object Detection and Recognition*. [S.l.]: Springer, 2019.

KOCH, S. et al. ABC: A big CAD model dataset for geometric deep learning. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2019. p. 9601–9611.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2012. p. 1097–1105.

KUMAR, A. Computer-vision-based fabric defect detection: A survey. *IEEE transactions on industrial electronics*, IEEE, v. 55, n. 1, p. 348–363, 2008.

PENG, X. et al. Learning deep object detectors from 3d models. In: PROCEEDINGS of the IEEE International Conference on Computer Vision. [S.l.: s.n.], 2015. p. 1278–1286.

RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

RUSSELL, S.; RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. [S.l.]: Pearson, 2019. (Pearson series in artificial intelligence). ISBN 9780134610993.

SCIME, L.; BEUTH, J. Anomaly detection and classification in a laser powder bed additive manufacturing process using a trained computer vision algorithm. *Additive Manufacturing*, Elsevier, v. 19, p. 114–126, 2018.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

WONG, M. Z. et al. Synthetic dataset generation for object-to-model deep learning in industrial applications. *PeerJ Computer Science*, PeerJ Inc., v. 5, e222, 2019.