# Classificação de Componentes Eletrônicos em Placa de Circuito Impresso utilizando Machine Learning

**Fabrício C. Guimarães, João V. Marques, Kluiwert V. T. Mota** [*]
**Francisco A. P. Januario** [**]

[*] *Faculdade de Tecnologia, Engenharia da Computação, Universidade Federal do Amazonas, AM, (e-mail: fabricioguimaraes@super.ufam.edu.br, joaomarques@super.ufam.edu.br, kluiwertvictor@super.ufam.edu.br).*
[**] *Departamento de Eletrônica e Computação, Faculdade de Tecnologia, Universidade Federal do Amazonas, AM, (e-mail: januario@super.ufam.edu.br)*

**Abstract:** In order to facilitate the identification of existing components on printed circuit boards, an application was developed using the help of the Flask framework, together with libraries for identification and classification of images such as OpenCV, implemented in an artificial neural network structure created through the library PyTorch, all libraries present in the Python programming language, we developed a prototype of a classifier of components present in a printed circuit. In this work, we use the help of a website written in HTML, CSS and JavaScript to receive the image that will then pass through the network for treatment and then classification of the components where we use Machine learning for identification. The results show problems encountered during development such as the inaccuracy of the results found by the system due to the cuts made in the images being inaccurate, in addition we have suggestions of what can be done to improve the result we want of at least 90% accuracy in identifying the components sought.

**Resumo**: Em busca de facilitar a identificação de componentes existentes em placas de circuitos impresso, foi desenvolvido uma aplicação utilizando o auxilio do framework Flask, juntamente com bibliotecas para identificação e classificação de imagens como OpenCV, implementados em uma estrutura de rede neural artificial criada através da biblioteca PyTorch, todas bibliotecas presentes na linguagem de programação Python, desenvolvemos um protótipo de classificador de componentes presentes em um circuito impresso. Neste trabalho, utilizamos o auxilio de um site escrito em HTML, CSS e JavaScript para o recebimento da imagem que então irá passar pela rede para tratamento e então classificação dos componentes onde utilizamos Machine learning para identificação. Os resultados mostram problemas encontrados durante o desenvolvimento como a inexatidão dos resultados encontrados pelo sistema devido aos cortes feitos nas imagens serem imprecisos, além disso temos sugestões do que pode ser feito para melhorar o resultado que desejamos de, pelo menos, 90% de exatidão na identificação dos componentes procurados.

*Keywords:* Neural Network; Circuits; Classification; Components; Images; Python; Machine Learning.

*Palavras-chaves:* Rede Neural; Circuitos; Classificação; Componentes; Imagens; Python; Machine Learning.

## 1. INTRODUÇÃO

A competitividade e a complexibilidade dos processos industriais, torna cada vez mais os processos produtivos eficientes. Com uma demanda alta da produção de produtos eletrônicos, torna-se imprescindível uma etapa para inspeção das placas de circuito impresso (PCI), desta forma é possível verificar erros como curto, componentes elevados e a quantidade de componentes, o que implica na necessidade da implementação de um software que seja capaz de realizar a inspeção.

FLECK et al (2006) afirmam que a busca por sistemas de controle compatíveis com a complexibilidade dos processos industriais demandou que muitas pesquisas fossem elaboradas, neste sentido o uso das Redes Neurais Artificiais (RNAs) torna-se um modelo compatível com o grau de complexibilidade que as indústrias demandam. As RNAs são algoritmos computacionais com modelo matemático inspirado na estrutura do cérebro, possibilitando a inserção simples do funcionamento em computadores.

Uma das maiores dificuldades encontradas no uso das redes neurais é a escolha da melhor arquitetura, uma vez que

esse processo é experimental e demanda tempo de execução para o treinamento da rede. Portanto, para que seja sanado o problema o uso de uma Rede Neural Convolucional (ConvNet / Convolutional Neural Network / CNN) é o mais adequado para essa situação, tendo em vista que ela é um algoritmo de aprendizado profundo que pode captar uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos da imagem e ser capaz de diferenciar um do outro.

Neste trabalho é apresentado um modelo de aprendizado profundo capaz de realizar identificações de componentes em placas de circuito impresso utilizando machine learning. Este modelo foi baseado na arquitetura de redes neurais de convolução (ConvNet / Convolutional Neural Network / CNN) possuindo fatores entre as camadas que auxiliam na recognição, promovendo uma maior precisão nos resultados.

Temos este artigo organizado da seguinte forma:

## 2. MACHINE LEARNING

Machine learning é uma técnica que tem por principio básico a elaboração de programas que melhoram seus algoritmos por meio da associações de diferentes dados, como imagens e tudo o que essa tecnologia consiga identificar (Mitchell, 1977).

Existem diversas aplicações bem-sucedidas de machine learning na solução de problemas reais, podemos citar(et al Faceli, 2011):

- Reconhecimento de palavras faladas;
- Predição de taxas de cura de pacientes com diferetes doenças;
- Detecção do uso fradulento de cartões de crédito;

Numa CNN cada camada é responsável por extrair determinadas informações dos dados de entrada. A informação flui através de cada camada da rede, com a saída da camada anterior, fornecendo a entrada para a camada seguinte da rede (Goodfellow, Bengio e Courville, 2016).

Pela capacidade de processamento que ela tem, podemos identificar características presente em uma imagem. Logo, podem ser utilizadas para classificar e reconhecer objetos.

A CNN pode ser dividida em duas partes (módulos) principais: convolução e classificação.

O módulo de convolução extrai elementos da imagem. O segundo módulo realizará a classificação dos dados extraídos durante a etapa de convolução. O resultado da classificação é direcionado para a saída da rede. Devido a essa característica de ter camadas para o pré-processamento dos dados, ela é um tipo de RNA. As suas camadas são:

- Camada de convolução: realiza a extração, mapeia o conteúdo da imagem, transforma em dados. Essa transformação é realizada na forma de blocos, onde filtros permitem a obtenção das informações da imagem.
- Pooling: a camada de pooling recebe os blocos que contém as informações extraídas na etapa de convolução. Ela simplifica a informação e repassa para camada totalmente conectada;
- Camada totalmente conectada: classifica as informações extraídas pelas camadas anteriores. A camada totalmente conectada achata o sub-bloco, contendo todas as informações extraídas.

Existem mais dois elementos importantes nas CNNs, a camada de dropout (regularização) e a função de ativação. A dropout reduz o overfitting (sobreajuste) da rede, ocorrendo quando a rede não é capaz de aplicar as relações aprendidas em dados novos.

A função de ativação, por sua vez, é responsável pelo aprendizado da rede, bem como pela relação entre as variáveis, decidindo quais neurônios serão ativados.

A aprendizagem de uma rede neural é adquirida a partir de seu ambiente. O ambiente gera uma estimulação na rede que irá modificar os pesos sinápticos com o objetivo de reduzir o erro observado, ao comparar a saída da rede com o resultado esperado. Por fim ela responde de uma maneira nova ao ambiente, logo após suas modificações (HAYKIN, 2001).

Para isso, um banco de dados de entrada é utilizado. A qualidade e quantidade dos dados utilizados impacta diretamente no treinamento e consequentemente na precisão da rede (GOODFELLOW; BENGIO; COURVILLE, 2016).

## 3. CLASSIFICADOR DE COMPONENTES

O classificador de componentes eletrônicos possui um processo minucioso para que o funcionamento aconteça, começando pelo recebimento da imagem através do site desenvolvido justamente para essa finalidade, seguindo para um processamento da imagem recebida mediante uma série de processos que facilitam para a próxima etapa do classificador e a mais importante, a própria classificação da imagem por meio de uma Rede Neural Convolucional (ConvNet / Convolutional Neural Network / CNN) que tem a função de identificar os componentes e encaminhar para a próxima etapa que seria apresentar o que foi encontrado como resultado, podemos observar este processo através do diagrama seguinte:

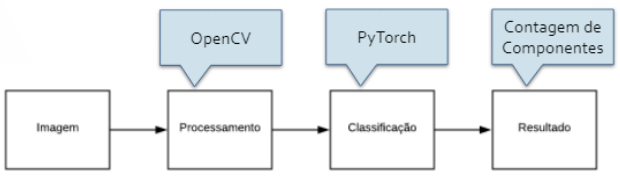


Figura 1. Diagrama da arquitetura

Cada etapa deste diagrama, foi realizado utilizando técnicas de processamento de imagem que existem na biblioteca OpenCV, nos levando ao processo da classificação em que a biblioteca PyTorch é utilizada para a criação da Rede Neural.

### *3.1 Pré-Classificação*

O processamento, ou pré-classificação, é uma das etapas mais importantes para facilitar a identificação e aumentar a porcentagem de acerto dos componentes presentes na

imagem do circuito. Com isso em mente, podemos dividir esse processamento em duas etapas importantes:

*Escala de cinza* A escala de cinza é muito utilizada para que tenha uma melhor visualização dos objetos presentes em imagens, esse processo de conversão da imagem é utilizado no inicio do classificador, logo no recebimento da imagem através do ambiente Web, a imagem já é convertida utilizando uma função presente na biblioteca OpenCV que permite está conversão de forma prática e rápida.

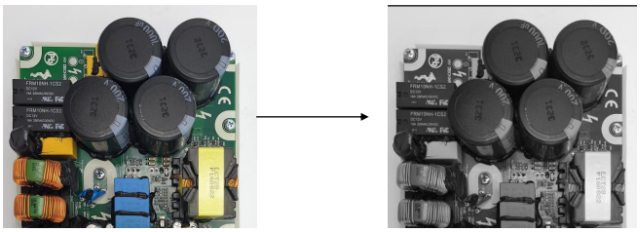

Figura 2. Conversão para escala de cinza

Além da vantagem de visualização computacional que a escala de cinza nos fornece, podemos também pontuar que ao utilizarmos, temos uma diminuição na matriz resultante que a imagem gera.

*Divisão da Imagem em Blocos* Esta etapa possui o objetivo de realizar a separação da imagem em 9 pedaços iguais, que permitem uma maior exatidão nos componentes que estão presentes em cada pedaço. Este processo é feito se aproveitando de funções de rotação da imagem e de métodos de divisão manual presentes na linguagem Python. Inicialmente, devemos fazer a rotação da imagem em $90^{o}$ o que nos permite trabalhar melhor na imagem, em seguida um recorte é feito para obter um segmento de 1/3 da imagem que ficam como faixas, este processo é repetido, porém, mudamos a localização que será realizada a divisão, obtendo assim a imagem inicial dividida em 3 faixas iguais, essas imagens então são revertidas a rotação em $90^{o}$ e armazenamos todos estes pedaços em uma lista.

```
# Para cortar a imagem quadrada em 3 faixas horizontais

def recort_t1(img, lista):
  #Rotação em 90°graus
  img2 = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
  altura = img2.shape[0] //3
  t1 = img2[:, (altura*2):(altura*3)]
  t1 = cv2.rotate(t1, cv2.ROTATE_90_COUNTERCLOCKWISE)
  lista.append(t1)
  return lista
```

Figura 3. Código do recorte em faixas

Em sequência, devemos realizar a divisão das faixas em 3 blocos iguais tendo assim a imagem dividida em 3x3, o processo para que isso ocorra é o mesmo apresentado anteiormente, porém o corte anterior era feito com base apenas na altura da imagem, neste caso, juntamente devemos nos preocupar com a largura, no restante não à diferença no processo, podemos observar este processo de forma mais clara a partir da figura 4.

```
def bloco_1(img, lista_blocos):
  img2 = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
  largura = img2.shape[0] //3 #100
  altura = img2.shape[1]
  p1 = img2[:largura, :altura]
  p1 = cv2.rotate(p1, cv2.ROTATE_90_COUNTERCLOCKWISE)
  lista_blocos.append(p1)
  return lista_blocos
```

Figura 4. Código da divisão em blocos

Pode-se notar pelas figuras que estes processos foram todos criados em funções, onde faz-se necessário um código principal da divisão da imagem, onde é feito esta chamada das funções, em que se utiliza duas listas para o primeiro e segundo processo do corte. Devemos lembrar que este processo pode ser feito de diversas formas, porém a que obtivemos sucesso na implementação foi desta forma.

```
def corta3x3(img):
  img = get_square(img, 300)
  lista_faixas = []
  lista_blocos = []

  lista_faixas = recort_t1(img, lista_faixas)
  lista_faixas = recort_t2(img, lista_faixas)
  lista_faixas = recort_t3(img, lista_faixas)

  for i in range(3):
    lista_blocos = bloco_1(lista_faixas[i], lista_blocos)
    lista_blocos = bloco_2(lista_faixas[i], lista_blocos)
    lista_blocos = bloco_3(lista_faixas[i], lista_blocos)

  return lista_blocos
```

Figura 5. Código principal da divisão

### *3.2 Classificação da imagem*

O classificador de imagem tem como principal objetivo identificar e contar os componentes nas imagens pré-processadas. Para esse processo, as imagens que estão em listas, como visto no tópico passado, são passadas ordinalmente como entrada da rede neural convulacional para a classificação. Por sua vez, a rede utiliza de um modelo pré-treinado para a previsão das saídas de cada fragmento da imagem original. Cada imagem é redimensionada para o tamanho padrão que a rede foi treinada, para que se possa trabalhar com um padrão. Logo após esse processo, a imagem deixa de ser um elemento visual e passa a ser uma matriz de tensores, que é a melhor forma que foi encontrado para como a CNN irá trabalhar. Com isso, é possível para a imagem passar pelas camadas do modelo da rede que possui uma alternancia entre o processo de pooling e o processo de convolução, sendo que o pooling é utilizado para agrupar e preparar um mapa de características condensadas. Uma arquitetura similar ao processo descrito pode ser visto na figura 6:

Como é possível observar, camadas de convolução e pooling são essenciais para a busca do resultado da classificação, onde são tratadas com a função de ativação Relu, que é uma função que retorna 0 para todos os valores que sejam negativos, e retorna o valor inserido para os positivos,

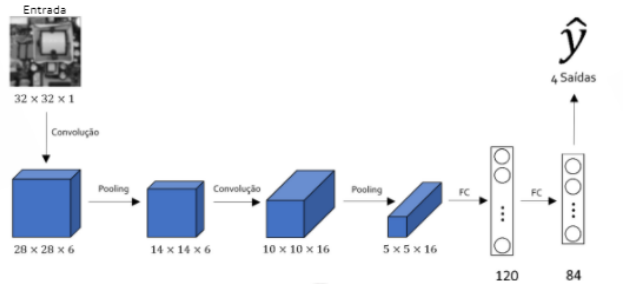


Figura 6. Arquitetura das camadas do Classificador CNN

além disso, como temos o resultado 0 em valores negativos, ocorre um aumento na velocidade do treinamento da rede. Após a passagem por todas as camadas da rede, um valor numérico representará o resultado da operação de classificação, que interpretado é capaz de descrever qual componente foi encontrado na imagem inicial de entrada. Este processo é repetido para cada fragmento da imagem original que foi dividido no processo de pré-classificação. Após a rede ser treinada, temos que carregar-la no Colaboratory para ocorrer os testes de funcionamento, podemos ver como é realizada o carregamento da rede a partir da figura 7.

```
caminho_modelo = '/content/drive/MyDrive/dataset electronic component/modelo.pth'

model = CNN()
model.load_state_dict(torch.load(caminho_modelo))
model.eval()
learning_rate = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Figura 7. Código de carregamento da CNN

Podemos notar que a rede treinada já esta presente em uma pasta no ambiente do Google Drive, e é feita somente a chamada e carregamento da mesma. Em sequência, é onde o processo de classificação realmente acontece para a identificação dos componentes separados, sem realizar o acumulo da quantidade de componentes encontrados, podemos ver no código da figura 8 como esse processo ocorre, nos retornando o resultado 'relé' devido a imagem inicialmente posta como forma de teste que foi utilizada.

```
def verifica_imagem(imagem_PIL):
  imagem = transform_prediction(imagem_PIL).unsqueeze(dim=0).to(device)
  prediction = model(imagem)
  resultado = str(torch.argmax(prediction))
  resultados = {
    "tensor(0)": "Fusível",
    "tensor(1)": "Relé",
    "tensor(2)": "Capacitor Eletrolítico",
    "tensor(3)": "LED"
  }
  return resultados.get(resultado, "Nenhum")

verifica_imagem(Image.open('/content/drive/MyDrive/dataset electronic component/dataset/test/relay/relay054.jpg'))
```

```
'Relé'
```

Figura 8. Código de classificação dos componentes

Como dito anteriormente, o processo subsequente é feito assim que toda lista de imagens for verificada pela rede neural, então é feita a contagem de componentes da imagem inicial passada pelo usuário e este resultado é mostrado na tela do cliente da aplicação, ou seja, a tela do classificador presente na aplicação WEB.


Figura 9. Resultado da aplicação no ambiente WEB

## 4. BASE DE DADOS

Em uma rede neural em que se é aplicada o método de analise Machine Learning, é preciso que a rede possua uma grande variedade de informações sobre aquilo que se está analisando, sendo estas informações, quando olhamos apenas para o projeto aqui abordado, majoritariamente composto por imagens dos componentes selecionados para que ocorra está analise em busca de calcular a quantidade presente no circuito.

A base de dados foi dividida em imagens de 4 tipos de componentes eletrônicos muito predominantes em placas de circuitos, sendo eles:

- Fusível de cartucho
- Relê elétrico
- Capacitor eletrolitico
- LED

A razão para a escolha destes componentes está na facilidade de serem encontrados, estando em diversos tipos diferentes de placas de circuitos, que causa uma facilitação para o desenvolvimento pois o treinamento da rede neural se torna menos complexo, nos levando a poder nos concertrarmos na exatidão em que a rede irá detectar os elementos que selecionamos em uma placa.

A procura por uma base de dados com imagens de circuitos tanto reais quanto 3D foi extensa, e no fim, não obtivemos sucesso em encontrar. A razão da procura, era sobretudo para testes de detecção com diferentes tipos de fotos de placas sendo elas reais ou não. Sem sucesso na busca, se fez necessário a criação manual de uma base de dados para os testes acontecerem, portanto selecionamos 50 imagens reais e 30 imagens 3D dos circuitos, estas imagens foram utilizadas apenas para testes e não estiveram presentes no treinamento da rede.

Com a finalidade de treinarmos a rede com os componentes selecionados, utilizamos uma base de dados com diversos elementos presentes em placas de circuito, esta base de dados foi adquirida através do site Kaggle, onde se faz presente uma variedade de bases de dados voltadas para Machine Learning. As imagens são pré preparadas para funcionarem como treinamento devido a possuirem diversos ângulos de um mesmo componente, dando mais informação para que a rede aprenda a identifica-lo.

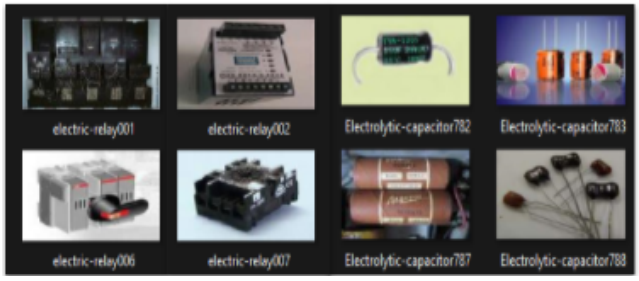


Figura 10. Base de dados de treinamento

Esta base de dados possui em um todo 1551 imagens de componentes que foram divididas em imagens que seriam usadas para o treinamento e imagens para os testes, a quantidade de imagens que um componente detém, não é igual para todos os componentes selecionados, porém eles possuem o suficiente de imagens para que a rede seja treinada de forma efetiva.

| Quantidades de Componentes PCI | | | |
|---|---|---|---|
| Componente | Imagens de Treino | Imagens de Teste | Total |
| Fusível | 157 | 68 | 225 |
| Relé | 310 | 133 | 443 |
| Capacitor Eletrolítico | 282 | 121 | 403 |
| LED | 336 | 144 | 480 |
| | 1085 | 466 | 1551 |

70%

Figura 11. Base de dados dividida

## 5. ARQUITETURA WEB

### *5.1 Diagrama da arquitetura*

A Figura 12 mostra a arquitetura da aplicação, onde a primeira etapa consiste no cliente utilizar o seu navegador Web, posteriormente ele acessa a aplicação e por fim ele escolhe entre utilizar o classificador de componentes PCI ou a trilha de aprendizado.

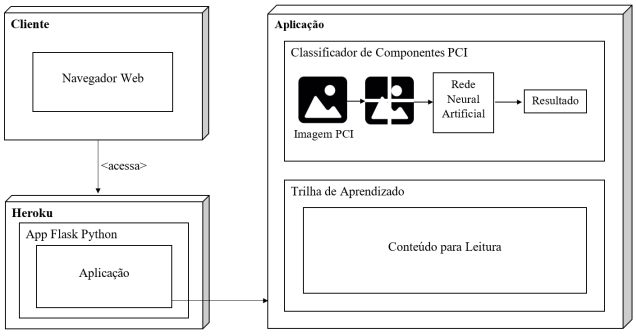


Figura 12. Arquitetura da aplicação.

### *5.2 Frameworks Utilizados*

Durante o processo de criação do site foram utilizados alguns frameworks, sendo as tecnologias utilizadas nesse processo divididas no front-end e back-end.

No front-end utilizamos:

*HTML5* como linguagem marcação de hipertexto, sendo utilizada na construção das páginas web;

*CSS3* para a estilização das páginas;

*JAVASCRIPT* para deixar o site mais dinâmico.

No back-end utilizamos:

*Python* como linguagem de programação;

*Flask* como framework para o desenvolvimento;

*OpenCV* realizará o processamento da imagem;

*PyTorch* realizará a classifição da imagem.

## 6. AVALIAÇÃO EXPERIMENTAL

### *6.1 Ambiente de Testes*

Em consequência da produção do projeto ter sido realizada de maneira remota, não foi utilizado nenhum hardware para simulação do experimento. Em suma, o projeto foi desenvolvido através de um Notebook que tirou proveito do ambiente de desenvolvimento disponibilizado pelo Google conhecido como Colaboratory, em que os membros do projeto realizavam em conjunto os códigos necessários para partição da imagem e classificação, este código está em conexão com uma pasta no Google Drive, onde ficam localizadas as imagens que desejamos para teste manual ou treinamento da rede, este código desenvolvido é então anexado ao ambiente de hospedagem HerokuAPP que funciona como a conexão entre o front-end e o back-end do projeto.

O Front-end de nosso projeto foi desenvolvido através da IDE conhecida como Visual Studio Code, em que o código era escrito para então ser passado para o software de código aberto, NodeJs. Este software é o responsável pela execução do código escrito em JavaScript, a partir daqui o processo é o mesmo presente no back-end em que o código é passado para o ambiente de hospedagem HerokuAPP permitindo a execução do projeto.

### *6.2 Resultados*

Os testes realizados inicialmente demonstraram um resultado insatisfatório em relação a precisão de detecção dos componentes quando estão realmente presentes nas placas. A experimentação do classificador em uma imagem em que apresente apenas o componente selecionado, nos mostra uma precisão de aproximadamente 80%. Em contrapartida, o resultado encontrado com a placa completa do circuito com componentes diversos é inferior ao esperado. Foram realizadas tentativas para o melhoramento da porcentagem de exatidão através de métodos de ”Canny”, onde o intuito é dividir a imagem de forma que os componentes ficassem unitarios em cada partição da imagem original, porém não foi obtido sucesso na implementação deste método.

Em contrapartida, com a utilização de outros métodos próprios para divisão da imagem, obtivemos sucesso em transformar-la da forma como era necessário para que o

projeto desse continuidade, apesar da falta de exatidão por parte do classificador quando possuimos diversos componentes presentes em uma unica partição da imagem original, podemos ver um exemplo disso na figura 9, em que os números de capacitores presentes é incerto em relação a real quantidade presente na imagem de teste, porém isso pode ser resolvido com a implementação dos métodos de "Canny"como citado anteriormente. No momento, com o método de partição atual, a porcentagem média de exatidão que possuímos no classificador com as imagens de treino é de 73,52%, sendo que tiveram uma média de acerto nas imagens de 789 em relação a 1085 imagens presentes no banco de imagens de treinamento, em outro ponto, utilizando o banco de imagens de teste que consiste de 466 imagens, tivemos uma média de acerto de 346 imagens, o que nos traz para uma média de exatidão de 74,32% para as imagens de teste. Como podemos analisar pelas porcentagens encontradas a partir de testes, o classificador possui uma chance de erro de 1/4 aproximadamente, podemos ver isto claramente na tabela 1.

| | Imagem de treino | Imagem de teste |
|---|---|---|
| Acertos | 789 | 346 |
| Quant. de imagens | 1085 | 466 |
| Porc. de exatidão | 73,52% | 74,32% |

Tabela 1. Resultados de exatidão

## 7. CONCLUSÃO

Neste artigo foi proposto um modelo de aprendizado profundo capaz de realizar identificações de componentes em placas de circuito impresso. O modelo foi baseado na arquitetura de redes neurais de convolução (CNN) possuindo fatores entre as camadas que auxiliam na recognição, promovendo uma maior precisão nos resultados. Os resultados obtidos mostraram a viabilidade do projeto.

No decorrer do desenvolvimento do projeto, vislumbra-se algumas possibilidades de continuidade e melhorias:

- A implementação de um hardware com uma câmera, para simulação mais precisa do ambiente industrial.
- Adição do processo de detecção de bordas de CANNY.
- Adição de mais componentes para serem identificados.

## AGRADECIMENTOS

Neste momento, é com imensa satisfação que agradecemos ao Projeto Super - Projeto de Educação e Pesquisa da Universidade Federal do Amazonas em parceria com a empresa Samsung - e ao Prof. Me. Francisco de Assis Pereira Januário de pela oportunidade que nos foi concedida de trabalhar com machine learning.

## REFERÊNCIAS

CUNHA, Leornado Cardoso da. Redes neurais convolucionais e segmentação de imagens – Uma revisão bibliográfica. Orientadora: Luciana Castanheira. 2020. 51 f. Monografia (Graduação) - ENGENHARIA DE CONTROLE E AUTOMAÇÃO, Universidade Federal de ouro Preto, Ouro Preto, 2020. Disponivel em: http://www.monografias.ufop.br/handle/35400000/2872. Acesso em: 15 de nov. 2021.

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

FLECK, Leandro et al. REDES NEURAIS ARTIFICIAIS: PRINCÍPIOS BÁSICOS. Revista Eletrônica Científica Inovação e Tecnologia, Paraná, v. 1, n. 13, p. 47-57, jan./jun. 2016.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning. MIT Press. 2016.

HAYKIN, S. Redes Neurais- Princípios e Práticas. BOOKMAN, São Paulo, 2ª ed. 2001.

LUDERMIR, TERESA BERNARDA. Inteligência Artificial e Aprendizado de Máquina : estado atual e tendências. DOI: https://doi.org/10.1590/s0103-4014.2021.35101.007. Dissponível em: https://www.youtube.com/watch?v=qenRdsVgKVo. Acesso em: 01 dez. 2021.

MITCHELL, T. Machine Learning. S. l.: McGraw Hill, 1997.

BARSTUĞAN, MÜCAHID AND CEYLAN, RAHIME. International Artificial Intelligence and Data Processing Symposium (IDAP), CLASSIFICATION OF MAMMOGRAM IMAGES BY DICTIONARY LEARNING, 2017,p. 1-4.

AMRANE, MERIEM AND OUKID, SALIHA AND GAGAOUA, IKRAM AND ENSARİ, TOLGA. Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT),BREAST CANCER CLASSIFICATION USING MACHINE LEARNING, 2018, p. 1-4.

KAVANA, V AND NEETHI, M. International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), FAULT ANALYSIS AND PREDICTIVE MAINTENANCE OF INDUCTION MOTOR USING MACHINE LEARNING, 2018, p. 963-966.