Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração (PROFICAM)
Escola de Minas, Universidade Federal de Ouro Preto (UFOP)
Associação Instituto Tecnológico Vale (ITV)

Dissertação

DEEP LEARNING E *DEVICE EDGE* NA IMPLEMENTAÇÃO DE DETECTOR DE RASGO DE CORREIA TRANSPORTADORA DE MINÉRIO DE FERRO

Emerson Klippel

Ouro Preto
Minas Gerais, Brasil
2021

Emerson Klippel

DEEP LEARNING E *DEVICE EDGE* NA IMPLEMENTAÇÃO DE DETECTOR DE RASGO DE CORREIA TRANSPORTADORA DE MINÉRIO DE FERRO

Dissertação apresentada ao Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração da Universidade Federal de Ouro Preto e do Instituto Tecnológico Vale, como parte dos requisitos para obtenção do título de Mestre em Engenharia de Controle e Automação.

Orientador: Prof. Ricardo Augusto Rabelo Oliveira, D.Sc.

Ouro Preto
2021

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

K65d Klippel, Emerson.
Deep Learning e Device Edge na implementação de detetor de rasgo de correia transportadora de minério de ferro. [manuscrito] / Emerson Klippel. - 2021.
77 f.: il.: color., gráf..

Orientador: Prof. Dr. Ricardo Augusto Rabelo Oliveira.
Dissertação (Mestrado Profissional). Universidade Federal de Ouro Preto. Programa de Mestrado Profissional em Instrumentação, Controle e Automação de Processos de Mineração. Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração.
Área de Concentração: Engenharia de Controle e Automação de Processos Minerais.

1. Beneficiamento de minério. 2. Correias transportadoras. 3. Aprendizado de máquina - Deep learning. I. Oliveira, Ricardo Augusto Rabelo . II. Universidade Federal de Ouro Preto. III. Título.

CDU 681.5:622.7

Bibliotecário(a) Responsável: Sione Galvão Rodrigues - CRB6 / 2526

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Emerson Klippel

"Deep Learning e Device Edge na Implementação de Detector de Rasgo de Correia Transportadora de Minério de Ferro"

Dissertacao apresentada ao Curso de Mestrado Profissional DO PROGRAMA DE PÓS GRADUAÇÃO EM INSTRUMENTAÇÃO, CONTROLE E AUTOMAÇÃO DE PROCESSOS DE MINERAÇÃO da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Mestre

Aprovada em 3 de Agosto de 2021

Membros da banca

Doutor - Ricardo Augusto Rabelo Oliveira - Orientador(a) Universidade Federal de Ouro Preto
Doutora -Andrea Gomes Campos Bianchi -Universidade Federal de Ouro Preto
Doutor -Edson Jorge de Matos - Universidade Federal do Pará
(Participação por videoconferência)
Doutor -Agnaldo José da Rocha Reis - Universidade Federal de Ouro Preto

Ricardo Augusto Rabelo de Oliveira, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 03/08/2021

Documento assinado eletronicamente por **Ricardo Augusto Rabelo Oliveira**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 03/11/2021, às 08:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0239583** e o código CRC **1E0A7DE8**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: 3135591692 - www.ufop.br

A minha esposa Dolores e meu
filho Emerson Júnior

Agradecimentos

Agradeço primeiramente ao Grande Arquiteto do Universo que por sua graça nos presenteou com o dom da inteligência e curiosidade, alicerces de toda a nossa ciência.

Agradeço minha esposa pela paciência, compreensão, carinho e apoio durante mais esta importante etapa de minha vida.

Agradeço ao meu filho pelo apoio, discussões e revisões no material nas etapas de desenvolvimento.

Agradeço aos professores do curso de mestrado pelo compartilhamento do seu conhecimento conosco, em especial ao meu orientador, pelo constante apoio em todas as etapas do trabalho.

Finalmente agradeço aos líderes da VALE Mauro Brombley e Gustavo Bastos que nos idos de 2018 viabilizaram, apesar de todos os desafios da área na época, o meu ingresso no mestrado.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), Código de Financiamento 001; do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq); da Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG); e da Vale SA.

"Os grandes homens parecem-nos grandes audaciosos; no fundo, eles obedecem mais que os outros. Uma voz soberana instrui-os. É porque um instinto provindo dela aciona-os que tomam, sempre corajosamente e as vezes com grande humildade, o lugar que lhes dará mais tarde a posteridade, ousando ações e arriscando invenções frequentemente em desacordo com seu meio, sendo até alvo de seus sarcasmos. Eles não tem medo, porque, por mais isolados que pareçam, não se sentem sós. Têm ao seu favor Aquele que tudo decide afinal. Pressentem seu futuro império"(A.-D Sertillanges)

Resumo

Resumo da Dissertação apresentada ao Programa de Pós Graduação em Instrumentação, Controle e Automação de Processos de Mineração como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

DEEP LEARNING E *DEVICE EDGE* NA IMPLEMENTAÇÃO DE DETECTOR DE RASGO DE CORREIA TRANSPORTADORA DE MINÉRIO DE FERRO

Emerson Klippel

Agosto/2021

Orientador: Ricardo Augusto Rabelo Oliveira

Transportadores de correias estão entre os principais ativos utilizados em usinas de beneficiamento de minério de ferro, sua disponibilidade e confiabilidade impactam diretamente na performance global dessas plantas tanto em aspectos de saúde e segurança das pessoas como em financeiros. Entre os modos de falhas desses dispositivos são comuns os rasgos nas correias transportadoras de borracha vulcanizada devido a diversos agentes como materiais cortantes e perfurantes provenientes da mina e ou sucatas mecânicas diversas. Os sistemas atuais de detecção de rasgo baseados em dispositivos eletromecânicos, óticos, eletromagnéticos ou de eletrônica integrada as correias não possuem a confiabilidade, robustez e facilidade de manutenção necessárias aos ambientes de mineração. Sendo assim o presente trabalho apresenta a implementação de um sistema de sensoriamento utilizando captura de imagens com detecção em tempo real de rasgos e ou anomalias do tapete da correia através de modelos *deep learning*. A implementação do *deep learning* deverá atender a requisitos de baixa demanda computacional permitindo sua aplicação em computador de borda de baixo custo e sem conectividade externa para processamento das informações, alinhado aos preceitos de *edge AI*. Para o treinamento do modelo será construído *dataset* específico com imagens de situações de rasgo e normais de correias reais. Todo o processo de treinamento e conversão do modelo para uso com *device edge* será abordado na metodologia, bem como as estratégias de execução das coletas de imagens, construção de protótipos e testes de campo. Para os testes executados com a versão final do protótipo a precisão de detecção de rasgo foi de 98%, mostrando a viabilidade

do uso de *deep learning* em conjunto com *device edge* para detecção de rasgos de correia transportadoras.

Palavras-chave: deep learning, detecção rasgo, computação de borda.

Macrotema: Usina; **Linha de Pesquisa:** Tecnologias da Informação, Comunicação e Automação Industrial; **Tema:** Transportadores de Correia; **Área Relacionada da Vale:** Manutenção de Correias Transportadoras.

Abstract

Abstract of Dissertation presented to the Graduate Program on Instrumentation, Control and Automation of Mining Process as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DEEP LEARNING AND DEVICE EDGE IN RIP DETECTOR IMPLEMENTATION FOR IRON ORE CONVEYOR BELT

Emerson Klippel

August/2021

Advisor: Ricardo Augusto Rabelo Oliveira

Conveyor belts are one of the main assets used in iron ore facilities, their disponibility and reliability impacting directly on the overall performance of these plants in health and safety as well as financial aspects. One of the most common failures in these conveyor belts is mechanical tear due to sharp materials coming from the mine or scraps from machinery. Eletromechanical, optical, electromagnetic or electronic systems integrated in conveyor belts don´t have the realiability, robustness and ease of maintenance required in mining areas. Thus, this paper presents the implementation of a sensing system utilizing image capture with live tear or anomalies detection in theconveyor belts through deep learning. The implementation of deep learning must meet requirements of low computational demand, allowing its application in low-cost edge computers without external connectivity for information processing, in line with edge AI paradigms. For model training, a specific dataset will be built with images of longitudinal rips and normal situations of real belts. The entire process of training and converting the model for use with device edge will be covered in the methodology, as well as the execution strategies for image collection, prototype construction and field tests. For the tests performed with the final version of the prototype, the longitudinal rip detection precision was 98%, showing the feasibility of using deep learning in conjunction with device edge to detect conveyor belt tears.

Keywords: deep learning, rip detection, edge device.

Macrotheme: Ore Plant; **Research Line:** Information, Communication and Industrial Automation Technologies; **Theme:** Belt Conveyors; **Related Area of Vale:**Conveyor Belt Maintenance.

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

AI Inteligência Artificial

APU Audio Processig Unit

BSM Britagem Semimóvel

CNN Convolutional Neural Network

DNN Deep Neural Network

DVP Digital Video Port

FFT First Fourier Transform

GNU General Public License

GPIO General Purpose Input/Output

GPU Graphics Processing Unit

IDE Integrated Development Environment

ILSVRC ImageNet Large Scale Visual Recognition Chalenge

KPU Knowledge Processing Unit

NR22 Norma Regulamentadora 22 - Segurança e Saúde Ocupacional na Mineração

QVGA Quarter Video Graphics Array

ReLU Retified Linear Units

s stride

SCCB Serial Camera Control Bus

SoC System on Chip

UART Universal Asynchronous Receiver/Transmitter

Sumário

1. Introdução

Os transportadores de correias são os principais ativos empregados em usinas de beneficiamento de minério de ferro. Como exemplo o complexo de beneficiamento de minério de ferro de Carajás utiliza 332 transportadores de correias com comprimento total de 114 km.

São aplicados em larga escala no transporte do minério de ferro entre as diversas etapas do processo de uma usina, possuindo custo de operação e flexibilidade superior se comparados a outros métodos de transporte, principalmente se foram considerados os altos volumes movimentados nessas plantas. Paradas nesses equipamentos impactam diretamente na produtividade do processo de beneficiamento, Ribeiro *et al.* (2016). Na Figura 1.1 é mostrada a usina de Carajás com 30 correias transportadoras.

Figura 1.1: Transportadores Correia Usina 1 Carajás. Fonte: KLIPPEL, 2019.

Um transportador de correia é constituído por estrutura física metálica, cavaletes de sustentação, mesa de impacto, rolos (roletes) de carga e retorno, tambores de acionamento, contrapeso e retorno, correia de borracha vulcanizada e sistema de acionamento eletromecânico. O transportador possui dois pontos de transferência de material, um de entrada e outro de saída, esses pontos são os chutes de carregamento e de descarga respectivamente, Swinderman *et al.* (2012). O chutes são caixas metálicas reforçadas que direcionam o fluxo do material. na Figura 1.2 é mostrada a configuração típica de um transportador de correia de minério de ferro.

A correia de borracha é o elemento mais susceptível a falhas com origem nos processos de operação e manutenção e dentre essas a mais comum é o rasgo do tapete por blocos de minério lamelares ou sucata metálica proveniente da mina ou gerada por falhas de manutenção, Hardygóra *et al.* (1999). A Figura 1.3 apresenta a situação do rasgo causado por bloco de jaspelito, tipo de minério de ferro, originado na mina e não britado pelo sistema de britagem da entrada da usina de Carajás.

Os rasgos em correias transportadoras são difíceis de controlar mesmo sendo ado-

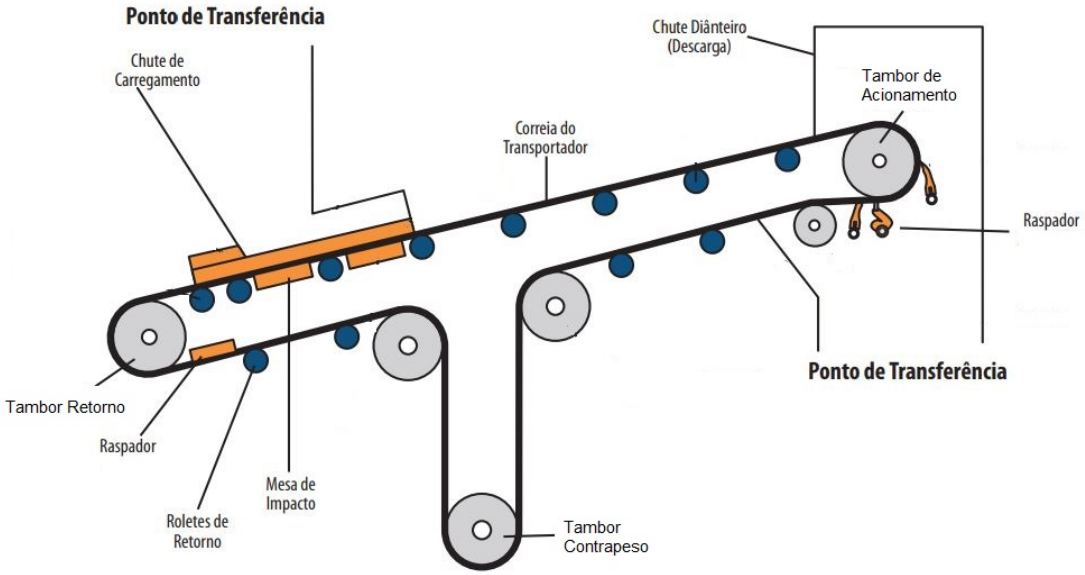


Figura 1.2: Transportadora de Correia. Fonte: Adaptado de Martin Engineering, 2012.

Figura 1.3: Correia rasgada por mataco lamelar. Fonte: KLIPPEL, 2019.

tadas técnicas com objetivo de minimizar suas ocorrências. Dentre as técnicas podem ser destacadas: ajustes da abertura dos britadores, inspeção periódica dos componentes constituintes do sistema de transportadores (chutes de descarga, rolos de carga, rolos de retorno, raspadores de correia, etc), detectores de metal e telas, Swinderman *et al.* (2012).

Para minimizar os efeitos dos rasgos, com a redução de sua extensão, e garantir o retorno a mais rápido possível a operação são utilizados sensores de rasgo. Estes sensores, quando detectam um rasgo na correia, enviam uma sinal digital para o sistema de controle da planta. O sistema de controle é responsável por enviar imediatamente os comandos necessários a parada emergencial da correia.

1.1 Justificativa

Os prejuízos financeiros originados por rasgos de correias estão associados às perdas de produção pela necessidade da substituição da correia ou da realização de grandes reparos nestas. Em algumas situações essas paradas podem levar mais de 24 horas. O valor médio do metro de correia vulcanizada com alma de aço é de R$ 1.000,00, logo um rasgo de 1000m de uma correia média gera um prejuízo direto de R$ 1.000.000,00, considerando somente a substituição da correia, sem levar em conta os demais custos operacionais envolvidos.

Como exemplo nas usinas de Carajás, segundo Sousa *et al.* (2019), só no segundo semestre 2019 ocorreram 5 rasgos de correias com danos diversos nos sitemas Britagem Semimóvel (BSM) conforme indicado na Tabela 1.1 incluindo duas perdas totais da correia.

Tabela 1.1: Rasgos Correia Usina Carajás. Fonte: Adaptado de Sousa (2019).

Circuito	**Correia**	**Capacidade (t/h)**	**Data**	**Comprimento (m)**	**Rasgo (m)**	**Perda**
BSM3	TR117K07	7000	30/08/19	1450	850	59%
BSM3	TR121K08	7000	02/09/19	245	245	100%
BSM3	TR117K06	7000	12/09/19	3120	1000	32%
BSM2 e 4	TR113K02	8000	17/09/19	2100	60	3%
BSM2 e 4	TR113K02	8000	18/09/19	2100	2100	100%

Aliada as perdas econômicas a execução de substituição completa ou reparo parcial de correias transportadoras traz grandes riscos às equipes de manutenção, principalmente se for considerada a não preparação prévia para a execução da atividade, a manutenção é realizada de forma corretiva, num ambiente agressivo e envolvendo atividades complexas como o corte e retirada da correia danificada, o transporte da nova correia e seu posicionamento sobre os rolos de carga e tambores, ainda existem as etapas de emenda, com sua vulcanização além dos processos de desativação e ativação do sistema de esticamento.

Ainda na execução das atividades de reparo ou substituição da correia são empregados equipamentos de grande porte como guindastes com capacidade de 100 t, guindautos de 10 t, prensas de vulcanização de emenda, máquinas de corte e sistemas de tração de correia manual, na Figura 1.4 e mostrada uma das etapas do preparo para manutenção. Numa troca de correia chegam a ser envolvidos mais de 20 profissionais, entre operadores de equipamentos, vulcanizadores, técnicos de campo, supervisores, técnicos de segurança e engenheiros de manutenção, todos expostos a riscos como corte, quedas de nível, prensamento e esmagamento de membros, batidas contra, projeção de partículas, queda de material, ruído entre outros.

Do exposto é evidente que o processo de manutenção e ou substituição de uma correia transportadora é complexo, demanda tempo, gera impactos financeiros além de trazer riscos diretos para as pessoas.

Figura 1.4: Troca Correia Patio Usinas Carajás. Fonte: Flavio Pinheiro, 2020.

Assim são empregadas técnicas de sensoriamento para detecção de rasgo. As técnicas atuais possuem limitações de sensibilidade e confiabilidade de detecção e perdas totais das correias são observadas, vide Tabela 1.1. No Capítulo 2 essas técnicas são estudadas e detalhadas.

Diante deste contexto é justificado o estudo e aplicação de novas tecnologias para desenvolvimento de uma solução para detecção de rasgos de correia de forma confiável, de fácil manutenção, escalabilidade e baixo custo.

1.2 Objetivos

O objetivo geral do trabalho é desenvolver um protótipo de solução de detecção de rasgo de correia utilizando imagens capturadas em tempo real e modelo *deep learning* sendo executado e tomando decisões localmente sem necessidade de conectividade para tal, alinhado ao paradigma de *AI on Edge*, conforme Deng *et al.* (2020b).

Para atingirmos o objetivo geral precisaremos atingir objetivos específicos conforme metodologia indicada na Figura 1.5. Os principais objetivos específicos são:

- Selecionar a plataforma computacional adequada para a o desenvolvimento do protótipo do detector. Essa seleção diz respeito tanto ao hardware quanto do modelo *deep learning* a ser utilizado. Os métodos de treinamento do modelo e sua conversão para uso com *edge device* também serão investigados além dos testes iniciais de toda a plataforma;

- Capturar, em campo, imagens de rasgo de correia tanto em condições estáticas quanto em dinâmicas, tratar as imagens e criar *dataset* de treinamento do modelo. Com esse dataset realizar o treinamento do modelo e sua otimização e conversão para uso em *edge device*;

- Construir o protótipo e realizar os testes de campo usando os modelos desenvolvidos e treinados durante os estudos e visitas a campo. Esse mesmo protótipo capturará novas imagens das condições de testes para aprimoramento do modelo.

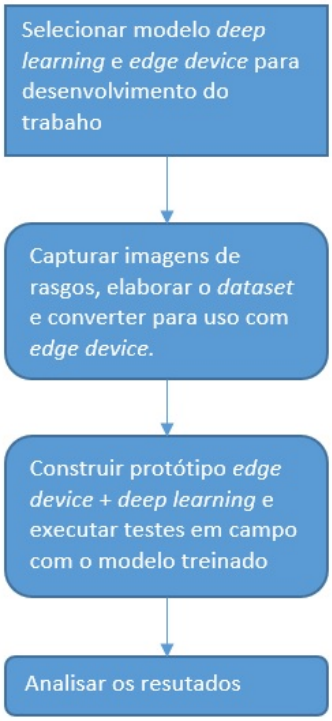


Figura 1.5: Metodologia macro utilizada no trabalho. Fonte: KLIPPEL, 2020.

1.3 Organização do Trabalho

O trabalho traz em sua organização sete capítulos. O primeiro apresenta os aspectos introdutórios do estudo incluindo as justificativas para a sua realização, considerando os cenários existentes na época de seu início. No capítulo 2 é realizado o estudo de todo o referencial teórico utilizado no desenvolvimento do protótipo, incluindo os conceitos de Inteligência Artificial (AI), paradigma de *edge AI* e apresentação das principais técnicas de sensoriamento aplicadas nos ambientes de usinas de beneficiamento de minério de ferro.

O capítulo 3 apresenta os trabalhos relacionados, considerando uso de visão computacional e inteligência artificial, na detecção de falhas em correias transportadoras ou estruturas e materiais semelhantes.

Os materiais e métodos, incluindo a definição da plataforma computacional, modelos adotados no projeto e os processos de treinamento e conversão estão agrupados no capítulo 4.

Os capítulos 5 e 6 trazem os resultados e discussões além da conclusão do trabalho, respectivamente. Nas discussões serão utilizados as indicadores de performance definidos

previamente no capítulo 4.

O capitulo 7 indica trabalhos futuros, principalmente no que diz respeito as adequações necessárias a transformação do protótipo em um produto final para aplicações em ambiente industrial.

1.4 Contribuições

No desenvolvimento do trabalho foram realizadas contribuições de natureza técnica, científicas, industriais e de saúde e segurança.

1.4.1 Contribuições Técnicas

As contribuições técnicas realizadas durante a construção do protótipo e treinamento do modelo *deep learning* foram:

1. Modelo de inteligência artificial treinado para detecção de rasgos de correia e convertido para uso com computador de borda;
2. *Scritps* em uPyton para detecção de rasgos de correia;
3. *Dataset* com imagens de rasgo de correia para uso em novos estudos.

1.4.2 Contribuições Científicas

A contribuição cientifica do trabalho diz respeito a demostração da viabilidade do uso de modelos de inteligência artificial sendo executados localmente em computadores de borda para tratamento de imagens em tempo real e detecção de rasgos de correias. Em paralelo será validado o processo de conversão de modelos neurais treinados para aplicações em nuvem ou computadores de maior porte para uso em computadores de borda, considerando todo o processo de otimização utilizado pelas ferramentas de compilação desses modelos.

1.4.3 Contribuições Industriais

O desenvolvimento da tecnologia para detecção de rasgos a partir de inteligência artificial, sendo executada em borda, trará mais uma opção de sensoriamento em relação as técnicas atuais. Consequentemente são esperados tempos de parada menores com redução dos prejuízos financeiros no processo produtivo.

1.4.4 Contribuições de Saúde e Segurança

Com danos menores nas correias devido a uma melhor detecção de rasgo serão reduzidos os tamanhos dos reparos necessários diminuindo assim o tempo de exposição à risco e esforço das equipes de manutenção corretiva de correias.

2. Referencial Teórico e Fundamentação Científica

2.1 Sensores de Detecção de Rasgo de Correia

Como mostrado na introdução, várias técnicas são usadas para evitar os rasgos de correias, mas mesmo assim eles ocorrem, e os sensores de rasgo são a última barreira que visam minimizar esses impactos ao máximo. O efeitos do rasgo completo de uma correia são muito maiores que um rasgo de 50m de extensão, relativamente mais fácil de ser manutenido.

Os detectores de rasgo são do tipo eletromecânico, eletrônicos. Esses sensores são intertravados com o sistema de controle da correia e devem garantir a parada da correia imediatamente no caso de seu acionamento.

Dentre os eletromecânicos os mais utilizados são os sensores de fios (ripcords) e os de bandeja. Os ripcords baseiam-se na interação do rasgo da correia, através de uma borda ou cabo solto pelo rasgo que romperá os fios de naylon do sensor o acionando. Esse mecanismo é construído conforme patente de Kerr (1984). Os sensores de bandeja são acionados através de acumulo de material que escapa pelo rasgo e se acumula sobre uma bandeja causando o acionamento do sensor. Os sensores de bandeja são construídos conforme patente de Lima (2015). Na Figura 2.1 são mostrados os dois tipos de sensores.

(a) Sensor ripcord.

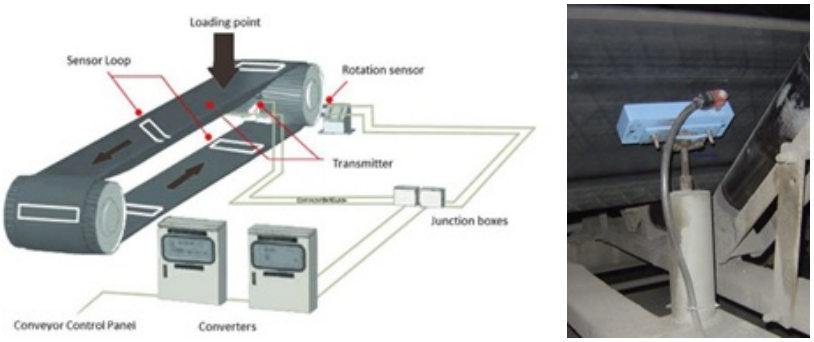
(b) Sensor bandeja.

Figura 2.1: Sensores de rasgo eletromecânicos. Fonte: Emerson Klippel, 2020.

Ambos os sensores são de simples manutenção e baixo custo mas possuem pouca confiabilidade de detecção devido a necessidade de interação do corte com o elemento sensor, interação essa ocorrida muitas vezes somente quando o dano é significativo inviabilizando, inclusive, o reparo no tapete e implicando na completa substituição da correia transportadora.

Para os sensores de rasgo eletrônicos existem aqueles incorporados ao tapete da correia com detectores instalados ao longo da estrutura mecânica de suporte da correia, dentre estes se destacam os eletromagnéticos e os capacitivos.

Os sensores eletromagnéticos, de forma geral baseiam-se na patente de Lee (1978) e consideram a interação entre uma bobina excitadora, instalada próxima a correia, e elementos sensores incorporados ao tapete da correia instaladas em intervalos regulares, esses elementos sensores são formados por bobinas curto-circuitadas. Por indução eletromagnética o sinal gerado pela bobina excitadora interage com o elemento sensor e, em caso da abertura do elo, a interação magnética sofrerá variações significativas detectadas pela eletrônica do sistema. Existem variações com formatos variados de elos sensores, números de bobinas exitadoras e receptoras, mas o princípio permanece o mesmo. Na Figura 2.2 é mostrado o diagrama do sensor e um exemplo de instalação em campo.



(a) Diagrama. (b) Instalação.

Figura 2.2: Sensor de rasgo eletromagnético. Fonte: www.conveyorbeltguide.com, 2020.

O segundo tipo de sensor eletrônico é o capacitivo. Estes utilizam um transmissor eletrônico gerando uma frequência de 200kHz num dos lados da correia, esse sinal é transmitido, através de acoplamento capacitivo, para o outro lado da correia onde existe um receptor desse sinal. Para promover o acoplamento são instaladas finas placas metálicas no interior da correia na direção de sua largura. No caso de rasgo da correia essas placas são rompidas impedindo o correto acoplamento e causando a detecção do evento e a parada do equipamento. O método em questão foi patenteado por Snyder (1974) e variações são encontradas no mercado. Na Figura 2.3 é mostrado o diagrama do sensor capacitivo.

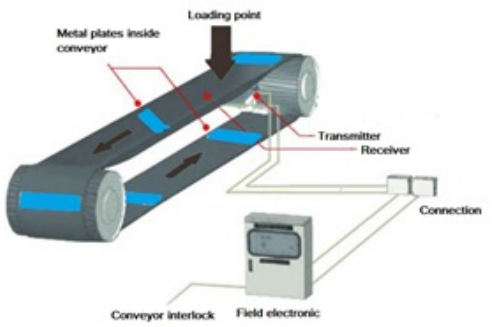


Figura 2.3: Diagrama sensor rasgo capacitivo. Fonte: www.conveyorbeltguide.com, 2020.

Ambas as soluções possuem boa confiabilidade de detecção mas apresentam como

desvantagens a necessidade da instalação de elementos detectores na própria correia, aumentando o custo do ativo além de implicar na utilização de elementos sensores de fornecedores específicos diminuindo a flexibilidade de intercâmbio de tapetes de correia durante os processos de manutenção. Em outras palavras, cada correia somente poderá ser aplicada no transportador que possuir a mesma tecnologia de detecção empregada na confecção da correia.

Outra grande desvantagem no uso dos sensores eletromagnéticos e capacitivos é o seu não funcionamento em correias transportadoras com malha de aço incorporada ao tapete, maioria dos casos em minas e usinas de grande porte, além de serem bastante sensíveis a interferência eletromagnética com falsos positivos sendo comuns. A Tabela 2.1 compara as tecnologias de detecção comumente aplicadas nos ambientes de mineração de minério de ferro.

Tabela 2.1: Comparativo Tecnologias Detecção Rasgo. Fonte: Eng. Manutenção Usinas Carajás

Tecnologia	**Vantagens**	**Desvantagens**
Eletromecânico (ripcord e bandeja)	Instalação simples Manutenção simples Baixo custo	Baixa sensibilidade de detecção
Eletrônicos (eletromagnético e capacitivo)	Alta sensibilidade	Instalação e manutenção complexas Sensível a interferência (falso positivo)

2.2 Deep Learning

As redes neurais, suas estruturas e métodos de treinamento vem sendo estudadas a décadas, desde o primeiro modelo matemático de um neurônio proposto por McCulloch e Pitts em 1943 (FERNANDES, 2005).

Durante essa evolução várias arquiteturas de rede e métodos de treinamento foram desenvolvidas permitindo a implementação de sistemas computacionais capazes de realizar tarefas anteriormente exequíveis apenas por humanos.

Importante ressaltar o aspecto de não absoluta flexibilidade das arquiteturas das redes neurais. Conforme Heaton (2015) as redes neurais não são projetadas para resolver todos os tipos de problemas. Cada uma deve ser aplicada a um conjunto específico de situações, tarefas ou problemas a serem resolvidos.

Com a evolução da complexidade dessas arquiteturas, e de seu processo de treinamento, surgem os modelos *deep learning*, esse termo foi utilizado pela primeira vez em meados do ano 2000, justamente para enfatizar o fato dessas redes possuírem maior número de parâmetros treináveis e capacidade de aprendizado mais profundo a partir dos dados usados em seu processo de treinamento. Um dos tipos mais comuns de Deep Neural Network (DNN) são as Convolutional Neural Network (CNN) (KELLEHER, 2019).

2.3 Redes Neurais Convolucionais

As redes neurais convolucionais - CNN são comumente aplicadas na implementação de soluções de problemas que envolvam imagens estáticas ou capturadas em tempo real.

Seu é funcionamento é baseado em como o cortex visual dos animais funcionam, com células respondendo a estímulos que dependem da posição do estímulo no campo visual e células que respondem ao formato do estímulo (traços, pontos, curvas, etc) independentemente da posição. Essas segundo tipo de células é estimulado pelo primeiro grupo numa estrutura hirárquica. Esse comportamento do cortex visual foi descoberto por Hubel e Wiesel em 1965 em experimentos realizados através da medição dos sinais cerebrais de gatos sedados submetidos a padrões de imagens simples (KELLEHER, 2019).

Como observado nos experimentos citados alguns neurônios funcionam extraindo características das imagens, filtrando traços, pontos, bordas, curvas e etc. Esse processo de filtragem de características com invariância espacial é a base da implementação das CNN.

Uma das primeiras das CNN foi desenvolvida em 1980 por Fukushima com avanços significativos promovidos por LeCun, Bottou, Bengio e Haffner em 1998. A partir desse trabalho, ainda em 1998, LeCun e colaboradores propõe a estrutura da rede LeNet5 para classificação de imagens de dígitos e caracteres manuscritos (HEATON, 2015). A estrutura da LeNet-5 é mostrada na Figura 2.4 com todas suas camadas.

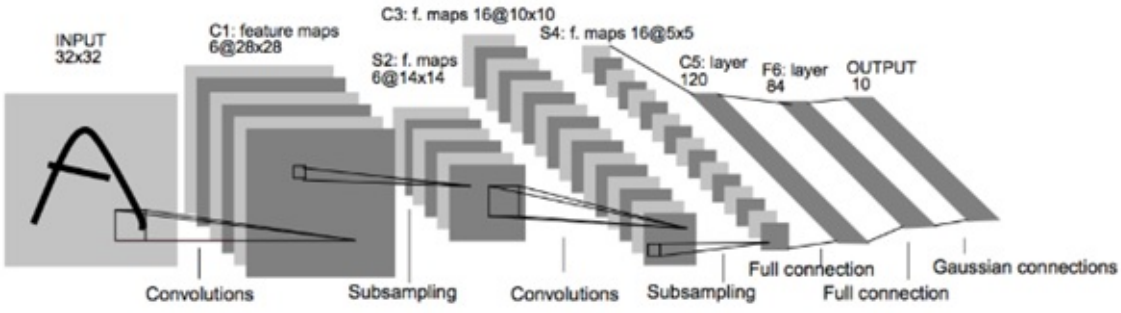


Figura 2.4: Rede Neural Convolucional LeNet-5. Fonte: LECUN, 1998.

Inspirado na estrutura da LeNet5 foi desenvolvido modelo AlexNet por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton. Esse modelo possuia 8 camadas, 65 mil neurônios e 60 milhões de parametros treináveis sendo o vencedor do ImageNet Large Scale Visual Recognition Chalenge (ILSVRC) de 2012, com uma taxa de erro menor que 16% ultrapassando completamente os resultados do estado da arte na classificação de imagens baseado em visão computacional. Esse modelo mostrou todo o potencial das CNN iniciando, de fato, a era moderna do *deep learning* (KOUL *et al.*, 2019).

A estrutura geral de uma DNN pode ser vista na Figura 2.5 e quanto mais profundo for o modelo mais complexas, abstratas e distintas características da imagem serão extraídas possibilitando a identificação de imagens com precisão (HOESER & KUENZER, 2020).

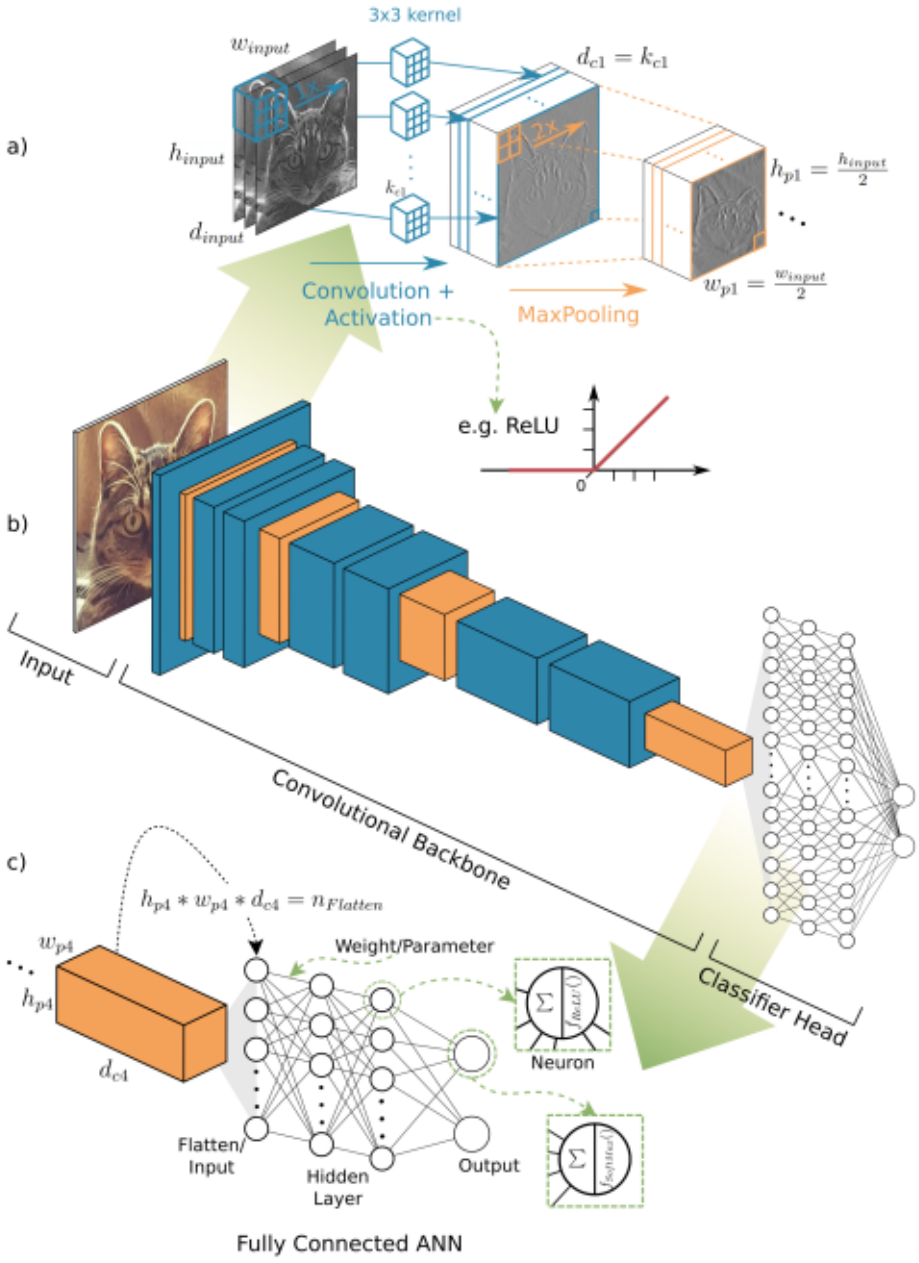


Figura 2.5: *Deep Neural Network. Fonte: HOESER, 2020.*

A DNN pode ser decomposta, para efeito de estudos, em três partes: a entrada, núcleo convolucional e classificador de saída. Os pixels da imagem de entrada passam por uma sequencia de operações de convolução (filtros), funções de ativação e operações de redução dimensional - *polling* para extração de características de alto nível da imagem, A partir dessas características extraídas, a saída composta por uma rede *feedforward*, realiza a classificação indicando a probabilidade de ocorrência da imagem (HOESER & KUENZER, 2020).

No núcleo convolucional serão extraídas características da imagem como bordas, linhas horizontais, linhas verticais, agrupamentos de cores. Essa extração é realizada com

a aplicação de filtros e, segundo Heaton (2015), quanto mais filtros forem aplicados na camada convolucional mais características serão extraídas da imagem, sendo essas características cada vez mais abstratas a medida que a profundidade do modelo aumenta.

Cada filtro ou *kernel* é composto por uma matriz de dimensão f x f. Na Figura 2.5, em a), é indicado um filtro de dimensão 3x3, matriz azul na figura. O filtro será deslizado sobre a imagem em número de passos dependente das dimensões da imagem w x h, dimensões do filtro f x f, com deslocamentos em pixels conforme o hyperparametro passo (s – stride) e o preenchimento (p – padding) (KELLEHER, 2019).

O preenchimento é um mecanismo para adequação do tamanho da imagem as dimensões do filtro, garantindo área na imagem para o filtro realizar a completa varredura em número de passos inteiros. O preenchimento da área não existente na imagem é realizado com zeros.

A relação entre passos executados pelo filtro sobre uma imagem em sua largura (w) e altura (h) são dados pelas relações seguintes, adaptadas de Heaton (2015), essas relações precisam ser números inteiros:

$$relacao_horizontal = (w - f + 2p)/(s + 1) \tag{2.1}$$

$$relação vertical = (h - f + 2p)/(s + 1) \tag{2.2}$$

Entre a entrada da camada convolucional e o mapa de características produzindo pelo processo de convolução com o filtro existirão um número de pesos dados pela função adaptada de Heaton (2015):

$$numeropesos = f.f.numerofiltros \tag{2.3}$$

Os pesos do *kernel* são compartilhados sobre toda a imagem reduzindo as demandas de processamento e desta forma é possível extrair as mesmas características da imagem em posições distintas pois o mesmo filtro foi aplicado em toda a imagem (HEATON, 2015). Os valores dos pesos utilizados pelos filtros são determinados durante o processo de treinamento da DNN não sendo definidos manualmente pelo projetista da rede (KRIZHEVSKY *et al.*, 2012).

Após o processo de convolução com os filtros são produzidos mapas de caraterísticas, tanto quantos os números de filtros. As operações executadas pelos filtros, durante a convolução, são lineares e para inserir não linearidade são usadas funções de ativação Retified Linear Units (Retified Linear Units), estas impedem que os valores dos pesos dos filtros assumam valores negativos durante o processo de treinamento da rede neural(HOESER & KUENZER, 2020). Na Figura 2.6 são mostrados, como exemplo, 96 mapas de características obtidos durante o processo de treinamento da AlexNet.

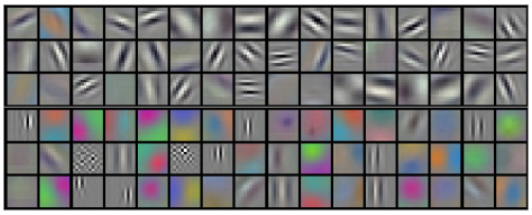


Figura 2.6: Mapas de Características AlexNet. Fonte: KRIZHEVSKY, 2012.

Em seguida os mapas de características da imagem são submetidos ao processo de subamostragem, denominando de *polling*, indicado na Figura 2.5, em a). Nesse processo ocorre a redução da resolução do mapa com o acréscimo da informação semântica contida em seus pixels e redução da relevância espacial associada, em outras palavras, a agrupamento de pixels percebidos pelo mapa de características passa a não depender de sua posição (HOESER & KUENZER, 2020).

A subamostragem é realizada com a aplicação de uma estrutura semelhante a utilizada pelos filtros, com a diferença de não serem utilizados pesos nem a estratégia de preenchimento nas bordas da imagem original. Nesse processo uma vez definida a estrutura do subamostrador f x f, sendo f a dimensão do amostrador, e a distância de seu deslocamento a cada passo, stride (s) o mesmo é deslizado sobre a imagem e o conjunto de pixel é compactado conforme regras definidas previamente. As regras podem ser: pixel com maior valor da amostra (max-pooling), pixel com menor valor da amostra (min-pooling) ou média dos pixels do amostra (avg-pooling).

As dimensões da imagem obtidas pelo processo de sub amostragem são dadas, conforme fórmulas adaptadas de Heaton (2015):

$$largura\_imagem\_compactada = (w - f)/(s + 1) \tag{2.4}$$

$$altura\_imagem\_compactada = (h - f)/(s + 1) \tag{2.5}$$

Desta forma o hiperparametros definidos durante o projeto da DNN relativos à camada de subamostragem são a dimensão do amostrador – f e o deslocamento – s a cada passo de deslizamento. Também durante o desenho da rede é definida a regra de compactação a ser utilizada: max-pooling, min-pooling ou avg-polling.

A camada final da rede neural convolucional será uma rede feedforward com entradas conectadas a última camada do núcleo convolucinal, essa conexão é indicada na Figura 2.5, em c). O número de entradas da camada feedforward será exatamente igual ao número de pixel dos mapas de características da última camada do bloco convolucional. A saída dessa camada e sua função de ativação terão relação intrínseca com o problema a ser resolvido. Em aplicações que envolvam classificação de imagens a função de ativação da saída será softmax, em que cada saída indicará a probabilidade da respectiva classe de

imagem. (HEATON, 2015).

2.4 Edge AI

O paradigma de computação de borda, *Edge*, surgiu da demanda de processamento de dados o mais próximo possível de sua fonte endereçando problemas de velocidade, latência, confiabilidade e custos da transmissão dessas informações para tratamento centralizado. Dependendo do tipo de serviços fornecidos pelo *Edge* este é classificado como: *Near Edge*, *Far Edge*, *Enterprise Edge* e *Device Edge* (MALIK & GUPTA, 2020). Na Figura 2.7 é mostrado, de forma resumida, as principais características e serviços fornecidos por cada tipo.

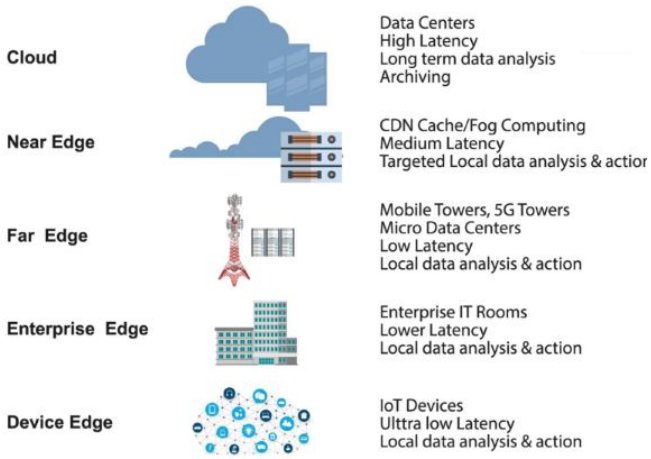


Figura 2.7: Tipos Edge. Fonte: MALIK, 2020.

O processamento distribuído com quantidades massivas de dados sendo produzidas próximos ao mundo físico abre todo um conjunto de novas oportunidades para a implementação de sistemas de AI. Daí a convergência entre os *Device Edge* e os modelos de AI é natural e cria o paradigma de *AI on Edge*, (DENG *et al.*, 2020b). Nesse conceito todo o poder dos modelos DNN são aproveitados principalmente na realização de inferências rápidas, sem perda de tempo por latência de comunicação entre o dispositivo próximo do mundo físico, como por exemplo um instrumento no campo, e o processamento do modelo em um computador central localizado na nuvem ou num *data center* empresarial.

Na implementação do paradigma de *AI on Edge* é necessário levar em conta as limitações de velocidade de processamento, disponibilidade energética e tamanho de memória dos dispositivos *Edge*. Um *road-map* para a implementação de *AI on Edge* é proposto por Deng *et al.* (2020b) e considera a adaptação de modelos existentes convencionais, criação de *framework* de treinamento e inferência específicos além de mecanismos aceleração da execução dos modelos a nível de hardware, detalhes dessa proposta podem ser vistos na Figura 2.8.

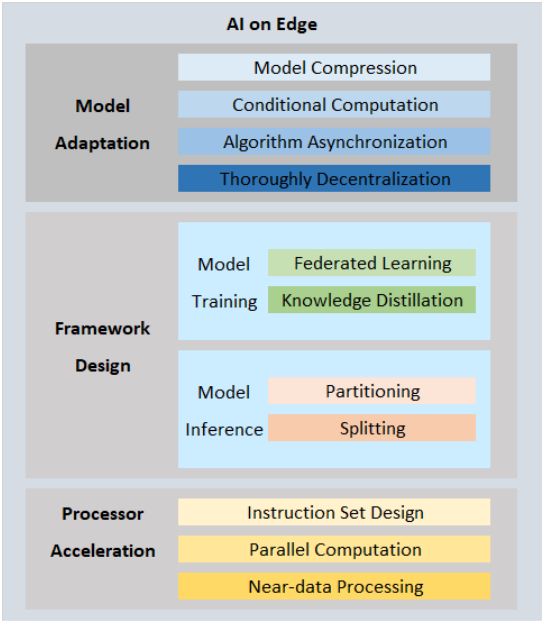


Figura 2.8: Road-map AI on Edge. Fonte: DENG, 2020.

Na adaptação dos modelos são utilizadas abordagens de compressão como *parameter pruning and sharing*, *low-rank factorization* e *knowledge transfer*. Zhang *et al.* (2019) descreve de forma resumida a abordagem de cada uma das técnicas:

- *Parameter pruning and sharing* - elimina parâmetros sem informação relevantes no modelo treinado;
- *Low-rank factorization* - utiliza matrizes para determinar os parâmetros representativos do modelo treinado;
- *Knowledge transfer* - usa um modelo previamente treinado para treinar um modelo compactado na execução das mesmas tarefas.

Aliado a essas técnicas são usados algoritmos para quantização dos pesos da DNN com redução do número de bits necessários para sua representação chegando, no limite, a binarização da rede com a representação de cada peso por um único bit (CHENG *et al.*, 2017), com redução do tamanho de memória utilizado pelo modelo.

Com relação aos *frameworks* de treinamento estes são categorizados em: centralizado com o treinamento sendo executado em computador central ou na nuvem; descentralizado com todo o treinamento da DNN adaptada sendo realizado localmente no *device edge*; e hibrido com o treinamento de um modelo centralizado a partir dos parâmetros obtidos nos treinamentos realizados localmente, o modelo final é propagado para todos os dispositivos (ZHOU *et al.*, 2019);

Para a inferência do modelo os principais *frameworks* são a *edge-based mode* com o modelo sendo executado inteiramente no *device edge* e *edge-cloud mode* com parte do modelo sendo executado localmente e parte na nuvem.

Para a aceleração dos modelos DNN são usadas estratégias como a construção de set de instrução específicos para manipulação destes, uso de processadores paralelos executando os modelos além de memória local, próxima ao núcleo de processamento.

Uma arquitetura genérica de acelerador de redes neurais é mostrada na Figura 2.9, sendo composta por um array de elementos processadores especializados - PE, cada um destes com um buffer de memória para compensar latências do barramento de comunicação. No PE as entradas são multiplicadas pelos pesos da rede, o somatório desses produtos é realizado e as funções de ativação são calculadas. Os PEs permitem a implementação dos processos de convolução, *polling* e *feedforward* das DNN (DENG *et al.*, 2020a).

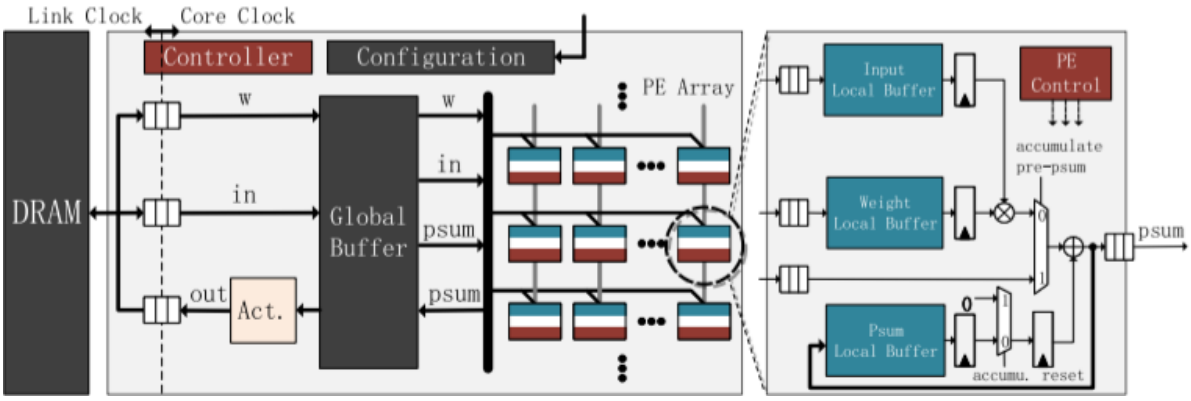


Figura 2.9: Arquitetura Típica de Aceleradores para Redes Neurais. Fonte: DENG, 2020.

3. Trabalhos Relacionados

Muitos problemas de ordem prática podem ser resolvidos com o uso de visão computacional e inteligência artificial, incluindo problemas existentes no ambiente industrial. Neste capítulo estudamos algumas aplicações dessas tecnologias nas detecção de defeitos e anomalias em correias transportadoras, incluindo ainda uma caso de detecção de falhas em ruas pavimentadas, devido a similaridade das características dos danos com os das correias.

Na detecção prematura de defeitos no tapete da correia encontramos Netto (2019) que propõe uma solução utilizando laser e câmera. O feixe de laser é utilizado para realçar as imperfeições contidas no tapete da correia, no lado de sua carga. Uma câmera captura as imagens que passarão por processamento da imagem como segmentação da imagem do laser, transformação unidimensional, extração de curvatura, análise estatística, agrupamento e detecção do defeito. Os defeitos na superfície da correia estarão associados a pontos altos na curvatura extraída pelo sistema e são analisados estatisticamente antes da sinalização de defeitos. Em paralelo, a partir da transformação unidimensional, é criada a reconstrução de uma aproximação 3D da correia. Os testes realizados com o sistema propostos obtiveram precisão média de detecção de 92% para ambientes simulados e 70% para ambientes simulados com ruído.

Em Hou *et al.* (2019) encontramos a proposta de uma solução para detecção de rasgos em correias utilizando imagens e sons produzidos pelo equipamento. A imagem da correia será capturada por uma câmera instalada no lado do retorno da correia, ou seja, lado oposto ao carregamento de material. A imagem será adquirida e processada com aplicação de filtros para redução de ruídos na imagem, binarização por limite adaptativo para criação de contraste entre a imagem do rasgo e o fundo normal do tapete, processamento morfológico de abertura para remoção de pequenos elementos da imagem, extração de domínio conectado para determinação do centro de gravidade do elementos de imagens destacados e conectados e segmentação crescente da região para reforço dos pixels do corte e seu entorno. Após esse processamento os números de pixels brancos são contados e a situação é classificada como correia boa, correia danificada e correia rasgada.

O som produzido pela correia é capturado por uma matriz com 4 microfones instalados na parte de baixo da câmera instalada próxima a correia. O sinal captado pelos microfones é pré-processado com ênfase nas altas frequências e coleta de um frame do som captado. As características do som previamente processado são extraídas com a combinação da técnica de Short-time energy e MFCC – Mel-Frequency Cepstral Coeficients, esses valores combinados são processados por um modelo GMM-UBM (Gausssian Misture Model – Universal Background Model) treinado com som produzidos pela correia em boas condições, com danos menores ou rasgos. O treinamento é realizado considerando a mesma combinação do short-time energy e MFCC. A classificação da situação da correia

é obtida a partir do GMM.

A partir da combinação da detecção do rasgo por imagem e pelo som o sistema toma a decisão de manter a correia rodando, emitir alarmes ou parar o equipamento. Nos experimentos realizados, a partir de um simulador de correias em laboratório, foram obtidos acurácia de detecção da ordem de 86,7%.

No trabalho de Santos *et al.* (2020) encontramos a aplicação de DNN na detecção de sujeira em transportadores de correias. Nessa pesquisa uma DNN foi treinada a partir de um conjunto de 73 fotografias das estruturas dos transportadores em duas situações: transportador com a estrutura limpa e transportador com a estrutura suja, esse conjunto permitiu a criação de duas classes de saída para o modelo, ou seja, transportador limpo ou transportador sujo. As imagens foram pré-processadas com seu redimensionamento e corte aleatório, para adequar ao número de pixels das entradas das redes utilizadas. Como o número de imagens era limitado, foram utilizadas técnicas de *Data Augmentation*. No treinamento da rede neural foram utilizadas técnicas de transferência de conhecimento com redes ResNet18 e VGG16 treinadas inicialmente com o dataset ImageNet. As camadas finais das redes selecionadas foram alteradas de forma a terem apenas duas classes de saída. As redes foram treinadas em 100 épocas fixas. Com a rede ResNet18 foi obtida acurácia de 81,8% e com a VGG16 95,5%, indicando a viabilidade do uso do método de detecção de sujeira em elementos de transportadores de correias.

Situações de detecção de falhas com características visuais semelhantes a proposta em nossa pesquisa nós encontramos no estudo realizado por Majidifard *et al.* (2020) onde redes neurais convolucionais dos frameworks YOLO (You Only Look Once) e Faster R-CNN (Faster Region Convolutional Neural Network) são treinadas a partir de um dataset de imagens de danos em asfalto de ruas e rodovias. O dataset para treinamento foi obtido a partir de imagens do Google Street View utilizando-se a API do Google em conjunto de software de extração de imagens escrito em Python especificamente para essa aplicação. Foram obtidas 7.237 imagens classificadas posteriormente por especialistas como pertencendo a nove categorias de danos ao pavimento. Os dois modelos de rede foram previamente treinados utilizando o Microsoft COCO, dataset este com mais de 2 milhões de objetos classificados em 80 classes. A partir desses pesos iniciais as redes foram treinadas com as 5789 imagens do dataset e os testes de performance foram realizados a partir das 1148 imagens do restante das imagens de danos no pavimento obtidas no trabalho. Os resultados de precisão e F1 (*overall accuracy*) para R-CNN foram 78% e 65% respectivamente. Para YOLO-v2 foram obtidos precisão de 93% e F1 de 84%, mostrando ser esse modelo mais adequado para esse tipo de detecção.

Os resultados que encontramos nos trabalhos estão consolidados na Tabela 3.1. Esses resultados encorajam o prosseguimento do desenvolvimento de nossa proposta de detecção de rasgo de correias com uso de AI on Edge.

Tabela 3.1: Resultados dos Trabalhados Avaliados Durante a Pesquisa

Trabalho	**Resultados**	**Autor**
Visão computacional e laser para detectar rasgo em correia	Precisão de 70% em simulação com ruídos	Neto (2019)
Visão computacional e processamento de ruído da correia por MFCC para detectar rasgo em correia	Acurácia de 86,7% em testes de laboratório com simulador de correia	Cheng (2019)
Modelo DNN VGG16 para detecção de sujeira em estruturas de correias transportadoras	Acurácia de 95,5% a partir imagens coletadas de estruturas de correias aplicadas no campo	Santos (2019)
Detecção de danos em asfalto com uso de YOLO-V2	Precisão de 93% na classificação imagens do dataset Street View da Google	Majidifard (2020

4. Materias e Métodos

A proposta central do nosso trabalho é a implementação de um sistema de detecção de rasgo de correias baseado em captura de imagens em tempo real e reconhecimento desses danos no tapete usando DNN. Tanto a captura da imagem quanto o processamento na rede neural deverão ser realizados em *device edge*, instalado próximo ao equipamento, não dependendo de nenhum tipo de conexão externa para realização da inferência e tomada de decisão para parada do equipamento.

O dispositivo responsável pela execução dos algoritmos de decisão e execução da DNN deverá possuir pequenas dimensões, processamento adequado e baixo custo, permitindo a escalabilidade da solução.

A escolha do *device edge*, direcionará a seleção do modelo de inteligência artificial mais adequado, considerando o problema a ser resolvido e as restrições, já citadas, além da seleção do *framework* para treinamento do modelo, sua compactação e compilação para uso no hardware selecionado.

Diante deste cenário o presente capítulo aborda em detalhes o *device edge*, *framework* e modelo selecionados para o desenvolvimento do projeto. O detalhamento para a elaboração do *dataset*, construção do protótipo e testes de campo também serão abordados.

4.1 Device Edge para Deep Learning

Considerando as premissas e diretrizes definidas para o trabalho foram estudadas as plataformas disponíveis no mercado brasileiro, sendo identificadas como possibilidades o Raspberry PI, Jetson Nano Nvidia, SiPEED MAiX. Os principais parâmetros avaliados para cada uma das plataformas são indicados na Tabela 4.1. Dentre as plataformas foi selecionada a SiPEED devido ao custo, dimensões reduzidas, e acelerador de hardware para para uso com CNN.

Tabela 4.1: Comparativo Entre *Device Edge*

	Raspberry	**Jeston Nvidia Nano**	**SiPEED MAiX**
Processador	ARM Cortex- A53	Quad-core a57	K210 - RISC-V - 64bits
Velocidade	1,2 GHz	1,43 GHz	400 MHz
RAM	1 GB	4 GB	8 MB
EPROM	Somente cartão SD	16 GB	128 MB
Recursos AI	NA	GPU	KPU
SO/Linguagem	Raspian / Python	Ubuntu / SDK JetPack	uPython
GPIO	40	40	48
Dimensões	85x56x17	100x80x29	52x39x10
Custo	R$ 380,00	R$ 1500,00	R$ 230,00

Nas atividades de pesquisa do trabalho nós utilizamos as placas de desenvolvimento SiPPED MAiX BIT e MAiX DOCK. O kit SiPEED é fornecido com a placa de desenvolvimento, um display 2,4 polegadas e uma câmera VGA de 3M pixels e distância focal de 3,6mm. Esses componentes são mostrados na Figura 4.1.

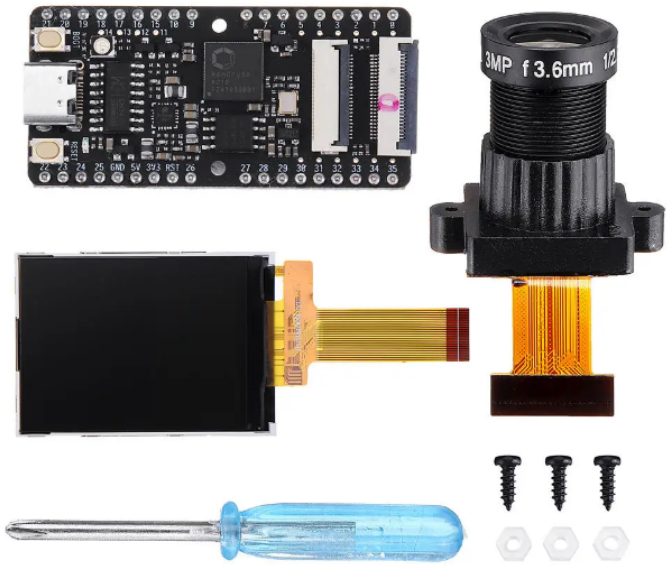


Figura 4.1: Kit Desenvolvimento SiPEED MAiX BIT. Fonte: SiPEED, 2019.

Essas placas de desenvolvimento utilizam como base o Kendryte K210, um System on Chip (SoC) voltado para aplicações de visão e audição computacionais além de inteligência artificial. Seu núcleo de processamento é composto por 2 processadores de 64 bits com arquitetura RISC-V operando a 400MHz, uma Knowledge Processing Unit (KPU) para aceleração de CNN, uma Audio Processig Unit (APU) para processamento de áudio, um First Fourier Transform (FFT) para processamento em hardware de transformada de Fourier, uma SRAM de 8 MB, *Timer*, uma Universal Asynchronous Receiver/Transmitter (UART), conjunto com 40 pinos de entrada e saída de uma General Purpose Input/Output (GPIO) entre outros dispositivos de comunicação e interface (SEEED, 2020). O diagrama em blocos do K210 e mostrado na Figura 4.2), o chip foi projetado e fabricado pela companhia chinesa CANAAN sendo um componente relativamente novo no mercado, a primeira revisão do documento de referência encontrado por nós foi de setembro de 2018 (CANAAN, 2019).

Dos dispositivos internos do K210 os mais importantes para o desenvolvimento do presente trabalho são o seu acelerador de CNN, a KPU, e a interface de vídeo, o Digital Video Port (DVP). Ambos os módulos são apresentados a seguir.

Conforme definido por CANAAN (2019), no datasheet do K210, a KPU é um processador de uso geral contento implementações nativas para *batch normalization*, funções de ativação e operações de *pooling*. A arquitetura da KPU é apresentada na Figura 4.3.

Outras características importantes da KPU são:

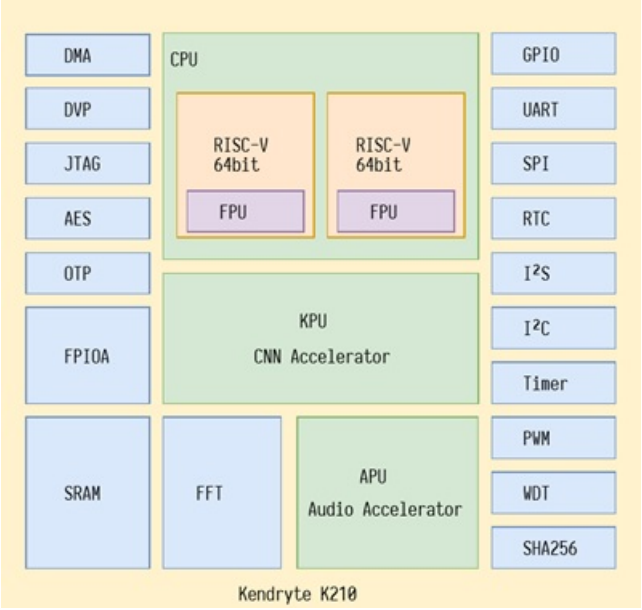


Figura 4.2: Diagrama em Blocos do K210 Kendryte. Fonte: CANAAN, 2019.

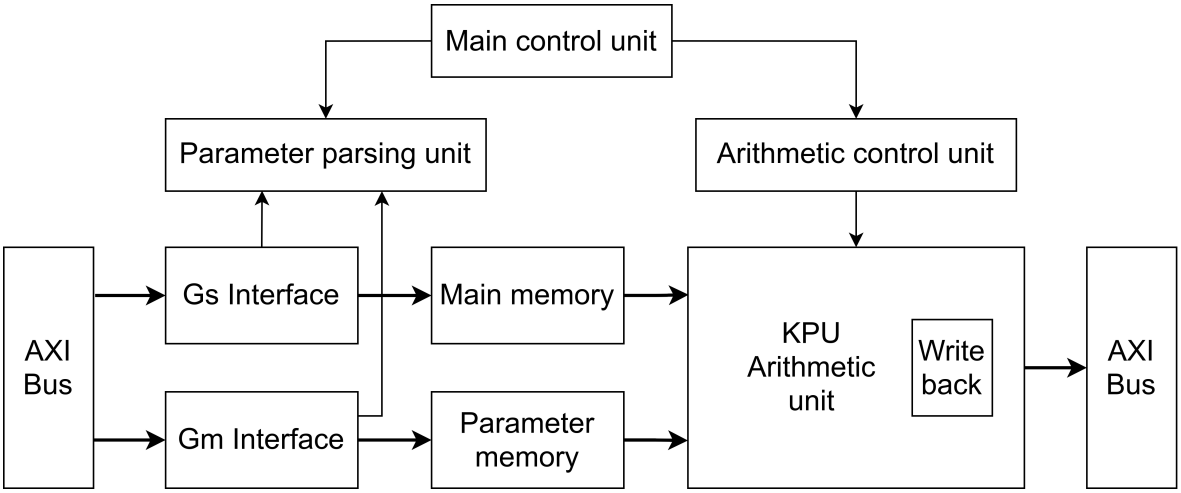


Figura 4.3: Arquitetura da KPU do K210 Kendryte. Fonte: CANAAN, 2019.

- Suporte a 2 núcleos convolucionais de dimensões 1x1 e 3x3;
- Suporte a qualquer tipo de função de ativação;
- Para operações em tempo real o limite de tamanho do modelo é de 5,9MB;
- Para operações sem requisitos de tempo real o limite é dado pelo tamanho da memória flash menos o tamanho do programa em execução;
- Não existe limite de número de camadas da rede nem de seus parâmetros configuráveis de forma individual.

A linguagem de programação, utilizada no desenvolvimento das aplicações, nas placas da SiPEED é o MicroPython, implementação do Python 3.4 "padrão" para execução em plataformas baseadas em microcontroladores (GEORGE & SOKOLOVSKY, 2020). A versão atual do MicroPython é a 1.12.

Para o uso dos recursos da KPU foram implementados métodos em MicroPython pelo fabricante do K210 Kendryte numa biblioteca denominada KPU, com destaque para os principais listados na Tabela 4.2.

Tabela 4.2: Métodos MicroPython KPU – K210 Kendryte.

Método	**Descrição**
import KPU as <kpu>	Importa a biblioteca KPU e a instancia no objeto <kpu>
<task>=<kpu>. load(<local>)	Carrega o modelo de <local>para o objeto <task>. O <local>pode ser no cartão SD ou na memória FLASH.
<fmap>= <kpu>.forward (<task>, <img>,<layer>)	Calcula os features da imagem <img>para a rede neural <task>na camada desejada <layer>e as atribui ao objeto <fmap>
<kpu>.fmap_free(<fmap>)	Libera o objeto <fmap>
<info>= <kpu>.netinfo(<task>)	Lista as informações das camadas do modelo
<kpu>.set_outputs(<task>, 0, 1, 1, <x>)	Configura o número de saidas <x>do modelo conforme a configuração utilizada no treinamento, números de classes.
<kpu>.deinit(<task>)	Libera a memória ocupada pelo objeto <task>
<kpu>.init_yolo2(<task>,x,y,<anchor>)	Inicializa a KPU para utilização de Yolo-V2
<code>= kpu.run_yolo2(<task>,<img>)	Executa o detector yolo na <img>retornando o lista de objetos <code>com a identificação das classes e a indicação do box da classe na imagem

O Digital Video Port (DVP) consiste na interface de vídeo com suporte ao protocolo Serial Camera Control Bus (SCCB). A resolução máxima admitida no DVP é de 640x480 com suporte aos formatos de entrada de cor YUV422 e RGB565, com 16 bits de resolução de cor. A imagem recebida pelo DVP pode ser direcionada diretamente tanto para a KPU quanto para o display da placa SiPEED.

No caso específico do kit de desenvolvimento usado a câmera fornecida é a modelo OV2640, com um sensor CMOS UXGA (1632x1232). A câmera é completamente configurável pelo barramento SCCB e, no caso de nossa pesquisa, para garantir a compatibilidade com o DVP e KPU, foi utilizada a resolução Quarter Video Graphics Array (QVGA) de 320x240 e formato de cor RGB565.

Para o uso das funcionalidades do DVP, da mesma forma que na KPU, foram implementados métodos para acesso e controle das imagens em MicroPython. Os métodos integram a biblioteca denominada sensor e os principais são indicados na Tabela 4.3.

Tabela 4.3: Métodos MicroPython DVP – K210 Kendryte.

Método	Descrição
import sensor	Importa a biblioteca sensor para tratamento de imagens pelo DVP
sensor.reset()	Inicializa a camera e o DVP
sensor.set_pixformat(sensor.<RGB565>)	Configura o modo de cor do DVP entre RGB565 ou YUV422
sensor.set_framesize(sensor.QVGA)	Configura as dimensões do frame entre QVGA e VGA
sensor.set_vflip(<OP>)	Inverte a imagem verticalmente. <OP>=0 sem inversão, <OP>=1 com inversão
sensor.run(<OP>)	Controla a captura de imagens da câmera. <OP>=0 câmera parada, <OP>=1 camera funcionando
<img>= sensor.snapshot()	Carrega a imagem capturada pela câmera no objeto <img>
sensor.skip_frames([n, time])	Salta um certo número [n] de frames ou por um intervalo de tempo [time]
sensor.set_windowing((<x>,<y>))	Configura a janela do sensor para as dimensões indicadas por <x>e <y>

4.2 Modelo Deep Learnig para Detecção

4.2.1 Seleção do Modelo

A KPU do Kendryte K210 do SiPEED utiliza o formato proprietário .kmodel para a execução do modelo e sua inferência. O modelo DNN convencional será compilado pelo nncase, um compilador para *AI accelerators* desenvolvido pelo fabricante do K210 (SUNNYCASE, 2020). De forma geral os modelos compiláveis indicados pelo fabricante são *MobileNet V1, YOLO V1 e TensorFlow Lite.*

Para a nossa aplicação foi selecionado o modelo *MobileNet* configurado como classificador, nesse caso o modelo infere, na imagem analisada, a probabilidade de existência das imagens que foi treinado a identificar, cada uma dessas imagens associada a uma classe.

A nossa escolha pelo uso da *MobileNet* foi fundamentada no fato desse modelo ser eficiente em termos de reconhecimento de detalhes finos na imagem, ter boa acurácia e baixo custo computacional, como demonstrado no trabalho de Howard *et al.* (2017). A comparação entre os diferentes versões da *MobileNet* e o modelo *Inception v3*, considerado o estado da arte na identificação de detalhes finos, pode ser visto na Tabela 4.4, nesse comparativo foi utilizado o *Standford Dogs dataset*, composto por 20.580 imagens agrupadas em 120 classes, nesse caso raças de cachorro (KHOSLA *et al.*, 2011).

A arquitetura da *MobileNet* é composta por 28 camadas. Na *MobileNet* os filtros con-

Tabela 4.4: Comparativo Performance Modelos com *Standford Dog Dataset*. Fonte: Howard *et al.* (2017)

Modelo	**Acurácia**	**Parâmetros (milhões)**
Inception - V3	84%	23,2
1,0 MobileNet-224	83,3%	3,3
0,75 MobileNet-224	81,9%	1,9
1,0 MobileNet-192	81,9%	3,3
0,75 MobileNet-192	80,5%	1,9

volucionais convencionais foram substituídos por duas camadas, uma chamada denominada *depthwise convolution* e outra *pointwise* com efeito de redução drástica do tamanho do modelo e sua demanda computacional (HOWARD *et al.*, 2017).

Na Tabela 4.5 é mostrada a estrutura do modelo. Importante notar que a camada de saída é composta por um classificador com função de ativação *softmax*, com indicação percentual da probabilidade de certeza da classificação realizada pelo modelo, e número de saídas igual ao número de classes a serem identificadas. Originalmente o modelo possui 1000 saídas, no caso de nosso trabalho o modelo será modificado para duas saídas, classes, uma indicando a situação normal da correia e outra indicando a situação de rasgo.

No processo de treinamento da *MobileNet* utilizaremos a técnica de *transfer learning*, aproveitando o treinamento prévio da DNN realizado com o *dataset ImageNet*, para as camadas convolucionais. De fato o treinamento do nosso modelo será realizado nas camadas *feed forward* da saída.

4.2.2 Treinamento do Modelo e Conversão para Edge AI

O ecosistema *Edge AI* é bastante fragmentado e cada dispositivo requer que os modelos DNN convencionais, após seu treinamento, sejam convertidos para os formatos executáveis por seu *hardware*. Normalmente cada fabricante desenvolve sua ferramenta de conversão do modelo treinado, em *frameworks* como Keras, Darknet, TensorFlow para o formato apropriado. Por exemplo, o SiPEED usa o compilador nncase para converter o modelo para o formato usado na KPU do K210, os chips Nvidia Movidus usam o *Open VINO toolkit* o Google Edge TPU usam um compilador proprietário (KLIPPEL *et al.*, 2021).

Para a conversão do modelo *MobileNet* para uso com a KPU do K210 do SiPEED usamos o *framework aXeleRate*. Esse sistema é baseado na plataforma Keras-TensorFlow sendo composto por um conjunto de scripts escritos em python, no formato de um bloco de notas jupyter para execução na plataforma Google Collaboratory (MASLOW, 2020).

O *aXeleRate* possui estrutura modular permitindo seu uso com modelos como MobileNet, NASNetMobile, ResNet e Yolo, em suas configurações como classificadores, detetores ou segmentadores. A principal funcionalidade do *aXeleRate* é a automação de todo o pro-

Tabela 4.5: Arquitetura *MobileNet*. Fonte: Howard *et al.* (2017)

Type / Stride	**Filter Shape**	**Input Size**
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 128	56 x 56 x 128
Conv dw / s2	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x 28 x 128
Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
5× Conv dw / s1	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 512	14 x 14 x 512
Conv dw / s2	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv / s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Avg Pool / s1	Pool 7 x 7	7 x 7 x 1024
FC / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Classifier	1 x 1 x 1000

cesso de treinamento, conversão e compilação do modelo para uso direto com *Device Edge*, no caso de nosso trabalho o formato é o .kmodel, executável pela KPU do K210.

O fluxo de treinamento, conversão do modelo, compilação e uso no SiPEED podem ser vistos na Figura 4.4 sendo composto pelos seguintes passos, conforme Klippel *et al.* (2020):

1. *Dataset* de imagens é armazenado no Google Drive para treinamento no Keras;
2. Após o treinamento o modelo treinado é entregue no formato .h5;
3. O modelo no formato .h5 é compactado pelo pelo TensorFlow sendo entregue no formato .tflite;
4. O modelo .tflite é compilado pelo nncase sendo convertido para o formato .kmodel;
5. A estrutura no formato .kmodel é gravada no cartão SD Card do SiPEED;
6. O modelo .kmodel é executado pela KPU do SiPEED.

Os parâmetros para o treinamento são passados para uma função do *aXeleRate* através de um dicionário de dados com informações de tipo de rede, arquitetura, tamanho da

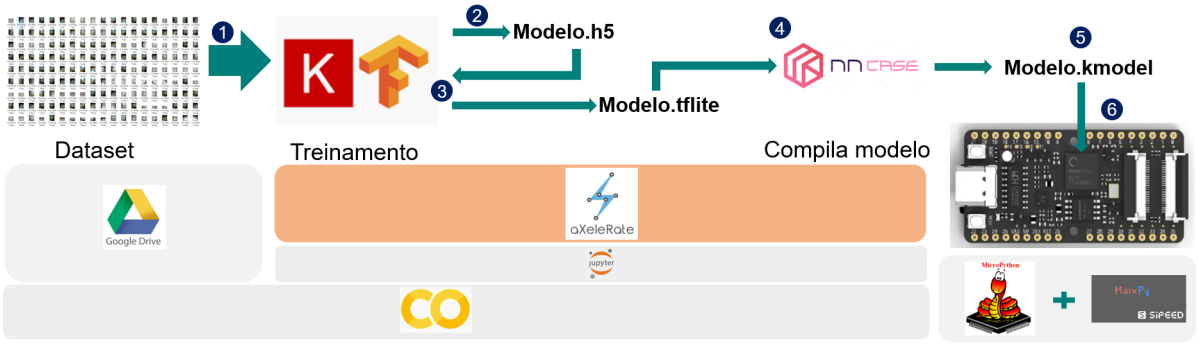


Figura 4.4: Execução Treinamento e Compilação *aXeleRate*. Fonte: KLIPPEL, 2020.

entrada, números de neurônios das camadas de saída, informações de pesos, número de épocas, endereço da pasta de treinamento, números de repetições, endereço da pasta de validação, métricas de validação, *batch size*, taxa de aprendizado, endereço para salvar o modelo, opção de *data augmentation*, formato de saída do modelo treinado e convertido. Para avaliação de performance os parâmetros de configuração da função são os mesmos indicados anteriormente, incluindo apenas o caminho para a o modelo treinado.

Após o processo de treinamento o modelo é disponibilizado numa pasta da máquina virtual do Google Collaboratoy conforme a indicação do dicionário de configuração, normalmente o nome da pasta é *"classifier"*. O *aXeleRate* retorna também com informações da acurácia do treinamento e dos testes de validação.

4.3 Metodologia

4.3.1 *Dataset* para Treinamento

O nosso modelo DNN operará como classificador possuindo duas saídas: uma indicando situação de correia normal e outra condições de correia com rasgo. Para o treinamento do modelo será utilizado um *dataset* com imagens de correias em condição normal e com rasgo. A captura das imagens foi realizada com o protótipo construído no item 4.3.2 para garantirmos a mesma resolução e amplitude de cores, eliminando etapas de compactação nas imagens, exemplos dessas imagens capturadas podem ser vistos na Figura 4.5.

Para a captura das imagens com rasgo utilizamos a estratégia de confecção destes pela equipe de manutenção de correias. Esses rasgos foram simulados por profissionais experientes e reproduzem, da melhor forma possível, as situações encontradas nesses eventos, a Figura 4.6 mostra uma dessas execuções. As imagens para a situação normal das correias foram obtidas de seções próximas aos locais de corte, com o objetivo de garantir a uniformidade do contexto da cena avaliada. Para alguns testes também pintamos o tapete da correia com tinta preta para a simulação da situação de rasgos como pode ser visto na

(a) Com Rasgo. (b) Sem Rasgo.

Figura 4.5: Imagem Capturadas pelo SiPEED em 224x224. Fonte: KLIPPEL, 2020.

Figura 4.7.

Figura 4.6: Simulação de Rasgo no Tapete da Correia. Fonte: KLIPPEL, 2020.

O *dataset* evoluiu a medida que os testes em campo avançaram, partindo de imagens estáticas, Figura 4.8 até imagens com a correia em operação, Figura 4.9. Durante o desenvolvimento nós consideramos o maior número possível de situações incluindo variações de iluminação, inserção de elementos estruturais da correia nas imagens, mudança do angulo de captura da foto, entre outras avaliadas por nós nos testes de campo.

Algumas imagens do *dataset* podem ser vistas no Apêndice A.

4.3.2 Construção do Protótipo

Para os testes de campo nós construímos quatro protótipos, partindo de uma versão bem simples e chegando até uma solução para instalação permanente. As duas primeiras

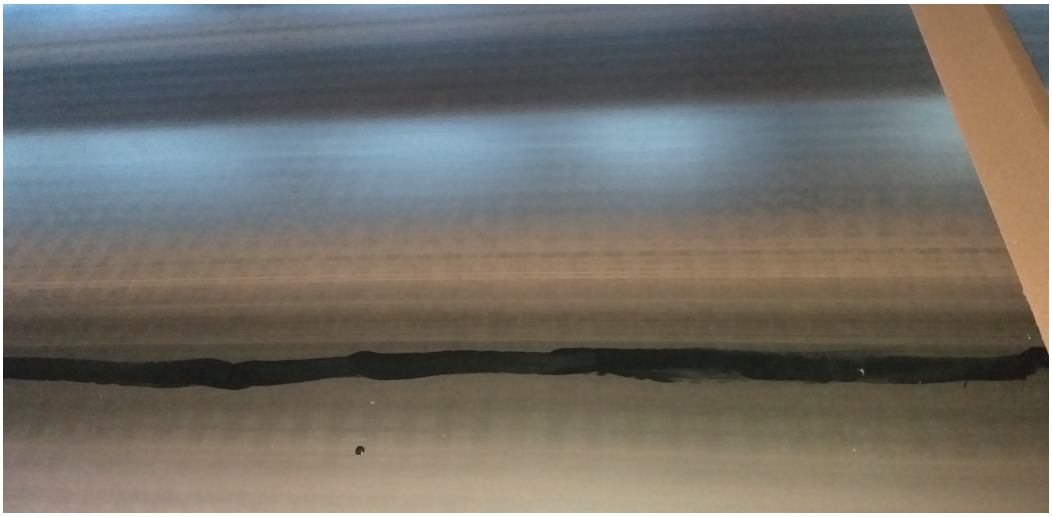
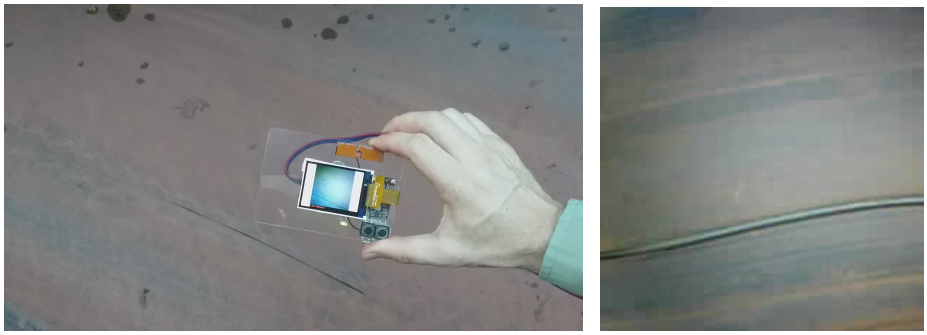


Figura 4.7: Pintura na Correia Simulando Rasgo. Fonte: KLIPPEL, 2020.



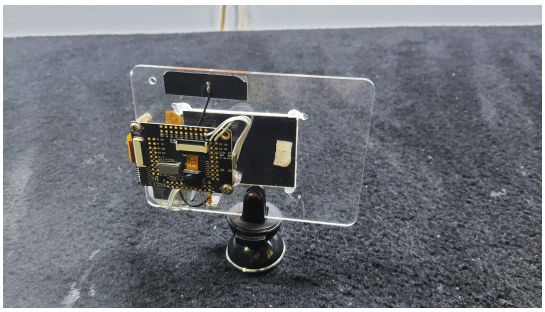
(a) Captura Imagem (b) Imagem 224x224.

Figura 4.8: Imagem Estáticas Capturadas pelo SiPEED em 224x224. Fonte: KLIPPEL, 2020.

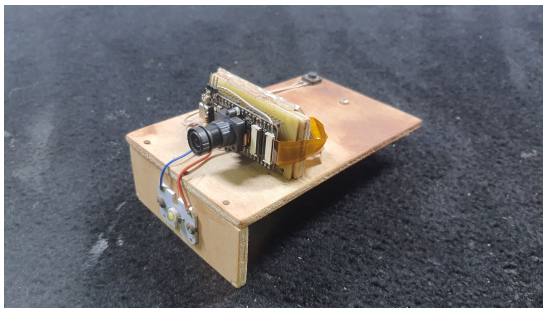
(a) Captura Imagem (b) Imagem 224x224.

Figura 4.9: Imagem Dinâmicas Capturadas pelo SiPEED em 224x224. Fonte: KLIPPEL, 2020.

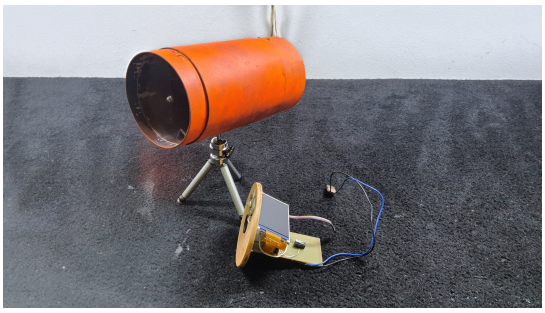
versões foram construídas sem proteção contra intempéries, a terceira foi construída com proteção contra poeira e a quarta com proteção contra poeira e umidade. As quatro versões são vistas na Figura 4.10.



(a) Primeira.



(b) Segunda.



(c) Terceira.

(d) Quarta.

Figura 4.10: Versões Protótipo Construídas no Trabalho. Fonte: KLIPPEL, 2020.

Para as três primeiras versões nós usamos uma banco de bateria de 5V @ 10000mA para alimentação elétrica. A última versão foi construída de forma completa com fonte integrada de 90 à 250 VAC e interface a relé para conexão ao sistema de controle da correia transportadora, o diagrama da eletrônica da última versão é mostrado na Figura 4.11.

Para os testes de campo foram desenvolvidos três programas em MicroPython, linguagem de programação baseada no Python 3 e otimizada para aplicações em microcontroladores e *embedded systems* (TOLLERVEY, 2017). A Integrated Development Environment IDE utilizada por nós foi a MaixPy IDE 0.2.4 desenvolvida pela SiPEED sendo a licença gratuita e nos moldes de General Public License (GNU). O *firmware* utilizado no SiPEED foi o maixpy_v0.5.0_33_gfcd6d8a_minimum_with_ide_support.bin.

O primeiro programa (Programa 1) nós construímos exclusivamente para transformar o SiPEED em uma máquina fotográfica comum, com disparos manuais ou temporizados. O *script* utilizou os métodos de configuração do sensor, incluindo os associados a redução do tamanho da imagem a resolução de 224x224 pixels e o método de captura e armazenamento da imagem em um objeto que era armazenado, em seguida, no cartão SD do SiPEED. Este programa foi utilizado, principalmente, na elaboração dos primeiros *datasets* com imagens estáticas.

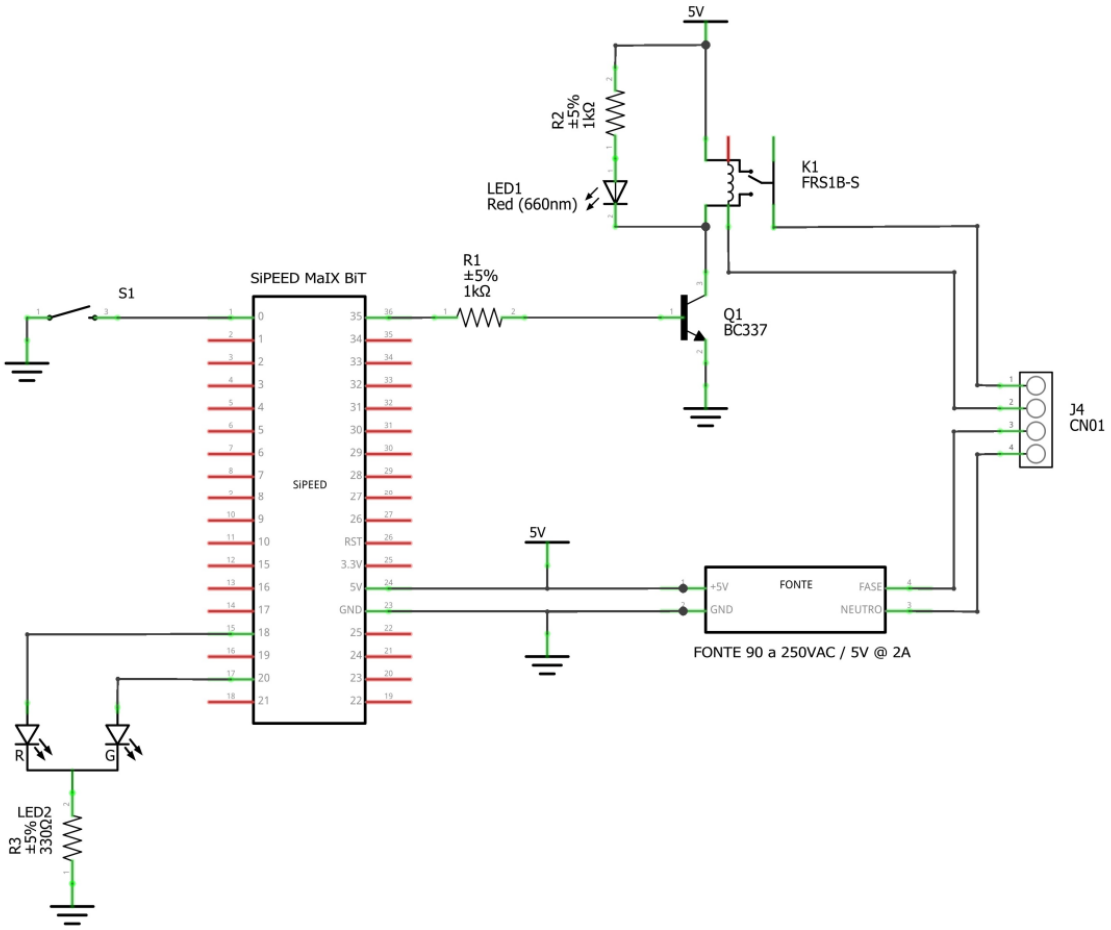


Figura 4.11: Esquema Eletrônico Protótipo Final. Fonte: KLIPPEL, 2021.

No segundo programa (Programa 2) nós utilizamos os recursos da KPU do K210 através de seus respectivos métodos. Nesse caso o objetivo do *script* era capturar imagens, na maior velocidade possível, e realizar a classificação das mesmas através do modelo executado na KPU. No caso da imagem ser classificada como contendo rasgo esta era armazenada numa pasta denominada "evento", em seguida um arquivo .txt recebia uma entrada contendo o nome do arquivo de imagem e o valor de sua classificação, entre 0 e 1. Para as imagens classificadas como não contendo rasgo essas eram armazenadas e registradas da mesma forma, a cada 10 segundos, em outra pasta e arquivo .txt.

É importante ressaltar a necessidade da KPU ser configurada como o mesmo número de saídas do modelo carregado para execução, no momento de sua instanciação no programa, isso não é feito de forma automática e existe a necessidade de uso de método específico para tal. No nosso caso o modelo *MobileNet* foi treinado como classificador possuindo duas saídas, configuração esta atribuída a KPU.

O terceiro programa (Programa 3) foi desenvolvido para ser usado na comparação dos resultados de validação do processo de treinamento no *aXeleRate* com a execução da mesma validação no SiPEED, utilizamos ele também para verificar a performance do modelo a medida que o *dataset* evoluiu. A funcionalidade desse *script* é semelhante ao

Programa 2, com a diferença das imagens classificadas terem origem em uma pasta do cartão SD, imagens com rasgo e sem rasgo, outro ponto é a não execução do processo de captura de imagens de forma temporizada.

Os códigos fontes finais, de cada uma dos programas, podem ser vistos no Apêndice B. Os principais pontos de atenção estão comentados nesses códigos.

4.3.3 Testes de Campo

Para o desenvolvimento do trabalho nós propusemos um conjunto de testes com objetivos específicos, indicadores de performance e matrizes de confusão. Os testes e objetivos são indicados a seguir:

1. Testes do Processo de Treinamento do Modelo e Conversão para Edge Device - Validar o processo de treinamento da DNN *MobileNet*, sua compactação e compilação para uso no SiPEED;

2. Teste Imagens Estáticas em Campo - Validar a captura de imagens, processo de treinamento e detecção da DNN para imagens de rasgo estáticas e testar a aplicabilidade do modelo assim treinado para detecção de rasgos reais;

3. Teste Imagens Dinâmicas em Campo - Validar a captura de imagens, processo de treinamento e detecção da DNN para imagens de rasgo dinâmicas e testar a aplicabilidade do modelo assim treinado para detecção de rasgos reais.

Para a coleta de imagens de correias e realização dos testes propostos nós realizamos visitas técnicas a Usina de Beneficiamento de Minério de Ferro de Carajás, PA, essas visitas e atividades sempre foram realizadas com o suporte e acompanhamento das equipes de manutenção e operação local, visando garantir a segurança das pessoas e minimizar possíveis impactos no processo produtivo.

Para a avaliação da performance dos testes nós utilizamos os indicadores acurácia, precisão, *recall* e F1. Esses indicadores são calculados a partir das detecções corretas e incorretas de cada uma das classes. As formulas de cálculo dos indicadores são definidas nas Equações (4.1), (4.2), (4.3) e (4.4) sendo TP o número de detecções de rasgo corretas, FP número de detecções de rasgo incorretas, TN número de detecções de situação normal corretas (ausência de rasgo) e FN número de detecções normal incorretas.

$$acuracia = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

$$precisao = \frac{TP}{TP + FP} \tag{4.2}$$

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 = \frac{2 * precisao * recall}{precisao + recall} \tag{4.4}$$

De forma geral a precisão avalia o quão bem o modelo está identificado a classe de interesse, no caso as situações de rasgo, em relação ao total de situações de rasgo existentes, ou seja o quão preciso ele é. A acurácia mede o quanto o modelo acertou tanto para as predições de situações de presença da classe de interesse quanto para as outras classes. Para o indicador *recall* este avalia a performance do modelo com relação a detecção de falsos negativos, no nosso caso situações em que a correia estava sem rasgo mas o sistema detectou com rasgo. O F1 ou *f-score* trata do balanço entre as métricas precisão e *recall* (SHUNG, 2018).

Outra ferramenta utilizada por nós foi a matriz de confusão, sendo esta uma tabela para visualização da performance da predição do modelo na classificação entre duas ou mais classes (MOHAJON, 2020). Os parâmetros de entrada da matriz são os mesmos utilizados nos cálculos dos indicadores de performance adotados. Na Figura 4.12 é mostrada o modelo de matriz de confusão utilizada nos testes.

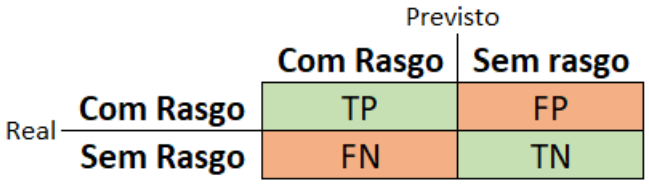


Figura 4.12: Matriz de Confusão Utilizada nas Avaliações. Fonte: KLIPPEL, 2020.

No capítulo seguinte serão detalhados a execução dos treinamentos e de todos os experimentos e testes realizados no desenvolvimento de nosso trabalho.

5. Resultados e Discussões

Neste capítulo nós apresentamos os resultados do processo de treinamento e compactação da *MobileNet*, pelo *framework aXeleRate* para uso no SiPEED. Também será abordada a performance da DNN sendo executada no *device edge* para a detecção de rasgos de correia, nos testes propostos na metodologia.

5.1 Testes do Processo de Treinamento do Modelo e Conversão para Edge Device

Para a validação do processo de treinamento e conversão do modelo DNN pelo *aXeleRate* para uso no SiPEED, nós realizamos um teste em laboratório. Nesse teste utilizamos uma bancada forrada com carpete de cor preta, sobre esse carpete colocamos um barbante de cor escura para simular o rasgo. Assim obtivemos duas condições, uma simulando o rasgo e outra simulando a condição normal. Exemplos das imagens usadas nos testes são vistas na Figura 5.1

(a) Rasgo Simulado (b) Normal Simulado

Figura 5.1: Imagens para Validação do Processo de Treinamento e Conversão do Modelo. Fonte: KLIPPEL, 2020.

Para o processo de treinamento foram usadas 50 imagens com o barbante e 50 sem barbante. Para a validação foram usadas 5 imagens normais e 5 imagens com a simulação do rasgo. O treinamento da *MobileNet* foi realizado em 10 épocas pelo *aXeleRate* rodando no Google Colab com tempo total de 1 minuto. A configuração completa do modelo e processo de treinamento são parametrizados no *aXeleRate* através de seu dicionário de dados carregado no objeto *config* mostrado a seguir:

```
1 config = {
2             "model": {
3                         "type": "Classifier",
4                         "architecture": "MobileNet7_5",
```

```
5                       "input_size": 224,
6                       "fully-connected": [100,50],
7                       "labels": [],
8                       "dropout": 0.5
9                       },
10              "weights": {
11                       "full": ,
12                       "backend": "imagenet",
13                       "save_bottleneck": False
14                       },
15              "train": {
16                       "actual_epoch": 10,
17                       "train_image_folder": "drive/My Drive/Modelo_Correia_00_-
Simulado/Treinamento",
18                       "train_times": 4,
19                       "valid_image_folder": "drive/My Drive/Modelo_Correia_00_-
Simulado/Validacao",
20                       "valid_times": 4,
21                       "valid_metric": "val_accuracy",
22                       "batch_size": 4,
23                       "learning_rate": 1e-3,
24                       "saved_folder": "classifier",
25                       "first_trainable_layer": ,
26                       "augumentation": True
27                       },
28              "converter": {
29                       "type": ["k210","tflite"]
30                       }
31      }
```

Os principais parâmetros configurados para o modelo são seu tipo, a arquitetura utilizada, tamanho da entrada, estrutura da camada de saída e o dropout. Esses valores são os indicados nas linhas de 3 a 8 do dicionário de configuração. Durante o processo de treinamento utilizamos a técnica de *transfer learning*, para evitarmos treinar o modelo do zero. No nosso caso utilizamos a *MobileNet* treinada a partir do *dataset ImageNet*. Esse parâmetro é indicado na linha 12. As configurações do processo de treinamento são realizadas nas linhas de 16 a 26 incluindo as pastas de origem das imagens de treinamento e validação, números de exposições das imagens no processo de treinamento, métrica de validação do processo de treinamento, uso ou não de data *augumentation* além dos hyperparametros número de épocas e *batch size*. Nesse ponto também é definida qual a

primeira camada a ser treinada sendo adotada por nós a estratégia de somente treinarmos as camadas *fully-connected*, opção *default do aXeleRate*, linha 25. O último parâmetro configurado para o *aXeleRate* é o processo de conversão, partindo do formato tflite e sendo compilado para .kmodel, formato utilizado pelo K210 do SiPEED. Essa configuração é feita na linha 29.

Os resultados de acurácia do processo de treinamento podem ser vistos no gráfico da Figura 5.2.

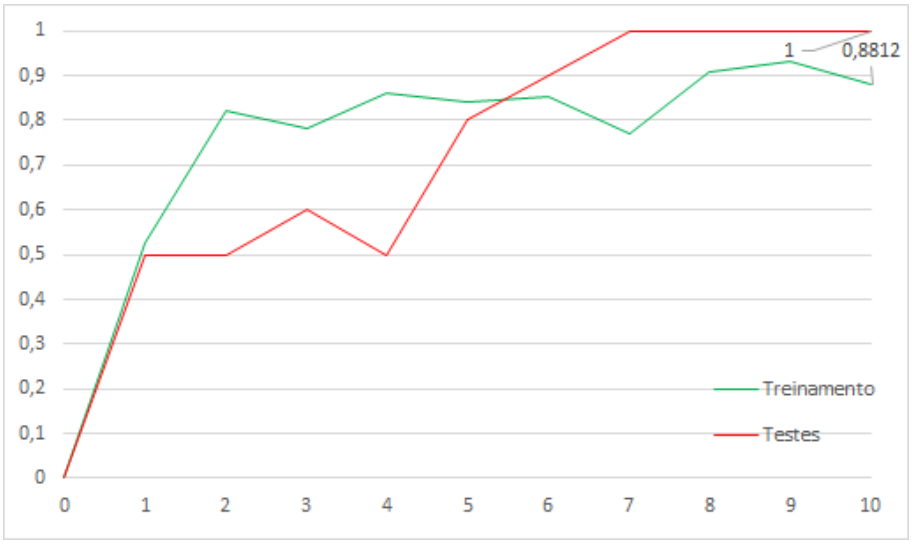


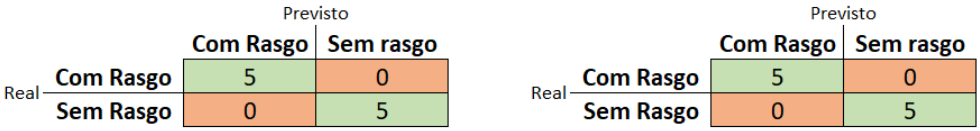
Figura 5.2: Gráfico Acurácia Treinamento no *aXeleRate*. Fonte: KLIPPEL, 2020.

Após o treinamento e compilação do modelo, as mesmas imagens de validação usadas no *aXeleRate* foram classificadas pela KPU do SiPEED, usando o Programa 3, com os resultados comparativos para os indicadores apresentados na Tabela 5.1. As matrizes de confusão são vistas na Figura 5.3 e exemplos das mesmas imagens de validação classificadas pelo *aXeleRate* e SiPEED são mostradas nas Figuras 5.4 e 5.5

Tabela 5.1: Comparativo Validação *aXeleRate* e KPU do SiPEED. Fonte: KLIPPEL, 2020.

Indicador	**Validação Google Colab**	**Validação KPU SiPEED**
Precisão	1	1
Recall	1	1
F1	1	1

Os resultados obtidos nesses testes mostraram a viabilidade do processo de treinamento da *MobileNet* pelo *aXeleRate* e Google Colab bem como sua conversão e compilação para uso na KPU do SiPEED. Para o número e tipos de imagens utilizadas, no teste, não foram detectadas perdas significativas nos indicadores de performance adotados por nós. Outro ponto a se ressaltar é que a partir da sétima época a acurácia dos testes permaneceu constante e os pesos do modelo não foram atualizados pelo *framework* de treinamento



(a) aXeleRate (b) KPU SiPEED

Figura 5.3: Matrizes de Confusão aXeleRate x KPU SiPEED. Fonte: KLIPPEL, 2020.

(a) Simulado Rasgo. (b) Sem Rasgo.

Figura 5.4: Imagens de Validação Classificadas pelo Modelo no aXeleRate. Fonte: KLIPPEL, 2020.

(a) Simulado Rasgo. (b) Sem Rasgo.

Figura 5.5: Imagens de Validação Classificadas pelo Modelo na KPU SiPEED. Fonte: KLIPPEL, 2020.

para evitar *overfitting* do modelo, conforme definido na respectiva opção no dicionário de dados.

5.2 Resultados Testes Campo

5.2.1 Teste Imagens Estáticas - Modelo Treinado com Imagens Estáticas

Para a construção do *dataset* de imagens estáticas nós usamos a sucata do tapete de uma correia transportadora danificada. Essa sucata possuía as marcas de desgaste e sujeira comuns a correias em operação, tornando as imagens muito próximas as situações reais. Nessa sucata de correia foram realizados rasgos, pela equipe de manutenção de correias transportadoras da usina, semelhantes aos identificados nas situações de falhas encontradas por essas equipes.

Nós fotografamos os rasgos simulados com o SiPEED rodando o Programa 1. Para a condição normal da correia nós tiramos fotos, com o SiPEED, de regiões próximas aos rasgos e de regiões sujas da correia. Como essa correia possuía regiões danificadas, essas também foram fotografadas para compor o *dataset*, na Figura 5.6 são mostrados exemplos dessas imagens. O total de imagens do *dataset*, após tratamento manual onde foram retiradas imagens muito parecidas e desfocadas, era de 100 imagens com dano e 100 imagens de condição normal da correia.

(a) Com Rasgo. (b) Sem Rasgo. (c) Dano Real.

Figura 5.6: Imagens Estáticas da Correia para Treinamento. Fonte: KLIPPEL, 2020.

Com o data *dataset* preparado nos executamos o treinamento da DNN no *aXeleRate* usando as mesmas configuração utilizadas para o modelo no item 5.1. O treinamento com 10 épocas foi realizado em 1 minuto, usando uma Graphics Processing Unit (GPU) do Google Colab, atingindo a acurácia de 95% com as imagens de validação. Na Figura 5.7 é mostrado o resultado do processo de treinamento.

Também neste teste nós usamos as mesmas imagens de validação, usadas no processo treinamento, para comparar as performance de execução do modelo na KPU do SiPEED. Os resultados obtidos para os indicadores adotados não mostrados na Tabela 5.2 e na Figura 5.8 são mostradas as matrizes de confusão, para a validação no *aXeleRate* e SiPEED. Exemplos das imagens de validação classificadas são mostradas nas Figuras 5.9 e

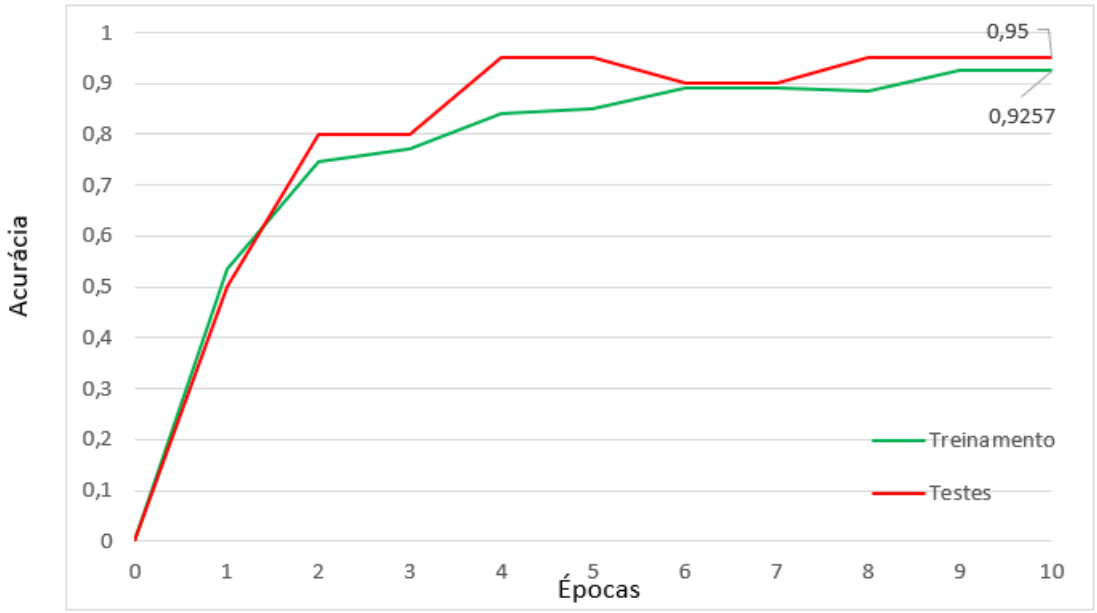
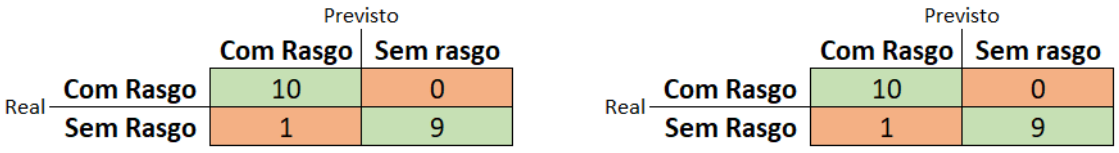


Figura 5.7: Gráfico Acurácia Treinamento no *aXeleRate*- Imagens Estáticas. Fonte: KLIPPEL, 2020.

5.10.

Tabela 5.2: Comparativo Validação *aXeleRate* e KPU do SiPEED- Imagens Estáticas. Fonte: KLIPPEL, 2020.

Indicador	Validação Google Colab	Validação KPU SiPEED
Precisão	1	1
Recall	0,91	0,91
F1	0,95	0,95



(a) aXeleRate

(b) KPU SiPEED

Figura 5.8: Matrizes de Confusão aXeleRate x KPU SiPEED- Imagens Estáticas. Fonte: KLIPPEL, 2020.

Importante notar que uma das imagens de validação foi justamente de um dano real existente na sucata da correia. Essa imagem foi corretamente classificada demonstrando a capacidade de generalização do modelo nos encorajando no sentido de solicitar um teste dinâmico em campo, com um rasgo sendo simulado em uma correia em processo de substituição.

(a) Com Rasgo. (b) Sem Rasgo. (c) Dano Real.

Figura 5.9: Imagens Estáticas Classificadas pelo aXeleRate. Fonte: KLIPPEL, 2020.

(a) Com Rasgo. (b) Sem Rasgo. (c) Dano Real.

Figura 5.10: Imagens Estáticas Classificadas pela KPU do SiPEED. Fonte: KLIPPEL, 2020.

O teste foi por nós solicitado junto a equipe de manutenção de correias transportadoras da unidade da VALE onde trabalhamos, no caso Usina de Beneficiamento de Minério de Ferro de Carajás, em Parauapebas no PA. A data agendada para o teste foi o dia 08/06/2020, nesse dia o protótipo foi instalado próximo a correia e a equipe de manutenção realizou 3 rasgos de 3m de comprimento, um no centro da correia, outro na borda e um entre a borda e centro. Na Figura 5.11 é mostrado o local de instalação do protótipo e na Figura 5.12 um dos rasgos simulados na correia.

Após a preparação a correia foi acionada e os rasgos passaram na frente do protótipo por 10 vezes, como se tratavam de 3 rasgos o protótipo foi exposto a 30 situações de rasgo, simultaneamente um trecho de correia normal, antes de cada rasgo, foi considerada totalizando mesmo número de exposições do protótipo a condição normal. Os resultados desses testes são mostrados na matriz de confusão da Figura 5.13 e os valores dos indicadores de performance são mostrados na Tabela 5.3.

Com esses resultados evidenciamos a completa incapacidade do uso do modelo Deep Neural Network, treinado com imagens estáticas, para a detecção de rasgos, mesmo nas condições controladas do teste. Uma das passagens do rasgo, em frente ao protótipo,

Figura 5.11: Protótipo Instalado Próximo a Correia para Teste. Fonte: KLIPPEL, 2020.

é mostrada na Figura 5.14, note que o rasgo é perceptível mas o protótipo não o detecta. Mesmo não detectando o protótipo tirou fotografias da correia para uso em novo

Figura 5.12: Conjunto de Rasgos Simulados na Correia. Fonte: KLIPPEL, 2020.

		Previsto	
		Com Rasgo	**Sem rasgo**
Real	**Com Rasgo**	0	30
	Sem Rasgo	0	30

Figura 5.13: Matriz de Confusão para Primeiro Teste em Campo. Fonte: KLIPPEL, 2020.

Tabela 5.3: Indicadores Perfomance Primeiro Teste em Campo. Fonte: KLIPPEL, 2020.

Indicador	**Teste**
Precisão	0
Recall	-
F1	-

treinamento do modelo.

Com umas das imagens de rasgo do teste de campo, capturadas pelo protótipo nos disparos temporizados, realizamos um novo teste em laboratório com o SiPEED usando o Programa 3 e o mesmo modelo. O resultado foi repetido com a falha na classificação do rasgo. Esse resultado é evidenciado na Figura 5.15.

Figura 5.14: Momento da Passagem do Rasgo em Frente ao Protótipo. Fonte: KLIPPEL, 2020.

5.2.2 Teste Imagens Dinâmicas - Modelo Treinado com Imagens Dinâmicas

Com o protótipo construído e usando os aprendizados e resultados dos testes anteriores, do modelo Deep Neural Network, organizamos as atividades de campo para captura e experimentos com imagens de rasgo dinâmicas, ou seja, com a correia em movimento. Essas coletas de imagens e testes foram organizadas em três campanhas.

Na primeira campanha utilizamos tinta preta para simular situações de rasgos. Esses rasgos simulados foram pintados com uma largura de 1cm e comprimento de aproxima-

Figura 5.15: Classificação Incorreta pela KPU do SiPEED em Teste de Laboratório. Fonte: KLIPPEL, 2020.

damente 3m. Para realização desses testes escolhemos uma correia que podia ser parada e bloqueada, no caso uma correia de pilha de emergência, sem impactar no processo produtivo. Cabe ressaltar que o bloqueio citado trata-se de um procedimento de segurança previsto na Norma Regulamentadora 22 - Segurança e Saúde Ocupacional na Mineração (NR22), que impede o acionamento indevido da correia durante qualquer atividade aí desenvolvida (DAMINELLI, 2014).

No processo de captura de imagens nós coletamos 200 imagens dos rasgos simulados, e 200 imagens da correia normal. A esse conjunto de imagens adicionamos as imagens do *dataset* estático do Item 5.2.1, totalizando 300 imagens de rasgo e mesmo número sem rasgos no *dataset* de treinamento. A imagem da pintura é mostrada na Figura 5.16, uma captura de imagem do rasgo com a correia em movimento é mostrada na Figura 5.17.

O processo de treinamento foi realizado no aXeleRate, com as mesmas configurações utilizadas anteriormente. O treinamento levou 3 minutos atingido uma acurácia de 93%, conforme pode ser visto no gráfico da Figura 5.18. Os resultados da validação do modelo *MobileNet* tanto no aXeleRate quanto na KPU do SiPEED são apresentados na Tabela 5.4 e nas matrizes de confusão da Figura 5.19. Como de praxe, na Figuras 5.20 e 5.21 mostramos exemplos de imagens de validação classificadas.

Tabela 5.4: Comparativo Validação aXeleRate e KPU do SiPEED- Pintura Rasgo. Fonte: KLIPPEL, 2020.

Indicador	**Validação Google Colab**	**Validação KPU SiPEED**
Precisão	0,93	0,93
Recall	0,97	1,00
F1	0,95	0,97

Os resultados obtidos nestes testes foram bastante promissores, principalmente se considerarmos que o modelo treinado com o conjunto de imagens estáticas e dinâmicas conseguiu detectar os rasgos em movimento, simulados com pintura, com uma precisão de

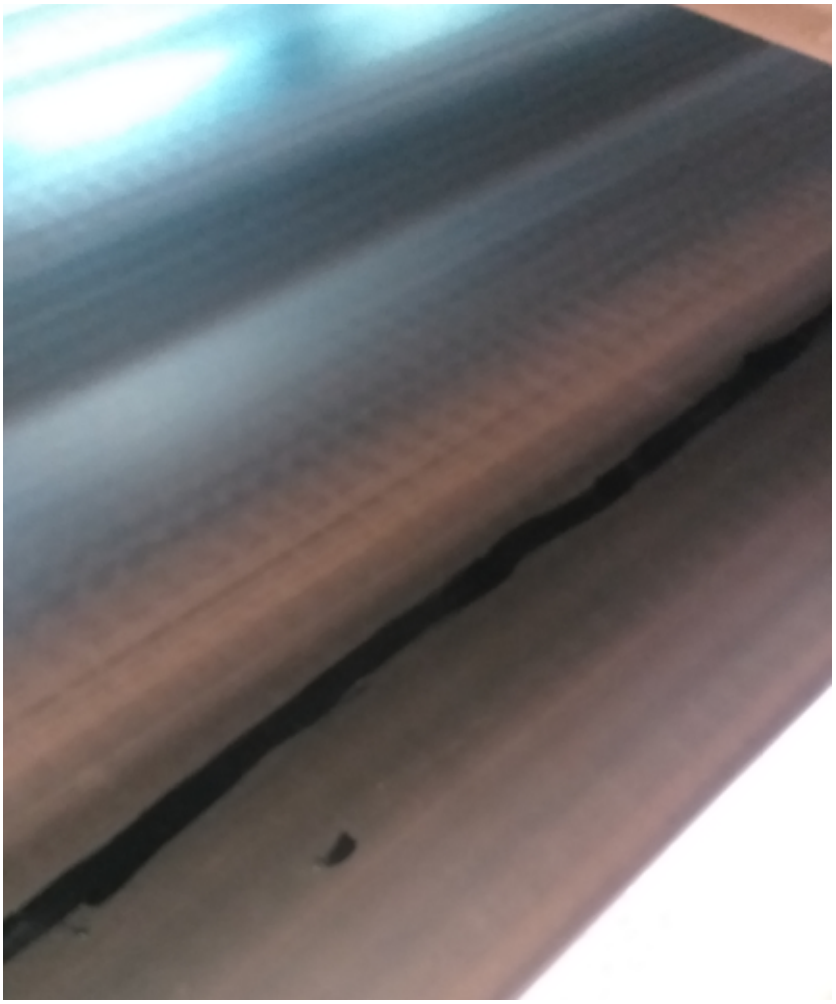


Figura 5.16: Pintura Simulando Rasgo na Correia. Fonte: KLIPPEL, 2020.

93% na KPU do SiPEED. Importante destacar que no final destes testes nos foi sugerido realizar marcas na correia, riscos na superfície do tapete. Essas marcas não danificam a correia e conseguem simular melhor possíveis situações de rasgo. Essa sugestão partiu de profissionais de manutenção, após terem acesso as imagens de rasgo simuladas por pintura e capturadas pelo protótipo.

Na segunda campanha, conforme sugestão das equipes de manutenção, foram simulados rasgos riscando levemente a superfície da correia. Um exemplo desse tipo de rasgo, bem como da instalação do protótipo, podem ser vistos na Imagem 5.22.

Com essa estratégia foram coletadas mais 100 imagens de rasgo simuladas e imagens normais, essas imagens foram acrescentadas ao *dataset* perfazendo 400 imagens. Nesse momento, com o objetivo de melhorarmos a capacidade de generalização do modelo, aplicamos a técnica de *data augmentation* com o uso de um *script* escrito em python para realizar a operação de h-flip, que consiste na inversão horizontal da imagem (DAWSON, 2019). Após esse processo o *dataset* de treinamento passou a contar com 800 imagens de rasgo e 800 imagens sem rasgo.

Figura 5.17: Imagem da Pintura do Rasgo com a Correia em Movimento. Fonte: KLIP-PEL, 2020.

Com o novo *dataset* nos realizamos o treinamento com o aXeleRate, novamente mantivemos as mesmas configurações dos hyper-parâmetros do modelo *MobileNet* usados anteriormente. O treinamento levou 7 minutos para ser executado atingindo uma acurácia de 96,8%. A performance do processo de treinamento é mostrada na Figura 5.23, os comparativos dos resultados da validação no *framework* aXeleRate e KPU do SiPEED são apresentados na Tabela 5.5 e nas matrizes de confusão da Figura 5.24, imagens de exemplos de classificação estão nas Figuras 5.25 e 5.26.

Com esse treinamento nós preparamos um novo teste em campo com a realização de novos riscos, as marcas realizados no dia anterior desapareceram em função da operação normal da correia. Nós realizamos 3 riscos na correia e a acionamos de forma que esses riscos fossem expostos ao protótipo por 10 vezes, totalizando 30 eventos para detecção. O protótipo detectou os rasgos e as condições normais, consideradas antes de cada risco. O

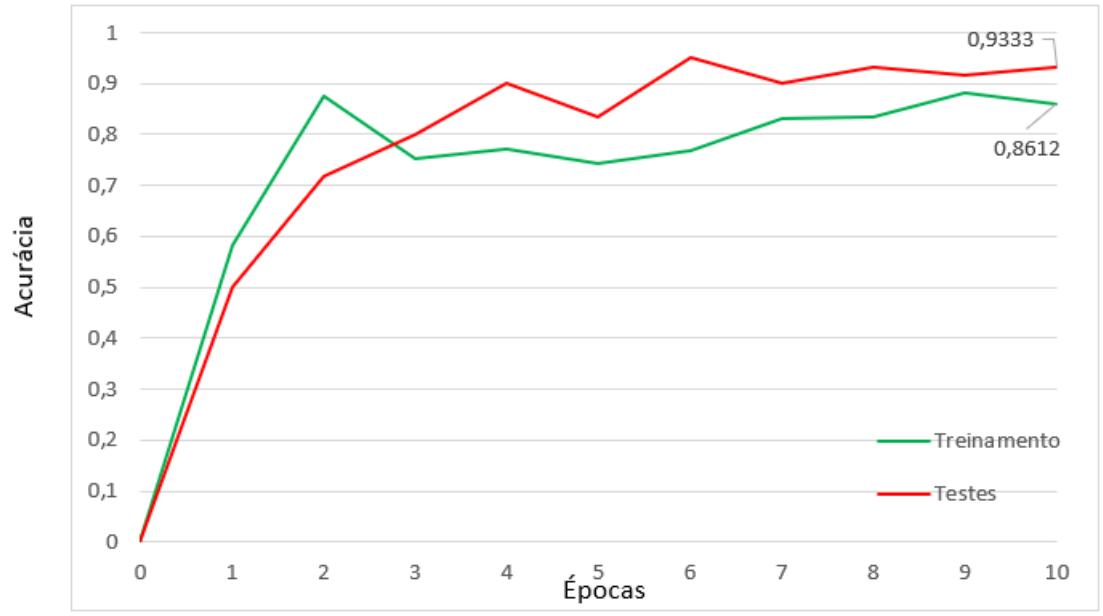


Figura 5.18: Gráfico Acurácia Treinamento no *aXeleRate*- Pinturas de Rasgo. Fonte: KLIPPEL, 2020.

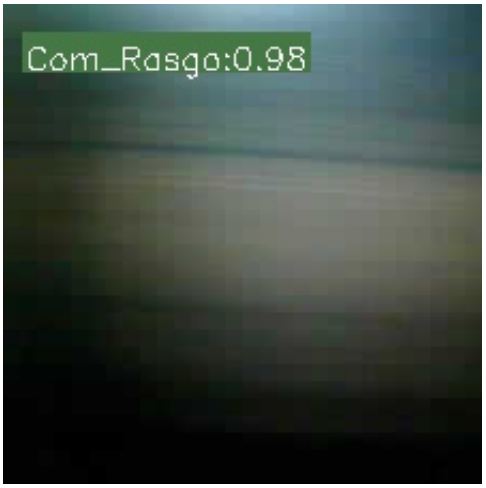
		Previsto	
		Com Rasgo	**Sem rasgo**
Real	**Com Rasgo**	28	2
	Sem Rasgo	1	29

(a) aXeleRate

		Previsto	
		Com Rasgo	**Sem rasgo**
Real	**Com Rasgo**	28	2
	Sem Rasgo	0	30

(b) KPU SiPEED

Figura 5.19: Matrizes de Confusão aXeleRate x KPU SiPEED- Pintura Rasgo. Fonte: KLIPPEL, 2020.



(a) Rasgo Simulado.

(b) Sem Rasgo.

Figura 5.20: Pintura Rasgo Classificadas pelo aXeleRate. Fonte: KLIPPEL, 2020.

(a) Com Rasgo Simulado. (b) Sem Rasgo.

Figura 5.21: Pintura Rasgo Classificadas pela KPU do SiPEED. Fonte: KLIPPEL, 2020.

Tabela 5.5: Comparativo Validação *aXeleRate* e KPU do SiPEED- Correia Riscada. Fonte: KLIPPEL, 2020.

Indicador	**Validação Google Colab**	**Validação KPU SiPEED**
Precisão	0,96	0,94
Recall	0,97	0,97
F1	0,97	0,95

performance do protótipo está na Tabela 5.6 e matriz de confusão da Figura 5.27,exemplos de imagens de rasgo e condição normal detectadas no campo são mostradas na Figura 5.28.

Tabela 5.6: Indicadores Performance Teste em Campo. Fonte: KLIPPEL, 2020.

Indicador	**Teste**
Precisão	0,93
Recall	0,93
F1	0,93

A performance do conjunto *Device Edge* e Deep Neural Network foi satisfatória considerando os resultados de precisão obtidos. Importante ressaltar que as imagens de treinamento foram obtidas com iluminação natural e nos testes foi utilizada iluminação artificial com luz branca, proveniente de leds, outro ponto é que as marcas de treinamento foram realizadas em momentos diferentes das marcas de testes, essas mudanças em características do contexto da imagem mostram a capacidade de generalização do modelo na detecção dos rasgos. Esses resultados nos encorajaram a solicitarmos um novo teste de campo com a confecção de danos na correia para treinamento e detecção.

Para a campanha 3 nós solicitamos o apoio da equipe de operação no sentido de utilizarmos um equipamento que sabíamos estar em processo de desmobilização na época, no

Figura 5.22: Simulação de Rasgo na Correia com Risco no Tapete. Fonte: KLIPPEL, 2020.

caso a recuperadora de roda de caçambas com tag de identificação RP152K01. Esse equipamento possui uma correia transportadora de largura 2m, taxa de operação de 6000t/h e velocidade da correia de 4,5m/s. O equipamento foi liberado para a execução da campanha e nesta foram realizados 5 testes de detecção e uma coleta de imagens para treinamento do modelo. Na Figura 5.29 mostramos o equipamentos onde os testes foram realizados e na Figura 5.30 a correia onde o rasgos foram realizados.

O primeiro teste foi realizado utilizando o modelo treinado na campanha 2 e consistiu na confecção de 2 rasgos na correia, um próximo a borda e outro entre a borda e a dobra central da correia. Esses rasgos simulados tinham comprimento de 2m. O protótipo foi instalado próximo da correia com o campo de visão apontado para a lateral da correia, conforme mostrado na Figura 5.31, a correia foi acionada e os rasgos passaram em frente do protótipo por 20 vezes, totalizando 40 exposições de rasgo e 40 exposição sem. Os

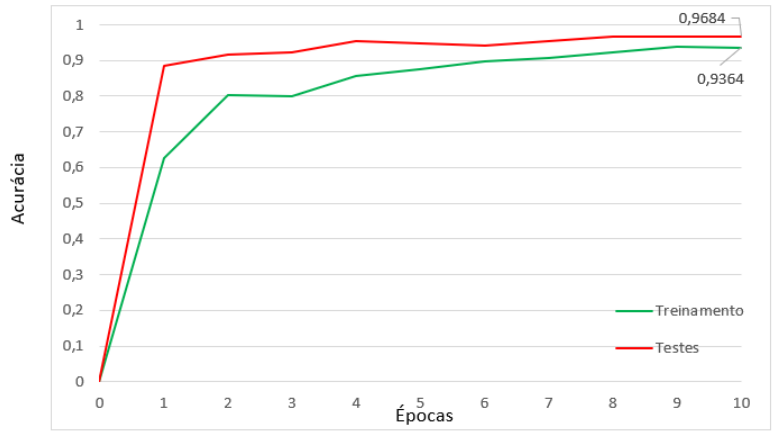


Figura 5.23: Gráfico Acurácia Treinamento no *aXeleRate*- Correia Riscada. Fonte: KLIPPEL, 2020.

		Previsto	
		Com Rasgo	Sem rasgo
Real	Com Rasgo	76	3
	Sem Rasgo	2	77

(a) aXeleRate

		Previsto	
		Com Rasgo	Sem rasgo
Real	Com Rasgo	74	5
	Sem Rasgo	2	77

(b) KPU SiPEED

Figura 5.24: Matrizes de Confusão aXeleRate x KPU SiPEED- Correia Riscada. Fonte: KLIPPEL, 2020.

(a) Com Rasgo.

(b) Sem Rasgo.

Figura 5.25: Correia Riscada Classificada pelo aXeleRate. Fonte: KLIPPEL, 2020.

resultados desse teste são apresentados na Tabela 5.7 e na matriz de confusão da Figura 5.32.

O *recall* obtido nos testes foi adequado mas a precisão foi abaixo dos valores relatados nos trabalhos relacionados estudados no Capítulo 3. Isso reforçou a necessidade, já mapeada por nós, de coletas de novas imagens e treinamento do modelo.

(a) Com Rasgo. (b) Sem Rasgo.

Figura 5.26: Correia Riscada Classificada na KPU do SiPEED. Fonte: KLIPPEL, 2020.

		Previsto	
		Com Rasgo	**Sem rasgo**
Real	**Com Rasgo**	28	2
	Sem Rasgo	2	28

Figura 5.27: Matriz de Confusão para Teste em Campo. Fonte: KLIPPEL, 2020.

(a) Rasgo Detectado. (b) Falso Positivo.

Figura 5.28: Exemplos de Imagens de Detecção em Campo. Fonte: KLIPPEL, 2020.

Tabela 5.7: Performance Primeiro Teste na RP152K01. Fonte: KLIPPEL, 2020.

Indicador	**Teste**
Precisão	0,78
Recall	0,97
F1	0,86

Figura 5.29: Equipamento para Testes - RP152K01. Fonte: KLIPPEL, 2020.

A coleta das imagens foi realizada no mesmo dia aproveitando os mesmos rasgos realizados para os testes. A posição do protótipo foi variada de forma que no seu campo de visão entrassem componentes estruturais da correia, como rolos e cavaletes. Nessas condições foram coletadas 350 imagens dos rasgos e 350 imagens sem rasgos.

As novas 350 imagens foram adicionadas as 400 imagens do *dataset* anterior, sem *data augmentation*, totalizando 750 imagens com rasgos e 750 imagens sem rasgo. O treinamento foi realizado no aXeleRate considerando os mesmos parâmetros configurados

Figura 5.30: Correia da Lança da RP152K01. Fonte: KLIPPEL, 2020.

nos treinamento anteriores. O treinamento levou 6 minutos, atingindo a acurácia de 97,6%, mostrada no gráfico da Figura 5.33. Da mesma forma que nos treinamentos anteriores, realizamos a comparação das validações do modelo original com sua versão compactada e executada pela KPU do SiPEED, esses resultados são mostrados na Tabela 5.8 e nas matrizes de confusão da Figura 5.34. Nas Figuras 5.35 e 5.36 nós também mostramos exemplos das imagens de validação classificadas pelo SiPEED e aXeleRate.

Tabela 5.8: Comparativo Validação *aXeleRate* e KPU do SiPEED- RP152K01. Fonte: KLIPPEL, 2020.

Indicador	**Validação Google Colab**	**Validação KPU SiPEED**
Precisão	1,00	1,00
Recall	0,99	0,97
F1	0,99	0,99

Figura 5.31: Protótipo Instalado na Correia da RP152K01. Fonte: KLIPPEL, 2020

		Previsto	
		Com Rasgo	**Sem rasgo**
Real	**Com Rasgo**	31	9
	Sem Rasgo	1	39

Figura 5.32: Matriz Confusão Teste 1 - RP152k01. Fonte: KLIPPEL, 2020.

A partir desse modelo treinado realizamos mais 4 testes, com variações do campo de

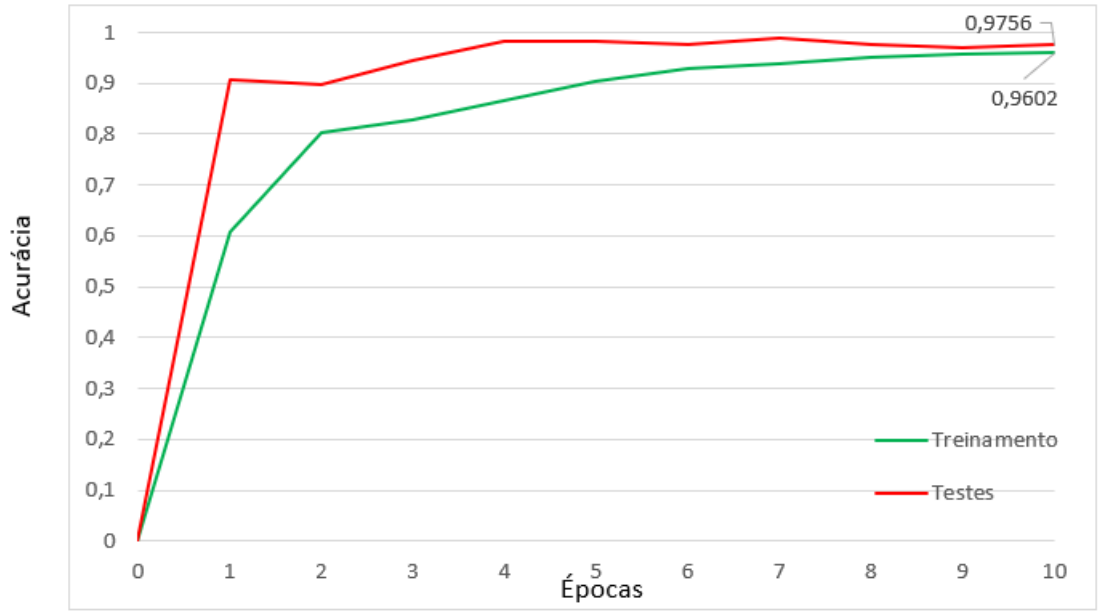


Figura 5.33: Gráfico Acurácia Treinamento no *aXeleRate*- RP152K01. Fonte: KLIPPEL, 2020.

		Previsto	
		Com Rasgo	Sem rasgo
Real	Com Rasgo	75	0
	Sem Rasgo	1	74

(a) aXeleRate

		Previsto	
		Com Rasgo	Sem rasgo
Real	Com Rasgo	75	0
	Sem Rasgo	2	73

(b) KPU SiPEED

Figura 5.34: Matrizes de Confusão aXeleRate x KPU SiPEED- RP152K01. Fonte: KLIPPEL, 2020.

(a) Com Rasgo.

(b) Sem Rasgo.

Figura 5.35: Imagens Validação Classificadas pelo aXelerate na RP152K01. Fonte: KLIPPEL, 2020.

(a) Com Rasgo. (b) Sem Rasgo.

Figura 5.36: Imagens Validação Classificadas pela KPU do SiPEED na RP152K01. Fonte: KLIPPEL, 2020.

visão do protótipo. No teste 1 o protótipo focalizou completamente a lateral da correia; No teste 2 o protótipo focalizou a lateral da correia e sua parte inferior. Nos testes 3 e 4 o campo de visão do protótipo cobriu a lateral da correia e componentes estruturais do transportadores como cavaletes e rolos de carga, com variações na direção do campo de visão.

Em cada um dos testes foram usados os rasgos utilizados no treinamento, não foi viável a confecção de novos rasgos devido a limitações na disponibilidade da equipe de manutenção de correias para novas intervenções. Para cada teste a correia do equipamento RP152K01 foi acionada e os rasgos passaram por 30 vezes em frente do protótipo totalizando 60 exposições, da mesma forma foram consideradas 60 exposições de situação normal da correia, sempre precedendo os pontos de simulação de rasgos. Esses testes foram realizados com iluminação natural com nível de luminância médio medido de 8500Lux.

Os resultados dos testes, para o indicadores adotados, são mostrados na Tabela 5.9. A performance do *Device Edge* executando a *MobileNet* treinada com o *dataset* elaborado em nosso trabalho foi bastante estável, considerando as condições e variações impostas nos experimentos. A maior perda de performance ocorreu no teste 2 devido a este focalizar a correia até o limiar da borda, e nesta existiam desgastes e danos que acabaram sendo detectados como rasgo. Nas Figuras 5.37, 5.38, 5.39 e 5.40 nos mostramos exemplos de cada uma das detecções realizadas no campo, pelo protótipo.

Tabela 5.9: Teste em Campo com Modelo Treinado com Imagens da Correia da RP152K01. Fonte: KLIPPEL, 2020.

Indicador	**Teste 1**	**Teste 2**	**Teste 3**	**Teste 4**
Precisão	1,0	0,98	1,0	0,98
Recall	0,98	0,94	0,98	0,98
F1	0,99	0,96	0,99	0,98

(a) Rasgo. (b) Sem Rasgo.

Figura 5.37: Detecções Realizadas pelo Protótipo Teste 1 - RP152K01. Fonte: KLIPPEL, 2020.

(a) Rasgo. (b) Sem Rasgo.

Figura 5.38: Detecções Realizadas pelo Protótipo Teste 2 - RP152K01. Fonte: KLIPPEL, 2020.

(a) Rasgo. (b) Sem Rasgo.

Figura 5.39: Detecções Realizadas pelo Protótipo Teste 3 - RP152K01. Fonte: KLIPPEL, 2020.

(a) Rasgo. (b) Sem Rasgo.

Figura 5.40: Detecções Realizadas pelo Protótipo Teste 3 - RP152K01. Fonte: KLIPPEL, 2020.

6. Conclusão

Nós consideramos que o objetivo geral do projeto foi alcançado tendo-se em vista a performance do protótipo, construído em conformidade com os paradigmas de *edge AI*, nos testes de campo conduzidos em um equipamento real da operação das usinas da VALE de Carajás. Nesses testes o protótipo realmente foi capaz de detectar a presença de rasgos com uma precisão superior a 98%. Apesar dos testes terem sido conduzidos em condições controladas, estes resultados mostram a completa viabilidade do uso da tecnologia de DNN em conjunto de *device edge* para a solução de um problema real, no âmbito industrial, abrindo todo um campo de novas aplicações para detecção de outros modos de falhas ou mesmo condições operacionais aí presentes.

O modelo DNN *MobileNet* selecionado por nós mostrou, nos diversos testes realizados durante o desenvolvimento do trabalho, capacidade suficiente para realizar a detecção dos rasgos simulados, mesmo em condições bastante severas de iluminação e interferências nas imagens, classificadas no modelo. Esses resultados confirmaram a capacidade de *fine grained recognition* já relatada por Howard *et al.* (2017) em seu trabalho. Este funcionamento adequado, do modelo selecionado, atende um dos nossos objetivos específicos de nossa pesquisa.

Ainda com relação ao modelo conseguimos demonstrar a não perda de performance, para a nossa aplicação, no processo de compactação e compilação para uso com *device edge*. Essa demonstração foi realizada através das comparações dos valores dos indicadores, adotados no trabalho, para as classificações das imagens de validação de cada treinamento, executada no *Keras* e Google Colaboratory e na KPU do SiPEED, após compactação. Não ocorreram variações superiores a 3% em todas as validações executadas e comparadas. Cabe ressaltar que todo esse processo de treinamento, compactação e compilação foram automatizados pelo *framework* aXeleRate, selecionado por nós para uso em nossa metodologia.

Com a seleção do *device edge* SiPEED, em conformidade com os requisitos definidos no trabalho, construção dos protótipos e desenvolvimento dos programas para testes, além da execução de repetidas campanhas de coletas de imagens e testes em campo, atingimos o objetivo específico de definir uma plataforma viável para uso em ambiente industrial. Essa plataforma se mostrou absolutamente confiável e não foi detectada nenhuma falha nos testes tanto de bancada como em campo.

O *dataset* construído, com mais de 700 imagens de rasgo, tanto estáticas quanto dinâmicas, permitiu o adequado treinamento do modelo *MobileNet* e representa, além do atendimento de mais um objetivo específico, uma contribuição significativa para o desenvolvimento de novos trabalhos científicos na temática de detecção de falhas de correias transportadoras.

Também entendemos termos contribuído, com uma pequena parcela, na desmisti-

ficação do uso e aplicação do paradigma de *edge AI* em ambientes industrias, sendo a metodologia, modelos e *device edge* utilizados por nós aplicável na solução de outros problemas e situações reais do dia a dia industrial. Essa contribuição é evidente tanto que o artigo, escrito por nós sobre essa temática, foi o vencedor como melhor artigo na categoria aplicações industriais do evento *23rd International Conference on Enterprise Information Systems - ICEIS - 2021*.

7. Trabalhos Futuros

Como próximo trabalho propomos a instalação fixa da última versão do protótipo em uma correia em operação, isso para realização dos estudos de performance em operação continua, principalmente no que diz respeito a variações de condições ambientais como iluminação, poeira e sujeira na correia. Nesse estudo futuro entendemos ser possível a implementação de detecção de outros modos de falhas como desalinhamento, borda e cabos soltos, aperto excessivo de raspadores entre outros. Aqui também deverão ser estudados mecanismos para identificação de reparos realizados na correia que podem interferir no processo de detecção de rasgo.

Outra oportunidade de desenvolvimento é uso da metodologia utilizada em nosso trabalho, além da DNN e *edge device* escolhidos por nós, para uso em detecção de outras situações presentes no ambiente de operação de usinas e minas de minério de ferro. Entendemos que nosso protótipo, desde que o modelo seja devidamente treinado, poderá ser aplicado em detecção de sobrecargas de correia, entupimentos de chutes de descarga, desbalanceamento de cargas de minério de ferro em vagões e prevenção de situações de avalanche de minério de ferro.

Referências Bibliográficas

CANAAN. "K210 Datasheet - V 0.1.5". 2019. Disponível em: https://cdn.hackaday.io/files/1654127076987008/kendryte_datasheet_-20181011163248_en.pdf. Acesso em 16 de maio de 2020.

CHENG, Y., WANG, D., ZHOU, P., et al.. "A survey of model compression and acceleration for deep neural networks", *arXiv preprint arXiv:1710.09282*, 2017.

DAMINELLI, P. B. "Fatores potencialmente causadores de acidentes do trabalho nas fases de montagem e manutenção em correias transportadoras", 2014.

DAWSON, R. "Image Augmentor - Codebox". 2019. Disponível em: https://github.com/codebox/image_augmentor. Acesso em 13 de julho de 2020.

DENG, L., LI, G., HAN, S., et al.. "Model compression and hardware acceleration for neural networks: A comprehensive survey", *Proceedings of the IEEE*, v. 108, n. 4, pp. 485–532, 2020a.

DENG, S., ZHAO, H., FANG, W., et al.. "Edge intelligence: the confluence of edge computing and artificial intelligence", *IEEE Internet of Things Journal*, v. 7, n. 8, pp. 7457–7469, 2020b.

FERNANDES, A. M. D. R. *Inteligência Artificial: noções gerais*. Visual Books, 2005.

GEORGE, D. P., SOKOLOVSKY, P. "MicroPython Documentation". 2020. Disponível em:Disponível em: http://docs.micropython.org/en/latest/index.html¿. Acesso em 16 de maio de 2020.

HARDYGÓRA, M., WACHOWICZ, J., CZAPLICKA-KOLARZ, K., et al.. *Conveyor belts*. WNT Warszawa, 1999.

HEATON, J. *AIFH, volume 3: deep learning and neural networks*. Heaton Research, 2015.

HOESER, T., KUENZER, C. "Object detection and image segmentation with deep learning on Earth observation data: A review-part I: Evolution and recent trends", *Remote Sensing*, v. 12, n. 10, pp. 1667, 2020.

HOU, C., QIAO, T., QIAO, M., et al.. "Research on Audio-Visual Detection Method for Conveyor Belt Longitudinal Tear", *IEEE Access*, v. 7, pp. 120202–120213, 2019.

HOWARD, A. G., ZHU, M., CHEN, B., et al.. "Mobilenets: Efficient convolutional neural networks for mobile vision applications", *arXiv preprint arXiv:1704.04861*, 2017.

KELLEHER, J. D. *Deep learning*. MIT press, 2019.

KERR, J. F. "Damaged conveyor belt detector". jul. 31 1984. US Patent 4,462,523.

KHOSLA, A., JAYADEVAPRAKASH, N., YAO, B., et al.. "Novel dataset for fine-grained image categorization: Stanford dogs". Em: *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, v. 2. Citeseer, 2011.

KLIPPEL, E., OLIVEIRA, R., MASLOV, D., et al.. "Towards to an Embedded Edge AI Implementation for Longitudinal Rip Detection in Conveyor Belt". Em: *Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, pp. 97–102. SBC, 2020.

KLIPPEL, E., OLIVEIRA, R. A. R., MASLOV, D., et al.. "Conveyor Belt Longitudinal Rip Detection Implementation with Edge AI", 2021.

KOUL, A., GANJU, S., KASAM, M. *Practical Deep Learning for Cloud, Mobile, and Edge: Real-World AI & Computer-Vision Projects Using Python, Keras & TensorFlow*. O'Reilly Media, 2019.

KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E. "Imagenet classification with deep convolutional neural networks", *Advances in neural information processing systems*, v. 25, pp. 1097–1105, 2012.

LEE, M.-H. "Conveyor belt monitoring system". maio 2 1978. US Patent 4,087,800.

LIMA, J. B. "Chave detectora de rasgo em correia transportadora de bandeja de tela". jan. 27 2015. Patente Brasileira PI 0804799-5 B1. Visualizado em 15 apr. 2020, https://www.escavador.com/patentes/320555/chave-detectora-de-rasgo-em-correia-transportadora-de-bandeja-de-tela?page=1.

MAJIDIFARD, H., JIN, P., ADU-GYAMFI, Y., et al.. "Pavement image datasets: A new benchmark dataset to classify and densify pavement distresses", *Transportation Research Record*, v. 2674, n. 2, pp. 328–339, 2020.

MALIK, A., GUPTA, A. *Artificial Intelligence at the Edge*. Amazon Publishing, 2020.

MASLOW, D. “Image Recognition With K210 Boards and Arduino IDE/Micropython”. 2020. Disponível em: https://www.instructables.com/id/Transfer-Learning-With-Sipeed-MaiX-and-Arduino-IDE/¿. Acesso em 17/05/2020.

MOHAJON, J. “Confusion Matrix for Your Multi-Class Machine Learning Model”. Disponível em: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826. Acesso em: 19 de junho de 2020, 2020.

NETTO, G. G. “Método de visão computacional baseado em laser para monitoramento de defeitos em correias transportadoras.” 2019.

RIBEIRO, B. G. C., SOUSA, W. T. D., LUZ, J. A. M. D. “Feasibility project for implementation of conveyor belts in an iron ore mine. Study case: Fabrica Mine in Minas Gerais State, Brazil”, *Rem: Revista Escola de Minas*, v. 69, n. 1, pp. 79–83, 2016.

SANTOS, A. A., ROCHA, F. A., REIS, A. J. D. R., et al.. “Automatic system for visual detection of dirt buildup on conveyor belts using convolutional neural networks”, *Sensors*, v. 20, n. 20, pp. 5762, 2020.

SEEED. “SiPEED-MAiX-BiT for RISC-V AI+IoT - Seeed Studio”. 2020. Disponível em: https://www.seeedstudio.com/Sipeed-MAix-BiT-for-RISC-V-AI-IoT-p-2872.html. Acesso em 05 de janeiro de 2020.

SHUNG, K. P. “Accuracy, Precision, Recall or F1?” Disponível em: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. Acesso em: 19 de junho de 2020, 2018.

SNYDER, F. “Conveyor belt rip detector”. fev. 12 1974. US Patent 3,792,459.

SOUSA, J., BRITO, J., OSCAR, H. “Relatório Consolidado – Rasgo Correias TRs-117k-07, TR-121k-08 e TR-113k-02”. out. 08 2019. Número VALE 1181.

SUNNYCASE. “Kendrite nncase”. 2020. Disponível em: <https://github.com/kendryte/nncase>.

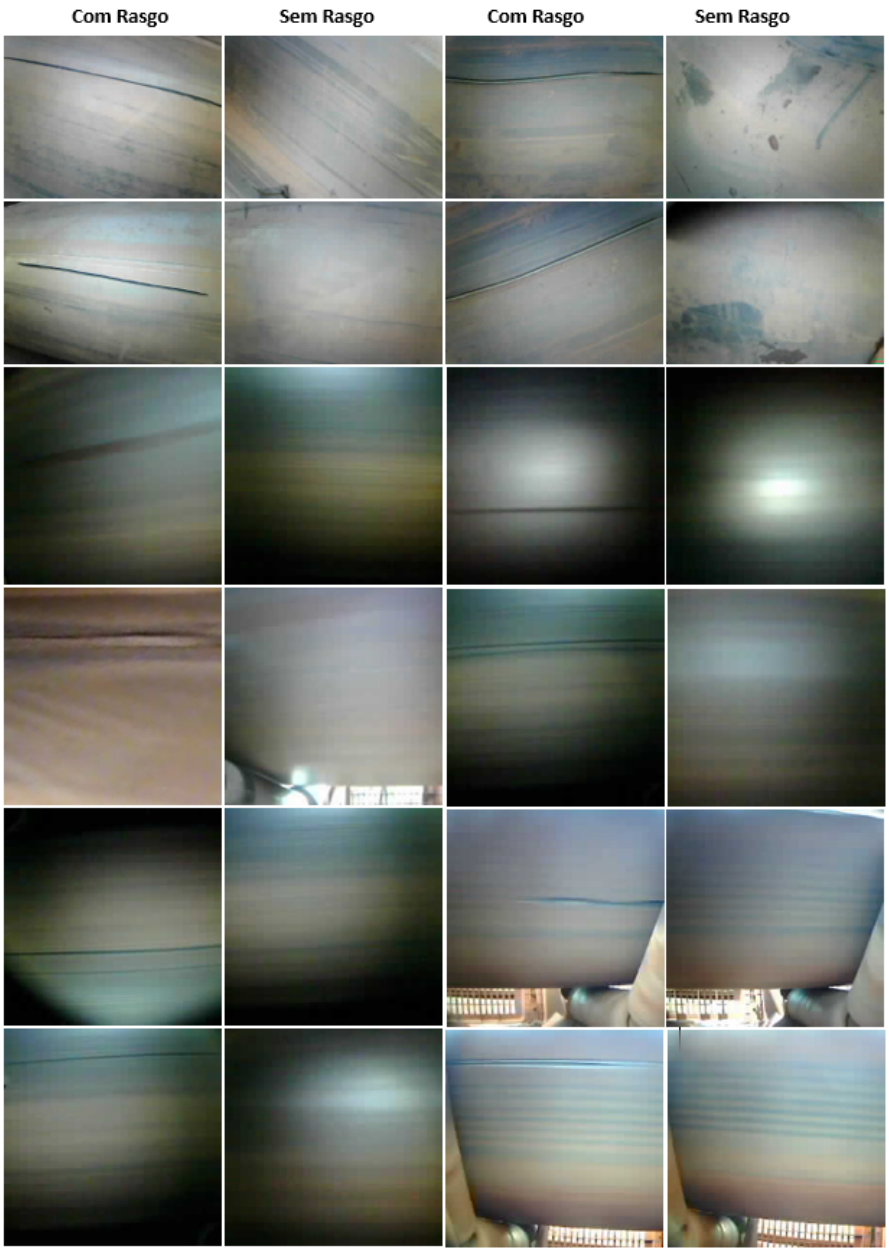
SWINDERMAN, R. T., MARTI, A., GOLDBECK, L. J., et al.. *Foundations- TM - Guia Prático para um Controle mais Limpo, Seguro e Produtivo de Pó e Material a Granel*. Martin Engineering, Ltda, 2012.

TOLLERVEY, N. H. *Programming with MicroPython: embedded programming with microcontrollers and Python*. ”O’Reilly Media, Inc.”, 2017.

ZHANG, X., WANG, Y., LU, S., et al.. "Openei: An open framework for edge intelligence". Em: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1840–1851. IEEE, 2019.

ZHOU, Z., CHEN, X., LI, E., et al.. "Edge intelligence: Paving the last mile of artificial intelligence with edge computing", *Proceedings of the IEEE*, v. 107, n. 8, pp. 1738–1762, 2019.

Apêndice A: Exemplos de Imagens do *Dataset*



Exemplos de Imagens do Dataset Desenvolvido Durante os Estudos. Fonte: KLIPPEL, 2020.

Apêndice B: Código Fonte em MicroPython

Programa 1

```
1 from fpioa_manager import *
2 import os, Maix, lcd, image, time, sensor
3 from Maix import FPIOA, GPIO
4
5 # Associando o pino IO 7 do K210 com pino 17 da placa
6 FPIOA().set_function(0,FPIOA.GPIO6)
7 FPIOA().set_function(1,FPIOA.GPIO7)
8
9 start=GPIO(GPIO.GPIO6,GPIO.IN,GPIO.PULL_UP)
10 stop=GPIO(GPIO.GPIO7,GPIO.IN,GPIO.PULL_UP)
11
12 # Associando o pino IO 0 do K210 com o pino ligado ao LED R da placa
13 fm.register(board_info.LED_B, fm.fpioa.GPIO4)
14 led_b=GPIO(GPIO.GPIO4,GPIO.OUT)
15 fm.register(board_info.LED_G, fm.fpioa.GPIO5)
16 led_g=GPIO(GPIO.GPIO5,GPIO.OUT)
17
18 led_b.value(1)
19 led_g.value(1)
20
21 sensor.reset(freq=10000000)
22 sensor.set_pixformat(sensor.RGB565)
23 sensor.set_framesize(sensor.QVGA)
24 sensor.set_windowing((224, 224))
25 #sensor.set_auto_gain(1,6)
26 sensor.set_vflip(1)
```

```
27 sensor.run(1)
28 sensor.skip_frames(40)
29
30 lcd.init(color=(0,100,100))
31 lcd.draw_string(10,10,"AGUARDANDO", lcd.WHITE, lcd.RED)
32 path = "Sem fotos"
33 i = 0
34 while True:
35 while start.value() == 1:
36 led_b.value(0)
37 img = sensor.snapshot()
38 lcd.display(img)
39 lcd.draw_string(10,10,"AGUARDANDO", lcd.WHITE, lcd.RED)
40 lcd.draw_string(200,220,path, lcd.WHITE, lcd.BLUE)
41
42 led_b.value(1)
43 lcd.draw_string(10,10, "FOTO", lcd.WHITE, lcd.RED)
44 led_g.value(0)
45
46 #time.sleep(1)
47 i = i + 1
48 path = "/sd/DCIM/Imagem"+ str(i)+ ".jpg"
49 img = sensor.snapshot()
50 lcd.display(img)
51 lcd.draw_string(10,30, path, lcd.WHITE, lcd.RED)
52 img.save(path)
53 lcd.draw_string(10,10,"AGUARDANDO", lcd.WHITE, lcd.RED)
54 time.sleep(1)
```

Programa 2

```
1 #tested with maixpy_v0.5.0_33_gfcd6d8a_minimum_with_ide_support.bin
2 import sensor, image, lcd, time, utime, os, re
3 import KPU as kpu
4 from Maix import GPIO, FPIOA
5 from machine import Timer,PWM
6
7 # Associando o pino IO 3 e 4 do K210 com pinos 18 e 20 da placa
```

```
8 FPIOA().set_function(18,FPIOA.GPIO3)
9 FPIOA().set_function(20,FPIOA.GPIO4)
10 FPIOA().set_function(35,FPIOA.GPIO5)
11 FPIOA().set_function(0,FPIOA.GPIO6)
12 led_g=GPIO(GPIO.GPIO3,GPIO.OUT)
13 led_r=GPIO(GPIO.GPIO4,GPIO.OUT)
14 rele=GPIO(GPIO.GPIO5,GPIO.OUT)
15 button=GPIO(GPIO.GPIO6,GPIO.IN,GPIO.PULL_UP)
16
17 # Incialização do Display e da Camera
18 lcd.init()
19 sensor.reset(freq=10000000)
20 sensor.set_pixformat(sensor.RGB565)
21 sensor.set_framesize(sensor.QVGA)
22 sensor.set_windowing((224, 224))
23 #sensor.skip_frames(40)
24 sensor.set_vflip(1)
25 lcd.clear()
26 lcd.draw_string(30,50, "Incializacao Realizada!", lcd.WHITE, lcd.BLACK)
27
28 # MODELO
29 # Classes
30 labels=['Com Rasgo', 'Sem Rasgo']
31 lcd.draw_string(30,70, "Labels Criados!", lcd.WHITE, lcd.BLACK)
32
33 # Carregando o Modelo a Partir do cartão SD
34 #a = kpu.deinit(task)
35 task = kpu.load('/sd/Modelo11.kmodel')
36 lcd.draw_string(30,90, "Modelo carregado", lcd.WHITE, lcd.BLACK)
37
38 # Parametrizando a Saída do Modelo Conforme Modelo Treinado - Número de
Classes devem ser iguais, no caso são duas classes logo temos 2 saidas
39 kpu.set_outputs(task, 0, 1, 1, 2)
40 lcd.draw_string(70,110, "Ultima layer carregado", lcd.WHITE, lcd.BLACK)
41
42 # Loop Principal
43 i = 0
44 k = 0
45 rele.value(1)
```

```
46 tempo_i = time.time()
47 tempo_f = time.time()
48
49 def cria_pasta(path_aux):
50 pastas = os.listdir(path_aux) # lista as pastas dentro de Foto
51 if (len(pastas)==0):
52 path_aux = path_aux + "/Test_Bed1"
53 os.mkdir(path_aux) # Cria o diretório Foto no local /sd/
54 pasta = "/Teste_Bed1"
55 else:
56 pasta = pastas[len(pastas)-1]
57 aux = re.sub('[ô-9]', ",pasta) # Remove da string pasta tudo que não é número
58 num = int (aux) # Contem o número da última pasta
59 path_aux = path_aux + "/Test_Bed"+str(num+1)
60 os.mkdir(path_aux)
61 return path_aux
62
63 path=cria_pasta("/sd/Foto")
64 path1 = path +"/Evento"
65 os.mkdir(path1)
66 path2 = path +"/Foto"
67 os.mkdir(path2)
68
69 temp_i2= time.ticks_ms()
70 temp_f2= time.ticks_ms()
71
72 tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PWM)
73 ch = PWM(tim, freq=500000, duty=0, pin=22)
74 duty=1
75 SP = 3
76 aux_led=True
77
78 #Prepara o arquivo com dados dos eventos de rasgo detectados
79 f = open(path1+'/Com_Rasgo.txt','w')
80 f.write(Ärquivo com informações dos eventos detectados"+"")
81 f.close()
82 #Prepara o arquivo com dados das fotos sem rasgo tiradas de forma temporizada
83 g = open(path2+'/Sem_Rasgo.txt','w')
84 g.write("Arquivo com informações das fotos temporizadas"+"")
```

```
85 g.close()
86
87 while(True):
88 temp2 = temp_f2 - temp_i2
89 temp_f2 = time.ticks_ms()
90 kpu.memtest()
91 img = sensor.snapshot()
92
93 #Controle LED
94 ganho = sensor.get_gain_db()
95 if (button.value()==0):
96 while(button.value()==0):
97 w = 5
98 if (aux_led):
99 aux_led=False
100 else:
101 aux_led=True
102 if (aux_led):
103 ERR = ganho - SP
104 duty = duty + (0.1*ERR)
105 if (duty¿80):
106 duty = 80
107 if (duty¡0):
108 duty = 0
109 else:
110 duty = 0
111 ch.duty(duty)
112 #img = img.rotation_corr(z_rotation=90.0) uncomment if need rotation correction
- only present in full maixpy firmware
113 #a = img.pix_to_ai()
114 fmap = kpu.forward(task, img)
115 plist=fmap[:]
116 pmax=max(plist)
117 max_index=plist.index(pmax)
118 a = lcd.display(img)
119
120 # Rasgo Detectado
121 if max_index==0:
```

```
122 lcd.draw_string(48, 224, "%.2f :%s "%(pmax, labels[max_index].strip()),lcd.RED,
lcd.BLACK)
123 led_r.value(1)
124 led_g.value(0)
125 tempo_f = time.time()
126 tempo_acionado = tempo_f - tempo_i
127 if(tempo_acionado¿3):
128 rele.value(0)
129 i = i + 1
130 foto = path1+"/Evento"+ str(i)+ ".jpg"
131 img.save(foto)
132 f = open(path1+'/Com_Rasgo.txt','a')
133 f.write(foto+", "+"%.2f :%s "%(pmax, labels[max_index].strip())+"")
134 f.close()
135 # Correia Normal
136 else:
137 lcd.draw_string(48, 224, "%.2f :%s "%(pmax, labels[max_index].strip()),lcd.GREEN,
lcd.BLACK)
138 led_r.value(0)
139 led_g.value(1)
140 tempo_i= time.time()
141 tempo_f= time.time()
142 rele.value(1)
143
144 if (temp2¿10000):
145 k = k +1
146 temp_i2= time.ticks_ms()
147 temp_f2= time.ticks_ms()
148 foto = path2+"/Foto"+ str(k)+ ".jpg"
149 img.save(foto)
150 g = open(path2+'/Sem_Rasgo.txt','a')
151 g.write(path1+", "+"%.2f :%s "%(pmax, labels[max_index].strip())+"")
152 g.close()
153 lcd.draw_string(200,0,"Evento: "+ str(i), lcd.WHITE, lcd.RED)
154 lcd.draw_string(0,0,path[8:], lcd.WHITE, lcd.RED)
155 lcd.draw_string(200,180,"Foto "+ str(k), lcd.WHITE, lcd.RED)
156 lcd.draw_string(250,200,str(round(duty,1))+"%",lcd.WHITE, lcd.BLACK)
157 lcd.draw_string(250,220,str(round(ganho,1))+"dB",lcd.WHITE, lcd.BLACK)
158
```

```
159 a = kpu.deinit(task)
160 with open("Le_Imagem.py") as f:
161 exec(f.read())
```

Programa 3

```
1 from fpioa_manager import *
2 import os, Maix, lcd, image, time
3 from Maix import FPIOA, GPIO
4 import KPU as kpu
5
6 # Associando o pino IO 6 do K210 com pino 0 da placa
7 FPIOA().set_function(0,FPIOA.GPIO6)
8 start=GPIO(GPIO.GPIO6,GPIO.IN,GPIO.PULL_UP)
9
10 # inicializando o LCD
11 lcd.init(color=(0,100,100))
12 lcd.direction(lcd.YX_LRUD)
13
14 # lendo arquivo de configuração para extrair path das fotos e quantidades destas
15 f=open('/sd/Teste_Modelo_Config.txt','r')
16 info=f.readlines()
17 f.close()
18 path_com_rasgo = info[0].strip()
19 num_com_rasgo = int(info[1].strip())
20 path_sem_rasgo = info[2].strip()
21 num_sem_rasgo = int(info[3].strip())
22 path_modelo = info[4].strip()
23 path_resultado = info[5].strip()
24 # Apresentado os valores carregados do arquivo de configuração
25 lcd.draw_string(10,10,"Configuracoes carregadas!", lcd.WHITE, lcd.RED)
26 lcd.draw_string(10,30,path_com_rasgo, lcd.WHITE, lcd.RED)
27 lcd.draw_string(10,50,str(num_com_rasgo), lcd.WHITE, lcd.RED)
28 lcd.draw_string(10,70,path_sem_rasgo, lcd.WHITE, lcd.RED)
29 lcd.draw_string(10,90,str(num_sem_rasgo), lcd.WHITE, lcd.RED)
30 lcd.draw_string(10,110,path_modelo, lcd.WHITE, lcd.RED)
31 lcd.draw_string(10,130,path_resultado, lcd.WHITE, lcd.RED)
32
```

```
time.sleep(2)
lcd.clear()

# Inicializando a KPU e carregando o modelo
labels=['Com rasgo', 'Sem rasgo']
lcd.draw_string(10,10, "Labels Criados!", lcd.WHITE, lcd.BLACK)

# Carregando o Modelo a Partir do cartão SD
# Necessário firmware maixpy_v0.5.0_33_gfcd6d8a_minimum_with_ide_support.bin
task = kpu.load(path_modelo)
lcd.draw_string(10,30, "Carregado: ", lcd.WHITE, lcd.BLACK)
lcd.draw_string(10,30, path_modelo, lcd.WHITE, lcd.BLACK)

# Parametrizando a Saída do Modelo Conforme Modelo Treinado - Número de Classes devem ser iguais,
# no caso são duas classes logo temos 2 saidas
kpu.set_outputs(task, 0, 1, 1, 2)
lcd.draw_string(10,50, "Ultima layer carregada", lcd.WHITE, lcd.BLACK)
time.sleep(2)
lcd.clear()




# função que recebe o caminho e classifica a foto com a rede neural
def analisa (path_fotos,num):
cont = int(num)
path = path_fotos + str(i)+ ".jpg"
classe = "rasgo"
certeza = 0.99
lcd.clear()
img_read = image.Image(path)
#lcd.display(img_read)
lcd.draw_string(10,30,path, lcd.WHITE, lcd.RED)
#Classifica a imagem
kpu.memtest()
img = img_read

```

```
70 #img = img.rotation_corr(z_rotation=90.0) uncomment if need rotation correction
- only present in full maixpy firmware
71 #a = img.resize(224,224)
72 a = img.pix_to_ai()
73 fmap = kpu.forward(task, img)
74 #a = lcd.display(img)
75 plist=fmap[:]
76 pmax=max(plist)
77 max_index=plist.index(pmax)
78 a = lcd.display(img)
79
80
81 #lcd.draw_string(10,220,"Classe: "+classe, lcd.WHITE, lcd.RED)
82 #lcd.draw_string(200,220,"Certeza: "+str(certeza), lcd.WHITE, lcd.GREEN)
83 #a = img.draw_string(0,200,"
84 #b = img_read.draw_string(0,200,"
85 a = lcd.display(img)
86 lcd.draw_string(10,30,path[30:], lcd.WHITE, lcd.RED)
87 lcd.draw_string(48, 224, "
88 #lcd.draw_string(10,20,path[30:], lcd.WHITE, lcd.RED)
89 #img.save(path_result)
90 f.write(path+", "+"
91
92
93
94 path = "Sem fotos"
95 i = 0
96 aux = True
97 roda = True
98 f = open(path_resultado,'w')
99
100 clock = time.clock()
101 while roda:
102 #while start.value() == 1:
103 # lcd.draw_string(10,0,"Ready", lcd.WHITE, lcd.RED)
104
105 if aux:
106 i = i + 1
107 analisa(path_com_rasgo,i)
```

```
108 time.sleep(1)
109 if i==num_com_rasgo:
110 i = 0
111 aux = False
112 else:
113 i = i + 1
114 analisa(path_sem_rasgo,i)
115 time.sleep(1)
116 if i==num_sem_rasgo:
117 roda = False
118 lcd.draw_string(10,0,"Analise finalizada!", lcd.BLUE, lcd.RED)
119 f.close()
```