Towards to an Embedded Edge AI Implementation for Longitudinal Rip Detection in Conveyor Belt

Emerson Klippel∗† , Ricardo Augusto Rabelo Oliveira† , Dmitry Maslov‡ , Andrea Gomes Campos Bianchi†,
Saul Emanuel Delabrida Silva† , and Charles Tim Batista Garrocho†
∗*Vale Company, Paruapebas, Brazil*
emerson.klippel@vale.com
†*Computing Department, Federal University of Ouro Preto Ouro Preto, Minas Gerais, Brazil*
rrabelo@gmail.com, andrea@ufop.edu.br, saul@sdelabrida.com, ctgarrocho@gmail.com
‡*TinkerGen, Seeed Studio, Shenzhen, China*
dmitrywat@gmail.com

***Abstract*— The use of deep learning on edge AI to detect failures in conveyor belts solves a complex problem of iron ore beneficiation plants. Losses in the order of thousands of dollars are caused by failures in these assets. The existing fault detection systems currently do not have the necessary efficiency and complete loss of belts is common.**

Correct fault detection is necessary to reduce financial losses and unnecessary risk exposure by maintenance personnel. This problem is addressed by the present work with the training of a deep learning model for detecting images of failures of the conveyor belt. The resulting model is converted and executed in an edge device located near the conveyor belt to stop it in case a failure is detected.

The results obtained in the development and tests carried out to date show the feasibility of using Edge AI to solve complex problems in a mining environment such as detecting longitudinal rips and stimulate the continuity of the work considering new scenarios and operational conditions in the search for a robust and replicable solution.

Keywords*— *Artificial intelligence, deep learning, edge computing

I. Introduction

Conveyor belts are one of the most utilized resources in iron ore industrial plants, easily reaching up to dozens of kilometers in a mid-sized plant. The main component of a conveyor belt is its vulcanized rubber belt with its internal structure made of fabric (canvas) or steel cables.

The conveyor belt is very sensible to piercing and sharp elements, this being critical especially where load transfer from one belt to another occurs.

In some cases, the cut may lead to the complete loss of the asset with impacts on the maintenance costs, production losses, and exposure to health and safety hazards of the workers. Fig. 1 shows an example of a real conveyor belt rip by an element present in the iron ore being transported.

Rip detection instruments can be categorized in two groups, electromechanical and electronic integrated in the conveyor belts. The electromechanical have as their main advantage its installation and maintenance costs but have low sensibility and efficiency, primarily for their dependence on direct interaction with the tear or its effects on the sensor element. The electronics integrated to the belt have good efficiency of detection but they have high cost and low flexibility.

Fig. 1. Example of longitudinal rip in the conveyor belt.

The specific contribution of this work is the study to use deep learning algorithms being executed in the device edge to detect belt failures from real-time images of its surface.

In our project, all image processing and decision making will be carried out on the device edge, without any dependence on centralized processing, which is the difference compared to other solutions based on the detection of image failures but with centralized processing.

In this sense, this work presents an Edge artificial intelligence (AI) approach for an industrial context. More specifically, is used deep learning in an embedded device in a decentralized way with the manipulation of information in real-time and local decision making combined with restrictions of energy consumption, cost, and dimensions.

II. Theoretical Reference

The constant development of AI algorithms has enabled it to perform complex tasks previously feasible only by humans and allowing the creation of applications for daily use.

A. *Deep Learning*

At the same time the models acquired the ability to perform practical tasks, their complexity increased. Modern models have dozens of layers and thousands of neurons leading to millions of trainable parameters. It can be said that the confluence of these millions of trainable parameters, massive amounts of data, modern training techniques, and specialized processors led to deep learning. Deep learning algorithms are widely used in image classification and detection [1].

In the classification and detection of images, models must have a high learning capacity. This feature is achieved by using deep layers in deep convolutional network models. Deep learning is composed of a set of technologies for optimization such as dropout, rectified linear units (ReLU) activation functions, and convolutional neural network architectures. Another important technology applied in deep learning is the transfer of knowledge. This technique allows the use of previously trained models that will be refined in the final training process [2].

The typical workflow of the deep learning process is shown in Fig. 2. Once the problem to be solved is defined, the development will pass through three stages: the first is the collection, preparation, and validation of the data for training; the second is the process of learning with the choice of the model, the task to be performed, training method and training; the last stage is the validation of the model with an analysis of its performance considering perceived errors, model bias, spurious correlations [3].

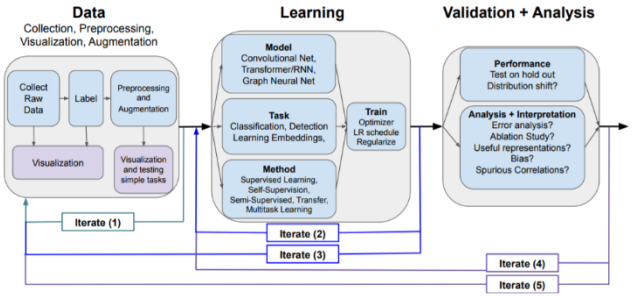

Fig. 2. Deep learning workflow [3].

B. *Edge AI*

The need for local information processing leads to the demand for distributed computing. The Edge computing paradigm is born out of this need and at a high level can be divided into Device Edge, Enterprise Edge, Far Edge, and Near Edge.

Device edges are the closest to the real world, being the Internet of Things (IoT) with the capacity to collect information from the environment, analyze and take actions without the need to exchange data with a centralized system [4].

The confluence of AI with Device Edge created the concept of Edge AI. Edge AI is in its early stages, attracting many companies and researchers to its study and application development. Considering the restrictions on energy consumption, processing power and memory size existing in Device Edge, development efforts have focused on framework design, model adaptation and processor acceleration [5].

The design of the frameworks and adaptation of the models are mainly addressed by the manufacturers of these devices. In our studies, we verified the existence of solutions such as the software of the Google Coral platform [6], the nncase compiler [7], and the OpenVino toolkit from Intel [8].

Neural network accelerators are processors designed and optimized for performing deep neural network operations. These are built considering the energy consumption, processing speed, and memory size restrictions available on device edges. Neural network accelerators have as their main objective inference and not training. The optimization of its operation is achieved through specific processing and memory architectures for operation with deep neural networks [9].

III. Related Works

Many works and studies have been carried out to use deep learning algorithms for intrusion detection, fire detection, structural failures, machine defects, product quality verification.

Detection of defects on the surface of conveyor belts is proposed with the use of computer vision combined with laser light to detect tears in the belt conveyor [10]. The images were captured using a CMOS flat matrix installed in the lower region of the belt. The laser image on the captured belt in real-time was treated with suitable filters and in the simulations carried out the system was able to detect tears quickly and accurately.

Longitudinal rip in conveyor belt detection with the combined use of sounds and images reached an accuracy of 86.7% in laboratory simulations [11]. The images obtained from the belt are processed with the application of filters, binarization, and extraction and bit counting associated with the existence of a tear or not. In parallel, a microphone array captures the noise produced by the belt, which is processed using Mel-Frequency Cepstral Coefficients (MFCC) and Gaussian Mix Model - Universal Background Model (GMM-UBM). This processing identifies the possible signature of the tear on the belt. The audio and video identifications are combined to produce the result of a belt with or without a tear.

A study for the use of convolutional neural networks and images for the detection of dirt in mechanical structures of conveyor belts has been developed with promising results [12]. In this study, two network architectures were used at RsNet18 and VGG16. These were trained from 73 photographs of the clean and dirty belt structure. The accuracy results for identifying the presence of dirt or not was 81.8% for the ResNet18 architecture and 95.5% for the VGG16.

Failure detection situations with visual characteristics similar to the belt tear are addressed using the frameworks YOLO (You Only Look Once) and Faster R-CNN (Faster Region Convolutional Neural Network) [13]. In this study, the models were trained in the identification of defects in the street asphalt. The image dataset was obtained from Google Street images. Each defect in the image was classified as belonging to 1 out of 9 categories, classified manually by a

specialist. The total number of classified images was 7,237. The results were satisfactory for both the YOLO-v2 and Faster R-CNN networks with precision results equal to 93% and 75%, respectively and F1 (Overall Accuracy) with values of 84% and 65%, respectively.

IV. Deep Learning Applications in Edge AI

The SiPEED board was used to implement the damage detector using deep learning on Edge AI. SiPEED presented the best cost-benefit relation compared to the other existing platforms available at the beginning of this work. The SiPEED model used is illustrated in Fig. 3, while the technical specifications in Table I was used to choosing the board.

TABLE I. Cross-Platform Comparison.

Parameter	Raspberry Pi 3	Jetson Nvidia Nano	SiPEED MAiX BiT
Processor	ARM Cortex – A53	Quad-core a57	K210 – RISC – V
Clock (GHz)	1.2	1.43	0.40
RAM (GB)	1	4	0.008
AI resources	NA	GPU	KPU
OS / Language	Raspian / Python	Ubuntu / SDK JetPack	uPython
Power rating (W)	15	10	5
Costs (US$)	75.00	194.00	21.00

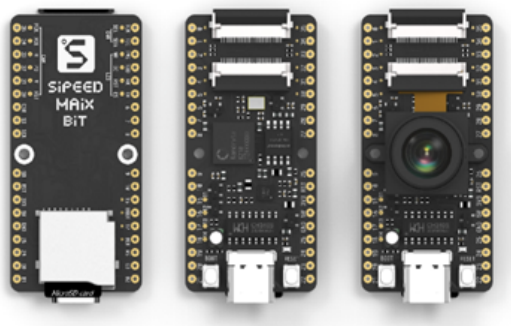

Fig. 3. SiPEED MAiX BiT

A. SiPEED Architecture

The SiPEED platform was developed from the K210 Kendryte, a System on Chip (SoC) aimed at computer vision and hearing applications, has an integrated convolutional neural network accelerator. Fig. 4 shows the K210 block diagrams.

SiPEED boards can be programmed using C or Micropython programming languages. C SDK and Micropython firmware both have a specific set of libraries for manipulation of convolutional neural networks, computer vision, and sound or voice. When flashed with Micropython firmware, SiPEED boards can be programmed using MaixPy IDE, an integrated development environment derived from OpenMV, that allows connection to the device, code execution, and debugging with visual feedback.

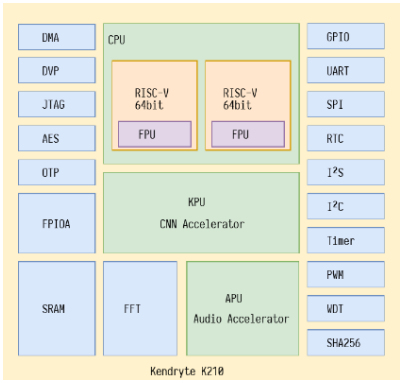

Fig. 4. Block diagram of K210

The K210 neural network accelerator is called Knowledge Processor Unit (KPU) and is a processor for convolution operation, batch normalization, activation, and pooling operations. It can detect objects and faces in real-time, Fig. 5 shows the K210 Kendryte KPU block diagram.

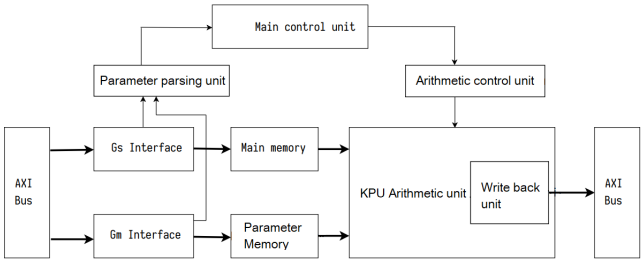

Fig. 5. Block diagram of KPU

The KPU supports a wide range of tensor operations used in common network architectures, such as Conv2D, DepthwiseConv2D, MaxPool2D, Relu6, and others(20+ in total). Model compilation to K210 format(.kmodel) is performed using nncase software developed by the K210 manufacturer. The manufacturer of the K210 is the Chinese company Canaan Creative.

B. Training Framework

The aXeleRate framework was used in training the deep learning model implemented in Edge AI. aXeleRate is based on Keras-TensorFlow and consists of a set of scripts optimized to be executed in a jupyter notebook running on the Google Collaboratory platform [14].

AXeleRate has a modular structure, allowing users to combine different frontend architectures with a variety of feature extractors, such as MobileNet, NASNetMobile, ResNet, and others. Frontend defines the format of data output by model - in aXeleRate users can choose between a classifier, YOLOv2 detector, and SegNet-basic semantic segmentation network. The data in front of the images are preprocessed and fed into the feature extractor part of the network. The resulting feature vectors are used by the network frontend to classify the image, output the bounding boxes or segmentation masks, depending on the type of frontend.

The main feature of aXeleRate is the automatic conversion of trained models to the necessary format for later use on Edge AI devices. The Edge AI devices ecosystem is currently very fragmented, each device requires the model to be converted into its own format in

order to accelerate inference. The conversion is done using different tools that are often not compatible with each other. For example, K210 uses nncase converter, Nvidia Movidius chips use OpenVINO toolkit and Google Edge TPU uses a proprietary model compiler.

The process of using aXeleRate is shown in Fig. 6 with the main steps indicated by the blue circumferences The first step consists of loading the images from the dataset stored in Google Drive for training in the TensorFlow-Keras framework (indication 1). After training, the model is delivered in .h5 format, for classifiers (indication 2). Next, the .h5 model returns to TensorFlow (indication 3), to be converted to .tflite format (indication 4), and then to be compiled in nncase. The nncase compiler performs the compression, parameterization, and compilation of the model to the .kmodel format (indication 5). The .kmodel model is executed by the KPU from the device's SD card (indication 6).

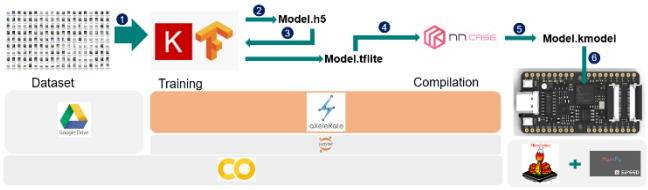

Fig. 6. Training and compilation with aXeleRate

C. Deep Learning Model Selection

We selected the MobileNet deep learning model from those compiled by nncase. This architecture is efficient in terms of fine-grained recognition, accuracy, and low computational cost [15]. Version 0.75 MobileNet-224 v1 was used in the project. The comparison between the different versions of MobileNet and the fine-grained recognition benchmark, in this case Inception v3, is shown in Table II. The Stanford Dogs dataset was used to assess this capacity of the compared networks.

TABLE II. MODEL COMPARISON USING STANFORD DOGS, ADAPTED FROM [15]

Model	Top 1 Accuracy	Million Parameters
Inception v3	84%	23.2
1.0 MobileNet-224	83.3%	3.3
0.75 MobileNet-224	81.9%	3.3
1.0 MobileNet-192	81.9%	1.9

V. METHODOLOGY

This section describes the training methodology for the deep learning algorithm, building the Edge AI prototype and field tests for the study.

A. Edge AI prototype construction

The prototype was built with the SiPEED board to carry out field tests and capture images of the conveyor belt. A prototype was built from the SiPEED board to obtain photos of the belt and field tests. The prototype is shown in Fig. 7.

Fig. 7. Edge AI prototype.

For the tests, three Python scripts were developed. The first to capture photos in the field with 224x224 resolution and storage on the SD card. The second for testing the model from the validation dataset previously stored on the SD card. The third for damage classification tests in the field with storage of the classified photos with this situation on the SD card.

B. Training the Deep Learning Model

The data set was developed to train the deep learning model. For the dataset 291 photos of the damaged belt (tear) and 291 photos of the normal belt were taken. The damage simulations were carried out by the maintenance team and pictures of these situations were taken with the belt stopped and in motion.

The photos were taken with SiPEED itself using the 224x224 resolution appropriate to the MobileNet input format. Examples of these images are shown in Fig. 8.

Data augmentation techniques were applied to increase the number of images available for training bringing the total number of images to 873 images with tears and 873 without. The images of each class were divided into training images 794 and 79 verification images.

(a) (b)

Fig. 8. Images of conveyor belt: (a) Without tear. (b) With tear.

The 0.75 MobileNet architecture was configured as a classifier, with 224 inputs, 2 fully-connected layers with 100 and 50 neurons, and a dropout of 0.5. The training was carried out using aXeleRate. Thirty training rounds were carried out and the learning rate adopted was 0.001. The initial weights of the model were based on training the model with an ImageNet dataset.

C. Experiments

To analyze the results, confusion matrices were used at the same time as the precision (1), recall (2) and overall accuracy F1 (3). Where TP is truly positive, FP is false positive, TN is true negative and FN is a false negative.

$$precision = \frac{TP}{TP+FP} \quad (1)$$

$$recall = \frac{TP}{TP+FN} \quad (2)$$

$$F1 = \frac{2*precision*recall}{precision+recall} \quad (3)$$

To test the KPU's performance, the second script developed for the prototype. In this script, the model previously trained and compiled by aXeleRate is loaded from the SD card. The same verification images used during the training process are stored on the SD card and pass through the KPU classification through the script.

The prototype positioned close to the belt so that the underside of the belt was in the SiPEED's field of view. The maintenance team simulated cuts in the belt and it was activated so that cuts would pass in front of the prototype. It was defined that each simulated cut would pass in front of the prototype 10 times.

VI. Results

This section presents the results of training the deep learning model as well as the performance of the prototype in field tests.

A. Training Model Performance

The 0.75 MobileNet-224 model was trained with the previously prepared image dataset. The training was carried out with the use of the aXeleRate platform being performed on Google Colab. The backend weights used in the training process was the imagenet.

The training was carried out in 24 minutes reaching a 97.4% accuracy as shown in the graph in Fig. 9.

Fig. 9. Training graph

The model was evaluated using the set of verification images separate from the original dataset. Altogether there were 79 images with tears and 79 without tears. The confusion matrix is shown in Fig. 10. The performance indicators can be seen in Table III.

		Predict: Rip	Predict: Normal
Real	Rip	75	4
	Normal	2	77

Fig. 10. Confusion matrix of the model - Google Colab.

TABLE III. Trained Model Performance at Google Colab.

Indicator	Value
Precision	95%
Recall	97%
F1	96%

B. Model Performance At Edge AI

The model trained and compiled by aXeleRate was loaded into SiPEED MaIX Bit to test its efficiency. The confusion matrix is shown in Fig. 11 and its performance indicators in Table IV.

		Predict: Rip	Predict: Normal
Real	Rip	69	10
	Normal	1	78

Fig. 11. Confusion matrix of the model - SiPEED.

TABLE IV. Trained Model Performance at SiPEED.

Indicator	Value
Precision	87%
Recall	99%
F1	93%

As can be seen, there is a loss of performance of 8.4% in accuracy and 3.1% in F1. The recall amount grows by 2.1%. These performance reductions do not prevent the use of Edge AI and deep learning models in the detection of tears.

Models are trained with weights and biases represented with 32-bit floating-point numbers, which are then quantized (i.e. discretized) to some specific values. We can then represent using integers instead of floating-point numbers. Converting 32-bit floating-point numbers to 8-bit integers reduces the memory usage by 4x and allows us to take advantage of specialized hardware for performing inference faster, but it comes with the price in the form of slight accuracy reduction. The precision loss in quantized models is a common occurrence - the accuracy difference between original and quantized models can be further lowered using larger calibration dataset or using quantization-aware training.

C. Field Test Performance

Field tests were carried out in 4 experiments. In tests, the prototype was installed next to the belt to visualize the simulated cuts. The installation location is shown in Fig. 12 and one of the simulated longitudinal cuts in Fig. 13.

Fig. 12. Prototype installation location

Fig. 13. Simulated longitudinal cut

In each test, the prototype was exposed to 10 regions with longitudinal rip and 10 regions without damage, in these situations the belt was in motion. For each situation, SiPEED took a photo to record the event.

The 4 experiments totaled 80 exposures with the results presented in the confusion matrix of Fig. 14 and the performance of Table V. One of the detected tears is shown in Fig. 15.

Comparing the results of the field tests with the tests carried out with SiPEED classifying images from the dataset, an increase of 3% in accuracy and 1% in F1 is verified. In the same comparison, there is a 2% reduction in Recall.

Field tests were performed during the day in situations of indirect sunlight. The lighting value measured in the test region was between 600 and 1500 lux.

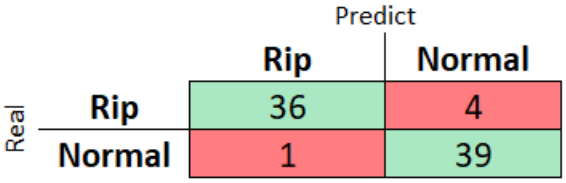

Fig. 14. Confusion matrix of field tests.

TABLE V. PERFORMANCE OF FIELD TESTS.

Indicator	Value
Precision	90%
Recall	97%
F1	94%

Fig. 15. Detection in field test – *Com rasgo* means *with tear*

VII. CONCLUSION

The feasibility of using edge AI running a deep neural network to detect longitudinal rip on a conveyor belt was demonstrated.

The process of converting and compiling the cloud-trained model for use on the device edge worked despite the 9% loss inaccuracy.

In the experiments conducted during the development of the work, the prototype reached an accuracy of 90% and an overall accuracy of 94% in the detection of the simulated longitudinal rip on the conveyor belt.

The results obtained encourage the advancement of the development of the work taking into account new operating conditions including conveyor belts with repairs and mud stains as well as environmental conditions such as higher dust levels and different lighting situations.

In parallel with the development of the solution, the study of similar works in search of parameters for a better evaluation of the overall performance of the developed system should be further developed.

ACKNOWLEDGMENT

The authors would like to thank Vale, CAPES, CNPq, and the Federal University of Ouro Preto for supporting this work.

REFERENCES

[1] A. Koul, S. Ganju, M. Kasan, Pratical Deep Learning for Cloud, Mobile, and Edge: Real-World AI & Computer-Vision Projects Using Pyhton, Keras & TensorFlow. 1st ed., O'Reilly Media, United States of America, 2020.

[2] A. Krizhesky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", Neural Information Processing System Conference, 2012.

[3] Raghu, Maithra, and Eric Schmidt. "A survey of deep learning for scientific discovery." arXiv preprint arXiv:2003.11755 (2020).

[4] A. Malik, A. Gupta, Artificial Intelligence at the Edge. 1st ed., Amazon Publishing, United States of America, 2020.

[5] L. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, & A. Y. Zomaya, "Edge intelligence: the confluence of edge computing and artificial intelligence", IEEE Internet of Things Journal, 2020.

[6] [NI], "Google Coral". https://coral.ai/software/. 2020.

[7] Sunnycase, "Kendrite nncase". https://github.com/kendryte/nncase. 2020.

[8] [NI], "OpenVino Intel". https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html. 2020.

[9] L. Deng, G. Li, S. Han, L. Shi and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey", IEEE, vol 108, No. 4, 2020.

[10] X. Li, L. Shen, Z. Ming, C. Zhang and H. Jiang, "Laser-based on-line machine vision detection for longitudinal rip of conveyor belt", Optik, v. 168, pp. 360-369, 2018.

[11] H. Chengchegn, Q. Tiezhu, Q. Meiying. X. Xiaoyan, Y. Yi and Z. Haitao. "Researhc on audio-visual detection method for conveyor belt longitudinal tear", IEEE Access, vol VII, pp. 120202-120213, 2019.

[12] A. A. Santos, F. A. S. Rocha, H. Azpúrua, A. J. R. Reis and F. G. Guimarães, "Automatic system for visual inspection of belt conveyors", SBA, 14° Intelligent Automation Symposium, pp. 1192-1197, 2019.

[13] Majidifard, Hamed, et al. "Pavement Image Datasets: A New Benchmark Dataset to Classify and Densify Pavement Distresses." Transportation Research Record 2674.2 (2020): 328-339.

[14] D. Maslov "Image Recognition With K210 Boards and Arduino IDE/Micropython". https://www.instructables.com/id/Transfer-Learning-With-Sipeed-Mai X-and-Arduino-IDE. 2020.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, & H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv preprint arXiv:1704.04861. 2017.