

Nomes: Guilherme Vilas Boas Ferreira da Silva Matrículas: 2017011836
Laura Pellizari Pereira 2019018756
Mateus Dumas Lima 2017017965

Anexo (links para os projetos públicos no Edge Impulse):

Modelo final: <https://studio.edgeimpulse.com/public/42837/latest>

Modelo original: <https://studio.edgeimpulse.com/public/42859/latest>

Modelo com transfer learning: <https://studio.edgeimpulse.com/public/43006/latest>

Existe também na pasta de entrega o arquivo .ino para a aplicação do modelo no Arduino e também um arquivo de python com o notebook onde se fez os primeiros testes para se verificar a viabilidade de um modelo para detecção de incêndios florestais.

Relatório:

Uma vez que o objetivo do trabalho era o de construir um modelo que conseguisse fazer a classificação de incêndios e não incêndios, foi necessário primeiramente obter uma base de dados que contivesse ambas as situações. Entretanto, apesar da procura por uma única que tivesse um grande número de dados, não se achou nenhuma que fosse robusta o suficiente para ser usada unicamente. Desta forma, decidiu-se por fazer a junção de diferentes datasets de fogo e não fogo, das mais diferentes formas. Os datasets utilizados neste trabalho foram:

1) <https://data.mendeley.com/datasets/gjmr63rz2r> :

O conjunto de dados projetado para problemas binários de detecção de incêndio ou não-incêndio na paisagem florestal. É um conjunto de dados balanceado que consiste em 1900 imagens no total, onde 950 imagens pertencem a cada classe. O conjunto de dados foi dividido em 80:20 para fins de treinamento e teste.

2) <https://www.kaggle.com/atulyakumar98/test-dataset> :

650 imagens de situações binárias de inocência ou não-incêndios em diferentes lugares, incluindo casos domésticos.

3) <https://www.kaggle.com/phylake1337/fire-dataset> :

Os dados foram coletados para treinar um modelo para distinguir entre as imagens que contêm fogo (imagens de fogo) e imagens regulares (imagens que não são de fogo), isto é, um problema de classificação binária. Os dados são divididos em 2 pastas, a pasta de imagens de fogo contém 755 imagens de fogo ao ar livre onde algumas delas contêm fumaça pesada. Já a outra consiste em imagens de não fogo que contêm 244 imagens da natureza.

4) Dataset pessoal composto por 64 imagens de não incêndio, mais especificadamente de situações de pôr do sol na floresta. Este grupo de dados foi criado com o objetivo de mitigar o problema identificado em testes dos primeiros modelos onde foi observado o problema onde o modelo classificava regularmente o pôr do sol como um incêndio.

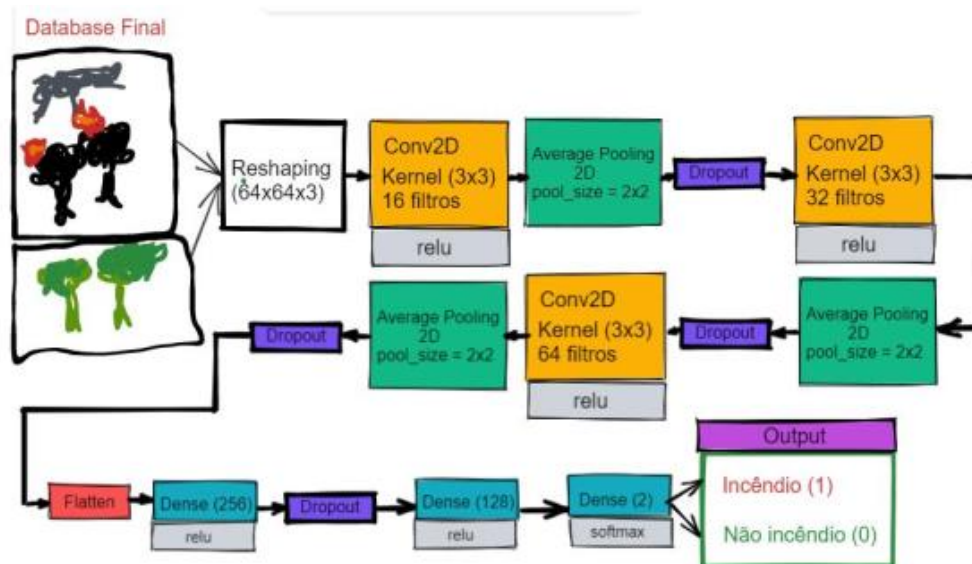
Depois de juntar os dados de cada dataset propriamente passou-se para a etapa de feature engineering, isto é, a de tratamento de dados. As implementações aplicadas nesta etapa foram a de redimensionamento e normalização.

Primeiramente, sabendo que os dados juntados a partir de diferentes datasets vinham com diferentes tamanhos, tinha-se o problema em que o modelo a ser treinado podia ter um viés anormal para um tipo de imagem, onde um dado que tivesse uma dimensão maior poderia receber uma prioridade maior por parte do modelo do que uma imagem de tamanho menor. Para se evitar isso, aplica-se o redimensionamento em todos os dados de entrada, ou seja, faz-se que com todas as imagens tenham o mesmo tamanho. Esse valor foi definido no final do trabalho como 32x32x3, onde se conseguiu um bom desempenho em termos de accuracy e uma redução de gasto de memória maior do que fazendo o redimensionamento para 64x64x3.

Além disso, é uma boa prática em um projeto de machine learning treinar dados que variem na faixa de 0 a 1. Para se conseguir isso, sabendo que as imagens são RGB e com isso cada pixel variará de 0 a 255, é necessário fazer a divisão de cada pixel pelo valor máximo possível, que é 255.

Passada esta etapa, prosseguiu-se para o desenvolvimento do modelo. Inicialmente, usou-se o modelo trabalhado no paper: “Automatic Fire and Smoke Detection Method for Surveillance Systems Based on Dilated CNNs”: <https://www.mdpi.com/2073-4433/11/11/1241/html>

Este modelo é representado pela figura abaixo:



Neste relatório se mostrará os resultados obtidos a partir da aplicação do modelo na plataforma Edge Impulse. A criação do impulso, junto com os resultados do treinamento deste modelo são mostrados nas imagens a seguir:

Image data

Axes

image

Image width

64

Image height

64

Resize mode

Fit shortest

For optimal accuracy with transfer learning blocks, use a 96x96 or 160x160 image size.

Image

Name

Image

Input axes

☒ image

Classification (Keras)

Name

NN Classifier

Input features

☒ Image

Output features

2 (fire, no_fire)

Output features

2 (fire, no_fire)

Save Impulse

Model

Model version: Quantized (int8)

Last training performance (validation set)

%

ACCURACY

94.0%

LOSS

0,28

Confusion matrix (validation set)

| | FIRE | NO_FIRE |
|----------|-------|---------|
| FIRE | 93.7% | 6.3% |
| NO_FIRE | 5.6% | 94.4% |
| F1 SCORE | 0.94 | 0.94 |



Após isso, aplicou-se o modelo nos dados ainda não usados de teste. O resultado foi o seguinte:

Model testing results

ACCURACY
90.50%



| | FIRE | NO_FIRE | UNCERTAIN |
|----------|-------|---------|-----------|
| FIRE | 92.1% | 6.6% | 1.4% |
| NO_FIRE | 8.3% | 88.9% | 2.8% |
| F1 SCORE | 0.92 | 0.91 | |

Depois de se trabalhar com este modelo, decidiu-se por tentar aplicar a técnica de transfer learning no dataset usado. Neste caso, apenas utilizou-se os parâmetros definidos por padrão pela plataforma, contando com o uso do modelo MobileNetV2 0.35. A criação do impulso, junto com o modelo e os resultados do treinamento deste são mostradas abaixo:

Image data

Axes
image

Image width
96

Image height
96

Resize mode
Fit shortest

For optimal accuracy with transfer learning blocks, use a 96x96 or 160x160 image size.

Image

Name
Image

Input axes
☒ image

Transfer Learning (Images)

Name
Transfer learning

Input features
☒ Image

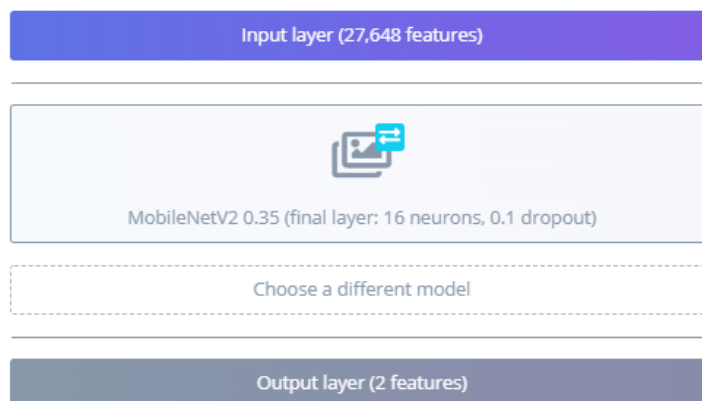
Output features
2 (fire, no_fire)

Output features

2 (fire, no_fire)

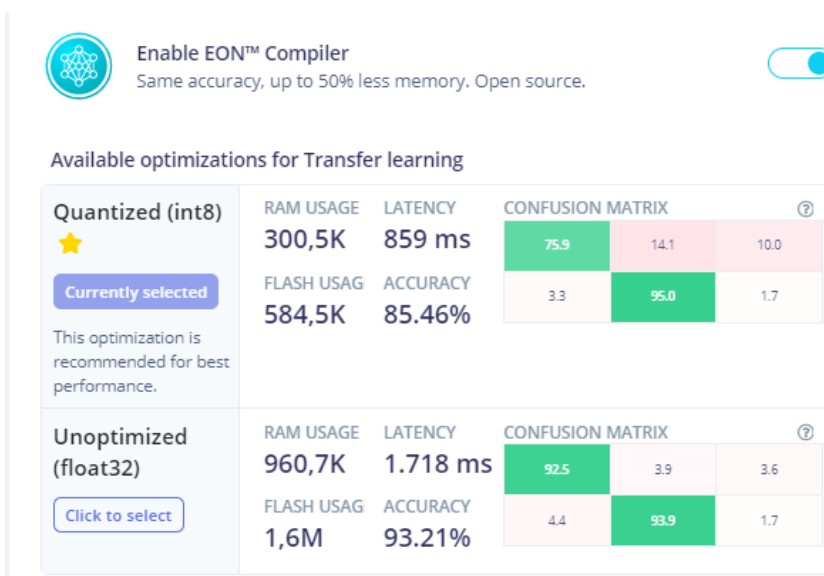
Save Impulse

Neural network architecture





Pode-se ver também o desempenho do modelo para os dados de teste de acordo com o método de quantização pós-treinamento:

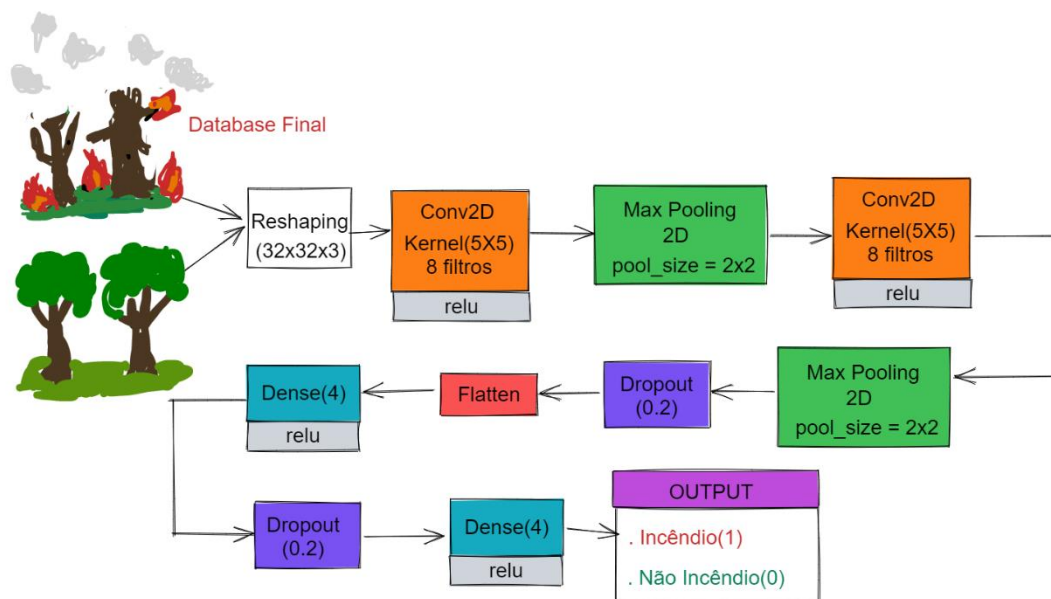


Após o teste deste modelo, decidiu-se por trabalhar em cima do primeiro apresentado, que foi construído a partir do bloco de classificação do tipo neural network. Com isso,

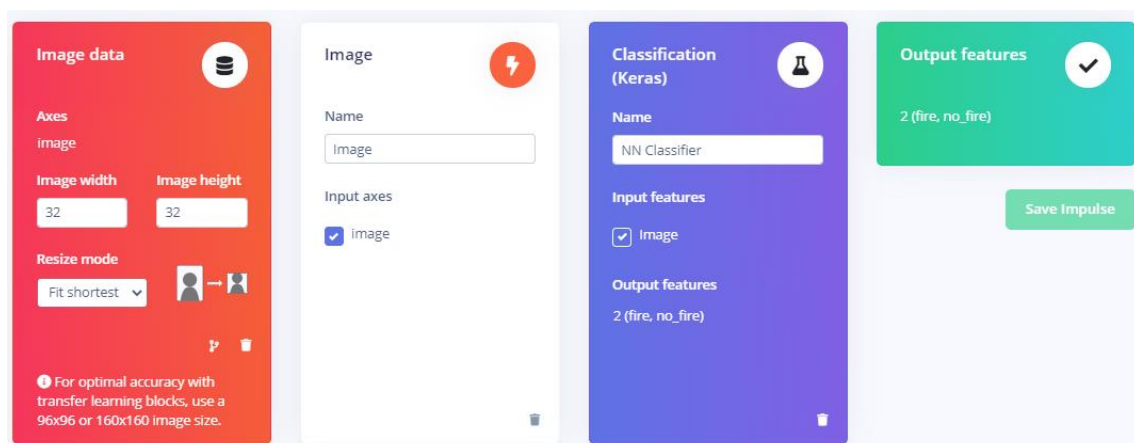
foram aplicadas diversas tentativas aonde se mudava alguns layers e parâmetros do modelo, a fim de se chegar a um que seja o mais otimizado possível, de acordo com as métricas de accuracy, gasto de memória e latência.

Ainda que o padrão era mantido, isto é, continuava-se a se usar layers convolucionais 2d, maxPooling, dropout e layers dense, tentou-se diminuir o número de neurônios nos layers dense assim como o número de filtros nos layers convolucionais, além de retirar alguns layers desses tipos que se julgavam desnecessários para o modelo no geral. Estas práticas foram realizadas sobretudo para reduzir o número de parâmetros totais do modelo e consequentemente diminuir o gasto de memória requerido.

O modelo final, que foi capaz de ter um bom resultado tanto em accuracy como gasto de memória e latência é apresentado na imagem abaixo:



O processo de criação do impulso no Edge Impulse, junto com os resultados do treinamento do modelo são mostrados nas imagens abaixo:





Já o desempenho do modelo com os dados de teste foi da seguinte maneira:



Com os resultados apresentados dos três diferentes modelos, é interessante compará-los de acordo com as métricas mais importantes de um projeto de tinyML que são o accuracy, gasto de memória e latência, de forma que possa se ter a ideia de qual modelo é o mais indicado para ser usado na placa de desenvolvimento utilizada, que no caso é o Arduino nano BLE 33 sense, que possui 256 KB de memória RAM e 1MB de memória flash. A tabela comparativa entre os modelos é mostrada a seguir:


| Modelo | Latência | Gasto de RAM | Gasto de Flash | Accuracy |
|-------------------|----------|--------------|----------------|----------|
| Original | 1,756s | 84,5 KB | 1100 KB | 90,77% |
| Transfer Learning | 0,859s | 300,5 KB | 584,5 KB | 85,46% |
| Final | 0,147s | 13,8 KB | 35,6 KB | 89,53% |

Analisando a tabela, pode-se perceber que tanto o modelo original quanto o de transfer learning não são funcionais para a aplicação proposta neste trabalho, pois ambos extrapolam os valores limites de memória do Arduino usado, onde o modelo original requer uma memória Flash maior do que a oferecida pelo dispositivo enquanto que o modelo construído a partir do Transfer Learning precisa de uma memória RAM maior do que o Arduino pode aguentar.

Enquanto isso, o modelo final oferece um bom desempenho em todas as métricas. Em relação ao gasto de memória, ele necessita de valores facilmente oferecidos pelo Arduino. Já com respeito ao accuracy, onde é mostrado o desempenho coletado com os dados de treino, se vê que ele mantém um valor bem próximo dos demais, na casa dos 90%. Por fim, analisando a latência, percebe-se o melhor desempenho se comparado com os demais, oferecendo uma latência bem baixa levando em conta os seus concorrentes.

Desta forma, concluindo que o modelo final se configurou como o melhor dentro de todos os modelos testados, pôde-se seguir para a etapa de deployment do modelo diretamente no Arduino, em conjunto com o módulo de câmera OV7675 que possui uma resolução de 176x144, no tipo QCIF.

Para fazer isso, bastou ir na seção Deployment no Edge Impulse, escolher a otimização que desse o melhor desempenho pro modelo que no caso foi a quantização do tipo int8 e então clicar em build.



Enable EON™ Compiler
 Same accuracy, up to 50% less memory. Open source.

Available optimizations for NN Classifier

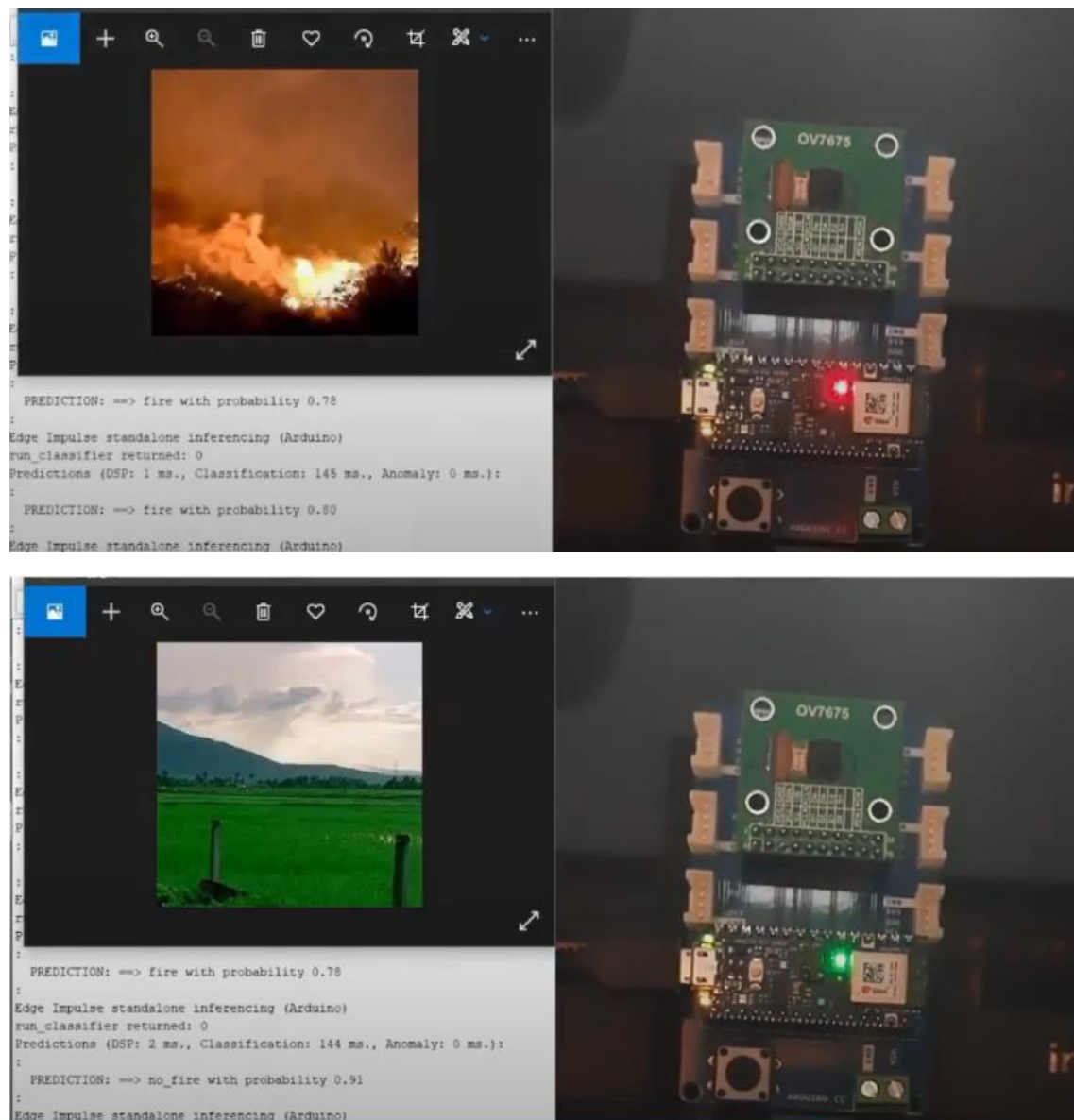
| | Quantized (int8) | Unoptimized (float32) | | | | | | | | | | | | |
|-------------------------|---|---|-----|-----|-----|------|-----|---|------|-----|-----|-----|------|-----|
| Quantized (int8) | <div>★</div> <div>Currently selected</div> <div>This optimization is recommended for best performance.</div> | <div>Unoptimized (float32)</div> <div>Click to select</div> | | | | | | | | | | | | |
| RAM USAGE | 13,8K | 46,7K | | | | | | | | | | | | |
| FLASH USAG | 35,6K | 47,4K | | | | | | | | | | | | |
| LATENCY | 147 ms | 691 ms | | | | | | | | | | | | |
| ACCURACY | 89.53% | 89.39% | | | | | | | | | | | | |
| CONFUSION MATRIX | <table border="1"> <tr> <td>92.9</td> <td>3.6</td> <td>3.6</td> </tr> <tr> <td>9.7</td> <td>86.2</td> <td>4.1</td> </tr> </table> | 92.9 | 3.6 | 3.6 | 9.7 | 86.2 | 4.1 | <table border="1"> <tr> <td>92.3</td> <td>3.6</td> <td>4.1</td> </tr> <tr> <td>9.7</td> <td>86.5</td> <td>3.9</td> </tr> </table> | 92.3 | 3.6 | 4.1 | 9.7 | 86.5 | 3.9 |
| 92.9 | 3.6 | 3.6 | | | | | | | | | | | | |
| 9.7 | 86.2 | 4.1 | | | | | | | | | | | | |
| 92.3 | 3.6 | 4.1 | | | | | | | | | | | | |
| 9.7 | 86.5 | 3.9 | | | | | | | | | | | | |

Estimate for Cortex-M4F 80MHz

Build

Com isso, foi possível baixar a biblioteca com o modelo pronto para ser aplicado no Arduino. Após isso, abriu-se o exemplo no arquivo static buffer e implementou-se as mudanças necessárias a fim de que se fizesse a inicialização da câmera e o tratamento de dados. Uma vez que a classificação tenha sido feita, o procedimento de pós inferência foi o seguinte: caso o incêndio tenha sido detectado acendia-se o led vermelho, em caso

contrário era o led verde que ficava aceso. Algumas imagens do funcionamento do modelo no Arduino são mostradas a seguir:



Foi possível ver com clareza o bom funcionamento do projeto na prática. No entanto, há algumas melhorias que podem ser feitas de forma a deixar o protótipo ainda mais eficiente em fazer as classificações. Elas são:

- . Usar uma câmera com maior resolução e qualidade, já que a usada possui certas deficiências em suas imagens capturadas que geram uma dificuldade maior no modelo em identificar cada situação, se comparado com as imagens que foram usadas no dataset de entrada.
- . Aplicar técnicas de janelamento nas classificações, onde se considera uma média de classificações consecutivas para se fazer a inferência, de forma a se evitar situações momentâneas que podem induzir ao modelo fazer uma classificação errada.
- . Implementar mais medidas de pós-inferência como a comunicação LoRa, pensando em reais aplicações deste modelo. Já que se sabe que um possível protótipo de detector de

incêndios florestais seria instalado em áreas afastadas da cidade, seria necessário um modo de comunicação que pudesse transmitir uma mensagem por longas distâncias e que também consumisse pouca energia, uma vez que a manutenção não seria trivial e facilmente feita. Nesta situação, a tecnologia LoRa atende a ambos requisitos. A fim de se economizar ainda mais energia, podia-se programar de tal forma o módulo LoRa onde ele seria ativado apenas depois do modelo identificar um incêndio, ou seja, o módulo ficaria “dormindo” todo o tempo em que detectasse não incêndios, e, portanto, salvaria ainda mais energia.

- . Aplicação da técnica de sensor fusion, em que se poderia utilizar os próprios sensores de temperatura e humidade do Arduino para fazer com que a classificação ficasse ainda mais robusta.

- . Utilização de uma câmara infravermelha, que neste caso seria capaz de corrigir completamente o problema onde o modelo em algumas vezes confundia o pôr do sol como uma situação de incêndio.