UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Lucas Felipe Moreira Magalhães

Detecção de Falhas em Cultivo de Milho

Uberlândia, Brasil

2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Lucas Felipe Moreira Magalhães

Detecção de Falhas em Cultivo de Milho

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Jefferson Rodrigo de Souza

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2019

Lucas Felipe Moreira Magalhães

Detecção de Falhas em Cultivo de Milho

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 20 de dezembro de 2019:

Prof. Dr. Jefferson Rodrigo de Souza
Orientador

Prof.ª Dra. Maria Adriana Vidigal de Lima

Prof. Dr. Paulo Henrique Ribeiro Gabriel

Uberlândia, Brasil
2019

Agradecimentos

Gostaria de agradecer aos meus pais por batalharem pela minha educação, por sempre me ajudarem em meu caminho e incentivarem o meu estudo. Aos meus avós que acreditaram em mim e por me acolherem e mostrarem a importância do caminho da educação. A minha madrinha Vivianny por ser minha heroína e por ter me acolhido e acreditado em meu futuro, e a meu amor Angélica por ter me apoiado toda esta trajetória.

"If you're not making someone else's life better, then you're wasting your time. Your life will become better by making other lives better." - Will Smith

Resumo

A civilização começou a utilizar a agricultura a bastante tempo, com o desenvolvimento da humanidade ao longo dos anos, trouxeram-se diversos avanços tecnológicos para o campo. Com o aumento das áreas de cultivo, então novas pesquisas sobre o desempenho e a qualidade iniciaram-se. Este trabalho de conclusão de curso apresenta o procedimento para a obtenção das falhas nas plantações de milho sob a área da Agricultura de Precisão. Assim sendo, a fim de melhorar a visualização das imagens aéreas, as quais foram extraídas por meio dos Veículos Aéreos Não Tripulados. Com a utilização do Aprendizado de Máquina juntamente aplicada com os recursos morfológicos aplicados à imagens, obtiveram-se as falhas nas plantações através do trabalho conjunto de ambas as técnicas computacionais.

Palavras-chave: Milho, Falhas, Aprendizado de Máquina e Operadores Morfológicos.

Lista de ilustrações

Lista de abreviaturas e siglas

GPS	Global Positioning System
MDIC	Ministério da Indústria, Comércio Exterior e Serviços
PDI	Processamento Digital de Imagens
VANT	Veículo Aéreo Não Tripulado
ML	Machine Learning
API	Application Programming Interface
EXIF	Exchangeable Image File Format
MART	Multiple Additive Regression Tress
HL	Hough Lines
HLP	Hough Lines Probabilistic
GSD	Ground Sample Distance

Sumário

1 Introdução

Em 2017, o Brasil bateu recorde de exportação de milho de acordo com o Ministério da Indústria, Comércio Exterior e Serviços (MDIC), ao todo foram exportadas 29,25 toneladas de milho. As plantações no Brasil são divididas em duas safras, que diferenciam a quantidade de hectares plantados em cada safra. Assim, com o uso de técnicas agrícolas em conjunto com tecnologia, os sensores e as máquinas para obter os melhores desempenhos em problemas no campo, então assume o nome de Agricultura de Precisão.

O uso de Veículos Aéreos Não Tripulados (VANTs) é importante na área de Agricultura de Precisão para obter dados no campo e que através de softwares proporcionam ótimas opções de ferramentas agrícolas, assim aumentando a excelência nas lavouras, possibilitando o melhor planejamento agrícola.

As imagens dos VANTs possibilitam o processamento por softwares em áreas de plantações em imagens com o objetivo de monitorar áreas, com alta qualidade espaço-temporal, e prover segurança aos agricultores e aumentando a competitividade no agronegócio.(OLIVEIRA H.; SOUZA et al., 2018).

Com o avanço digital, surgiram diversas tecnologias que providenciaram a criação deste trabalho com intuito de prover um programa que melhore o processo manual de contagem de falhas. Este trabalho realiza a contagem através de imagens obtidas por VANTs, possibilitando uma automatização desta tarefa e assim diminuindo o esforço manual. Assim proporcionando uma melhora significativa para os profissionais da área.

A contagem de falhas é um processo para se conseguir estimar várias informações, como por exemplo a qualidade de um cultivo realizado em duas áreas que possuem diferentes nutrientes. Outro exemplo é o cálculo para cultivos de sementes diferentes visando qual possui melhor produtividade da área.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolver um sistema de detecção de falhas de plantio em cultivo de milho usando processos morfológicos com ML para realizar a predição das falhas entre plantas.

1.1.2 Objetivo Específicos

- Caracterizar as imagens para uma separação angular.

- Realizar as etapas de processamento visual.
- Realizar a predição de plantas de milho.
- Utilizar operadores morfológicos para transformar as imagens.
- Definir os valores dos parâmetros para cada operador morfológico.
- Realizar a contagem de falhas de plantio na plantação de milho.

1.2 Organização do Trabalho

No primeiro capítulo, conceitua-se sobre o problema apresentado, além disto mostra os objetivos deste trabalho e a metodologia.

No segundo capítulo, descreve a fundamentação teórica, onde constitui as informações necessárias para o entendimento deste trabalho.

No terceiro capítulo, apresenta a metodologia, então apresentando as tecnologias utilizadas, assim como os frameworks.

No quarto capítulo, mostra o desenvolvimento, como ocorreu os operadores morfológicos, e a predição em uma imagem capturada pelo VANT sobre o plantio de milho.

O quinto capítulo apresenta a conclusão do trabalho com os resultados.

2 Fundamentação Teórica

Na atualidade, os técnicos e os produtores rurais já usufruem das tecnologias de Agricultura de Precisão, o seu conceito já abrange muitas plantações expondo as características do relevo, solo, vegetação e também os contextos passados.

A Agricultura de Precisão visa o melhoramento da cultura, independentemente do tamanho da área cultivada. Desta maneira, o responsável pela cultura deve observar, medir e registrar estas mudanças que acontecem no decorrer do ciclo agrícola. Tais observações ajudam a recomendar um tratamento específico para cada situação (BERNARDI et al., 2014).

2.1 VANT

Como a para a Agricultura de Precisão, os VANTs foram desenvolvidos para atender o mercado, então facilitou o estado atual do progresso tecnológico. Devido ao avanço tecnológico (da década de 80 até a atual), então diminuíram seu preço e dimensão.

O VANT é um dispositivo capaz de ficar acima do solo, controlado remotamente ou de forma autônoma, sem a presença de um ser humano dentro, e que pode ser equipado com GPS e sistemas inerciais. Também pode ser equipado com câmera RGB, multiespectral ou hiperspectral (JORGE et al., 2014).

2.1.1 Modelos de VANT

Hoje podem ser encontrados dois tipos de modelos de VANTs:

- Multi-rotores: Bastante utilizados pois, possuem capacidade de planar e fazer voos verticais, proporcionando mais utilidade para o dia a dia no campo quando se procura usar fotogrametria terrestre.
- Asa Fixa: Possui vantagem sobre os multi-rotores devido a capacidade de realizar voos mais longos, conseguindo realizar mapeamentos maiores que os multi-rotores, mas não possuem a capacidade de planar.

2.2 Aprendizado de Máquina

Os computadores são instrumentos importantes para a população no dia-a-dia. Com o aumento das informações que circulam pelo mundo, surgiram-se novas oportunidades para descobrirem novas áreas da tecnologia. O ML é a área de estudo que proporciona

aos computadores a habilidade de dominar um conhecimento sem possuírem um código explícito para aquele conceito (SIMON, 2013).

Dentre as técnicas computacionais mais conhecidas em ML, tem-se as Redes Neurais Artificiais, a qual tem como base do seu algoritmo reproduzir estrutura funcional da rede neural biológica. Assim para cada neurônio existe na estrutura um elemento, o perceptron. Os neurônios se comunicam através de sinapses, que são impulsos nervosos que são transmitidos entre eles. Analogicamente os perceptrons se comunicam com sinais, tais sinais são recebidos e processados de maneira que produzam uma saída. As redes neurais são organizadas em camadas, onde os perceptrons ficam conectados aos consecutivos. A figura 1 é exemplo de uma rede neural organizada em camadas.

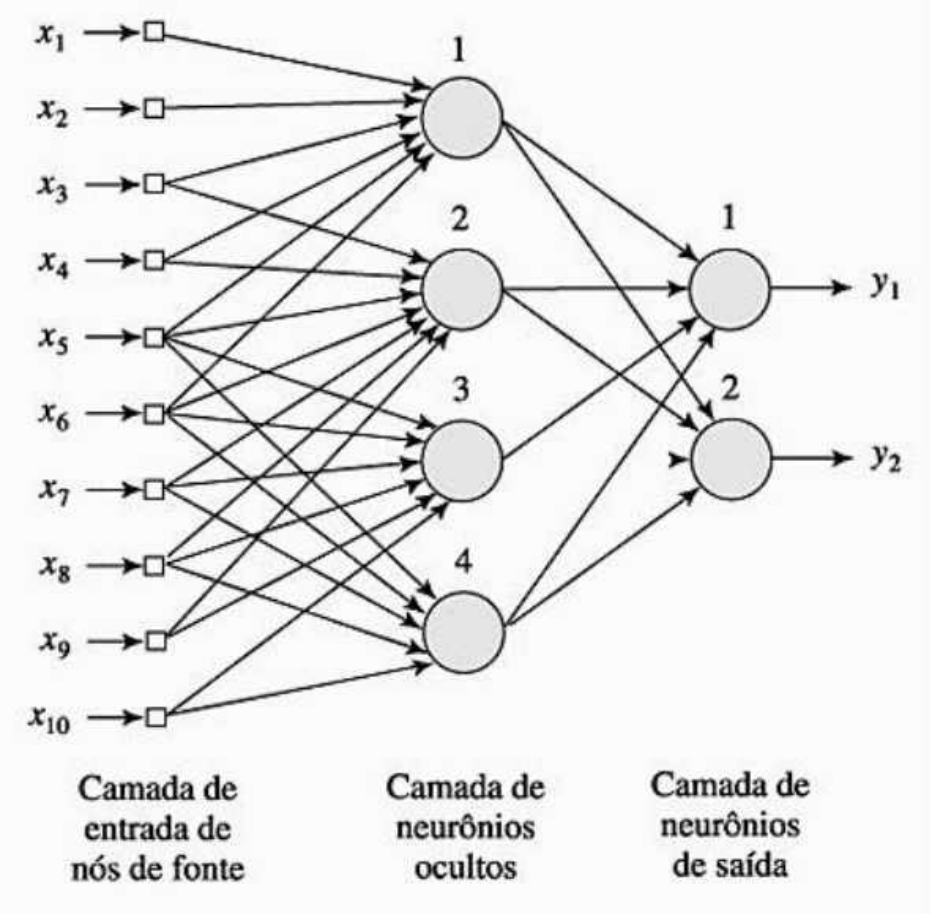

Figura 1 – Organização em camadas. (adaptado de Simon Haykin, 2001)

Conforme a figura 1, a rede neural possui quatro neurônios na camada oculta e em cada neurônio seis conexões locais, expressa-se para cada neurônio **j**, tal que $j \in [1, 2, 3, 4]$, o campo local induzido através da equação abaixo:

$$v_j = \sum_{i=1}^{6} w_i x_{i+j-1}, j = 1, 2, 3, 4$$

2.2.1 Aprendizado de máquina supervisionado

Um algoritmo de aprendizagem supervisionada é no qual à partir de um modelo esperado, prediz um rótulo para os dados ainda não vistos.

2.2.1.1 Árvores de Decisão

Árvore de decisão é um modelo de algoritmo de aprendizagem supervisionada com objetivo de realizar uma classificação através de uma variável (MICROSOFT, 2019a).

As árvores de decisão podem ser tipadas, dois tipos conhecidos são:

1. **Árvore de decisão de variável categórica:** A resposta da classificação é separada em classes. Por exemplo é prever a resposta de um valor que pode ser categorizado.
2. **Árvore de decisão de variável contínua:** A variável continua procura a resposta à partir de outras informações prevendo um valor desconhecido e de valor continuo.

O algoritmo de treinamento FastTree é uma implementação eficiente do algorítimo MART que é uma implementação dos métodos de aumento de árvores de gradiente para mineração de dados preditiva. Em aprendizado de máquina a técnica de aumento de gradiente é utilizada para problemas de regressão (MICROSOFT, 2018).

O algoritmo MART treina um conjunto de árvores que são árvores de decisão binárias. Os conjuntos de árvores produzidos são computados por etapas, onde cada árvore aproxima seu gradiente da função de perda (FRIEDMAN, 1999).

2.2.1.2 Clusterização

Outro tipo de aprendizado de máquina é a Clusterização, possui como objetivo organizar instâncias de dados em classes que possuem atributos semelhantes, pode ser utilizado para reconhecer padrões em uma coleção de informações. É útil de diversas maneiras, servindo como centroide, conectividade, densidade ou distribuição. A implementação mais conhecida é o K-Means(MICROSOFT, 2019b)

2.3 Morfologia Matemática

A morfologia matemática tem como intuito extrair informações através da geometria e topologia de um conjunto desconhecido, utilizando como base as transformações e um elemento estruturante para obter um conjunto conhecido. A base para toda esta lógica é a teoria de conjuntos (VALLE, 2017).

A morfologia matemática tem como estrutura uma imagem bidimensional, onde imagens binárias possuem pixeis pretos e brancos, ou seja, um conjunto de valores. O

operadores morfológicos são operações com funções estruturantes que modificam aquela entidade (BRADSKI G.; KAEHLER, 2008).

Através destes operadores morfológicos, é possível definir o milho como elemento estruturante, aplicar os transformadores e filtragens a partir da predição da imagem. Com isto, os operadores proporcionam benefícios como redução dos ruídos nas predições que não são importantes e estruturação das plantas para facilitar a contagem das falhas. Nas próximas seções seguem exemplos de operadores morfológicos utilizados.

2.3.1 Dilatação

A dilatação proporciona a criação de novas partículas, assim sendo, fechando os buracos e/ou conectando objetos de forma que aumenta os pixeis vizinhos da matriz utilizada, desta maneira conectando as plantas que estão por perto. Segue exemplo:

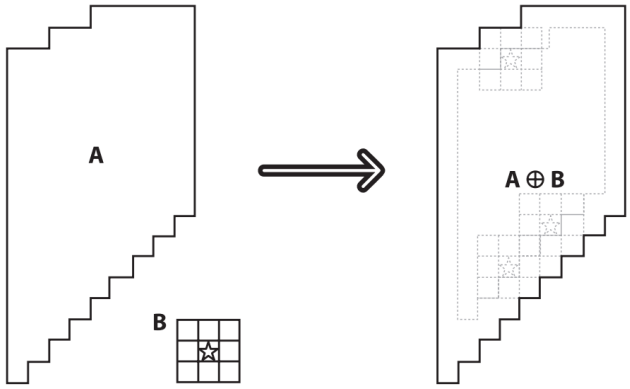

Figura 2 – Dilatação - adaptado de (BRADSKI G.; KAEHLER, 2008)

Na figura acima, utilizando é aplicado a dilatação na imagem **A** a partir do elemento **B**, com isto, para cada pixel da imagem é feita a validação aplicando o elemento no pixel. Se nos pixeis do elemento não existirem na imagem então é pintado estes pixeis, assim a imagem é aumentada.

2.3.2 Erosão

A erosão pela sua definição é a interseção de intervalo de pixeis conseguidos a partir da translação dos pixeis da imagem, assim acaba reduzindo o número de partículas, elimina os componentes irregulares ao elemento estruturante, aumento o espaço entre duas ou mais plantas, e permite realizar a separação de conjuntos de pixeis conectados. Segue exemplo:

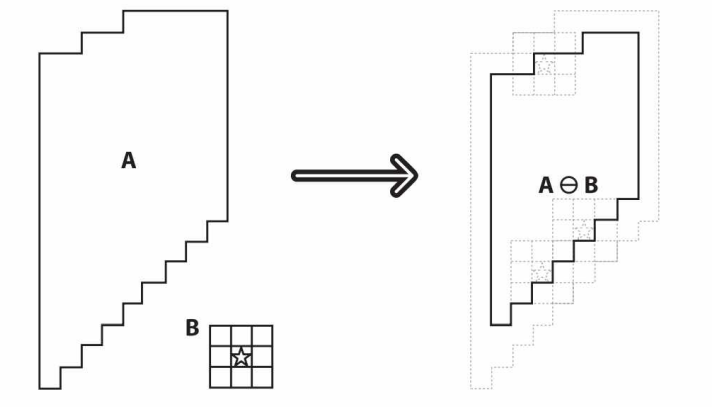

Figura 3 – Erosão - adaptado de (BRADSKI G.; KAEHLER, 2008)

O que acontece na Figura 3 é o inverso da Figura 2, ou seja, em vez de validar se não existir o pixel, é conferido se existe, então remove-se o pixel, assim ocorre uma diminuição da imagem, ou até mesmo o fim da imagem **A**.

2.3.3 Abertura

Possui a função de eliminar partículas de pixeis que não fazem parte da entidade desejada, ao mesmo tempo não desestruturando os modelos, ela é consiste em realizar o processo de erosão e em seguida o de dilatação, desta maneira o conjunto de pixeis é mais regular que o conjunto inicial e menos rico em detalhes que na imagem anterior. Segue exemplo:

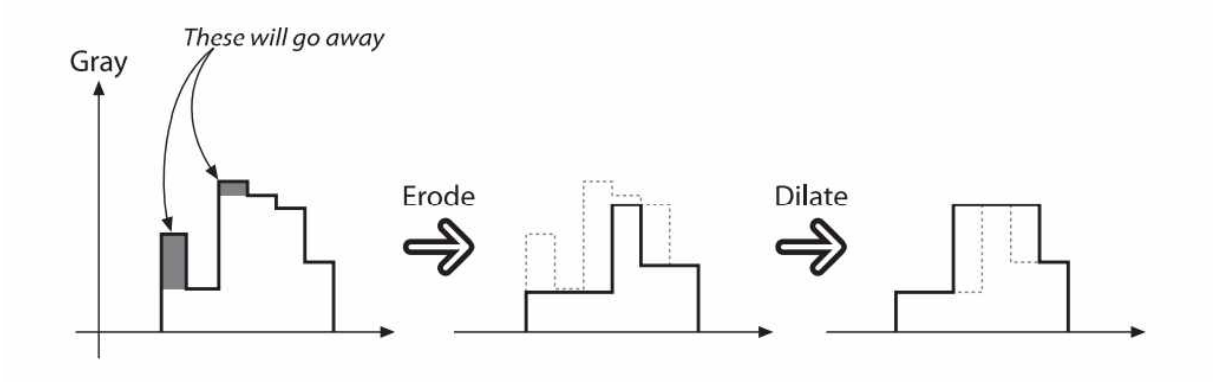

Figura 4 – Abertura - adaptado de (BRADSKI G.; KAEHLER, 2008)

2.3.4 Fechamento

O Fechamento possui a função de preencher espaços no interior dos componentes, porém inferior em tamanho da relação do elemento estruturante, pela sua definição, consiste em dilatar um conjunto de pixeis e em seguida o erodir, assim com este resultado,

aproxima-se do modelo proposto devido a sua forma mais regular e com menos detalhes, desta maneira suavizando a entidade. Segue exemplo:

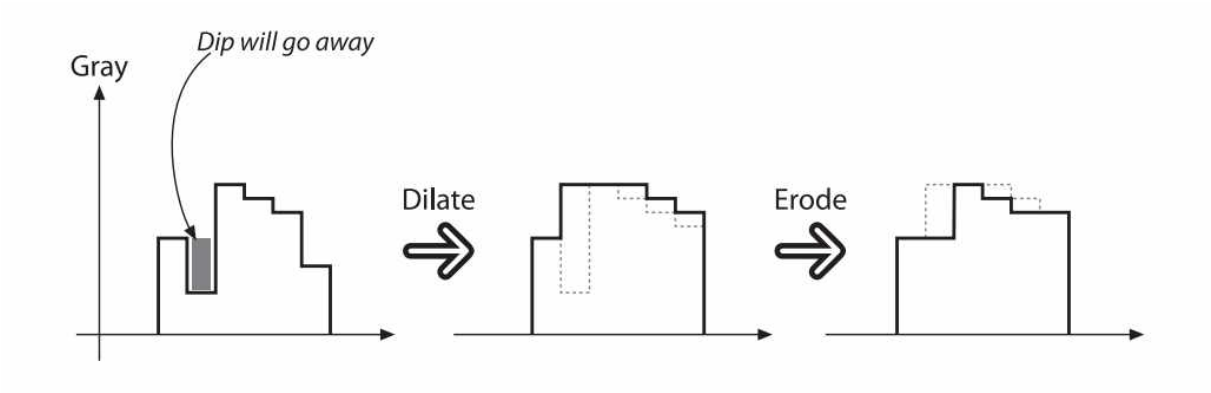

Figura 5 – Fechamento - adaptado de (BRADSKI G.; KAEHLER, 2008)

2.4 A Transformada Hough Lines

A transformação hough (BRADSKI G.; KAEHLER, 2008) é um algoritmo para encontrar formas em uma imagem, dentre elas linhas e círculos. Utiliza-se para encontrar estes modelos em imagens binárias (preto e branco). O princípio da transformação é que qualquer ponto em uma imagem seja uma parcela de um conjunto de linhas possíveis, e que este ponto esteja em uma reta, ou curva que possa ser desenhada uma forma. A transformada é um método de acumulação de informações que possibilita detectar qualquer curva, elipse ou linha que esteja pouco visível ou com muita distorção.

2.5 Espaços de Cores

O objetivo de um espaço de cor é caracterizar cores com algum padrão. Resumidamente um modelo de cores é classificado em um sistema de valores e subespaço nesse sistema, cada cor é representado por um único ponto. (GONZALEZ R.; WOODS, 1992)

Os principais modelos de espaços de cores são:

2.5.1 RGB

No espaço de cor RGB, cada cor apresenta 3 valores de componentes espectrais, sendo eles vermelho, verde e azul. Estas são cores são as cores primárias da luz. (GONZALEZ R.; WOODS, 1992) O cubo abaixo demonstra o modelo:

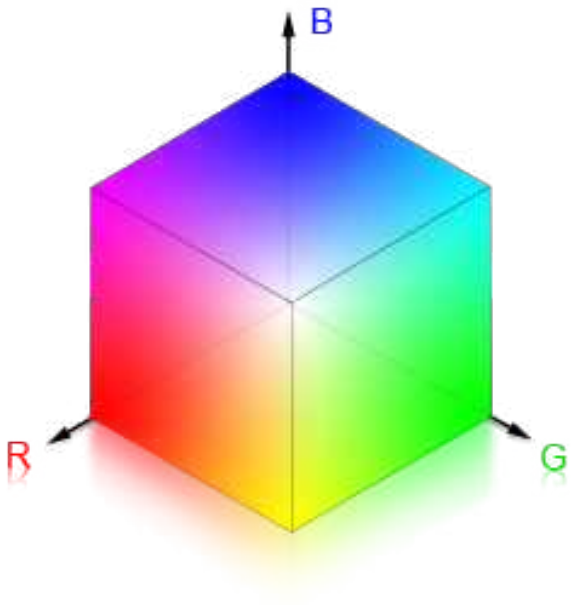

Figura 6 – Espaço de cor RGB. (GONZALEZ R.; WOODS, 1992)

2.5.2 CMY

Como o RGB são as cores primárias, CMY são as cores secundárias da luz, ou as cores primárias dos pigmentos. (GONZALEZ R.; WOODS, 1992) Dado uma cor RGB os valores desta cor em CMY são dados a partir da equação abaixo:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figura 7 – Equação. (adaptado de (GONZALEZ R.; WOODS, 1992)

2.5.3 HSV

O modelo foi desenvolvido baseado na mistura de cores, onde o sistema de coordenadas é descrito por um cilindro de cores e o subconjunto tem suas respectivas funções, o eixo H representa o ângulo em volta do eixo vertical, já o S representa a saturação da cor e o eixo V evidencia a escala de cinza. (AZEVEDO E; CONCI, 2003) O modelo é representado pela imagem abaixo:

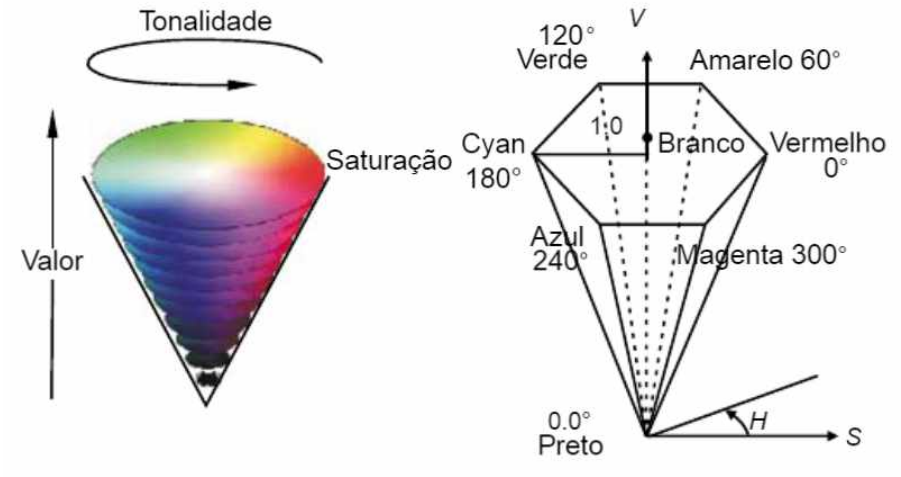

Figura 8 – Modelo HSV. adaptado de (AZEVEDO E; CONCI, 2003)

2.5.4 HLS

O modelo é uma alternativa do HSV, onde o sistema de coordenadas é descrito por um cilindro duplo de cores, onde a ponta superior representa a cor branca e a parte inferior a cor preta. O eixo H representa o ângulo em volta do eixo vertical, já o S representa a saturação da cor e o eixo L corresponde ao eixo vertical referente a luminosidade (AZEVEDO E; CONCI, 2003). O modelo é representado pela imagem abaixo:

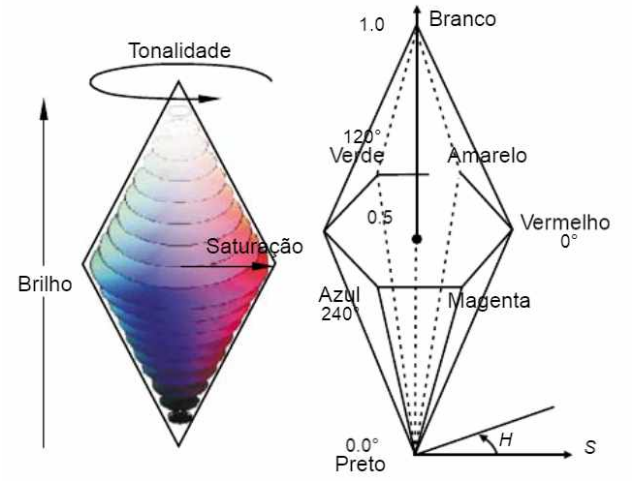

Figura 9 – Modelo HLS. (adaptado de (AZEVEDO E; CONCI, 2003)

2.5.5 LUV

O modelo representa um traço no diagrama cromático, fazendo um polígono possuindo todas as cores aptas de reproduzir, este modelo de representação não considera os fatores físicos do olho humano de perceber tal cor. O valor de cada item do subconjunto:

- L: Representa a luminosidade do pixel, quando os valores U e V estão em 0, o parâmetro corresponde a escala de cinza.
- U: Corresponde as mudanças de verde para vermelho, representado pelo aumento do valor.
- V: Corresponde as mudanças de azul para amarelo, representado pelo aumento do valor.

2.5.6 LAB

O modelo foi desenvolvido sobre a teoria de cores opostas, onde não pode existir duas cores opostas em um mesmo pixel, ou seja, não pode ter valor vermelho e verde juntos, ou azul e amarelo, com isto definiu-se que L é a luminosidade do pixel, A é a coordenada vermelho/verde e B é a coordenada amarelo/azul.(GUPTA, 2017)

2.5.7 YCrCb

YCrCb é um modelo bastante utilizado digitalmente. O parâmetro Y é o valor de luminosidade do pixel, após obter o valor da correção gama do pixel RGB. Cr é o valor da incidência do pixel na faixa de cor, que pode variar de vermelho a amarelo em relação a luminosidade e Cb é o valor da incidência do pixel na faixa de cor que pode variar de azul a amarelo em relação a luminosidade. (GUPTA, 2017)

3 Desenvolvimento

Neste capítulo é detalhado todo o processo para realizar a predição. Aponta-se os requisitos para realizar a contagem das falhas, faz-se um detalhamento dos métodos criados e das partes importantes do algoritmos. Além de mostrar os detalhes e resultados de alguns processos ao longo da execução da predição.

Na seção 3.1 são descrito as tecnologias utilizadas, além dos frameworks disponibilizados. Evidência as versões e os principais motivos pela escolha de cada um deles.

Na seção 3.2 é explicado como é realizada a captura das imagens, além do processo que é feito para criar as imagens de rótulo, para gerar os conjuntos de dados.

Na seção 3.3 é apresentado o passo a passo de toda a modelagem e o funcionamento do treinamento dos algoritmos, a fim de predizer as plantas de milho.

Na seção 3.4 é apresentado o passo a passo para realizar a predição da plantação de milho, além dos outros passos para encontrar as falhas.

3.1 Tecnologias

3.1.1 .Net Core

O .NET Core é um framework de código aberto provido pelo grupo .NET no GitHub e pela Microsoft, pode ser utilizado nos sistemas operacionais mais utilizados, pois possui compatibilidade com Windows, macOS e Linux (MICROSOFT, 2016c).

A versão 2.2 foi adotada, pois era a versão mais estável no momento da criação do projeto. Para utilização do framework, foi utilizado a linguagem de programação C# juntamente com os pacotes NuGet ML.NET, AspNetCore e OpenCvSharp.

O NuGet é um gerenciador de pacotes .NET, onde sua função é facilitar a integração com referências (bibliotecas .dll), proporcionando maior facilidade no uso de funções externas como o OpenCv, que para sua utilização é preciso compilar na máquina e instalar mais bibliotecas; adicionar referências em variáveis de ambiente, tornado o trabalho maior e assim proporcionando mais dependências para a funcionalidade.

3.1.1.1 AspNetCore

O ASP.NET é um conjunto de pacotes NuGet, o qual contém bastante bibliotecas que facilitam a criação de aplicativos de alto desempenho e multiplataforma. Ele proporciona a criação de API Web com sistema de injeção de dependência, além de possuir

diversas ferramentas de desenvolvimento de software (MICROSOFT, 2016a).

3.1.1.2 ML.NET

O ML.NET proporciona acrescentar o ML a aplicativos .NET. Com essa finalidade é possível realizar treinamentos e predições usando os dados disponíveis no aplicativo. Como são um conjunto de pacotes NuGet, então existem diversos algoritmos repartidos por bibliotecas prontos para a utilização (MICROSOFT, 2016b).

3.1.1.3 OpenCvSharp

O pacote OpenCvSharp trás consigo a biblioteca OpenCv e todas as funções que a biblioteca oferece quando compilada na máquina. Para o projeto ela forneceu todas funcionalidades úteis para realizar operações com imagens, redimensionamento, transformação em outros canais de cores, operadores morfológicos e transformadas (SHIMAT, 2016).

3.1.1.4 Photo.exif

Photo.exif é um pacote/biblioteca NuGet, que oferece a partir das imagens obtidas, as quais possuem as informações sobre as câmeras e as coordenadas geográficas.

3.1.2 Angular

É um framework de código aberto, mantido pela comunidade no GitHub, é aplicado na criação de aplicações web e dispositivos móveis, bastante utilizado com as linguagens TypeScript/JavaScript, mas também possui disponibilidade em outras (GOOGLE, 2016).

Foi escolhido para facilitar o envio dos arquivos para a API, pois é de rápido desenvolvimento e aplicação devido a disponibilização dos componentes em suas bibliotecas.

3.2 Produção dos conjuntos de dados

Com as imagens capturadas de um VANT, os quais são geralmente utilizadas para a criação de ortomosaicos. Então é realizada a copia delas para um repositório. Segue exemplo de uma imagem capturada por um VANT (Figura 10):

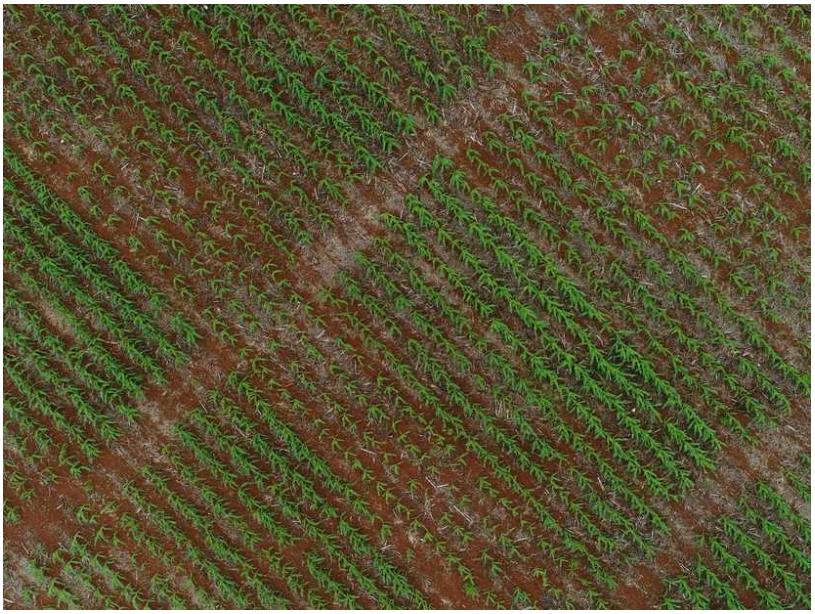

Figura 10 – Imagem capturada por VANT.

A partir desta imagem, é feita uma cópia dela e realizado um pré-processamento usando o programa Paint 3D (MICROSOFT, 2017) que possui uma ferramenta de pre-enchimento com parametrização da tolerância, que serve para quando aplicado em um pixel, expande-se ao pixeis vizinhos que possuem valores próximos ao valor informado do parâmetro.

Este passo tem como objetivo realizar a criação da imagem de Ground Truth, chamada neste trabalho também como imagem rótulo, onde possuí a função de identificar na imagem original o milho como resultado esperado.

A utilização desta ferramenta facilitou a criação de imagens rótulos, e com bastante aproximação do modelo procurado. Segue exemplo da imagem rótulo referente a imagem 10 criada com o uso da ferramenta (Figura 11):

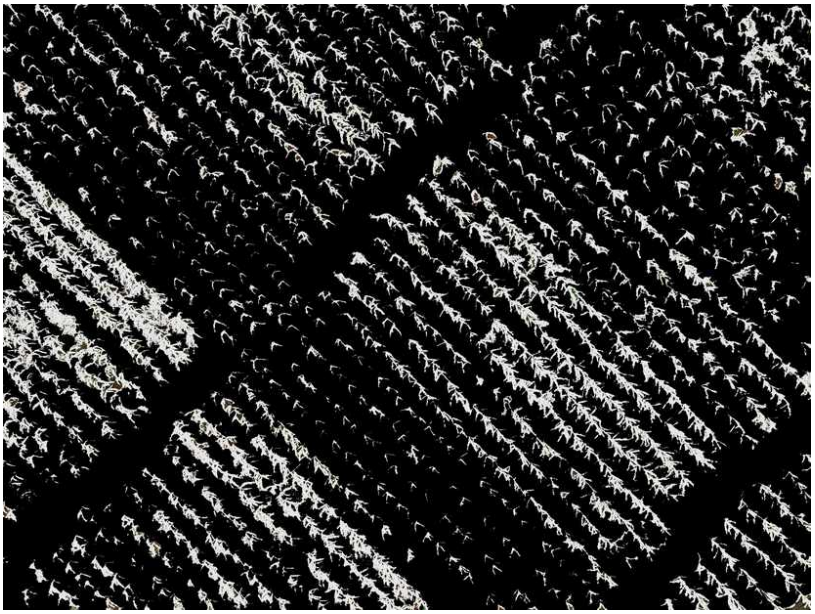

Figura 11 – Imagem rótulo.

3.3 Treinamento do Modelo

Foi utilizado as imagens capturadas por VANT, que possuem mesma similaridade em altura e com amostras diferentes de plantações, desta maneira o treinamento possui um aperfeiçoamento em detectar a planta de milho, e melhora a abrangência da predição.

3.3.1 Morfologia das imagens

As imagens são capturadas de plantações, alta resolução, e são entregues em formatos de arquivo TIFF, JPEG ou PNG. Possuem tamanhos grandes como arquivos.

Na aplicação, quando recebe-se as imagens para criar um treinamento, é necessário o envio da imagem de rótulo, assim a imagem tem sua resposta de predição pixel a pixel.

Quando recebe-se as imagens na requisição, elas são remanejadas ao BD em formato base64 (codificação), para serem usadas no treinamento através da hash retornada.

3.3.2 Transformação da imagem em vários tipos de imagens

Para realizar o treinamento das imagens enviadas à aplicação, é preciso enviar uma requisição a API, contendo a hash de treinamento. Desta maneira, recupera-se o BD as imagens e as carregam na memória. Com as imagens em memória, junto com seu rótulo aplica-se a redução da imagem (Figura 12) para melhorar o desempenho e tempo do processamento, reduz o custo de memória.

```
...ction-back\FaultDetectionService\Helpers\ImageHelper.cs                    1
244         public static void ResizeImages(int rows, int cols, Mat imgRGB, Mat
              labels, Mat imgLAB, Mat imgGray, Mat imgHLS, Mat imgLuv, Mat
              imgYCrCb, Mat imgHSV, Mat rLabels, Mat rImgRGB, Mat rImgLAB, Mat
              rImgGray, Mat rImgHLS, Mat rImgLuv, Mat rImgYCrCb, Mat rImgHSV)
245         {
246             Size size = new Size(cols, rows);
247             Cv2.Resize(imgLAB, rImgLAB, size, interpolation:
                  InterpolationFlags.Nearest);
248             Cv2.Resize(imgHLS, rImgHLS, size, interpolation:
                  InterpolationFlags.Nearest);
249             Cv2.Resize(imgLuv, rImgLuv, size, interpolation:
                  InterpolationFlags.Nearest);
250             Cv2.Resize(imgHSV, rImgHSV, size, interpolation:
                  InterpolationFlags.Nearest);
251             Cv2.Resize(imgRGB, rImgRGB, size, interpolation:
                  InterpolationFlags.Nearest);
252             Cv2.Resize(labels, rLabels, size, interpolation:
                  InterpolationFlags.Nearest);
253             Cv2.Resize(imgGray, rImgGray, size, interpolation:
                  InterpolationFlags.Nearest);
254             Cv2.Resize(imgYCrCb, rImgYCrCb, size, interpolation:
                  InterpolationFlags.Nearest);
255         }
```

Figura 12 – Função de redução do tamanho da imagem.

Logo é aplicada a transformação desta imagem em várias outras, onde são geralmente identificadas por siglas que cada letra significa um valor que representa uma característica da imagem.

```
...ction-back\FaultDetectionService\Helpers\ImageHelper.cs                    1
153         public static void ConvertColorImages(Mat imgRGB, Mat imgLAB, Mat
              imgGray, Mat imgHLS, Mat imgLuv, Mat imgYCrCb, Mat imgHSV)
154         {
155             Cv2.CvtColor(imgRGB, imgLAB, ColorConversionCodes.BGR2Lab);
156             Cv2.CvtColor(imgRGB, imgHLS, ColorConversionCodes.BGR2HLS);
157             Cv2.CvtColor(imgRGB, imgLuv, ColorConversionCodes.BGR2Luv);
158             Cv2.CvtColor(imgRGB, imgHSV, ColorConversionCodes.BGR2HSV);
159             Cv2.CvtColor(imgRGB, imgGray, ColorConversionCodes.BGR2GRAY);
160             Cv2.CvtColor(imgRGB, imgYCrCb, ColorConversionCodes.BGR2YCrCb);
161         }
```

Figura 13 – Função de conversão das imagens

Abaixo mostra-se o resultado das conversão da imagem original para as outras imagens.

1. LAB

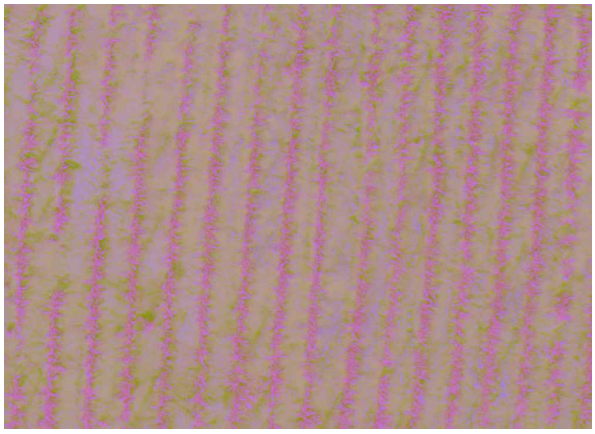

Figura 14 – Imagem LAB.

2. HLS

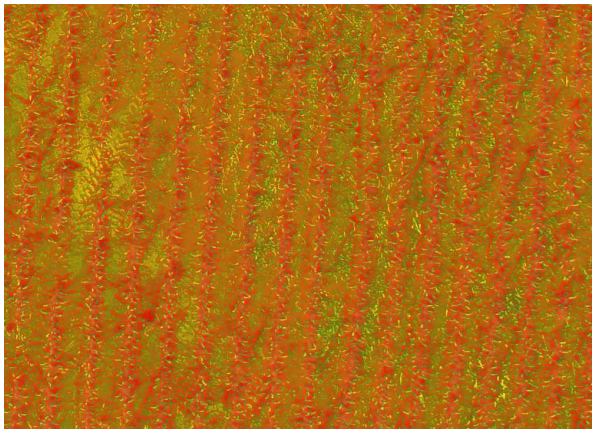

Figura 15 – Imagem HLS.

3. LUV

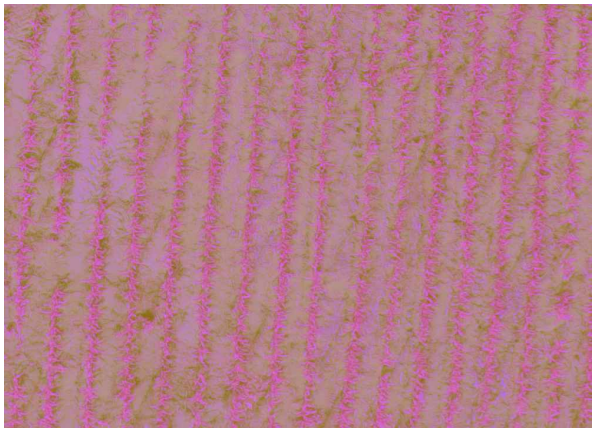

Figura 16 – Imagem LUV.

4. HSV

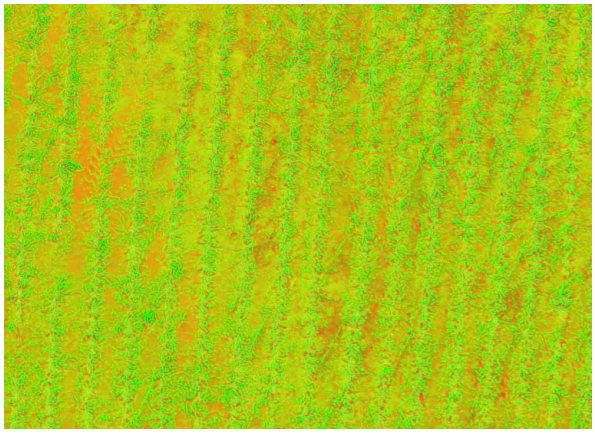

Figura 17 – Imagem HSV.

5. YCrCb

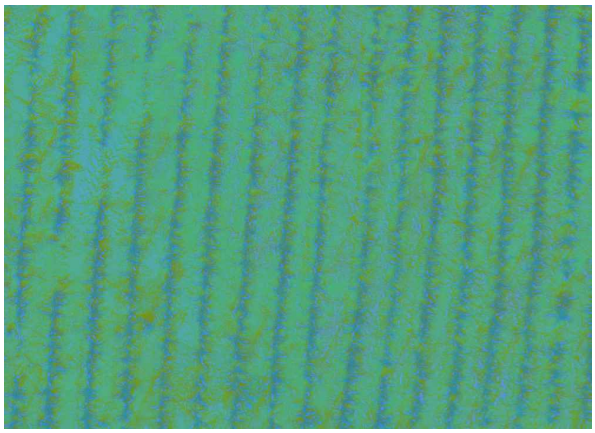

Figura 18 – Imagem YCrCb.

6. Gray

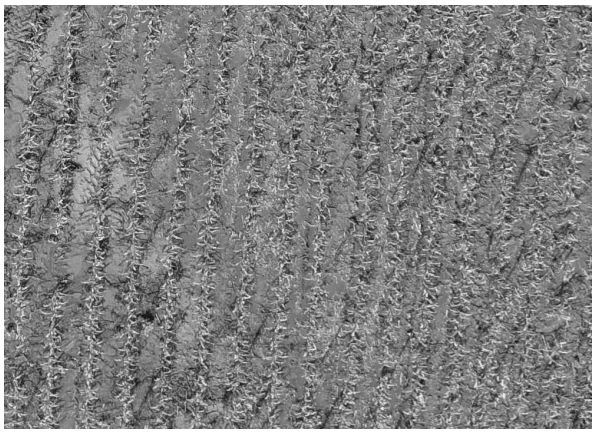

Figura 19 – Imagem Gray.

3.3.3 Criação do vetor de Características

Após realizar a transformação da imagem, para cada pixel da mesma é guardado em uma lista onde é informada a posição (linha, coluna) do pixel, além das informações de cada imagem e o valor inverso da escala de cinza (OffGray). Com isto é possível formar um vetor de 17 posições para cada pixel da imagem, onde cada posição tem sua importância no treinamento, tal vetor possui a respectiva forma:

[0]: Lab_L - representa o valor L da imagem LAB

[1]: Lab_A - representa o valor A da imagem LAB

[2]: Lab_B - representa o valor B da imagem LAB

[3]: Gray - representa o valor de cinza

[4]: GrayOff - representa o valor inverso de cinza

[5]: Hls_H - representa o valor H da imagem HLS

[6]: Hls_L - representa o valor L da imagem HLS

[7]: Hls_S - representa o valor S da imagem HLS

[8]: Hsv_H - representa o valor H da imagem HSV

[9]: Hsv_S - representa o valor S da imagem HSV

[10]: Hsv_V - representa o valor V da imagem HSV

[11]: YCrCb_Y - representa o valor Y da imagem YCrCb

[12]: YCrCb_Cr - representa o valor Cr da imagem YCrCb

[13]: YCrCb_Cb - representa o valor Cb da imagem YCrCb

[14]: Luv_L - representa o valor L da imagem LUV

[15]: Luv_U - representa o valor U da imagem LUV

[16]: Luv_V - representa o valor V da imagem LUV

Desta forma, a estruturação da imagem em formato de dado que é possível trabalhar para realizar os treinamento do modelo (Figura 20).

```
...on-back\FaultDetectionService\Helpers\TrainingHelper.cs                    1
315         public static Feature ExtractFeature(int i, int j, Mat rImgGray, Mat
              rImgLAB, float imgMeanGray, Mat rImgHLS, Mat rImgHSV, Mat rImgYCrCb,
               Mat rImgLuv)
316         {
317             byte grayValue = rImgGray.At<byte>(i, j);
318             Vec3b labValue = rImgLAB.At<Vec3b>(i, j);
319             Vec3b hlsValue = rImgHLS.At<Vec3b>(i, j);
320             Vec3b hsvValue = rImgHSV.At<Vec3b>(i, j);
321             Vec3b luvValue = rImgLuv.At<Vec3b>(i, j);
322             Vec3b ycrcbValue = rImgYCrCb.At<Vec3b>(i, j);
323
324             return new Feature
325             {
326                 Row = i,
327                 Cols = j,
328                 Gray = grayValue,
329                 GrayOff = grayValue - imgMeanGray,
330                 Lab_L = labValue[0],
331                 Lab_A = labValue[1],
332                 Lab_B = labValue[2],
333                 Hls_H = hlsValue[0],
334                 Hls_L = hlsValue[1],
335                 Hls_S = hlsValue[2],
336                 Hsv_H = hsvValue[0],
337                 Hsv_S = hsvValue[1],
338                 Hsv_V = hsvValue[2],
339                 Luv_L = luvValue[0],
340                 Luv_U = luvValue[1],
341                 Luv_V = luvValue[2],
342                 Ycrcb_Y = ycrcbValue[0],
343                 Ycrcb_Cr = ycrcbValue[1],
344                 Ycrcb_Cb = ycrcbValue[2]
345             };
346         }
```

Figura 20 – Função de separação de treinamento.

3.3.4 Separação do vetor em treinamento e teste

Para saber o resultado do treinamento, é preciso separar uma parte do conjunto de dados criado para realizar a predição dos dados no conjunto de teste (Figura 21).

```
...on-back\FaultDetectionService\Helpers\TrainingHelper.cs                    1
46          /// <summary>
47          /// Realiza a repartição dos dados para treinamento e teste
48          /// </summary>
49          /// <param name="mlContext"></param>
50          /// <param name="dataView"></param>
51          /// <returns></returns>
52          public static TrainTestData ConvertingFeaturesToTrainTestData(MLContext ↵
                mlContext, IDataView dataView)
53          {
54              try
55              {
56                  TrainTestData splitDataView = mlContext.Data.TrainTestSplit     ↵
                      (dataView, testFraction: 0.2);
57                  return splitDataView;
58              }
59              catch (Exception)
60              {
61                  throw;
62              }
63          }
```

Figura 21 – Função de separação de treinamento.

3.3.5 Treinamento

Antes de realizar o treinamento, foi efetuado uma normalização utilizando média e variância nos dados.

Para realizar o treinamento é preciso criar um transformador de dados, isto é, decide qual o modelo, que no treinamento do milho foi o FastTree. Neste algoritmo, para o treinamento é passado o vetor de valores retirados dos pixeis, junto com o pedido de normalização do vetor, além disto é passado como parâmetros o número de folhas e árvores, além da taxa de aprendizado (Figura 22).

O algoritmo FastTree foi escolhido devido ele possuir a oportunidade de passar a taxa de aprendizagem e a quantidade de árvores que serão utilizadas no treinamento, logo juntando estas duas parametrizações, tornou-se fácil realizar o treinamento. Para o treinamento realizado, utilizou-se da taxa de aprendizado de 0.2 e 10 árvores para realizar o treinamento.

```
...on-back\FaultDetectionService\Helpers\TrainingHelper.cs                    1
130                 case TrainEnum.FastTree:
131                     int numberOfLeavesFT = 2;
132                     int numberOfTreesFT = 10;
133                     int minimumExampleCountPerLeaf = 2;
134                     double learningRateFT = 0.2;
135
136                     return mlContext.Transforms.Concatenate("Features",
                        "Lab_L", "Lab_A", "Lab_B",
137                                 "Gray", "GrayOff", "Hls_H", "Hls_L", "Hls_S",
                        "Hsv_H", "Hsv_S", "Hsv_V",
138                                 "Ycrcb_Y", "Ycrcb_Cr", "Ycrcb_Cb", "Luv_L",
                        "Luv_U", "Luv_V")
139                             .Append(mlContext.Transforms.NormalizeMeanVariance
                        ("Features"))
140                             .Append
                        (mlContext.BinaryClassification.Trainers.FastTree(
141                                 labelColumnName: "TrueValue",
142                                 numberOfLeaves: numberOfLeavesFT,
143                                 numberOfTrees: numberOfTreesFT,
144                                 minimumExampleCountPerLeaf:
                        minimumExampleCountPerLeaf,
145                                 learningRate: learningRateFT));
```

Figura 22 – Método para modelar treinamento Fast Tree.

3.3.6 Cálculo das métricas

Após a realização do treinamento, é possível notar as métricas, as quais obtém-se a acurácia e entropia usando a parte do conjunto de dados que foi separada para testes.

3.4 Predição nos Dados de Teste

Após a realização do treinamento, para contar as falhas é preciso enviar uma imagem com informações Exif. Com estas informações é possível reconhecer as falhas e realizar a contagens. A imagem de exemplo possuí tamanho de 4000 x 3000 pixeis, tirada à uma altura de 30 metros em relação ao solo. Exemplo de imagem enviada para contagem de falhas (Figura 23):

Figura 23 – Imagem com plantação.

3.4.1 Criação do vetor de Características

Quando recebe-se uma requisição de contar as falhas sobre uma imagem é preciso fazer o processo de extração da imagem para cada pixel, transformando a imagem na lista. Cada item da lista representa um pixel com sua posição (linha, coluna) na imagem.

3.4.2 Carrega o Modelo e realiza a Predição

Na requisição, especifica-se qual a hash de treinamento para ser carregada a máquina de predição, e para cada item da lista é realizada a predição. Então, é montada a imagem através dos itens, formando uma imagem de fundo preto, onde cada pixel que possui predição positiva é setado como branco (milho) (Figura 24).

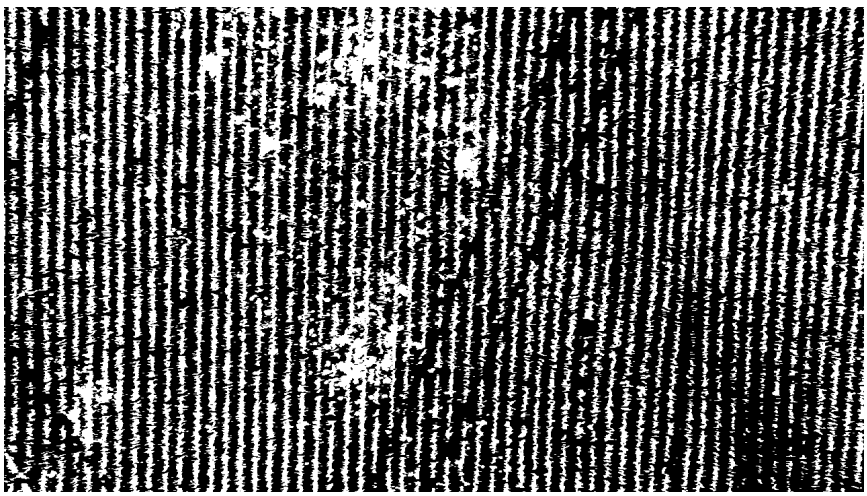

Figura 24 – Imagem contendo predições.

3.4.3 Criação dos elementos estruturantes para Operações Morfológicas

Para realizar as operações morfológicas, é criado um elemento estruturante, o qual é um quadrado de tamanho definido por 4x4 e outro elemento elemento o qual é um retângulo de 11x9.

3.4.4 Fechamento

Com a imagem pronta para trabalhar, é feito o processo de limpeza de ruídos e a modelagem dos milhos reconhecidos. Utiliza-se o fechamento, com elemento estruturante o quadrado, para preencher os espaços vazios nas plantas de milho (visão de cima do milho pode ter chão, mas não é falha) e para aumentar o tamanho do milho, para não considerar que cada planta de milho e seu vizinho não estejam longe, quando não exita falha de plantio.

Após este filtro a imagem possui um aumento das predições facilitando a algoritmo encontrar as linhas, conforme Figura 25.

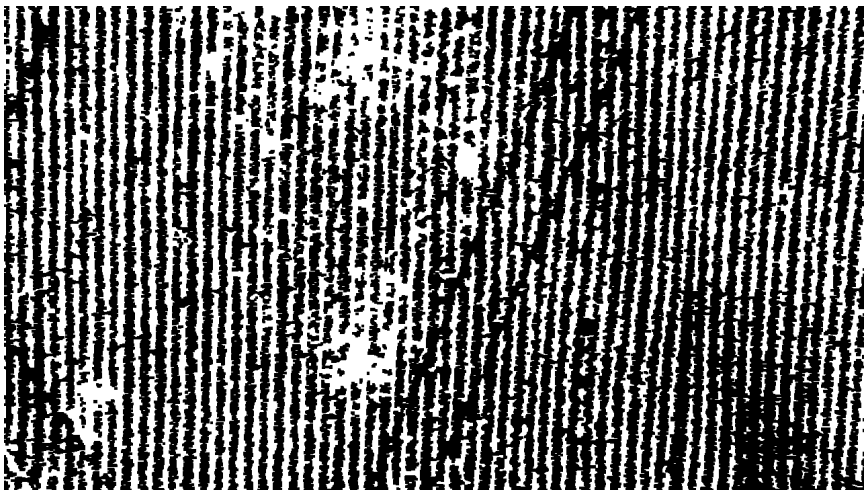

Figura 25 – Imagem criada pelo fechamento.

3.4.5 HLP

Após melhorar a imagem é preciso encontrar as linhas de milho para poder deduzir onde estão as falhas, para isto utilizou-se o algoritmo HLP, que consegue encontrar as linhas de forma eficiente e retornar um vetor de segmentos de linhas, que a partir deste vetor é feito a inteligência de calcular o ângulo que está a plantação da imagem enviada.

É criada uma nova imagem para armazenar as linhas que foram encontradas pela transformada. Com o vetor produzido pela transformada é feito o cálculo de dois pontos que formam uma linha que vai de uma borda até a outra utilizando a função do anexo A.

Após este filtro é criada uma imagem onde as linhas de plantio estão em branco e o chão da plantação está como preto, conforme Figura 26.

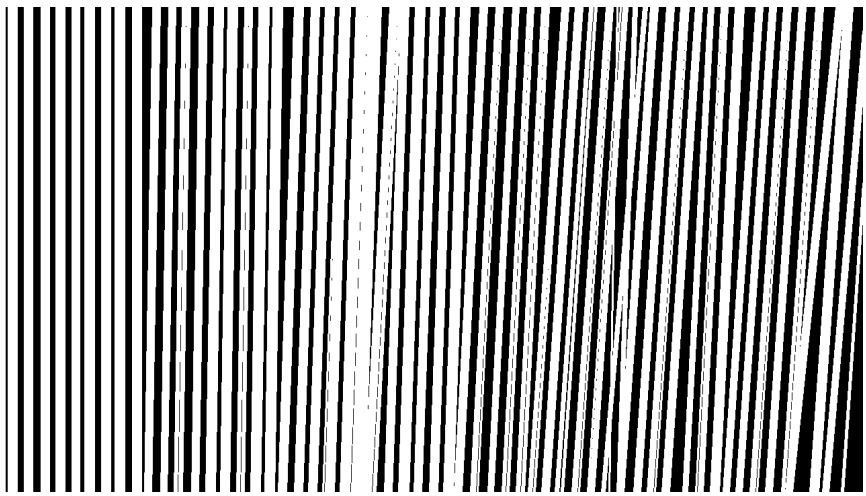

Figura 26 – Imagem criada pelo passo HLP.

3.4.6 Subtração da imagem de HL pela de Fechamento

Para realizar a subtração das imagens, é necessário aumentar a área das plantas para poderem ser subtraídas pelas linhas feitas pelo HLP. É preciso realizar inversão binária da imagem de fechamento e realizar a coloração das plantas de milho de vermelho.

Para encontrar as falhas é feita a subtração da imagem de HL (fundo branco, linhas pretas) pela imagem de fechamento (chão preto, plantas de milho marcados de vermelho). É encontrada uma imagem onde possui o chão preto, falhas de cor azulada, milhos com coloração branca, onde a junção dos milhos com as falhas é feita a linha criada pelo HLP (Figura 27).

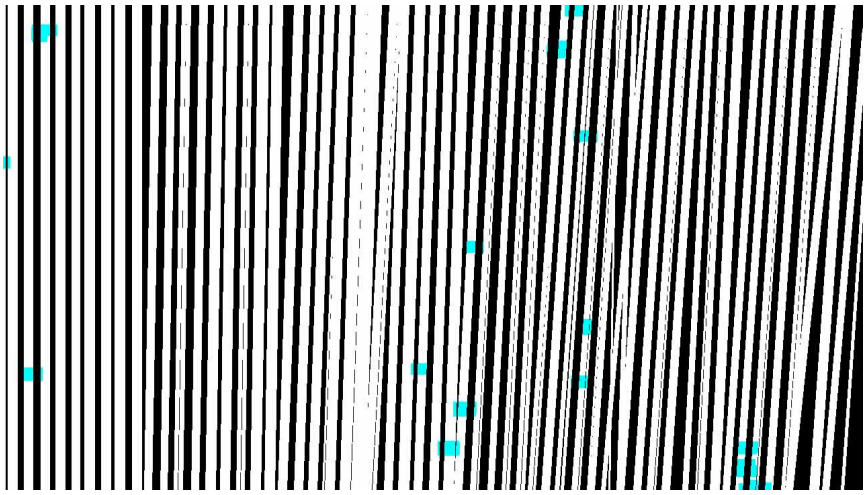

Figura 27 – Imagem após subtração.

3.4.7 Abertura para juntar os componentes

Para melhorar o acerto das falhas, colore-se os milhos de preto, e as falhas de branco, e assim realiza-se uma erosão para remover ruídos (linhas finas que sobram da subtração) com o tamanho do dobro elemento morfológico utilizado no fechamento. Seguido disto, é feito uma dilação para aumentar o tamanho das falhas e mostrar falhas que se encaixam melhor na imagem enviada para o teste, a imagem a seguir mostra o resultado da predição final (Figura 28).

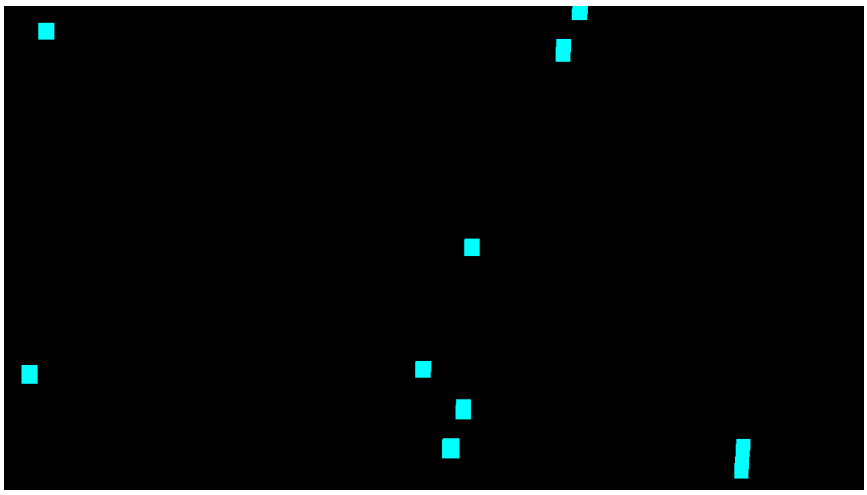

Figura 28 – Imagem contendo as falhas.

3.4.8 Contagem das falhas

Para encontrar os objetos das falhas da Figura 27 é utilizado o método de encontrar contornos. O contorno é um vetor de pontos, que forma uma figura poligonal, a partir deste polígono é encontrada a maior distância entre seus pontos para assim calcular a quantidade de falhas em metros lineares. Com a seleção destes pontos é realizado o somatório de todas as distâncias entre os pontos e depois é multiplicado pelo GSD. Desta maneira é encontrado o valor de metros de falhas.

A Figura 29 é o desenho de todos os contornos encontrados na Figura 28.

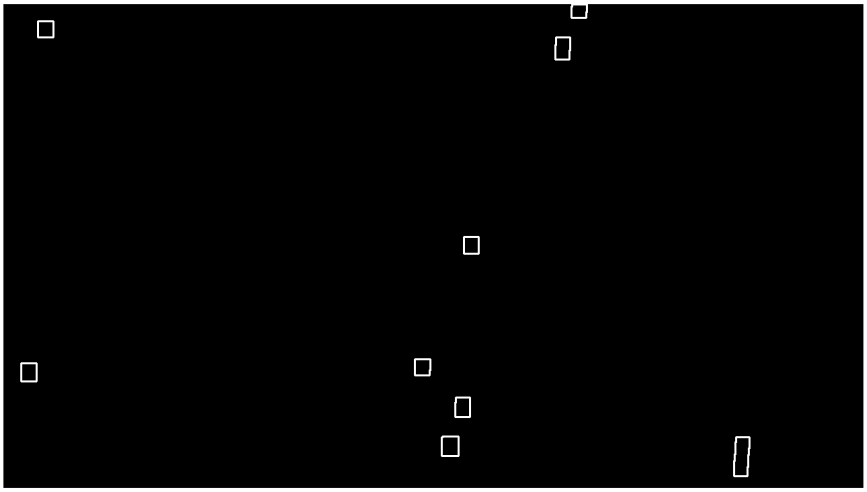

Figura 29 – Imagem contendo os contornos.

4 Resultados

Neste capítulo são apresentados os resultados obtidos desta pesquisa, tanto da fase de treinamento como os passos para a predição da contagem de falhas.

4.1 Treinamento

No treinamento, obtivemos uma máquina de predição que consegue predizer as plantações de milho. Então, marcando-as de branco. Este arquivo é salvo como arquivo junto com a API, e possui como nome a hash de treinamento. Também no treinamento é obtido as métricas de acurácia, entropia e f1 score, sendo elas:

1. Acurácia = 85,83%
2. Entropia = 0,85
3. F1 Score = 0,73

4.2 Predição

A imagem utilizada para predição, foi capturada em uma altura de 30 metros por um VANT chamado ebee senseFly. As Figuras a seguir ilustram o processo de predição. A Figura 30 representa a imagem original obtida pelo VANT.

Figura 30 – Imagem original tirada pelo VANT.

A Figura 31 exibe o resultado das predições.

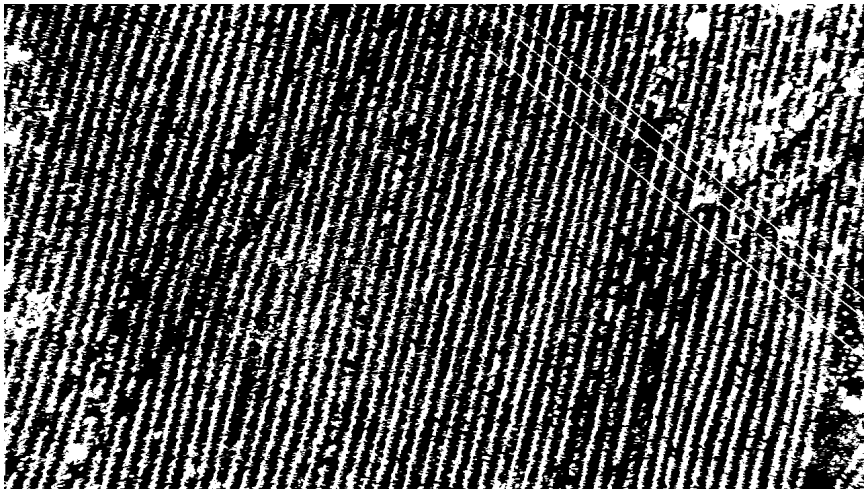

Figura 31 – Imagem com predição.

A Figura 32 apresenta um aumento significativo na predição devido ao fechamento.

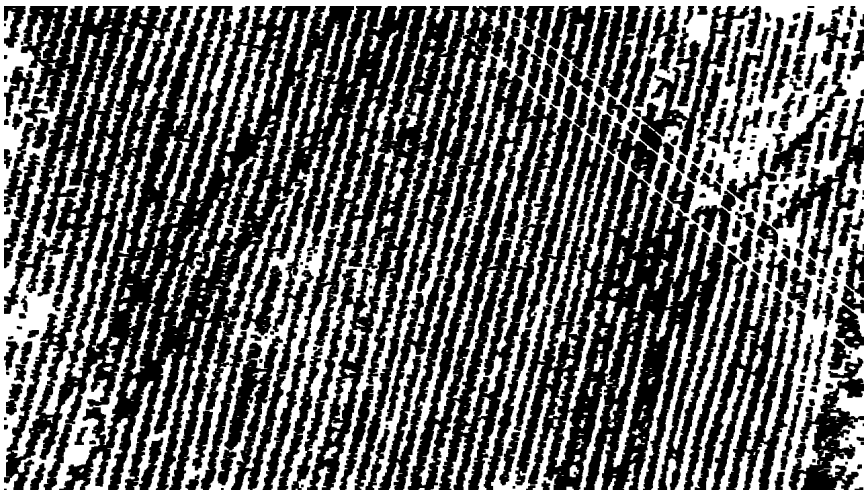

Figura 32 – Imagem com fechamento após predição.

Após o fechamento é realizado o processo HLP, onde cria-se uma imagem com as linhas, nota-se nas Figuras 33 e 34, algumas linhas que não são da plantação, mas devido a qualidade da imagem para prover resultados, é passível de acontecer.

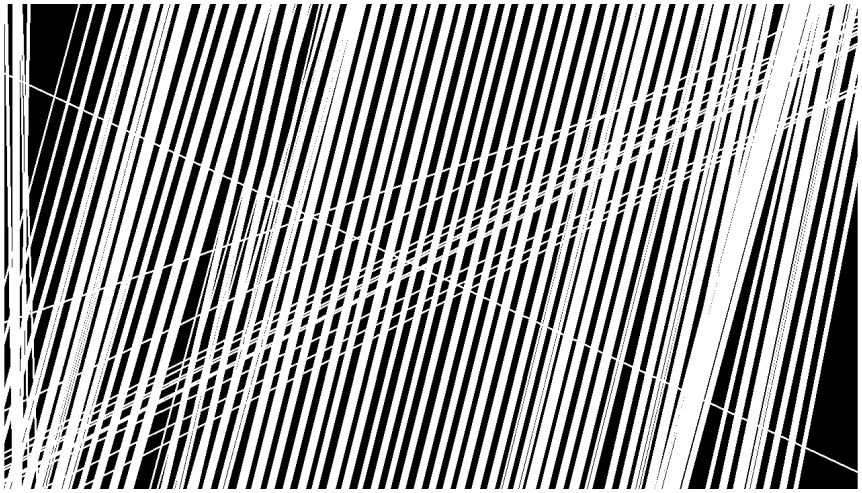

Figura 33 – Imagem após HLP.

A Figura 34 mostra o processo de subtração onde é encontrada as falhas.

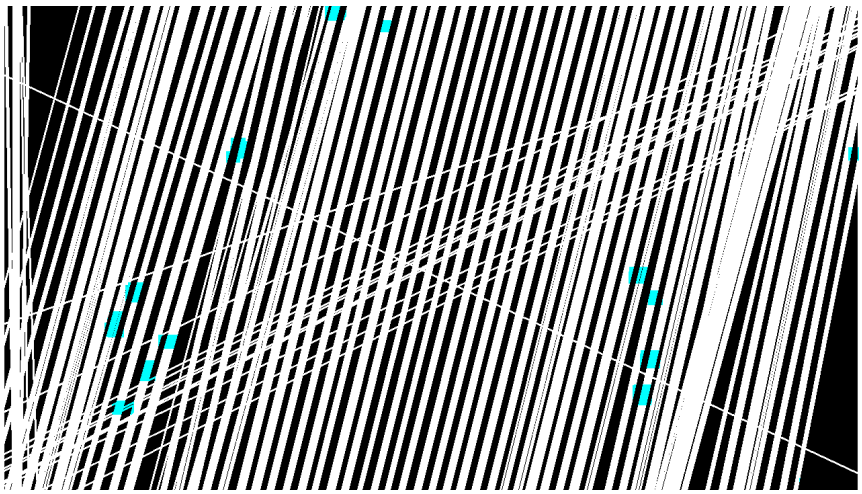

Figura 34 – Imagem após subtração.

Após colorir de preto as linhas brancas, somente sobra as falhas e ruídos, como mostra a Figura 35.

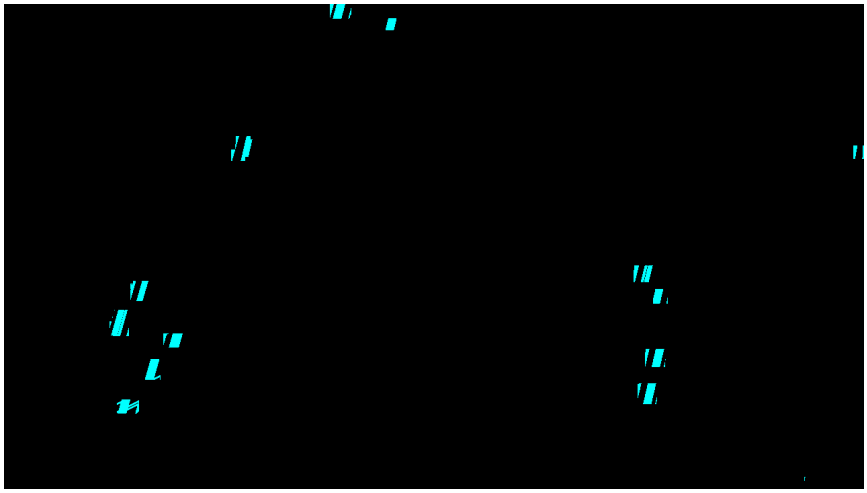

Figura 35 – Imagem somente com as falhas e ruídos.

Com os processos de erosão e dilatação são removido os ruídos e restando apenas as falhas como mostra na Figura 36.

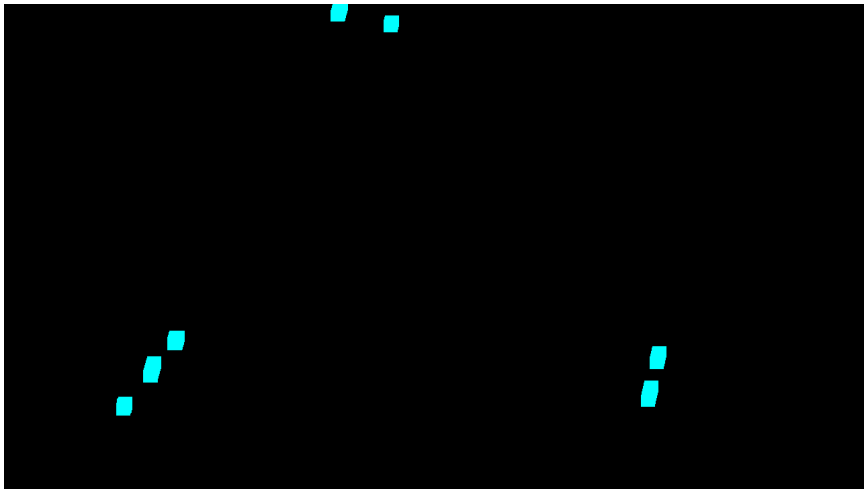

Figura 36 – Imagem com as falhas.

Figura 37 mostra as falhas na imagem original.

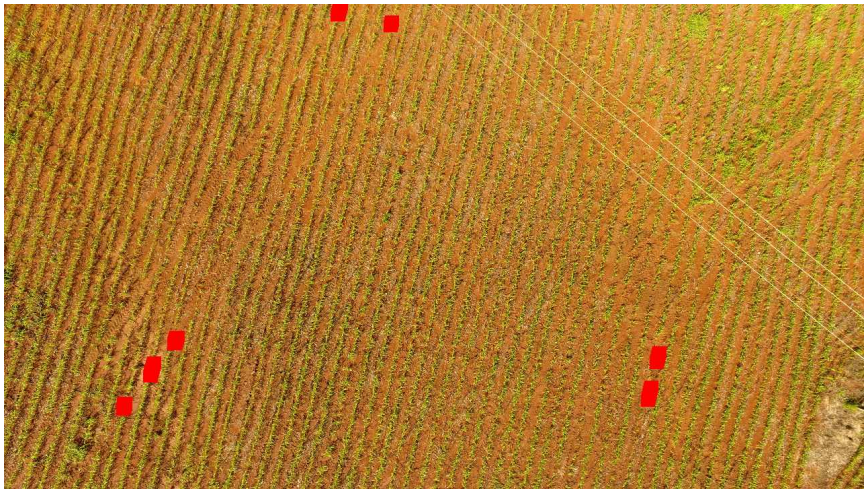

Figura 37 – Imagem original com as falhas.

4.3 Contagem das falhas

Após reconhecer as falhas e encontrar os contornos, é feito o cálculo das linhas dos polígonos, assim é criado a figura 38 apenas para visualizar estas distâncias.

Figura 38 – Imagem com as distâncias lineares.

Com as distâncias calculadas, a quantidade de falhas em metros lineares foi de 7.23 metros, e a quantidade de biomassa calculada, a partir da área dos objetos obtidos pelo método do contorno foi de 62.81 metros.

5 Conclusão

Este capítulo tem como objetivo falar sobre contribuições, os próximos trabalhos e a conclusão do trabalho.

5.1 Contribuição

Esta metodologia juntamente com a API, podem ser reusadas para realizar as contagens de falhas em outras culturas, como laranja, algodão, eucaliptos, soja e outros.

5.2 Trabalhos futuros

Para continuação deste trabalho, poder-se-ia otimizar o custo de memória em trabalhar com algumas outras estruturas e bibliotecas, considerando o uso de streams. Um ponto a melhorar é a otimização do processo de predição, gasta-se muita memória e tem algumas bibliotecas dependentes. Vale ressaltar, o uso do paralelismo em alguns métodos, além de redução de cálculos matemáticos que são usados em dois lugares para o mesmo objeto. Outro ponto é conseguir executar a aplicação em um VANT.

É possível melhorar a interface para integração dos testes ao tempo de o VANT somente utilizar um servidor para processar, e o mesmo só capturar as imagens e enviá-las. Outra sugestão é a criação de uma tarefa que buscaria as imagens de um servidor e realizaria automaticamente o treinamento e cálculo das falhas.

5.3 Conclusão

Esta pesquisa tem como importância em minimizar o trabalho de campo, reduzindo a mão de obra e o tempo de trabalho na contagem de falhas de plantio de milho, visto que hoje é um trabalho manual, sendo que com a utilização de VANT e ML podem melhorar os trabalhos e economizar nos custos e estimativas de erros.

A transformação das imagens em várias cores foi bem sucedido e teve muita importância no treinamento, pelo motivo de criar um vetor com todas aqueles valores de cores o treinamento conseguiu identificar bem as plantas. Porém, o método HLP teve problemas na identificação das carreiras de milho, pois em certas fotos as ervas daninhas são reconhecidas na predição e acabam fazendo linhas que não estão na carreira de milho.

Poderia-se treinar um modelo direcionado as falhas e neste modelo em vez de abordar um pixel por predição, abordar um bloco de pixeis onde três classes poderiam

ser utilizadas, sendo elas falhas, chão e milho.

Referências

AZEVEDO E; CONCI, A. *Computação Gráfica Teoria e Prática*. 4. ed. [S.l.]: Editora Campus-Elsevier, 2003. Citado 3 vezes nas páginas 6, 19 e 20.

BERNARDI, A. et al. *Agricultura de precisão: resultados de um novo olhar*. [S.l.]: Embrapa, 2014. 596 p. Citado na página 13.

BRADSKI G.; KAEHLER, A. Learning opencv. 2008. Citado 4 vezes nas páginas 6, 16, 17 e 18.

FRIEDMAN, J. Greedy function approximation: A gradient boosting machine. 1999. Disponível em: <https://statweb.stanford.edu/~jhf/ftp/trebst.pdf>. Citado na página 15.

GONZALEZ R.; WOODS, R. *Digital Image Processing*. [S.l.]: Prentice Hall, 1992. Citado 3 vezes nas páginas 6, 18 e 19.

GOOGLE. *Angular*. 2016. Disponível em: <https://github.com/angular/angular>. Citado na página 23.

GUPTA, V. *Color spaces in OpenCV (C++ / Python)*. 2017. Disponível em: <https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/>. Citado na página 21.

JORGE, L. et al. *Uso de veículos aéreos não tripulados (VANT) em agricultura de precisão*. [S.l.]: Embrapa, 2014. 109-134 p. Citado na página 13.

MICROSOFT. *ASP.NET Core*. 2016. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/core/?view=aspnetcore-2.2>. Citado na página 23.

MICROSOFT. *ML.NET*. 2016. Disponível em: <https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet>. Citado na página 23.

MICROSOFT. *.NET Core*. 2016. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/core/>. Citado na página 22.

MICROSOFT. *Paint 3D*. 2017. Disponível em: <https://www.microsoft.com/pt-br/p/paint-3d/9nblggh5fv99?activetab=pivot:overviewtab>. Citado na página 24.

MICROSOFT. *FastTree*. 2018. Disponível em: <https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fasttreeregressiontrainer?view=ml-dotnet#training-algorithm-details>. Citado na página 15.

MICROSOFT. *Classificação Binária*. 2019. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/machine-learning/resources/tasks#binary-classification>. Citado na página 15.

MICROSOFT. *Clustering*. 2019. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/machine-learning/resources/tasks#clustering>. Citado na página 15.

OLIVEIRA H.; SOUZA, J. et al. Failure detection in row crops from uav images using morphological operators. 2018. Citado na página 11.

SHIMAT. *OpenCvSharp*. 2016. Disponível em: <https://github.com/shimat/opencvsharp>. Citado na página 23.

SIMON, P. *Too Big to Ignore: The Business Case for Big Data*. [S.l.]: Wiley, 2013. Citado na página 14.

VALLE, M. E. Mathemathical morphology for color images: Total ordering approaches. 2017. Citado na página 15.

Anexos

ANEXO A – CalculatePoints

```
private (Point origin, Point end) CalculatePoints(Point
    p1, Point p2, int max_x, int max_y)
{
    double m = ((double)p2.Y - p1.Y) / ((double)p2.X -
        p1.X);

    if (double.IsInfinity(m))
        m = 0;

    Point origin;
    Point end;
    if (p1.X == p2.X)
    {
        origin = new Point(p1.X, 0);
        end = new Point(p1.X, max_y);
        return (origin, end);
    }
    else if (p1.Y == p2.Y)
    {
        origin = new Point(0, p2.Y);
        end = new Point(max_x, p2.Y);
        return (origin, end);
    }
    else
    {
        int x0, x1, y0, y1;

        x0 = (int)Math.Round((p1.Y - m * p1.X) / -m);
        y0 = 0;

        x1 = p1.X + (int)Math.Round((max_y - p1.Y) / m);
        y1 = max_y;


        if (x0 > max_x)
```

```
{
    x0 = max_x;
    y0 = p1.Y + (int)Math.Round(m * (max_x - p1.
        X));

    if (x1 > max_x)
    {
        x1 = p1.X + (int)Math.Round((0 - p1.Y) /
            m);
        y1 = 0;
    }
    else if (x1 < 0)
    {
        x1 = 0;
        y1 = p1.Y + (int)Math.Round(m * (0 - p1.
            X));
    }

    origin = new Point(x0, y0);
    end = new Point(x1, y1);
    return (origin, end);

}
else if (x0 < 0)
{
    x0 = 0;
    y0 = p1.Y + (int)Math.Round(m * (0 - p1.X));

    if (x1 > max_x)
    {
        x1 = p1.X + (int)Math.Round((max_x - p1.
            Y) / m);
        y1 = max_x;
    }
    else if (x1 < 0)
    {
        x1 = p1.X + (int)Math.Round((max_y - p1.
            Y) / m);
        y1 = max_y;
```

```
                }

                origin = new Point(x0, y0);
                end = new Point(x1, y1);
                return (origin, end);
            }
            else
            {
                if (x1 > max_x)
                {
                    x1 = max_x;
                    y1 = p1.Y + (int)Math.Round(m * (max_x -
                        p1.X));
                }
                else if (x1 < 0)
                {
                    x1 = 0;
                    y1 = p1.Y + (int)Math.Round(m * (0 - p1.
                       X));
                }

                origin = new Point(x0, y0);
                end = new Point(x1, y1);
                return (origin, end);
            }
        }
    }
```