Universidade Federal de Pernambuco
Centro de Informática
Graduação em Engenharia da Computação

Caroline Pereira Medeiros

Processamento de Imagem e Aprendizagem de Máquina para Classificação de Anomalias em Vias Públicas

Trabalho de Graduação

Recife
2018

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Engenharia da Computação

Caroline Pereira Medeiros

Processamento de Imagem e Aprendizagem de Máquina para Classificação de Anomalias em Vias Públicas

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Engenheiro da Computação.

Orientador: Adriano Augusto de Moraes Sarmento

Recife
2018

Agradecimentos

Agradeço a meus pais que durante os anos de estudo não mediram esforços para me fazer crescer tanto profissionalmente quanto como pessoa. Estes que em um ato altruísta sempre me deram força nos momentos de desânimo, apontaram meus erros e me orientaram sobre qual o melhor caminho a seguir. Para mim não existe forma de amor maior.

Agradeço a meus professores que durante toda a trajetória tiveram papel importante não só na construção de minha formação, mas também do Centro de Informática como um todo tornando este um centro de referência. Agradeço em especial ao Professor Adriano Sarmento que ao longo do desenvolvimento deste trabalho se mostrou sempre paciente e solícito a ajudar.

Agradeço também a todos os meus amigos de curso que compartilharam comigo as dificuldades, anseios e que contribuíram para tornar o caminho até a formação acadêmica mais leve e descontraído.

Por fim, agradeço ao meu grande amigo e companheiro Gedson Santos que tive a felicidade de conhecer durante o curso e que a partir de então passou a trilhar toda a trajetória e compartilhar os mesmos objetivos lado a lado comigo.

Resumo

Os defeitos nas estradas além de serem causados pela má qualidade dos materiais utilizados, o que intensifica a reincidência de buracos antigos, também são causados pela falta de capacidade em detectar essas falhas com agilidade e automaticidade. No Brasil as rodovias ainda são responsáveis pelo maior tráfego de pessoas e bens sendo aproximadamente 204.000 km pavimentadas apenas, dos quase 1,7 milhão de quilômetros de estradas que cortam o país. Este trabalho tem como objetivo a implementação de um algoritmo de detecção e classificação de falhas em vias públicas (buracos, rachaduras e manchas) utilizando técnicas de processamento de imagens e aprendizagem de máquina que seja rápido permitindo um acompanhamento automatizado, eficiente e barato do principal meio de circulação de cargas e pessoas no Brasil e no mundo. Com os resultados apresentados é possível ver o potencial do uso de algoritmos de classificação na detecção de falhas em vias em geral e a importância na escolha de características relevantes para seu uso no método proposto.

Palavras-chaves: buracos, anomalias, vias públicas, aprendizagem de máquina, processamento de imagem, python

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

No Brasil as maiores causas de acidente de trânsito ainda são decorrentes do excesso de velocidade e do consumo de bebidas alcoólicas[1], porém, um dado interessante mostra que desde a liberação do aplicativo de trânsito do DNIT em 2015, o DNIT Móvel, o maior número disparado de ocorrências registradas pelos usuários é com relação à presença de buracos nas principais vias públicas[2]. Este em conjunto com os outros fatores citados serve para aumentar a quantidade e intensificar a gravidade dos acidentes.

Os defeitos nas estradas além de serem causados pela má qualidade dos materiais utilizados[3], o que intensifica a reincidência de buracos antigos, também são causados pela falta de capacidade em detectar essas falhas com agilidade e automaticidade[4]. No Brasil as rodovias ainda são responsáveis pelo maior tráfego de pessoas e bens sendo aproximadamente 204.000 km pavimentadas apenas, dos quase 1,7 milhão de quilômetros de estradas que cortam o país[5].

Das perdas geradas sobre as mercadorias no setor frutífero 50% são causadas durante o manuseio e transporte da carga. Os caminhões de transporte são obrigados a circular lentamente e por longos períodos sob efeitos climáticos devido às más condições das rodovias o que justifica a alta porcentagem de desperdício no setor gerando grande perda de competitividade no mercado[5]. Sem falar nos acidentes fatais envolvendo esses veículos de grande porte[6].

Diversas técnicas foram pensadas para atacar o problema incluindo desde mapeamentos presenciais rotineiros, onde o responsável circula pelas vias mapeando os pontos danificados com o auxílio de dispositivos como acelerômetros, GPS[7] e câmeras[8]; até aplicativos de notificações alimentados pelos próprios usuários[9], sendo ambos os processos pouco automatizados. O processamento de imagem (do inglês, *image processing*) é um conjunto de técnicas computacionais para análise, aprimoramento, compressão e reconstrução de imagens. Geralmente se refere ao processamento de imagem digital mas o processamento de imagem óptica e analógica também é possível. Tem aplicações extensas em muitas áreas, incluindo astronomia, medicina e robótica industrial[10].

Uma série de trabalhos baseados em análise de vídeos e imagens[11] podem ser encontrados incluindo abordagens em imagens 2D[4][7] com observação da forma e textura das falhas, algumas incluindo reconstrução das imagens em 3D[11], e abordagens utilizando vídeos[12]. A maioria dos trabalhos encontrados utilizaram abordagens heurísticas ou de clusterização, porém nenhum dos trabalhos revisados utilizam classificação. Aprendizagem de máquina, ou inteligência artificial, diz respeito à implementação de software que consegue aprender autonomamente. É aplicada mais comumente em sistemas especialistas e de mineração de dados. E geralmente envolve abordagens com redes neurais artificiais e algoritmos genéticos.

1.1 Objetivo e Contribuições

O objetivo deste trabalho é implementar um algoritmo de detecção e classificação de falhas em vias públicas (buracos, rachaduras e manchas) que seja rápido permitindo um acompanhamento automatizado, eficiente e barato do principal meio de circulação de cargas e pessoas no Brasil e no mundo[13].

O software desenvolvido será testado em um computador do tipo desktop onde será avaliado qual algoritmo de aprendizagem de máquina apresenta o melhor desempenho na execução da aplicação. Pensando em aplicações futuras em hardwares com capacidade mais limitada, como Beaglebone e Raspberry, o método escolhido a ser testado é a análise 2D das imagens com foco na forma e textura para um melhor discernimento entre os diferentes tipos de falhas.

Com o auxílio das câmeras já presentes pela cidade, ou até mesmo câmeras no interior de veículos, as imagens seriam capturadas e enviadas via nuvem para o servidor do órgão responsável pelo monitoramento onde, com o auxílio de um computador, passariam por uma série de tratamentos e ao final do processamento classificaria as imagens dentre 3 tipos de falhas: buraco, rachadura ou mancha como mostra a Figura 1. Com o modelo desenvolvido espera-se obter um bom desempenho de tempo para que este possa ser aproveitado em aplicações de tempo real ou mesmo para uso em sistemas embarcados.

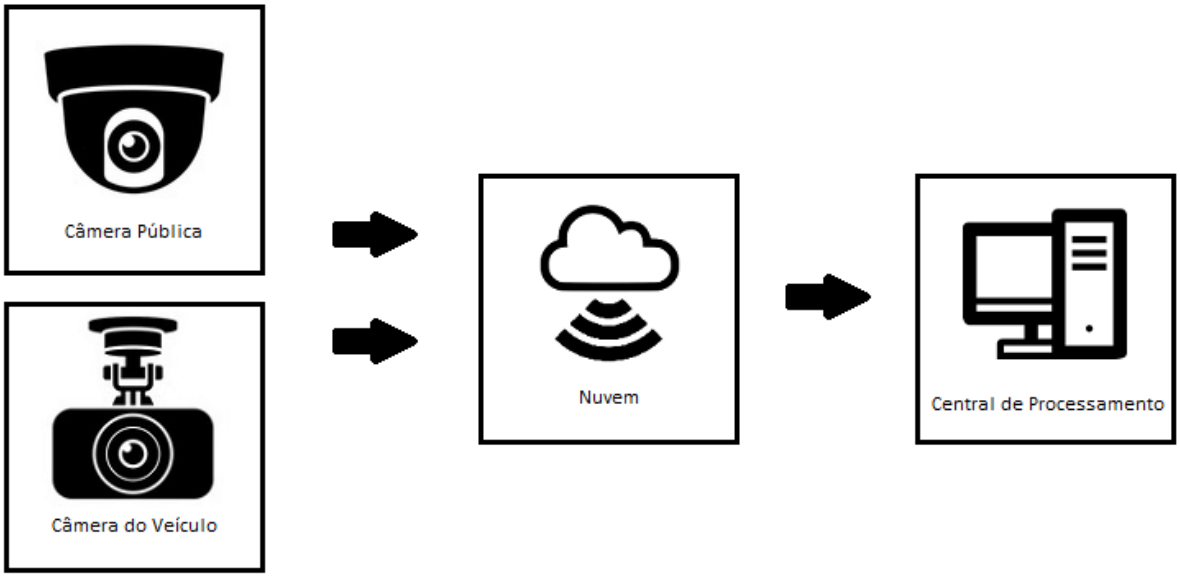


Figura 1 - Modelo de Coleta e Processamento de Imagens

Portanto uma das contribuições deste trabalho é utilizar técnicas de processamento juntamente com aprendizagem de máquina para a detecção e classificação de falhas em estradas. Alguns dos trabalhos existentes utilizam apenas técnicas de processamento de imagem. O uso de técnicas de aprendizagem

de máquina tornaria a análise mais adaptativa do que as abordagens heurísticas que tanto dependem do meio analisado.

Outra contribuição deste trabalho é testar diferentes algoritmos de aprendizagem de máquina trazendo uma comparação dos quatro algoritmos que melhor se destacaram neste experimento servindo como uma direção para novas pesquisas. Por fim, espera-se que o método desenvolvido sirva de base para futuros trabalhos que desejem testar novas características extraídas das imagens, juntamente com melhores formas de pré-processamento e até novos algoritmos de aprendizagem.

Esta monografia tem a seguinte estrutura: no Capítulo 2 serão descritos alguns trabalhos relacionados. Em seguida, no Capítulo 3, será apresentada a metodologia utilizada onde serão discutidas as etapas de pré-processamento, extração de características e classificação. No Capítulo 4 uma descrição do processo de implementação será explicitada mostrando as ferramentas utilizadas, explicando um pouco mais sobre a implementação do classificador e mostrando como configurar o ambiente de projeto. Por fim, no Capítulo 5, será feita uma reflexão dos resultados obtidos com base nos experimentos realizados utilizando as métricas de precisão, medida-F (F1-score) e sensibilidade (*recall*).

2 Trabalhos Relacionados

Ao longo do tempo diferentes métodos para detecção de falhas nas estradas foram estudados e desenvolvidos. Cada um deles utilizando mais ou menos recursos com alguns mais apropriados ou não para seu uso em hardwares de propósito específico. Este capítulo mostrará os métodos mais comumente utilizados com esse fim destacando-se aqueles baseados em vibrações[15-17], baseados em reconstrução 3D[18-24] e baseados em visão[25-29].

2.1 Métodos Baseados em Vibração

Com o uso de hardwares para a coleta dos dados, diversos métodos baseados em vibração foram desenvolvidos. Cada tipo de estresse, como buraco ou rachadura, pode causar diferentes impactos em veículos. Baseado nessa intensidade é possível identificar que tipo de falha o veículo está passando sobre.

Como vantagem esse método requer uma quantidade pequena de armazenamento de dados e é fácil em ser aplicado a modelos de tempo real. Porém, por retornar dados muito simples não traz muitos detalhes das características da falha como na abordagem baseada em visão que será discutido na seção 2.3. Além disso, tem uma dependência das condições de conservação de cada veículo necessitando de calibração manual e outros tipos de interferência como junções da pista e a perda de buracos localizados ao centro.

Alguns autores como De Zouysa et al.[15], Erickson et al.[16] e Mednis et al.[17] desenvolveram trabalhos com base nessa metodologia sendo os dois últimos para uso em aparelhos mobile.

2.2 Métodos Baseados em Reconstrução em 3D

Os métodos com base em reconstrução 3D da pista podem ser subdivididos em: métodos com uso de scanner a laser Chang et al.[18] e Li et al.[19], métodos com visão estéreo Wang[20], Hou et al.[21] e Staniek[22] e baseados em sensores do Microsoft Kinect Joubert et al.[23] e Moazzam et al.[24].

Através dos modelos digitais dos objetos criado a partir de pulsos de laser 3D refletidos[18] ou ainda usando técnicas de varredura transversal 3D de alta velocidade com o auxílio de laser infravermelho e câmera digital[19] é possível fazer com alta precisão a reconstrução e identificação de diferentes estresses encontrados em uma superfície. Apesar de sua precisão e sua possibilidade de uso em sistemas de tempo real o custo dos scanners a laser é significante e tem sido usado apenas para melhoria na precisão dos métodos em 3D.

Na abordagem com visão estéreo é possível fazer o uso da técnica com auxílio de duas ou mais câmeras[20-22], fazendo uma análise inicial da imagem 2D e dessa forma detectar falhas de uma superfície de dois ângulos diferentes e então

usá-las para recuperar propriedades para realizar a reconstrução 3D. Devido ao seu alto custo computacional esse tipo de solução acaba não sendo adequado para uso em sistemas de tempo real embora tenha uma alta acurácia.

O uso de sensor Kinect promete uma abordagem de baixo custo comparada ao uso de lasers infravermelho e câmeras industriais mas ainda é uma alternativa pouco testada[23][24]. Nessa abordagem utiliza-se a profundidade coletada das imagens para calcular o volume aproximado de cada buraco usando regra trapezoidal em curvas de área-profundidade.

2.3 Métodos Baseados em Visão

Neste tipo de abordagem, Koch and Brilakis[25], propuseram um método onde a imagem é dividida entre defeituosas e não defeituosas. As áreas com defeito tiveram seus formatos extraídos e suas texturas obtidas para serem comparadas com as regiões sem defeito. Caso fossem consideradas mais granularizadas eram classificadas como buraco. Este experimento teve auxílio de um robô controlado remotamente equipado com uma câmera para captura das imagens que foram divididas em grupos de treino e teste obtendo uma acurácia de 86%, precisão de 82% e 86% de recall (sensibilidade).

Um outro método sem a necessidade de equipamentos de alto custo, filtro e treinamento foi desenvolvido por Buza et al.[26] com o uso de processamento de imagem e agrupamento espectral. O método envolve a segmentação da imagem, extração da forma com agrupamento espectral, identificação e extração. Neste caso a precisão obtida foi por volta de 81% para um conjunto de 50 imagens.

Além dos métodos com imagem, existem estudos com o uso de vídeos que além da identificação permite determinar a magnitude dos buracos encontrados. Baseado em métodos com reconhecimento 2D e reconstrução 3D, Jog et al.[27] realiza a detecção e medida dos buracos, como largura, quantidade e profundidade, com o auxílio de uma câmera monocular equipada atrás de um carro. Como trabalho futuro o autor pretende estudar maneiras de detecção com reconstrução 3D em tempo real.

Lokeshwor et al.[28] propôs um método de detecção e classificação de falhas como buracos, rachaduras e manchas a partir de video clips. Neste método as imagens são separadas entre com defeitos e sem defeitos, onde o grupo com defeito tem suas imagens aprimoradas, segmentadas, tem suas características visuais extraídas (desvio padrão, circularidade e largura média) para detecção e classificação através de lógica de decisão e quantificação. Esta técnica tem sua acurácia piorada caso haja objetos, manchas negras, bueiros na pista que possam ser confundidos com alguns tipos de falhas.

Koch et al.[29] visando a melhoria do método baseado em imagem 2D, proposto por Koch e Brilakis[25], fez um método melhorado que atualiza a textura de áreas sem defeito e usa rastreamento de visão (vision tracking) para rastrear buracos em uma sequência de quadros. Desta forma, o desempenho na

comparação de textura e o tempo de computação foram melhorados consideravelmente. Porém, só é capaz de analisar um buraco por janela mas não em conjunto.

Os métodos baseados em vibração embora tenham vantagem na simplicidade dos dados, que serviria para uso em sistemas com hardware menos robusto, possuem grande desvantagem dos demais por sua dependência de calibração e dependendo da localização de algumas falhas pode deixar de detectá-las. Já o método baseado em reconstrução 3D embora possua alta precisão nos resultados não serviria para seu uso em dispositivos com baixo poder de processamento. Dessa forma, o método mais adequado seria o baseado em visão.

Porém, como foi visto a abordagem heurística traz muitas dependências que impossibilitariam seu uso quando expostas a condições diferentes às previstas ou mesmo a condições não previstas pelo método. Já o método aplicado neste trabalho, com o uso de classificação através de aprendizagem de máquina, poderá se adaptar a diferentes situações. Aprendizagem de máquina ainda traz a vantagem de poder melhorar seus resultados caso seja usado com novas características mais descritivas que venham a ser descobertas posteriormente.

3 Metodologia

Este capítulo descreve o método escolhido para realizar a classificação dos diferentes tipos de falhas presentes em vias públicas. As imagens escolhidas nos experimentos passam por um pré-processamento para só então haver a extração de suas características e por fim, a classificação.

O método escolhido foi testado em uma base de dados criada no Departamento de Elétrica e Eletrônica da Stellenbosch University na África do Sul[14]. Nesta base não houve preocupação em separar as estradas da paisagem ao redor o que afeta a eficácia do algoritmo desenvolvido, portanto, foi tratada isolando estas duas partes por meio de uma máscara. Além disso, a resolução das imagens foi diminuída para garantir um desempenho rápido e igualmente eficaz.

A base de dados foi dividida pelos autores entre imagens consideradas simples e complexas sendo estas compostas de buracos e manchas, tais como sombras de objetos e árvores. Esta não é composta de amostras de rachaduras o que foi tratado com o acréscimo de imagens extras pertencentes a esta classe.

O método proposto de detecção de buracos é baseado em técnicas de processamento de imagens e aprendizagem de máquina. As etapas do método proposto de detecção de anomalias são mostradas na Figura 2 e serão explicadas em detalhes nas próximas seções.

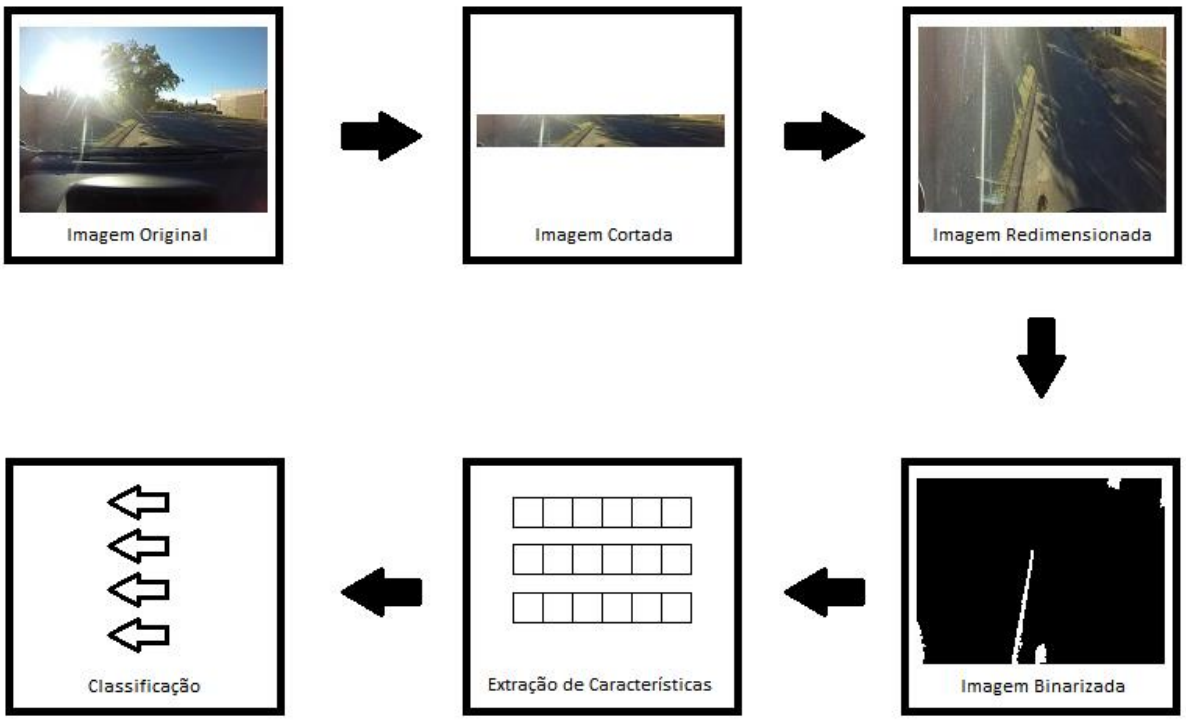


Figura 2 - Etapas para detecção de anomalias

3.1 Pré-processamento das imagens

O banco de imagens encontrado precisou passar por um pré-processamento antes da etapa de extração de dados e classificação. Embora a ideia central seja o uso de imagens tiradas a partir de postes presentes nas pistas, as imagens do banco usado foram tiradas a partir de uma câmera instalada no interior de um carro o que não deve modificar ou comprometer o método escolhido neste trabalho.

Estas precisaram ser redimensionadas já que possuíam uma resolução muito elevada comprometendo a velocidade de processamento do software. Além disso, elas precisaram passar por uma máscara para retirar elementos do carro como painel, parte do limpador de vidro e retrovisor de onde foram tiradas.

3.2 Extração de Características e Classificação

A título de classificação manual, os contornos foram classificados entre buraco, mancha, rachadura ou irrelevante. Buracos foram considerados como sendo falhas profundas geralmente com variação de cor mais acentuada em relação à área o redor e geralmente, mas não exclusivamente, com formato mais arredondado e com largura maior contidos nos limites da pista[28].

Manchas são consideradas como sendo sombras de árvores, sombras de carros ou outros objetos na pista, além, de manchas de óleo ou água. Estas possuem uma maior circularidade assim como os buracos, porém, sua variação de cor é menos acentuada[28]. Rachaduras são consideradas como sendo falhas menos profundas e de caráter mais alongado[28]. Os demais casos foram enquadrados como sendo ‘Irrelevante’.

Algumas abordagens com lógicas heurísticas são sensíveis às mudanças de parâmetros ocasionadas pela distância de captura das imagens e distorção (redimensionamento) sofrida. Além disso, a classificação de manchas sem a ajuda da análise de variação de cor tende a ser confundida com buracos ou até mesmo como algo irrelevante (ou seja, classificadas como ‘Irrelevante’). Diante disso, surgiu a ideia de aproveitar os parâmetros já anteriormente utilizados nestas abordagens heurísticas, juntamente com outros, para seu uso em conjunto a algoritmos de aprendizagem de máquina.

Os algoritmos de classificação escolhidos já estão implementados em bibliotecas de aprendizagem de máquina para Python. Para que isso fosse possível, algumas características foram extraídas dos contornos das imagens, entre elas: o desvio padrão, circularidade, alongamento, área, perímetro, largura e altura.

Estas características foram dispostas em um dicionário juntamente com a rotularização (labels) manual para uso nos métodos de aprendizagem. Dividiu-se o conjunto de imagens entre treino e teste e em seguida foi feita a predição para cada um dos algoritmos de aprendizagem de máquina testados. Estes resultados foram avaliados através do cálculo da precisão, medida-F e sensibilidade.

4 Implementação

Este capítulo trata do detalhamento do desenvolvimento do classificador de anomalias bem como as ferramentas, bibliotecas e configurações para a execução do classificador.

Neste trabalho, os modelos foram implementados respeitando algumas etapas: o pré-processamento de imagem para cortar, redimensionar e salvar as imagens tratadas em uma pasta e o extrator de características que lê essas imagens anteriormente salvas no pré-processamento, extrai características e as utiliza com aprendizagem de máquina para classificação das falhas.

4.1 Ferramentas

O sistema operacional escolhido para execução do software foi o Windows com a linguagem de programação escolhida sendo o Python. Além disso, a ferramenta Virtualenv foi utilizada para criação de um ambiente local já configurado com as dependências e bibliotecas necessárias. Estas bibliotecas consistem de funções de apoio ao processamento de imagens e de algoritmos de aprendizagem de máquina.

4.1.1 Windows SO

Neste projeto o sistema operacional utilizado foi o Windows 10 visto à larga utilização desse sistema e facilidade de instalação do ambiente de projeto utilizado. Para execução dos comandos de instalação e execução do software desenvolvido foi usado o Prompt de Comando (CMD) juntamente com Bloco de Notas para escrita do código, ambos presentes por padrão neste sistema operacional.

4.1.2 Virtualenv

Para facilitar a escalabilidade do software desenvolvido e uma prática e isolada configuração do ambiente de projeto este foi instalado com o auxílio da ferramenta de ambiente virtual, virtualenv. Dessa forma, as instalações das dependências necessárias para o projeto não influenciam na instalação global do ambiente de execução e vice-versa.

4.1.3 Python IDLE/Prompt

A linguagem de programação utilizada é o Python por ser largamente difundida, prática sintaxe, bastante suporte da comunidade acadêmica e grande quantidade de bibliotecas de processamento de imagens e algoritmos de aprendizagem de máquina.

4.1.4 Bibliotecas

Entre as bibliotecas de processamento de imagens utilizadas neste projeto estão a OpenCV e PIL. A biblioteca de aprendizagem de máquina escolhida foi a scikit-learn. Além destas, as principais bibliotecas auxiliares usadas foram a numpy, os e time.

4.2 Implementação do Classificador de Anomalias

4.2.1 Pré-Processamento

As imagens do banco de dados usado possuíam uma resolução elevada (3680x2760) o que poderia causar uma lentidão no processamento caso fossem utilizadas em sua forma original. Além disso, havia uma grande quantidade de elementos irrelevantes presentes externamente à pista.

Desta forma, foi necessário um redimensionamento das imagens que passaram a ter resolução de 511x384 e o uso de uma máscara para redução dos elementos externos (Figura 3). Para isso foi necessário o uso de bibliotecas do Python como a ‘os’ para leitura da pasta de imagens e PIL para abertura de cada imagem.

Figura 3 - Imagem original vs Imagem pré-processada

Note que o redimensionamento utilizado foi não proporcional o que mostrou ser melhor aproveitado posteriormente pela etapa de extração e classificação do que o redimensionamento proporcional. A Figura 4 mostra um exemplo de como ficaria a imagem sem o redimensionamento desproporcional:

Figura 4 - Imagem com redimensionamento proporcional

4.2.2 Extração de Características

Primeiramente as imagens pré-tratadas passam por uma extração dos contornos presentes em cada uma delas. Para isso, cada imagem é transformada para a escala cinza e em seguida, a imagem é binarizada usando a técnica de threshold.

No próximo passo, de cada imagem são extraídos seus contornos que passam por uma transformação de erosão, seguida de dilatação. Dessa forma, os ruídos presentes na forma de pontos em branco são em sua maioria retirados. O processo realiza ainda uma última erosão como forma de separar melhor um contorno do outro.

No final dessa etapa o resultado é uma imagem em preto e branco onde os contornos são dispostos em branco e o fundo em preto como mostra a Figura 5.

Figura 5 - Imagem após as transformações e extração de contornos

Apesar do tratamento sofrido a quantidade de contornos retornados ainda é grande exigindo uma seleção dos contornos mais relevantes. Para isso, é calculada a área de cada contorno que são filtrados permanecendo apenas os contornos com área maior do que $500cm^2$.

Terminada a etapa de transformação das imagens e seleção de contornos começa-se a extração de suas características visuais. Ao todo foram 7 características utilizadas onde são consideradas a textura da imagem, seu formato e sua dimensão. Dentre elas, estão o desvio padrão do contorno em relação ao seu

redor, sua circularidade, seu alongamento, a área do contorno, seu perímetro, sua largura e sua altura.

4.2.3 Classificação

Por fim, os dados extraídos foram dispostos em um dicionário de forma similar ao que acontece com o banco de dados 'load_iris' presente na biblioteca 'sklearn.datasets' do Python. Desse modo, torna-se possível a utilização desses dados em diferentes tipos de algoritmos de aprendizagem e predição presentes na biblioteca sklearn.

Dentre os algoritmos testados estão o Nearest Neighbors, Gaussian Process, Decision Tree, Random Forest, Multilayer Perceptron (Neural Net), Naive Bayes, Linear Support Vector Machine (Linear SVM), Radial Basis Function SVM (RBF SVM), AdaBoost e Quadratic Discriminant Analysis (QDA). Após a predição de cada algoritmo uma análise de sua precisão, medida-F e sensibilidade foi feita.

4.3 Configuração do Ambiente de Projeto

Primeiramente uma versão do Python deve ser instalada globalmente no sistema. A versão utilizada neste projeto foi o Python 2.7.11. Em seguida, supondo que o Python27 tenha sido instalado na pasta C:\, o virtualenv é instalado com o auxílio do comando pip como mostra a Figura 6.

```
$cd C:\Python27\Scripts\
$pip.exe install virtualenv
```

Figura 6 - Comando para instalação do virtualenv no Windows

Uma pasta para guardar o ambiente virtual bem como o próprio virtualenv são criados em [caminho-virtualenv] com o [nome-pasta] e [nome-projeto] escolhidos (Figura 7). Onde [caminho-virtualenv] é o endereço do diretório onde deseja-se criá-los, [pasta-virtualenv] o nome da pasta do ambiente e [nome-projeto] o nome da subpasta do ambiente do projeto.

```
$cd  [caminho-virtualenv]
$mkdir [pasta-virtualenv]
$cd c:\Python27\Scripts\
$virtualenv.exe [pasta-virtualenv]\[nome-projeto]
```

Figura 7 - Criação da pasta do ambiente virtual

Por fim, antes de compilar o projeto deve-se primeiro ativar o ambiente criado (Figura 8).

```
$cd [caminho-virtualenv]\[pasta-virtualenv]\[nome-projeto]\Scripts\activate
```

Figura 8 - Ativando o ambiente virtual

Feita a criação do ambiente virtual resta instalar cada uma das bibliotecas utilizadas pelo software desenvolvido. Para isso pode-se instalar usando os arquivos de extensão .whl [arquivo-wheel] em [pasta-wheel] disponibilizados juntamente com o código deste projeto executando os comandos da Figura 9. Onde [pasta-wheel] é o endereço da pasta com as bibliotecas necessárias e [arquivo-wheel] o arquivo de extensão .whl que deseja-se instalar.

```
$cd [pasta-wheel]
$pip install  [arquivo-wheel].whl
```

Figura 9 - Instalação das bibliotecas do projeto

Feito isso basta pegar o arquivo cv2.pyd da pasta “Wheel” e salvar na pasta “[pasta-virtualenv]\Lib\site-packages” do ambiente virtual que foi criado.

Após a instalação das bibliotecas e ativação do ambiente virtual, como mostrado acima, basta entrar na [pasta-fonte] do projeto e executar o [codigo-desejado] de extensão .py como mostrado na Figura 10. Onde [pasta-fonte] é o endereço da pasta com o código-fonte e [codigo-desejado] o nome do código que se deseja executar.

```
$cd [pasta-fonte]
$python [codigo-desejado].py
```

Figura 10 - Compilando os códigos-fonte do projeto

O programa inicia compilando o código de pré-processamento “PreProcessor.py” encontrado na pasta “Source” que salva o resultado dessa etapa na pasta “Cropped_Resized_Data”. Em seguida, executa o código de extração e classificação “Processor.py” em “Source” e salva seus resultados na pasta “Contours”. O projeto completo está disponível no Github[31].

5 Experimentos e Resultados

Este capítulo trará alguns resultados obtidos durante os experimentos mostrando exemplos das etapas de pré-processamento bem como do processamento do método proposto além de trazer uma análise do desempenho obtido através das métricas de precisão, medida-F, sensibilidade e tempo. O método proposto foi testado em uma base de dados criada no Departamento de Elétrica e Eletrônica da Stellenbosch University na África do Sul[14] e a quantidade de amostras utilizadas nos experimentos foram de 124 imagens.

5.1 Métricas de Análise

Como métricas de análise foram escolhidas a precisão, a medida-F e a sensibilidade de forma que os efeitos causados pela quantidade variante de amostras entre os diferentes tipos de classes (buraco, rachadura, mancha e irrelevante) fossem minimizados.

Antes de explicar tais métricas é importante entender o conceito de quatro parâmetros, também chamados métricas de confusão, utilizados no cálculo dessas três medidas de análise. São eles: verdadeiro positivo ou *true positive* (TP), verdadeiro negativo ou *true negative*(TN), falso positivo ou *false positive* (FP) e falso negativo ou *false negative* (FN).

O parâmetro TP denota os casos onde uma classe é na realidade positiva assim como seu valor predito. O parâmetro TN é para o caso em que o valor real é negativo assim como o valor da predição. Completando assim os casos de acerto. Já o parâmetro FP ocorre nos casos onde o valor na realidade é negativo mas a predição retornou positivo. De forma semelhante o parâmetro FN ocorre nos casos onde o valor real é positivo mas a predição deu negativo. Totalizando os casos de erro. A Tabela 1 mostra um esquema exemplificando cada um dos parâmetros explicados onde em azul encontram-se as predições corretamente avaliadas e em vermelho as predições que se deseja minimizar, ou seja, aquelas que foram preditas incorretamente[30].

	Valor Previsto		
Valor Real		+	-
	+	TP	FN
	-	FP	TN

Tabela 1 - Matriz de confusão genérica

Definido os parâmetros pode-se agora expressar cada uma das métricas em função deles.

No caso da acurácia a pergunta que se deseja responder é: “Do total (TP, TN, FP e FN), o quanto foi predito corretamente (TP e TN)?”. Neste caso a equação que expressa essa medida se torna[30]:

$$Acurácia = \frac{TP + TN}{TP + FN + FP + TN}$$

A precisão busca responder: “De todas as previsões positivas (TP e FP) quantas de fato eram positivas (TP)?”. Assim a precisão pode ser traduzida da seguinte forma[30]:

$$Precisão = \frac{TP}{TP + FP}$$

No caso da sensibilidade ela responde à pergunta: “De todas as classes de fato positivas (TP e FN) quantas foram previstas corretamente como positivas (TP)?”. A sensibilidade pode ser representada pela fórmula[30]:

$$Sensibilidade = \frac{TP}{TP + FN}$$

Por fim, tem-se a medida-F que nada mais é do que uma média harmônica da precisão e da sensibilidade, ou seja, ela indica a qualidade geral do modelo em questão. Ela pode ser expressa como[30]:

$$Medida\ F = \frac{2 * Precisão * Sensibilidade}{Precisão + Sensibilidade}$$

Para o caso onde as amostras entre as classes são divididas desproporcionalmente esta medida tem uma maior capacidade de minimizar esse tipo de interferência tornando-se uma medida mais adequada do que a acurácia. Como é o caso deste trabalho onde a quantidade de amostras das classes ‘rachadura’ e ‘buraco’ é muito menor do que as classes ‘mancha’ e ‘irrelevante’ esta foi a medida escolhida para substituir a acurácia.

5.2 Pré-processamento das Imagens

Na Figura 11 vemos um exemplo da imagem original do banco, utilizada nos experimentos, diminuída proporcionalmente em torno de 5 vezes, e sua respectiva cópia pré-processada e redimensionada desproporcionalmente.

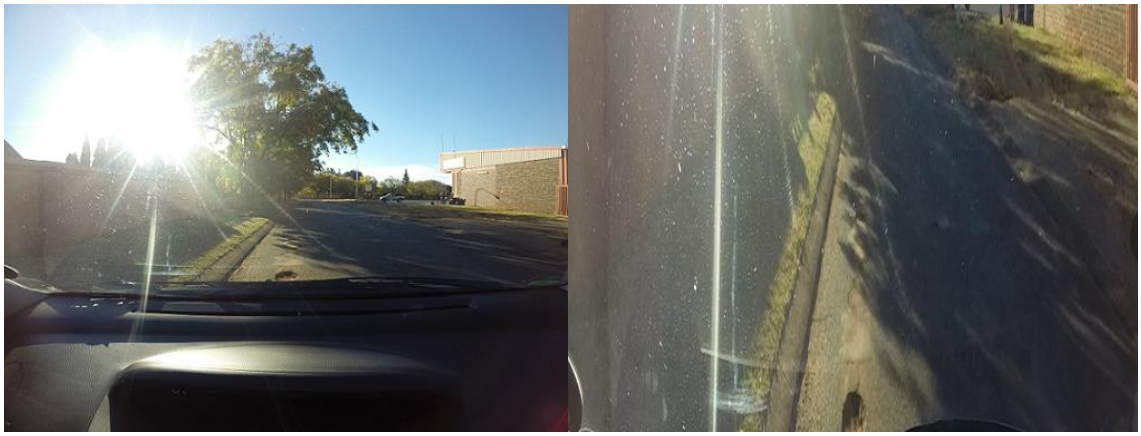
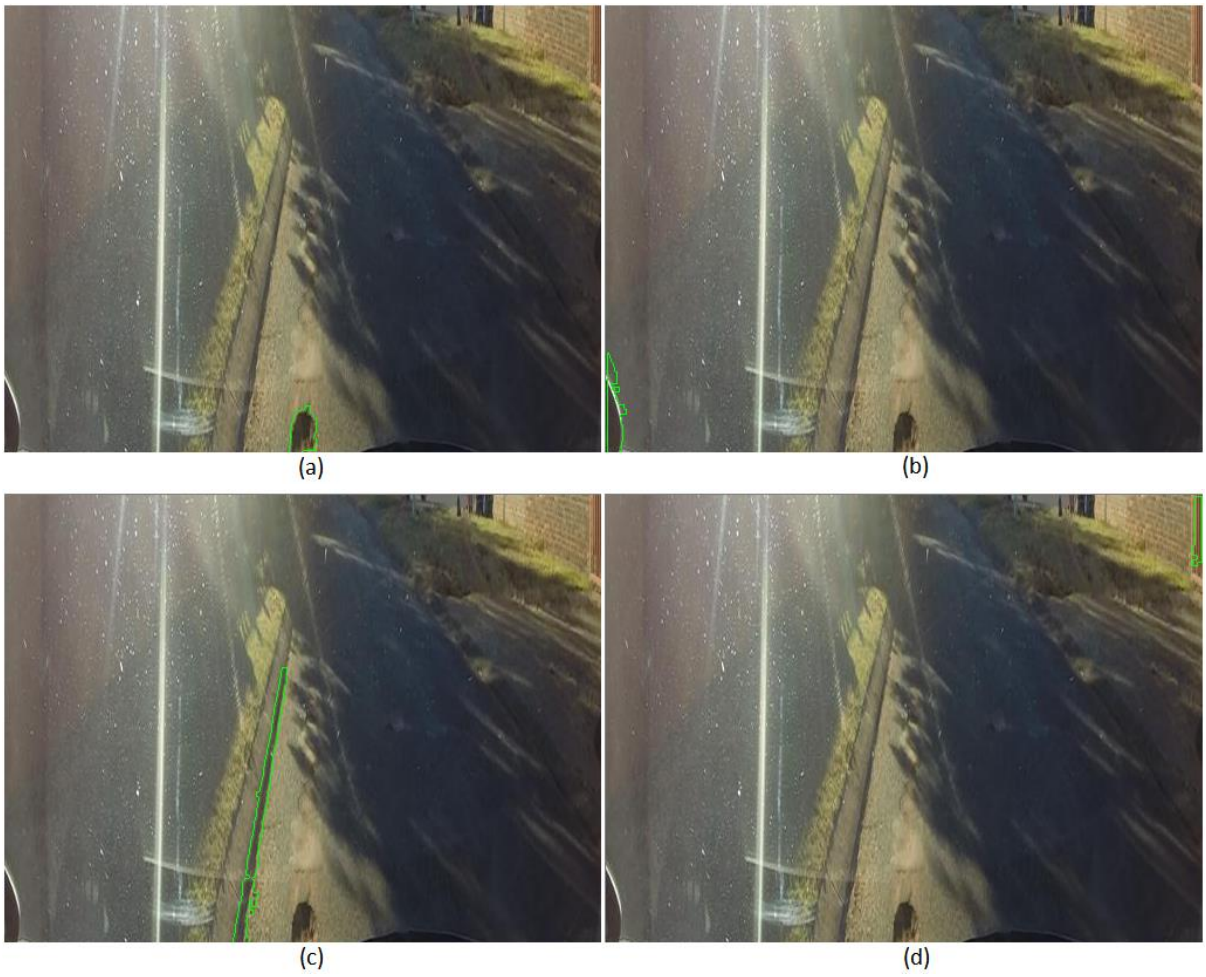


Figura 11 - Imagem original vs Imagem pré-processada

5.3 Processamento das Imagens

Após as etapas de pré-processamento, extração de características dos contornos foi possível obter alguns dos seguintes resultados da Figura 12.



(e)

Figura 12 - (a) Contorno de um buraco, (b) Contorno de algo irrelevante, (c) Contorno de uma mancha, (d) Contorno de algo irrelevante

Dos contornos da Figura 12 foi possível extrair as características apresentadas na Tabela 1.

Características	Contorno 1	Contorno 2	Contorno 3	Contorno 4	Contorno 5
Área	724,0	796,5	1825,0	362,0	444,0
Comprimento	119,1	218,5	559,6	139,7	89,3
Largura	24,0	17,0	47,0	9,0	27,0
Altura	41,0	86	236,0	60,0	21,0
Circularidade	0,6	0,2	0,1	0,2	0,7
Alongamento	3,1	38,0	667,9	75,2	1,8
Desvio Padrão	16,0	23,5	35,0	36,9	38,9

Tabela 2 - Características dos contornos da Figura 12

Com a extração das características, como as listadas na Tabela 2, os algoritmos Nearest Neighbors, Gaussian Process, Decision Tree, Random Forest, Neural Net, Naive Bayes, Linear SVM, RBF SVM, AdaBoost e QDA foram utilizados na classificação. Para cada um deles foram calculadas a precisão, medida-F e sensibilidade em relação a cada uma das classes de falhas (buraco, rachadura, mancha e irrelevante) para melhor explicitar o desempenho de cada algoritmo. Dentre eles, os quatro que melhor se destacaram em relação às métricas avaliadas foram o Linear SVM, QDA, Random Forest e Naive Bayes. Seus resultados se encontram nas Tabelas 3-6.

Falha	Precisão	Sensibilidade	Medida-F	Nº Amostras Teste
Buraco	0,75	0,30	0,43	10
Rachadura	0,00	0,00	0,00	4
Mancha	0,00	0,00	0,00	3
Irrelevante	0,59	0,95	0,73	21
Média/Total	0,52	0,61	0,51	38

Tabela 3 - Desempenho do algoritmo Linear SVM por classe de falha

Falha	Precisão	Sensibilidade	Medida-F	Nº Amostras Teste
Buraco	0,50	0,30	0,37	10
Rachadura	0,00	0,00	0,00	4
Mancha	0,00	0,00	0,00	3
Irrelevante	0,56	0,86	0,68	21
Média/Total	0,44	0,55	0,47	38

Tabela 4 - Desempenho do algoritmo QDA por classe de falha

Para as Tabelas 3-4 o tempo de execução total foi de 1,06s para os algoritmos Linear SVM e QDA juntos.

Falha	Precisão	Sensibilidade	Medida-F	Nº Amostras Teste
Buraco	0,67	0,20	0,31	10
Rachadura	0,00	0,00	0,00	4
Mancha	0,00	0,00	0,00	3
Irrelevante	0,57	0,95	0,71	21
Média/Total	0,49	0,58	0,48	38

Tabela 5 - Desempenho do algoritmo Random Forest por classe de falha

Falha	Precisão	Sensibilidade	Medida-F	Nº Amostras Teste
Buraco	0,57	0,40	0,47	10
Rachadura	0,00	0,00	0,00	4
Mancha	0,00	0,00	0,00	3
Irrelevante	0,55	0,76	0,64	21
Média/Total	0,46	0,53	0,48	38

Tabela 6 - Desempenho do algoritmo Naive Bayes por classe de falha

Por fim, para as Tabelas 5-6 o tempo de execução foi de 0,87s para os algoritmos Random Forest e Naive Bayes juntos.

Os resultados acima foram feitos em cima de uma base com 124 amostras. Neste primeiro caso pode-se notar que os algoritmos testados não obtiveram bom desempenho apresentando baixa precisão e medida-F e sensibilidade. Além disso, a quantidade de amostras pertencentes à classe ‘rachadura’ não é muito representativa já que a base escolhida não dispunha muito dela.

Por fim, as características referentes às classes ‘mancha’ e ‘irrelevante’, tais como a área e o comprimento, possuíam valores muito próximos podendo explicar a frequência de erro apresentada entre essas duas classes.

Visando retirar a interferência da baixa quantidade de amostras relativas à classe ‘rachadura’ e diminuir a taxa de erro provocada pela confusão entre as classes ‘mancha’ e ‘irrelevante’ foram realizados novos experimentos com a ausência do primeiro grupo e uma junção dos dois últimos grupos, respectivamente. Os resultados obtidos podem ser vistos nas Tabelas 7-10.

Falha	Precisão	Sensibilidade	Medida-F	Nº Amostras Teste
Buraco	0,67	0,40	0,50	10
Irrelevante	0,81	0,93	0,87	28
Média/Total	0,77	0,79	0,77	38

Tabela 7 - Desempenho do algoritmo Linear SVM por classe de falha

Falha	Precisão	Sensibilidade	Medida-F	N° Amostras Teste
Buraco	0,50	0,60	0,55	10
Irrelevante	0,85	0,79	0,81	28
Média/Total	0,76	0,74	0,74	38

Tabela 8 - Desempenho do algoritmo QDA por classe de falha

As Tabelas 7-8 obtiveram um tempo de execução de 0,72s para os algoritmos Linear SVM e QDA em conjunto.

Falha	Precisão	Sensibilidade	Medida-F	Nº Amostras Teste
Buraco	1,00	0,20	0,33	10
Irrelevante	0,78	1,00	0,88	28
Média/Total	0,84	0,79	0,73	38

Tabela 9 - Desempenho do algoritmo Random Forest por classe de falha

Falha	Precisão	Sensibilidade	Medida-F	Nº Amostras Teste
Buraco	0,50	0,70	0,58	10
Irrelevante	0,88	0,75	0,81	28
Média/Total	0,78	0,74	0,75	38

Tabela 10 - Desempenho do algoritmo Naive Bayes por classe de falha

Para as Tabelas 9-10 o tempo de execução foi de 0,79s para os algoritmos Random Forest e Naive Bayes em conjunto.

Com as modificações realizadas pôde-se notar uma melhoria na medida-F e precisão para os mesmos algoritmos anteriormente mostrados. Porém, a presença de elementos externos à pista ainda causa a presença de uma grande quantidade de elementos 'irrelevantes' com diferentes formatos o que dificulta os algoritmos de aprendizagem de máquina a criar um perfil distinto e característico para cada tipo de classe.

5.4 Outras Tentativas

As abordagens vistas funcionam, porém, com um tratamento mais elaborado na fase de pré-processamento das imagens os resultados poderiam melhorar consideravelmente. Com esse intuito uma tentativa de separação da pista dos demais elementos da imagem foi feita selecionando-se a parte com cor predominante e com variação próxima a ela. Com o pré-processamento inicial e a redução de elementos não importantes da imagem, geralmente, esta técnica retorna a cor da pista.

Após essa etapa o algoritmo convex hull[17] foi usado para separação total da pista. Porém, dependendo da iluminação da imagem os resultados podem não ser tão satisfatórios. Vale destacar ainda que o tempo de execução usando essa técnica aumenta consideravelmente acrescentando em torno de 9 minutos ao tempo geral de processamento. Isso se deve à necessidade em se percorrer toda a imagem ao se buscar a cor dominante e ao salvar posteriormente os pixels que se encontram em torno da cor dominante encontrada. As Figuras 13-16 mostram um caso de sucesso e de fracasso, respectivamente, utilizando a técnica explicada.

Figura 13 - Imagem com sucesso após a seleção da cor predominante

Figura 14 - Imagem com sucesso após a seleção da cor predominante e do algoritmo convex hull

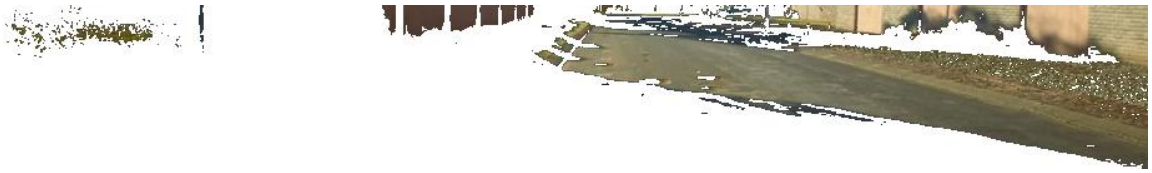


Figura 15 - Imagem sem sucesso após a seleção da cor predominante

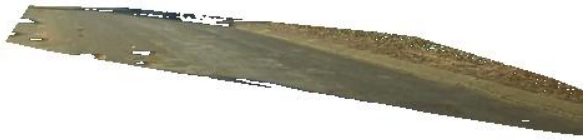


Figura 16 - Imagem sem sucesso após a seleção da cor predominante e do algoritmo convex hull

Fica claro que o uso dessa técnica não é adequado para aplicações de tempo real e nem possivelmente para sistemas embarcados e como não obteve bons resultados não houve o interesse em melhorar seu tempo de execução. Devido aos fracassos apresentados na tentativa descrita acima não foi considerado usá-lo com o método de processamento de imagem anteriormente proposto.

6 Conclusões e Trabalhos Futuros

Este trabalho teve por objetivo propor um método de detecção de falhas em vias públicas através do uso de algoritmos de aprendizagem de máquina utilizando as principais características usadas nesse tipo de aplicação extraídas de imagens 2D. Com o método foi possível obter precisão total de até 52% e medida-F de 51% para o caso com os quatro tipos de classes consideradas: ‘buraco’, ‘rachadura’, ‘mancha’ e ‘irrelevante’. E precisão de 84% e medida-F de 73% para o caso considerando apenas as classes ‘buraco’ e ‘irrelevante’.

A forma como a maioria dos artigos relacionados mostra a precisão e medida-F de seus algoritmos, considerando a estimativa geral sem distinção entre as classes, leva outros pesquisadores a pensarem que os métodos propostos fazem uma boa distinção entre as classes. O que pode acontecer é que a quantidade de elementos ‘irrelevantes’ na amostra de imagens analisada é tão grande que mesmo o algoritmo prevendo apenas elementos ‘irrelevantes’ a precisão e medida-F continuem sendo elevadas.

Uma boa forma de avaliar o real desempenho dos algoritmos para um determinado conjunto de imagens é fazendo a estimativa individual por classe de falha como foi feito neste trabalho. Dessa forma, fica mais evidente se o método proposto serve de fato para a aplicação em questão. O que não fica claro na maioria dos trabalhos relacionados estudados.

Os algoritmos de aprendizagem de máquina para classificação têm grande potencial para ser utilizado na detecção de falhas em vias, porém, é preciso que se estude uma forma melhor de separação entre os elementos externos e a pista como a tentativa mostrada na seção 5.3. Além disso, a presença de mais características principalmente as relacionadas à cor e textura pode servir para uma melhoria significante na previsão dos métodos existentes.

Como trabalhos futuros, vislumbra-se o uso de sistemas embarcados para diminuir o custo necessário para se implantar esse tipo de solução automatizada de forma mais abrangente, por exemplo, em transportes públicos ou carros particulares. Usando câmeras para capturar as imagens que seriam processadas em um hardware embarcado instalado internamente que enviaria apenas o resultado das análises para a nuvem.

Além de monitoramento para órgãos públicos esse tipo de solução embarcada poderia servir para benefício direto de motoristas com a criação de sistemas para alertar motoristas previamente sobre a existência de buracos ou animais em autoestradas. O tempo obtido nos experimentos foi relativamente baixo o que abre margem para o uso da técnica nesse tipo de aplicação de tempo crítico.

Referências

[1] JusBrasil. *As principais causas de acidentes de trânsito*. Disponível em: <https://paulocwb.jusbrasil.com.br/artigos/346024662/as-principais-causas-de-acidentes-de-transito>. Acesso em: 24 mar. 2018.

[2] O Carreteiro. *Buraco na pista é a notificação mais recebida pelo DNIT*. Disponível em:<http://www.ocarreteiro.com.br/buraco-na-pista-e-a-notificacao-mais-recebida-pelo-dnit/>. Acesso em: 24 mar. 2018.

[3] O Carreteiro. *Estradas federais têm defeito um mês após a entrega, afirma TCU*. Disponível em: <http://www.ocarreteiro.com.br/estradas-federais-tem-defeito-um-mes-apos-a-entrega-afirma-tcu/>. Acesso em: 24 mar. 2018.

[4] Koch, C.; Brilakis, I. *Pothole Detection in Asphalt Pavement Images*. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1474034611000036>. Acesso em: 26 mar. 2018.

[5] O Globo. *No Brasil, 80% das estradas não contam com pavimentação*. Disponível em: <https://oglobo.globo.com/brasil/no-brasil-80-das-estradas-nao-contam-com-pavimentacao-13710994>. Acesso em: 24 mar. 2018.

[6] O Carreteiro. *Faroeste Caboclo*. Disponível em: <http://www.ocarreteiro.com.br/revista/faroeste-caboclo/>. Acesso em: 24 mar. 2018.

[7] Ryu, S.; Kim, T.; Kim, Y. *Image-Based Pothole Detection System for ITS Service and Road Management System*. Disponível em: <https://www.hindawi.com/journals/mpe/2015/968361/>. Acesso em: 26 mar. 2018.

[8] Borges, P.S.; Carvalho, T. O. C.; Pires, T.; Torres, M.; Milian, F.M. *Embedded System for Detecting and Georeferencing Holes in Roads*. Disponível em: <http://ieeexplore.ieee.org/document/6096973/>. Acesso em: 26 mar. 2018.

[9] TranspoOnline. *Aplicativo DNIT MÓVEL permite o registro de ocorrências nas rodovias federais*. Disponível em: <http://transpoonline.com.br/lancamentos/aplicativo-dnit-movel-permite-o-registro-de-ocorrencias-nas-rodovias-federais/>. Acesso em: 26 mar. 2018.

[10] Britannica Academic. *Image Processing*. Disponível em: <http://academic.eb.com/levels/collegiate/article/image-processing/472572>. Acesso em: 24 mar. 2018.

[11] Kim. T; Ruy, S. *Review and Analysis of Potholes Detection Methods*. Disponível em:<https://www.researchgate.net/publication/308174730_Review_and_analysis_of_pothole_detection_methods>. Acesso em: 26 mar. 2018.

[12] Huidrom, L.; Das, L. K. *Robust Method for Automated Segmentation of Frames with-without Distress from Road Surface Video Clips*. Disponível em: <https://www.researchgate.net/publication/277608807_Robust_Method_for_Automated_Segmentation_of_Frames_withwithout_Distress_from_Road_Surface_Video_Clips>. Acesso em: 26 mar. 2018.

[13] Ilos. *Transporte de cargas e a encruzilhada do Brasil para o futuro*. Disponível em: <http://www.ilos.com.br/web/tag/matriz-de-transportes/>. Acesso em: 25 mar. 2018.
[14] Nienaber, S.; Booysen, M.J.; Kroon, R.S. *Dataset of images used for pothole detection*. Disponível em: <https://www.researchgate.net/publication/282807920_Dataset_of_images_used_for_pothole_detection> Acesso em: 28 mar. 2018.
[15] De Zoysa,K.; Keppitiyagama, C.; Seneviratne, G. P. ; Shihan, W.W.A.T. *A public transport system based sensor network for road surface condition monitoring*, Em *Proceedings of Workshop on Networked Systems for Developing Regions* (2007), 1-6.
[16] Erikson, J.; Girod, L.; Hull, B. *The pothole patrol: using a mobile sensor network for road surface monitoring*, Em *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services* (2008), 29-39.
[17] Mednis, A.; Strazdins, G.; Zviedris, R.; Kanonirs, G.; Selavo, L. *Real time pothole detection using Android smartphones with accelerometers*, Em *Proceedings of the International Conference on Distributed Computing in Sensor Systems and Workshops* (2001), 1-6.
[18] Chang, K.T.; Chang, J. R.; Liu, J. K. *Detection of pavement distress using 3D laser scanning technology*, Em *Proceedings of the ASCE International Conference on Computing in Civil Engineering* (2005), 1-11.
[19] Li, Q.; Yao, M.; Yao, X.; Xu, B. *A real-time 3D scanning system for pavement distortion inspection, Measurement Science and Technology*, Vol. 21, No. 1, (2009), 15702-15709.
[20] Wang, K. C. P.; *Challenges and feasibility for comprehensive automated survey of pavement conditions*, Em *Proceedings of 8th International Conference on Applications of Advanced Technologies in Transportation Engineering* (2004), 531-536.
[21] Hou, Z.; Wang, K. C. P.; Gong, W. *Experimentation of 3D pavement imaging through stereovision*, Em *Proceedings of the International Conference on Transportation Engineering* (2007), 376-381.
[22] Staniek, M. *Stereo vision techniques in the road pavement evaluation*, Em *Proceedings of the XXVIII International Baltic Road Conference* (2013), 1-5.
[23] Joubert, D.; Tyatyantsi, A.; Mphahlehle, J.; Manchidi, V. *Pothole tagging system*, Em *Proceedings of the 4th Robotics and Mechatronics Conference of South Africa* (2011), 1-4.
[24] Moazzam, I.; Kamal, K.; Mathavan, S.; Usman, S.; Rahman, M. *Metrology and visualization of potholes using the Microsoft Kinect sensor*, Em *Proceedings of the 16th International IEEE Annual Conference on Intelligent Transportation Systems* (2013), 1284-1291.
[25] Koch, C.; Brilakis, I. *Pothole detection in asphalt pavement images*, *Advanced Engineering Informatics*, Vol. 25 (2011), 507-515.

[26] Buza, E., Omanovic, S.; Huseinnovic, A. *Stereo vision techniques in the road pavement evaluation*, Em *Proceedings of the 2nd International Conference on Information Technology and Computer Networks*, (2013), 48-53.
[27] Jog, G. M.; Koch, C.; Golparvar-Fard, M.; Brilakis, I. *Pothole properties measurement through visual 2D recognition and 3D reconstruction*, Em *Proceedings of the ASCE International Conference on Computing in Civil Engineering* (2012), 553- 560.
[28] Lokeshwor, H.; Das, L. K.; Sud, S. K. *Method for automated assessment of potholes, cracks and patches from road surface video clips*, *Procedia – Social and Behavioral Sciences*, Vol. 104 (2013), 312-321.
[29] Koch, C., Jog, G. M.; Brilakis, I. *Pothole detection with image processing and spectral clustering*, *Journal of Computing in Civil Engineering*, Vol. 27, No. 4, (2013), 370-378.
[30] Exsilio Solutions. *Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures.* Disponível em: <http://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/>. Acesso em: 14 jun. 2018.
[31] Github. *Processamento de Imagem e Aprendizagem de Máquina para Classificação de Anomalias em Vias Públicas*. Disponível em: <https://github.com/carolpm/potholes-patches-cracks>. Acesso em: 18 jun. 2018.

Apêndices

APÊNDICE A – Código Fonte do Pré-Processador

```
import PIL
from PIL import Image
import os

def resize_image (img):
    wsize = img.size[0]/2
    hsize = img.size[1]/2
    img_resized = img.resize((wsize,hsize), PIL.Image.ANTIALIAS)
    return img_resized

def crop_image (img, x, y, w, h):
    img_cropped = img.crop((x, y, w+x, h+y))
    return img_cropped


def adjust_image_size (img):
    wsize = 511
    hsize = 384
    img_resized = img.resize((wsize,hsize), PIL.Image.ANTIALIAS)
    return img_resized


def save_image (img, destination):
        img.save(destination)
        img.close()

if __name__ == "__main__":
    source = 'Original_Data'
    destination = 'Cropped_Resized_Data'

    x = 0
    y = (1272/2)
    w = (3680/2)
    h = (500/2)

    for i in range(len(os.listdir(source))):
        path_s = os.listdir(source)
        path_d = range(len(path_s))
        path_d = os.path.join(destination, path_s[i])
        path_s = os.path.join(source, path_s[i])
        if (os.path.isfile(path_s)):
           files = path_s
        if (files.lower().endswith('.jpg')):
            jpgs = files
        img = PIL.Image.open(jpgs)
        img = resize_image(img)
        img = crop_image(img, x, y, w, h)
        img = adjust_image_size(img)
        save_image(img, path_d)
```

APÊNDICE B – Código Fonte do Processador

```
import numpy as np
import cv2
import math
import os
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.gaussian_process.kernels import RBF
from sklearn.ensemble import AdaBoostClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import classification_report
import time
import warnings
global_features = []
global_labels = []
global_label_names = []
global_classifiers = []
global_classifiers_names = []

'''Funcao que transforma a imagem para escala em preto e branco, binariza,
aplica erosao, dilatacao e, por fim, nova erosao
e em seguida a imagem binarizada eh invertida e retornada'''
def transform_image (img):
      thresh = 127
      maxValue = 255
      kernel = np.ones((5,5),np.uint8)

      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
      ret, th =
cv2.threshold(gray,thresh,maxValue,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
      opening = cv2.morphologyEx(th, cv2.MORPH_OPEN, kernel)
      erosion = cv2.erode(opening,kernel,iterations = 1)
      erosion = cv2.bitwise_not(erosion)
      return cv2.findContours(erosion, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

'''Funcao que retorna a quantidade de contornos de uma imagem'''
def num_contours (contours):
      return len(contours)
```

```
'''Funcao que retorna a area de um contorno'''
def area (contour):
        a = cv2.contourArea(contour)
        if a == None:
                a = 0.0
        return a

'''Funcao que retorna o comprimento de um contorno'''
def length (contour):
        l = cv2.arcLength(contour,True)
        if l == None:
                l = 0.0
        return l

'''Funcao que retorna apenas os contornos com area dentro de um range'''
def contour_filter(area):
        a = cv2.contourArea(area)
        if a == None:
                a = 0.0
        return a >= 300 and a <= 2000

'''Funcao que retorna a largura de um contorno'''
def width (contour):
        x,y,w,h = cv2.boundingRect(contour)
        if w == None:
                w = 0.0
        return w

'''Funcao que retorna a altura de um contorno'''
def heigth (contour):
        x,y,w,h = cv2.boundingRect(contour)
        if h == None:
                h = 0.0
        return h

'''Funcao que retorna a circularidade de um contorno'''
def circularity (contour):
        c
=(4*math.pi*cv2.contourArea(contour))/((cv2.arcLength(contour,True)**2))
        if c == None:
                c = 0.0
        return c

'''Funcao que retorna o alongamento de um contorno'''
def elongation (m):
        x = m['mu20'] + m['mu02']
        y = 4 * m['mu11']**2 + (m['mu20'] - m['mu02'])**2
        e = (x + y**0.5) / (x - y**0.5)
        if e == None:
                e = 0.0
        return e

'''Funcao que retorna o desvio padrao de um contorno'''
```

```
def standard_deviation (imgcontour):
        (means, std) = cv2.meanStdDev(imgcontour)
        for k in std:
                for m in k:
                        std = m
        if std == None:
                std = 0.0
        return std

'''Funcao que salva as imagens processadas na pasta destino
especificada'''
def save_image(img, destination_img):
        cv2.imwrite(destination_img, img)


def main():
        #Pasta origem dos contornos Pre-Processados
        source = 'Cropped_Resized_Data'
        #Pasta destino dos contornos retornados pelos classificadores
"Nearest Neighbors", "Gaussian Process", "Decision Tree", "Random Forest",
"Neural Net", "Naive Bayes"
        '''destination = 'Contours/Predict_Proba_Classifiers' '''
        #Pasta destino dos contornos retornados pelos classificadores
"Linear SVM", "RBF SVM", "AdaBoost", "QDA"
        destination = 'Contours/Decision_Function_Classifiers'
        path_s = os.listdir(source)
        path_d = range(len(path_s))
        dictionary = {}
        #Para classificar entre buraco, rachadura, mancha e algo
irrelevante
        dictionary['target_names'] = np.array(['pothole', 'crack',
'patch', 'irrelevant'], dtype='|S9')
        #Para classificar apenas entre buraco e algo irrelevante
        '''dictionary['target_names'] = np.array(['pothole',
'irrelevant'], dtype='|S9')'''
        dictionary['target'] = []
        dictionary['image_name'] = []
        dictionary['feature_names'] = np.array(['area', 'length', 'width',
'heigth', 'circularity', 'elongation', 'standard deviation'],
dtype='|S23')
        dictionary['data'] = []
        list_name = []
        list_features = []

        #Predict Proba Algorithms Names
        '''classifiers_names = [
                "Nearest Neighbors",
                "Gaussian Process",
                "Decision Tree",
                "Random Forest",
                "Neural Net",
                "Naive Bayes"
                ]'''
```

```
        #Decision Function Algorithms Names
        classifiers_names = [
                "Linear SVM",
                "RBF SVM",
                "AdaBoost",
                "QDA"
                ]

        global global_classifiers_names
        global_classifiers_names = classifiers_names
        #Predict Proba Algorithms
        '''classifiers = [
            KNeighborsClassifier(3),
            GaussianProcessClassifier(1.0 * RBF(1.0)),
            DecisionTreeClassifier(max_depth=5),
            RandomForestClassifier(max_depth=5, n_estimators=10,
max_features=1),
            MLPClassifier(alpha=1),
            GaussianNB()
            ]'''
        #Decision Function Algorithms
        classifiers = [
            SVC(kernel="linear", C=0.025),
            SVC(gamma=2, C=1),
            AdaBoostClassifier(),
            QuadraticDiscriminantAnalysis()
            ]

        global global_classifiers
        global_classifiers = classifiers

        totalNumContours = 0;

        for i in range(0, len(path_s)):
                list_name.append(path_s[i])
                path_d[i] = os.path.join(destination, path_s[i])
                path_s[i] = os.path.join(source, path_s[i])
                if os.path.isfile(path_s[i]):
                        arquivos = path_s[i]
                else:
                        continue
                if arquivos.lower().endswith('.jpg'):

                        jpgs = arquivos
                else:
                        continue
                img = cv2.imread(jpgs,1)

                imgcontours, contours, hierarchy = transform_image(img)
                #cv2.imshow('Contornos',imgcontours)
                contoursRange = contours
                contoursRange = filter(contour_filter, contours)
```

```
                j = 0
                imgcnt = np.zeros(imgcontours.shape[:2])
                imag = img.copy()
                imgsave = img.copy()
                print('\n-------------------------------------------------
-----------')
                print('Image {} ({})' .format(i+1, list_name[i]))
                print ('Number of Contours:
{}\n'.format(num_contours(contoursRange)))
                totalNumContours += num_contours(contoursRange)
                for contour in contoursRange:
                        lista = []
                        a = area(contour)
                        print ('Area[{}]: {}' .format(j, a))
                        l = length(contour)
                        print ('Length[{}]: {}' .format(j, l))
                        w = width(contour)
                        print ('Width[{}]: {}' .format(j, w))
                        h = heigth(contour)
                        print ('Heigth[{}]: {}' .format(j, h))
                        c = circularity(contour)
                        print ('Circularity[{}]: {}' .format(j, c))
                        m = cv2.moments(contour)
                        e = elongation(m)
                        print ('Elongation[{}]: {}' .format(j, e))
                        #imgcnt = np.zeros(imgcontours.shape[:2])
                        imag = img.copy()
                        imgcnt = cv2.drawContours(image=imgcnt,
contours=[contour], contourIdx=-1, color=(255, 0, 0),
thickness=cv2.FILLED)
                        imag = cv2.drawContours(image=imag,
contours=[contour], contourIdx=-1, color=(0, 255, 0))
                        imgsave = cv2.drawContours(image=imgsave,
contours=[contour], contourIdx=-1, color=(0, 255, 0))
                        std = standard_deviation(imgcnt)
                        std_f = 0.0
                        '''for k in std:
                                for m in k:
                                        std = m'''
                        print ('Standard Deviation[{}]: {}\n' .format(j,
std))
                        name_image = 'Image {} / Contour {}' .format(i+1,
j+1)
                        #cv2.imshow(name_image, imgcnt)
                        #cv2.imshow(name_image, imag)
                        cv2.waitKey(0)
                        cv2.destroyAllWindows()
                        lista.append(i)
                        lista.append(j)
                        lista.append(a)
                        lista.append(round(l, 2))
                        lista.append(w)
                        lista.append(h)
```

```
                        lista.append(c)
                        lista.append(round(e, 2))
                        lista.append(round(std, 2))
                        list_features.append(a)
                        list_features.append(l)
                        list_features.append(w)
                        list_features.append(h)
                        list_features.append(c)
                        list_features.append(e)
                        list_features.append(std)
                        dictionary['data'].append(list_features)
                        list_features = []
                        j+=1
                save_image(imgsave, path_d[i])
                print ('Total of Contours: {}\n'.format(totalNumContours))
                #cv2.imshow(name_image, imag)
                #cv2.imshow(name_image, imgcnt)
                cv2.waitKey(0)
                cv2.destroyAllWindows()

        dictionary['image_name'] = np.array(list_name, dtype='|S23')
        dictionary['data'] = np.array(dictionary['data'])
        #Labels para 4 classes de falhas
        #Caso a quantidade de imagens seja mudada alterar as labels abaixo
tambem
        dictionary['target'] = np.array([3, 0, 3, 0, 3, 3, 3, 3, 3, 0, 3,
2, 3, 3, 0, 3, 1, 3, 3, 3, 0, 0, 2, 0, 0, 2, 2, 3, 0, 2, 3, 0, 3, 3, 3, 0,
3, 3, 0, 2, 3,
                                            0, 3, 2, 3, 3, 0, 0, 0, 3, 3, 3,
0, 3, 3, 3, 0, 3, 3, 3, 3, 0, 2, 3, 3, 3, 0, 2, 3, 3, 3, 0, 3, 3, 3, 3, 3,
3, 1, 3, 0, 3,
                                            3, 3, 0, 0, 0, 0, 2, 1, 0, 1, 0,
0, 3, 0, 1, 3, 3, 3, 3, 0, 0, 1, 2, 3, 3, 3, 3, 3, 3, 0, 2, 3, 3, 3, 3, 3,
0, 3, 3, 3, 3, 3])

        #Labels para 2 classes de falhas
        '''dictionary['target'] = np.array([1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
0, 1, 1, 0, 1, 1,
                                            0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 0, 1,
                                            1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
0, 1, 1, 1, 1, 1])'''

        label_names = dictionary['target_names']
        labels = dictionary['target']
        feature_names = dictionary['feature_names']
        features = dictionary['data']
        global global_features
        global_features = features
        global global_labels
```

```
        global_labels = labels
        global global_label_names
        global_label_names = label_names

        warnings.filterwarnings("ignore")

def stats():
        # Binariza saidas. Exemplo: para a label = 4 o resultado sera 001.
        lb = preprocessing.LabelBinarizer()
        labels_bin = lb.fit_transform(global_labels)
        n_classes = labels_bin.shape[1]
        n_test_contours = int(0.31*global_labels.size)

        train, test, train_labels, test_labels =
train_test_split(global_features, labels_bin, test_size=n_test_contours,
shuffle=False)

        for index, k, clf in zip(range(1,len(global_classifiers)+1),
global_classifiers_names, global_classifiers):
                #Aprende a predizer cada classe contra as outras
                classifier = OneVsRestClassifier(clf)

                #Para os classificadores "Nearest Neighbors", "Gaussian
Process", "Decision Tree", "Random Forest", "Neural Net", "Naive Bayes"
                '''label_score = classifier.fit(train,
train_labels).predict_proba(test)'''
                #Para os classificadores "Linear SVM", "RBF SVM",
"AdaBoost", "QDA"
                label_score = classifier.fit(train,
train_labels).decision_function(test)

                preds = lb.inverse_transform(label_score)
                test_l = lb.inverse_transform(test_labels)

                print('  Real  ({}): {}' .format(k, test_l))
                print('Predicted({}): {}' .format(k, preds))
                #Calcula precisao, sensibilidade e medida-F
                print('Report({}):' .format(k))
                print(classification_report(test_l, preds, target_names =
global_label_names))
                print('---------')

        print("Runtime: %s seconds" % (time.time() - start_time))
        print('---------')
        warnings.filterwarnings("ignore")

if __name__ == "__main__":
        start_time = time.time()
        main()
        stats()
```