

**UNIVERSIDADE ESTADUAL DE CAMPINAS  
FACULDADE DE ENGENHARIA MECÂNICA**

Relatório Final  
Trabalho de Graduação

**Estudo Prático do Algoritmo Haar-Cascade via Integração API com  
Aplicação Híbrida**

Autor: Carlos Vinícius Araki Oliveira  
Orientador: Prof. Dr. Eric Fujiwara

Campinas, Julho de 2020



**UNIVERSIDADE ESTADUAL DE CAMPINAS  
FACULDADE DE ENGENHARIA MECÂNICA**

Relatório Final  
Trabalho de Graduação

**Estudo Prático do Algoritmo Haar-Cascade via Integração API com  
Aplicação Híbrida**

Autor: Carlos Vinicius Araki Oliveira  
Orientador: Prof. Dr. Eric Fujiwara

Trabalho de Conclusão de Curso apresentado à Comissão de Graduação da Faculdade de Engenharia Mecânica, como requisito para a obtenção do título de Engenheiro de Automação e Controle.

Campinas, 2020  
SP - Brasil

## Agradecimentos

Este estudo não seria plausível de existência sem o apoio de diversas pessoas, às quais presto minha singela homenagem.

Ao Prof. Dr. Eric Fujiwara, pelas orientações e dicas em como executar de melhor forma esse estudo. À família e o meu amor, pela motivação psicologicamente em progredir nesse trabalho. Aos amigos, pelo apoio demonstrado ao longo de todo o período em que me dediquei a este projeto.

# Lista de Figuras

3.1	Padrões Haar Features . . . . .	XIX
3.2	Diagrama do Algoritmo em Cascata . . . . .	XXIII
3.3	Relacionamento Banco de dados . . . . .	XXIV
3.4	Carta utilizada para treinamento . . . . .	XXV
3.5	Carta de retorno pós processamento . . . . .	XXVII
3.6	Tela Connection . . . . .	XXVIII
3.7	Tela App . . . . .	XXIX
3.8	Tela Get Picture . . . . .	XXX
3.9	Tela Options . . . . .	XXXI
3.10	Tela Compare . . . . .	XXXII
3.11	Fluxo de Telas . . . . .	XXXIII
4.1	Sensibilidade de Escala . . . . .	XXXVI
4.2	Acurácia . . . . .	XXXVII
4.3	Precisão . . . . .	XXXVII
4.4	Sensibilidade de Cor . . . . .	XXXIX



# Lista de Símbolos

$tPos$	True Positive
$fPos$	False Positive
$tNeg$	True Negative
$fNeg$	False Negative
$ACC$	Accuracy
$PPV$	Precision
$FDR$	False Discovery Rate
$NPV$	Negative Predictive Value
$FOR$	False Omission Rate
$TPR$	Sensitivity
$FNR$	Miss Rate
$FPR$	False Positive Rate
$TNR$	Specificity





# Sumário

<b>Lista de Símbolos</b>	<b>V</b>
<b>1 Introdução</b>	<b>IX</b>
1.1 Objetivos . . . . .	X
<b>2 Revisão da Literatura</b>	<b>XI</b>
2.1 Estado da Arte . . . . .	XI
<b>3 Materiais e Métodos</b>	<b>XIII</b>
3.1 Software . . . . .	XIV
3.1.1 Aplicativo . . . . .	XIV
3.1.2 API . . . . .	XIV
3.1.3 Banco de Dados . . . . .	XV
3.2 Hardware . . . . .	XV
3.2.1 Requisitos Computador Utilizado . . . . .	XV
3.2.2 Requisitos do Aparelho Mobile Utilizado . . . . .	XVI
3.3 Metodologia . . . . .	XVII
3.3.1 Análise Teórica . . . . .	XVIII
3.3.1.1 Haar-like features . . . . .	XVIII
3.3.1.2 Imagem Integral . . . . .	XIX
3.3.1.3 AdaBoost em Cascade . . . . .	XXI
3.3.2 Implementação . . . . .	XXIV
3.3.2.1 Construção API . . . . .	XXIV
3.3.2.1.1 ERD (Entity Relationship Diagram) . . . . .	XXIV
3.3.2.1.2 API Django integrado OpenCV . . . . .	XXV
3.3.2.2 Construção Aplicação Híbrida . . . . .	XXVII
<b>4 Resultados</b>	<b>XXXV</b>
4.1 Vantagens e Limitações . . . . .	XXXVIII
<b>5 Conclusão</b>	<b>XLI</b>
5.1 Sugestões para trabalhos futuros . . . . .	XLI
<b>A GitHub</b>	<b>XLVII</b>

<b>B Resultados de classificação</b>	<b>XLIX</b>
<b>C Imagens Utilizadas</b>	<b>LIII</b>

# Capítulo 1

## Introdução

Uma das principais razões da sobrevivência humana em tempos primitivos é a evolução tecnológica da comunicação em poder preservar e transmitir informação, possibilitando a organização humana em estado de sociedade, havendo a partir disso uma separação entre a Pré-História e a História [4]. As primeiras formas de preservar a informação ao longo do tempo é através da imagem.

Tão importante essa representação visual de algo existente na realidade do ser humano em um estado presente e que será protegido para utilização de um estado futuro, seja pelo mesmo indivíduo ou por outros membros da sociedade, que a evolução da escrita (uma representação gráfica de informação) foi criada de forma independente em regiões diferentes [12].

Depois de milênios de evolução tecnológica, o homem está entrando na era da Hiper História [4]. Onde além de gravar e transmitir informação, a tecnologia atual permite processar a mesma, sem o resguardo de um ser humano atuando diretamente.

Dentro do globo de tecnologias de processamentos de dados, um dos ramos com destaque acalorado nos últimos tempos é o processamento de imagem digital com um enfoque na identificação computacional de padrões [3] [13].

Esse tema traz um leque de possibilidades de melhoria em diversas áreas da sociedade, como na área biomédica [15] [21], controle de tráfego de trânsito [10] e até mesmo detecção de falhas estruturais por vibração [19].

Uma das maiores possibilidades de melhoria levantadas sobre o processamento de imagem é o tempo utilizado em paralelo com a capacidade computacional para identificar um padrão. Uma das primeiras pesquisas que obteve um resultado satisfatório nessa modalidade são os baseados nos classificadores Haar-Features, proposto por Viola e Jones com os resultados mostrados em [17][16] e [8]

Visto a estabilidade e robustez desses classificadores, esse trabalho o desenvolverá de forma prática. Identificando cartas de baralho a partir de um aplicativo desenvolvido para comunicação direta de uma API, que além processar a imagem, será responsável pela organização do banco de dados.


## 1.1 Objetivos

Os objetivos traçados neste trabalho de graduação estão enumerados a seguir:

1. Realizar uma análise teórica do algoritmo Haar Cascade
2. Criação prática de uma API para reconhecimento da carta de Número Três de Naipes de Espadas
3. Criação de uma aplicação simples que alimentará e consumirá os dados fornecidos pela API
4. Análise de desvantagens da utilização do Haar Cascade e base para trabalhos futuros

## Capítulo 2

# Revisão da Literatura

 ramo de visão computacional expandiu consideravelmente nos últimos anos. Com diferentes abordagens para a detecção dos algoritmos e de classificadores, essa seção vislumbra pincelar no Estado da Arte algumas dessas abordagens.

### 2.1 Estado da Arte

1. Fuzzy Supervised Classification of Remote Sensing Images [18]

Nesse artigo é descrito um método de classificação supervisionada fuzzy em informações geográficas representadas como conjuntos fuzzy. O algoritmo utilizado consiste em estimar os parâmetros fuzzy e executar uma partição nebulosa do espaço espectral.

2. Hyperspectral Image Classification Using Dictionary-Based Sparse Representation [2]

É apresentado nessa pesquisa um algoritmo de dispersão para classificação de imagens hiperespectrais. São também colocadas duas abordagens para melhoria na performance do classificador, a primeira é forçando o vetor Laplaciano para reconstrução da imagem e a segunda é utilizar um modelo de dispersão, onde a vizinhança dos pixels de teste é representado por uma combinação linear de amostras do algoritmo de treinamento

3. An Image Change Detection Algorithm Based on Markov Random Field Models [9]

Nesse artigo foca o problema de mudança de imagem a partir de modelos de campos aleatórios de Markov. É desenvolvido no trabalho um algoritmo ideal com o critério de MAP(Maximum a Posteriori) desenvolvido no mesmo modelo, depois são apresentados exemplos para ilustração e performance do algoritmo.



## Capítulo 3

# Materiais e Métodos

Nos materiais e métodos utilizados para implementação do algoritmo foi utilizado um fluxo completo com um aplicativo híbrido em comunicação de uma API(Application Programming Interface) externa. A motivação atrás dessa estruturação é a modularidade da API no quesito que , qualquer dispositivo possa fazer uma análise de reconhecimento de imagem por Haar-Cascade contanto que tenha os requisitos necessários para uma requisição Http/1.1.

Abaixo haverá a separação de todo aparato utilizado para implementação prática do estudo, assim como suas especificações. Em seguida, teremos as metodologias utilizadas para implementação assim como o desenho do fluxo de telas do aplicativo e o relacionamento do banco a partir de um esquemático.

## 3.1 Software

As especificações de softwares e tecnologia utilizadas podem ser categorizadas em três subseções nesse estudo: O aplicativo, a API e o banco de dados. Esse estudo foi utilizado apenas o localhost como endpoint das aplicações, as portas utilizadas para o funcionamento do sistema como um todo foram 8000 para aplicação WEB, 8080 para API, e 3306 para o banco de dados. As características são listadas a seguir.

### 3.1.1 Aplicativo

Para a criação do aplicativo foi utilizado IONIC 3 como framework. Essa escolha se deve pela facilidade de Cross-Platform deixando o projeto escalável para diversas plataformas de comunicação, como por exemplo uma expansão futura para iOS. Os requisitos principais utilizados para compilação das plataformas Android e Browser são apresentados a seguir, demais requisitos estão listados no package.json apresentados no GitHub no Apêndice [A] :

Node.js v10.16.0 (*Interpretador de JavaScript assíncrono, orientado a eventos* ), npm v6.9.0 (*Gerenciador de pacotes para JavaScript* ), ionic 5.4.13 –type ionic-angular (*SDK para aplicações híbridas* ), cordova v9.0.0 (*Estrutura de Desenvolvimento responsável pelo acesso de recursos nativos* ) e typescript 2.6.2 (*Superconjunto de JavaScript que adiciona principalmente tipagem estática* )

### 3.1.2 API

Para criação da API foi utilizado o Django devida a comodidade de integração do tratamento de requisições Http1.1 e a utilização do OpenCV. Os requisitos utilizados no estudo são apresentados a seguir :



Python 3.7 (*Linguagem de programação de alto nível* ), OpenCV 4.2 (*Biblioteca Multi-plataforma para desenvolvimento de visão computacional* ), opencv\_createsamples e opencv\_traincascade (*Aplicações do OpenCV que permitem criar amostras de imagens e treinamento do algoritmo Haar Cascade* ), Django 3.0.3 (*Framework de desenvolvimento rápido em Web*), matplotlib 3.1.3 (*Biblioteca para visualização de dados, no caso fotos*), NumPy 1.18.1 (*Pacote para tratamento de dados multidimensionais* ) e Pandas 1.0.3 (*Biblioteca para manipulação de análise de dados*)

### 3.1.3 Banco de Dados

No requisito do banco de dados foi utilizado MariaDB, compilado no sistema operacional de Win64 na versão 10.3.16 e com innodb version = 10.3.16 (*Especifica mecanismo de armazenamento por row locking* )

## 3.2 Hardware

No hardware há a utilização principal de dois componentes por parte do aplicativo por ser Híbrido. Um aparelho mobile Android, que será responsável pela requisição da imagem e interface com o usuário, e um computador ,que será utilizado não só como interface de usuário para requisição e visualização de dados no Servidor, mas também como o própria hospedagem da API, do banco de dados e as imagens salvas. Dito isso temos dois requisitos de Hardware em subseções logo em seguida.

### 3.2.1 Requisitos Computador Utilizado

O modelo do computador utilizado foi Lenovo Ideapad 330 15IKB com processador Intel Core i5-820U CPU 1.60GHz 1.80 GHz, 8 GB de memória RAM, 220 GB PCIe SSD, GPU Intel(R) UHD Graphics 620, Display : 39.62cms (15.6) HD (1366 x 768) e Câmera: 0.3-megapixel

### **3.2.2 Requisitos do Aparelho Mobile Utilizado**

O modelo do mobile foi o Motorola One Vision XT1970-1 (SKU) com Android versão 10 (QSA30.62-24), 4 GB RAM, 128 GB ROM e câmera 48MP (1.6 micrometro quad pixel, 12MP output)+5MP

### 3.3 Metodologia

Na metodologia abordaremos primeiro uma análise teórica sobre a ideia envolvida no artigo do Viola e Jones [16] no quesito dos algoritmos Haar-Cascade , logo em seguida abordaremos a metodologia prática utilizada para implementação funcional da base teórica.

Na metodologia prática podemos separar em três subseções: a forma que foi construída a aplicação híbrida desde o desenho de fluxo de telas e a construção da API e o seu relacionamento com o banco de dados em MariaDB assim como o esquemático de relacionamento.

### 3.3.1 Análise Teórica

Para entender a base teórica deste estudo analisaremos a fundo todo conceito por trás do Algoritmo Haar-Cascade apresentado em 2001 por [16]. Separaremos essa subseção em uma análise sobre os padrões Haar-like features com a Imagem Integral, em sequência com o funcionamento do classificador Adaboost em conjunto com o Cascade.

#### 3.3.1.1 Haar-like features

Os Haar Features são um conjunto de padrões como similaridade matemática com uma sequência Haar proposto pelo matemático Alfréd Haar [6] em 1909. Esse padrões são utilizados como base no artigo do Viola-Jones para o reconhecimento de contornos, linhas e diagonais de uma imagem. A partir desses padrões conseguiremos identificar similaridades na imagem que queremos reconhecer. Para uma imagem colorida, o reconhecimento do padrão deve ser feito após passar a imagem para uma escala de cinza e normalizar os pixels para diminuir influências de luminosidade. Após esse tratamento, o reconhecimento de um padrão Haar pode ser feito na imagem com uma região de interesse identificando as variações entre os pixels com tendência maior ao Branco e os com tendência maior ao Preto. É feito então um produto escalar entre a região de interesse e algum dos padrões Haar como apresentado a seguir com uma figura  $I$  de tamanho  $N \times M$  [20]:

$$\sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} I(i, j) 1_{branco} - \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} I(i, j) 1_{preto} = \delta \quad (3.3.1.1)$$

Se houver uma alta similaridade ( $\delta \rightarrow 1$ ) do resultado com o valor do Haar Feature o padrão é reconhecido pela região de interesse. A seguir alguns os padrões Haar Features utilizados neste estudo.

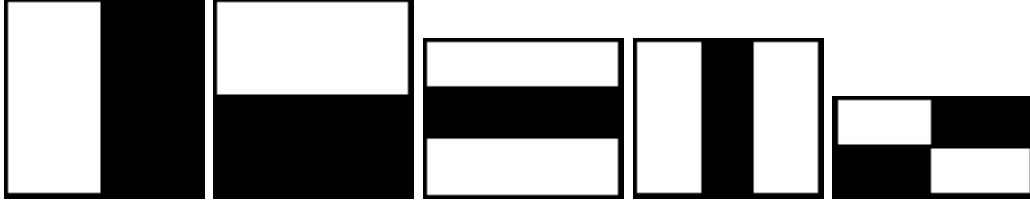


Figura 3.1: Padrões Haar Features

No entanto na equação pura (3.3.1.1) há um problema de número de cálculos para identificar o Haar Feature, pelo fato que é necessário passar por todas combinações de pixels para criação de regiões de interesse onde o cálculo será utilizado. Contornando esse problema de alta complexidade de processamento, introduziremos o conceito de Imagem Integral.

### 3.3.1.2 Imagem Integral

A imagem integral facilita o cálculo de soma de pixels em padrões retangulares. Nela, cada pixel é a soma escalar de toda região normalizada cumulativa anterior. Podemos então criar somas totais de regiões de interesse retangulares fazendo operações simples entre os quatro vértices do retângulo escolhido. Escolhida uma imagem integral  $ii$  relacionada a uma imagem real  $i$  e um pixel de interesse  $x, y$  utilizando o seguinte par de recorrências :

$$s(x, y) = s(x, y - 1) + i(x, y) \quad (3.3.2.2)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \quad (3.3.2.3)$$

Onde o artigo [16] esclarece que  $s(x, y)$  é soma cumulativa da linha com as variáveis de contorno  $s(x, -1) = 0$  e  $ii(-1, y) = 0$ .

Viola e Jones esclarece então a seguinte definição de cada pixel em uma imagem integral abaixo:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \quad (3.3.2.1)$$

Para ilustração demonstrarei abaixo com um exemplo prático, começando com uma imagem normalizada do cinza representada na tabela abaixo com a região de interesse marcada:

Tabela 3.1: Imagem Normalizada

.1	.1	.1	.1	.2
.1	.1	.1	.1	.1
.1	.1	.2	.1	.4
.1	.1	.1	.2	.1
.1	.1	.1	.1	.2

Utilizando a equação (3.3.2.1) na imagem normalizada temos que a seguinte imagem integral com os pontos de interesse:

Tabela 3.2: Imagem Integral

.1	.2	.3	.4	.6
.2	.4	.6	.8	1.1
.3	.6	1.0	1.3	2.0
.4	.8	1.3	1.8	2.6
.5	1.0	1.6	2.2	3.2

Somando as região de interesse em amarelo da imagem original temos que  $.1 + .1 + .1 + .1 + .1 + .1 + .1 + .2 + .2 = 1.1$ , agora utilizando a imagem integral obtemos o mesmo resultado com apenas os pontos de interesse em azul  $1.8 - .4 - .4 + .1 = 1.1$ . É fácil notar, que houve uma alteração substancial na quantidade de operações realizadas na imagem original(nove operações) contra a imagem integral(quatro operações).

Com os Haar Features selecionados e com a imagem normalizada e tratada como imagem integral, é necessário algum algoritmo que otimize e treine o classificador. No caso do artigo base [16] é utilizado uma variação do Algoritmo AdaBoost, que em conjunto numa cascata de classificadores para otimização de processamento a partir do nível de probabilidade do objeto de busca estar dentro de uma região de interesse foi denominada

como Haar-Cascade.

A combinação desse método AdaBoost em Cascata será melhor aprofundado na próxima sessão e concluirá a parte teórica utilizado no nesse trabalho.

### **3.3.1.3 AdaBoost em Cascade**

O algoritmo AdaBoost foi proposto por Freund e Schapire em [5] em que consiste em uma combinação de classificadores fracos que são criados a partir de ajustes de pesos entre positivos e negativos. No artigo de Viola e Jones [16] é citado também que há um interesse em reduzir o número de características na imagem para uma melhora na taxa de erro. Este fenômeno ocorre, pois com poucas iterações do classificador AdaBoost, características mais gerais da imagem considerada positiva são encontradas, após um número maior de iterações características específicas das imagens positivas atingem o classificador de acordo com a Haar-Feature.

Dito isso [16] representa uma adaptação do AdaBoost em pseudocódigo apresentado melhor a seguir:

1. É necessário um conjunto  $(x_n, y_n)$  de uma quantidade  $l$  imagens positivas ( $y_i = 1$ ) e uma quantidade  $m$  de imagens negativas ( $y_i = 0$ ).
2. Com essas imagens, é possível inicializar os pesos do algoritmo  $w_{1,i} = \frac{1}{2m}$  onde  $y_i = 0$  e  $w_{1,i} = \frac{1}{2l}$  onde  $y_i = 1$ .
3. Iniciamos um número de iterações  $t = 1, \dots, T$

- Normalize os pesos da seguinte forma a seguir, onde  $w_t$  é a distribuição de probabilidade e  $j$  é a representação de cada Haar Feature :

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

- Treine o classificador fraco  $h_j$  restrito para um único Haar Feature, obtendo o erro  $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$
- Escolha o melhor classificador fraco  $h_t$  verificando o menor  $\epsilon_t$
- Atualize os pesos dos  $w_t$  de acordo com o acerto do classificador da seguinte forma:

- Caso classificado corretamente  $w_{t+1,i} = w_{t,i} \frac{\epsilon_t}{1-\epsilon_t}$
- Caso contrário  $w_{t+1,i} = w_{t,i}$

4. Após as  $T$  iterações o classificador forte  $h(x)$  será a dado por uma combinação dos classificadores fracos  $h_t(x)$  da seguinte forma:  $h(x) =$

$$\begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{caso contrário} \end{cases}$$

$$\text{Onde } \alpha_t = \log \frac{1-\epsilon_t}{\epsilon_t}$$

Com o classificador vindo do AdaBoost pronto é necessário separar de modo otimizado a imagem considerada positiva dentro de uma imagem mais geral. Viola e Jones [16] propõe uma modificação na forma que o processamento de imagem é dado em um



classificador em cascata.

A ideia proposta consiste em separar a imagem original em sub-imagens descartando-as caso não há possibilidade nenhuma de ser um positivo de acordo com o classificador AdaBoost. Caso contrário, a sub-imagem é separada em novas sub-imagens colocadas como sendo o input do classificador e novamente verificando se não há possibilidade nenhuma de ser um positivo. Um evento em cascata de vários classificadores é exemplificado melhor na imagem a seguir:

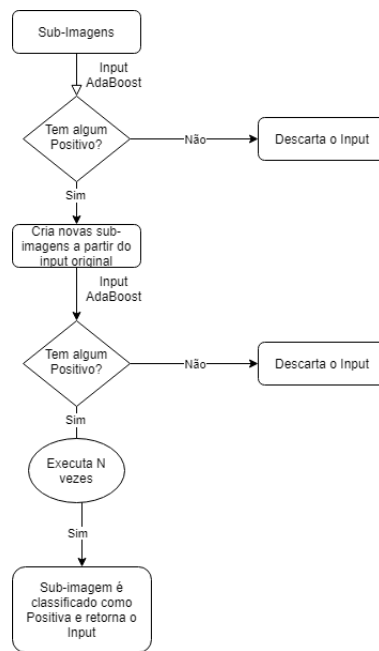


Figura 3.2: Diagrama do Algoritmo em Cascata

O interessante dessa combinação dos algoritmos é a melhora no processamento da imagem como um todo, pois grandes blocos de pixels de sub-imagens são descartadas caso seja certamente uma imagem negativa, tendo um enfoque apenas nas de possibilidade de sub-imagens positivas.

Com essa introdução teórica foi apresentado o algoritmo Haar-Cascade que será utilizado de modo prático nesse estudo na próxima sessão.

### 3.3.2 Implementação

Na metodologia prática será demonstrado como foi feito o planejamento e executado os elementos da teoria do algoritmo Haar-Cascade ,assim como os desenhos e resultados.

Primeiro abordaremos a construção da API que será responsável pela comunicação do banco de dados e seus relacionamentos quanto o processamento e premissas do algoritmo Haar-Cascade.

Após isso daremos seguimento à aplicação Híbrida que será o ponto de input e output do usuário, mas sem o processamento da aplicação teórica do algoritmo, também será colocado as integrações entre o Aplicativo e a API via Http/1.1.

#### 3.3.2.1 Construção API

##### 3.3.2.1.1 ERD (Entity Relationship Diagram)

Começamos pela modelação do banco de dados em que o diagrama de entidade e relacionamento é apresentado a seguir:

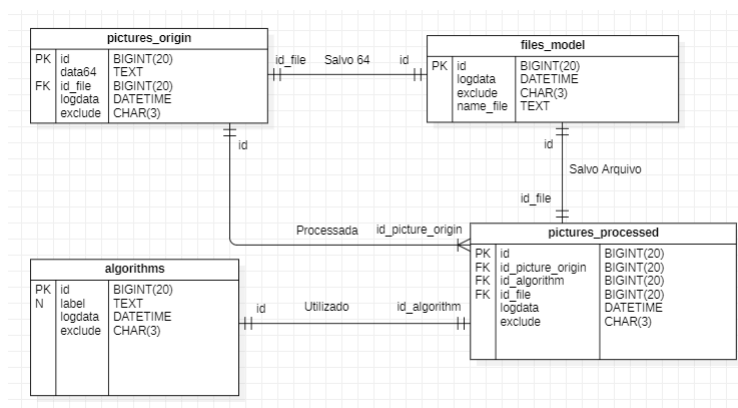


Figura 3.3: Relacionamento Banco de dados

No diagrama podemos perceber a entidade **pictures\_origin**, responsável por salvar no banco as características da imagem inicial capturada pré-processamento da API. Essa

entidade se relaciona diretamente com a entidade **files\_model** responsável por salvar e localizar uma identificação com o arquivo físico assim como algumas características do arquivo.

Para expansão de estudos futuros foi acrescida a entidade **algorithms** responsável por salvar os tipos de algoritmos utilizados para comparação em um estudo futuro.

E finalmente , se relacionando com todas as entidades anteriores, temos a entidade **picture\_processed** que relaciona a imagem de origem, com o algoritmos utilizado (No caso Haar-Cascade) e o arquivo físico criado no banco de servidores como resposta do processamento executado pela API.

### 3.3.2.1.2 API Django integrado OpenCV

O foco da utilização da API foi a criação de requisições que não só ajudam na criação do classificador, mas também processam a imagem.

Como apresentado em [Análise Teórica] para criação do classificador forte pelo método Haar Cascade necessitamos de um conjunto de imagens consideradas positivas e um outro conjunto de negativas. Utilizaremos as bibliotecas abertas do OpenCV para tratamento das amostras de imagens e treino do algoritmo Haar Cascade.

Para o estudo apresentado, faremos um exemplo de reconhecimento de carta de número três de um baralho comum apresentado em 3.4.



Figura 3.4: Carta utilizada para treinamento

Para criação das amostras pré-treinamento do algoritmo foi criada requisições que classificam imagens positivas, negativas e tratamento das cores para o cinza a partir da requisição apresentada a seguir:

---

```
1 POST      http://localhost:8080/createcascade HTTP/1.1
2 content-type: application/json
3
4 {
5   "path": "./traincascade/%caminho%",
6   "type": "%tipo%"
7 }
```

---

Onde *caminho* é pasta do servidor onde estão salvas as imagens que queira executar a operação de amostragem ou filtro de cor. O segundo parâmetro *tipo* pode ser '*positive*' caso o conjunto de imagens a ser tratada no servidor são positivas e deve-se colocar uma região de interesse, '*negative*' caso o conjunto de imagens a ser tratada no servidor são negativas e '*gray*' caso queira executar um filtro de cor para o cinza no conjunto de imagens localizada pelo *caminho*.

Após a criação de amostras positiva e negativas podemos executar o treinamento do Haar Cascade através da aplicação traincascade, em que o output será o classificador da nossa carta de número três. Para o estudo, foi utilizado 1800 amostras de imagens positivas que continham a carta desejada e 900 amostras negativas aleatórias.

Após a criação do classificador Haar Cascade em XML e atualização do código para utilização do próprio podemos retornar a identificação da imagem processada pela requisição demonstrada abaixo:

---

```
1 POST      http://localhost:8080/returnimage HTTP/1.1
2 content-type: application/json
3
4 {
5   "uuid": "%uuid%"
6 }
```

---

Onde *uuid* é chave primária da entidade `origin_picture` apresentada em 3.2. Essa requisição retorna a imagem processada da imagem como demonstrado a seguir:

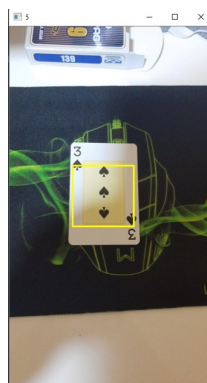


Figura 3.5: Carta de retorno pós processamento

### 3.3.2.2 Construção Aplicação Híbrida

Para construção da aplicação foi desenhado primeiro o fluxo de telas que trataremos o objetivo de cada uma, começaremos pela tela **Connection** à seguir:



Figura 3.6: Tela Connection

O objetivo da tela **Connection** 3.6 é entender quaisquer problemas de requisição da aplicação com a API através de uma Entidade simples com uma chave primária e um campo de texto. Com isso, temos uma aplicação simples de utilização de POST, GET, DELETE e PUT de requisições Http/1.1.

Depois de verificada nessa tela a possibilidade da utilização das requisições Http no aplicativo com a API podemos desenhar a aplicação como um todo. A próxima imagem demonstra a tela principal de listagem das imagens adquiridas.



Figura 3.7: Tela App

A tela **App** 3.7 é tela principal de listagem, edição e criação dos dados da entidade `pictures_origin` apresentado em 3.3. Nela podemos além de capturar a imagem da câmera, selecionar alguma imagem em específico através da chave primária pelo botão *Get Pic* que abre um modal apresentado na tela a seguir.

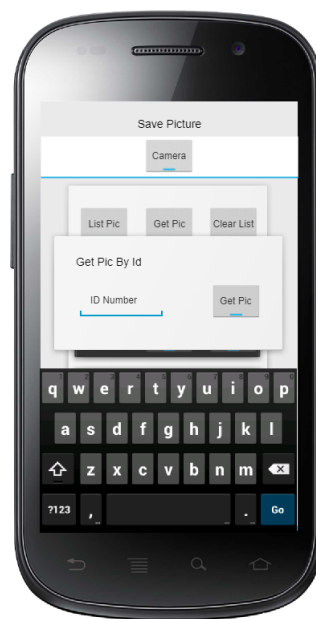


Figura 3.8: Tela Get Picture

A tela **Get Picture** 3.8 realiza um Get por chave primária da entidade `pictures_origin` apresentado em 3.3. Após o Get ser realizado há um redirecionamento para a página de Opções mostrado a seguir.





Figura 3.9: Tela Options

A tela **Options** 3.9 realiza o pré-relacionamento entre os tipos de algoritmos gravados no banco de dados. No caso deste estudo estamos utilizando apenas o Haar Cascade, mas a entidade foi criada para expansão do estudo em outros algoritmos no futuro.

O segundo select *Compare Image* relaciona o retorno de todos os dados da entidade `pictures_processed` com relacionamento com a imagem de origem capturada na entidade `pictures_origin`.

O primeiro botão *process* realiza o processamento na API e salva uma nova linha na entidade `pictures_processed`, enquanto o segundo botão *compare* realiza um get na entidade `pictures_processed` e captura os dados da imagem processada. Ambos os casos o fluxo continua na tela seguinte.

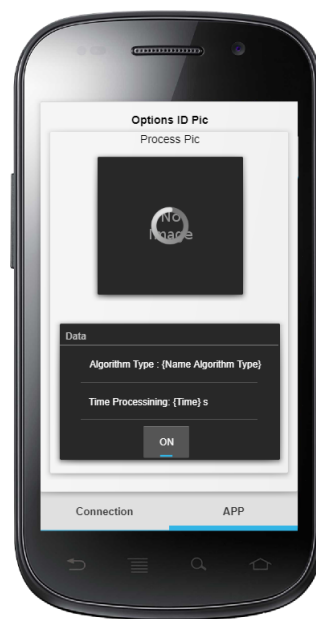


Figura 3.10: Tela Compare

A tela **Compare** 3.10 foi desenhada para visualização de dados da entidade `pictures_processed`, visualizando a imagem processada, o algoritmo utilizado, e o tempo de processamento feito pela requisição Http.

A ideia por trás desse fluxo completo é verificar primeiro se há comunicação com a API em todos os tipos de requisição Http relevantes que é o caso da tela connection para então executar de modo simples as operações de comunicação com a API.

O resultado final das telas do Aplicativo são mostrados a seguir:

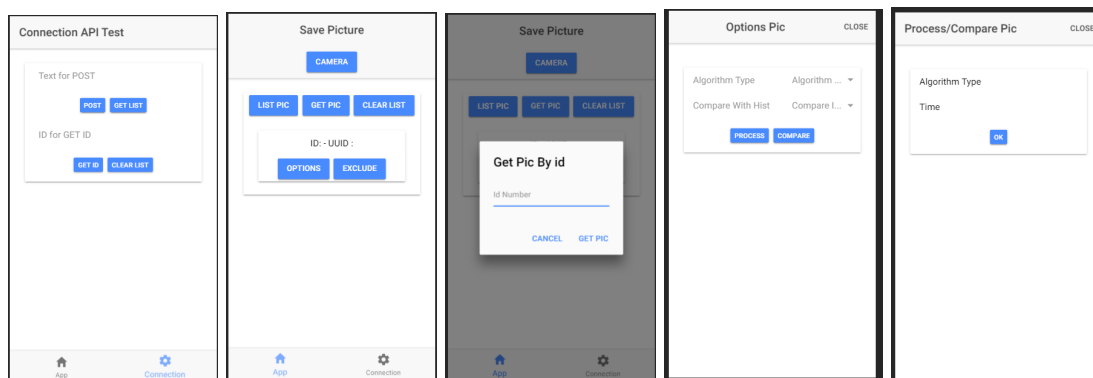


Figura 3.11: Fluxo de Telas

Todos os códigos utilizados e criados estão disponíveis no GitHub no apêndice A deste trabalho.



## Capítulo 4

# Resultados

Como apresentado na implementação na imagem 3.5, a API para o processamento reconheceu o padrão estabelecido como número 3, no entanto, para verificar a eficácia do método de modo qualitativo utilizaremos o método de Matriz de Confusão apresentado por Stephen V. Stehman em [14] e neste estudo com o algoritmos Haar Cascade os dados estão listados no apêndice B.

Para criação da matriz foram utilizadas 139 imagens positivas e 268 imagens negativas em relação a carta de número três com naipe de espadas. Essas imagens estão abertas ao público no apêndice C para verificação e testes dos dados.

Foi utilizado também na criação da matriz a variação dos parâmetros no Cascade de *scaleFactor* (que especifica o quanto a imagem original é reduzida por uma escala) e *minNeighbors* (representado por *minN* e que especifica a quantidade mínima de Haar-Features agrupados para ser considerado um candidato na identificação). Esses dois parâmetros permitem um ajuste de sensibilidade o classificador como demonstrado do exemplo 4.1



Figura 4.1: Sensibilidade de Escala

Na primeira imagem temos *scaleFactor* de 1.05 e *minNeighbors* de 20, enquanto na imagem subsequente *scaleFactor* de 1.08 e *minNeighbors* de 40 diminuindo a sensibilidade do algoritmo mas deixando mais robusto.

Para verificar a utilidade do classificador, utilizaremos então a matriz de confusão para verificar a acurácia (*ACC* 4.2) e precisão (*PPV* 4.3) em relação aos dois parâmetros já citados. Os dados estatísticos completos estão no apêndice B.

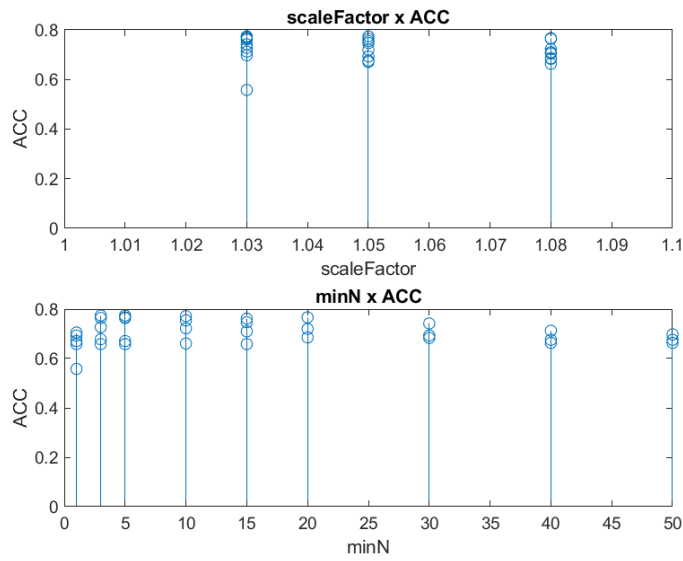


Figura 4.2: Acurácia

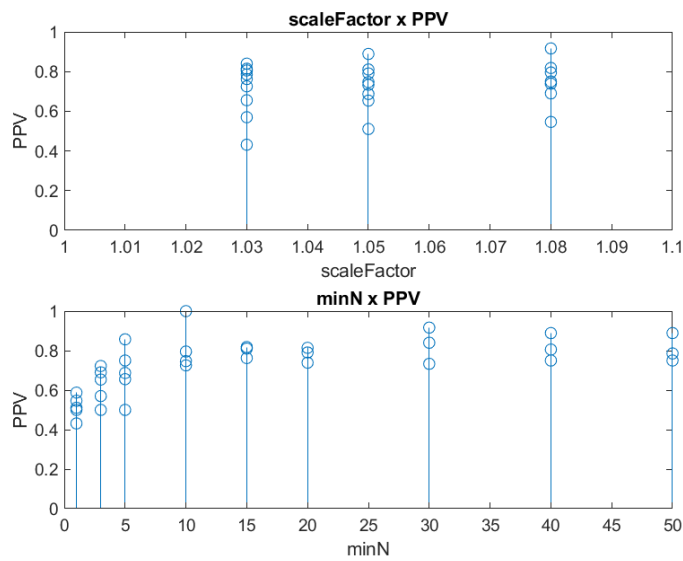


Figura 4.3: Precisão

Podemos perceber pelos dados apresentados que a acurácia tem uma menor sensibilidade tanto ao scaleFactor quanto ao minNeighbors ficando sempre com uma taxa de

acerto de 75 %. No entanto, a precisão é afetada por estes parâmetros.

Como podemos perceber pelo gráfico 4.3 a precisão do classificador tende a aumentar, isso ocorre porque a sensibilidade do Cascade diminui com os aumentos dos parâmetros de `scaleFactor` e `minNeighbors`, evoluindo a robustez do classificador.

Elucidado os pontos da eficácia do classificador, podemos aprofundar nas vantagens e limitações.

## 4.1 Vantagens e Limitações

Começando pelas vantagens, o tempo de processamento da requisição na API de retorno de imagem quando o classificador Haar Cascade já está pronto é por volta de 0.5 segundos. Esse tempo pode ser melhorado de acordo com um upgrade nos componentes de hardware utilizados neste trabalho.

Outra vantagem, é como a API detém todo o processamento de imagem e controle do Fluxo de Dados e a interface de comunicação são apenas as requisições do protocolo Http/1.1, qualquer aplicação com suporte deste protocolo pode acessar e utilizar esses dados.

A maior limitação é o tempo de treinamento do Haar Cascade, que com os parâmetros de positivos e negativos no estudo chegava um tempo de 40 minutos para criação do XML.

Outro ponto negativo é atualização manual do código para cada Haar Cascade criado pelas aplicação *traincascade* do OpenCV, necessitando cuidado para não haver a quebra da API no momento da atualização do classificador.

Outra oportunidade de melhoria identificada é a sensibilidade de cor. Por exemplo, o naipe de copas e naipe de espadas atingem uma semelhança notável que o Haar Cascade não consegue diferenciar quando há um pré processamento da imagem para a cor cinza. O resultado é demonstrado abaixo em 4.4:





Figura 4.4: Sensibilidade de Cor



## Capítulo 5

# Conclusão

Após o estudo realizado retiradas as limitações também citadas, o algoritmo Haar Cascade demonstra velocidade no processamento expansível para um reconhecimento em tempo real. Já a implementação modular da API permite uma escalabilidade em número de projetos ligados à ela.

Portanto podemos expandir o projeto para comunidade para criação de classificadores personalizados, e tornando mais acessível o conhecimento de reconhecimento de padrões.

### 5.1 Sugestões para trabalhos futuros

Para trabalhos futuros ou evolução do projeto é possível:

- Vincular outros algoritmos de reconhecimento de padrões na API para comparação de robustez e velocidade
- Vincular a API em um embarcado para uma aplicação modular física de reconhecimento de padrões
- Executar testes de robustez na API e verificar o quanto o protocolo de comunicação Http/1.1 afeta no desempenho

- Executar reconhecimento de padrões em tempo real via API
- Aplicar classificadores de cores no reconhecimento de padrões como utilizado em [7]
- Automatizar atualização de classificador na API via POST
- Ampliar a resposta de retorno do classificador para uma visualização tridimensional na aplicação Client em realidade virtual

# Referências Bibliográficas

- [1] Jean-Francois Cayula and Peter Cornillon. Edge detection algorithm for sst images. *Journal of atmospheric and oceanic technology*, 9(1):67–80, 1992.
- [2] Yi Chen, Nasser M Nasrabadi, and Trac D Tran. Hyperspectral image classification using dictionary-based sparse representation. *IEEE transactions on geoscience and remote sensing*, 49(10):3973–3985, 2011.
- [3] Márcio Portes de Albuquerque and Marcelo Portes de Albuquerque. Processamento de imagens: métodos e análises. *Rio de Janeiro, Brasil*, 12, 2000.
- [4] Luciano Floridi. Hyperhistory and the philosophy of information policies. In *The onlife manifesto*, pages 51–63. Springer, Cham, 2015.
- [5] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.
- [6] Alfred Haar. *Zur theorie der orthogonalen funktionensysteme*. Georg-August-Universitat, Gottingen., 1909.
- [7] Rein-Lien Hsu, Mohamed Abdel-Mottaleb, and Anil K Jain. Face detection in color images. *IEEE transactions on pattern analysis and machine intelligence*, 24(5):696–706, 2002.

- [8] Michael Jones and Paul Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3(14):2, 2003.
- [9] Teerasit Kasetkasem and Pramod K Varshney. An image change detection algorithm based on markov random field models. *IEEE Transactions on Geoscience and Remote Sensing*, 40(8):1815–1823, 2002.
- [10] Panos G Michalopoulos, Richard A Fundakowski, Meletios Geokezas, and Robert C Fitch. Vehicle detection through image processing for traffic surveillance and control, July 11 1989. US Patent 4,847,772.
- [11] Richard J Radke, Srinivas Andra, Omar Al-Kofahi, and Badrinath Roysam. Image change detection algorithms: a systematic survey. *IEEE transactions on image processing*, 14(3):294–307, 2005.
- [12] Denise Schmandt-Besserat. The evolution of writing. *Austin, Texas: University of*, 2014.
- [13] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.
- [14] Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.
- [15] Stanley R Sternberg. Biomedical image processing. *Computer*, pages 22–34, 1983.
- [16] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. IEEE, 2001.
- [17] Paul Viola, Michael Jones, et al. Robust real-time object detection. *International journal of computer vision*, 4(34-47):4, 2001.

- [18] Fangju Wang. Fuzzy supervised classification of remote sensing images. *IEEE Transactions on geoscience and remote sensing*, 28(2):194–201, 1990.
- [19] WJ Wang and PD McFadden. Early detection of gear failure by vibration analysis–ii. interpretation of the time-frequency distribution using image processing techniques. *Mechanical systems and signal processing*, 7(3):205–215, 1993.
- [20] Yi-Qing Wang. An analysis of the viola-jones face detection algorithm. *Image Processing On Line*, 4:128–148, 2014.
- [21] Yue Zhou, Li Shen, and Jie Yang. Feature analysis method of tongue image by chinese medical diagnosis based on image processing. *Infrared and Laser Engineering*, 31(6):490–494, 2002.





## Apêndice A

# GitHub

Todos os código realizados estão colocadas no GitHub, disponíveis para o público baixar, gerar qualquer alteração, estudo e evolução: [GitHub Carlos Araki TG2](#)



## Apêndice B

# Resultados de classificação

Primeira Matriz é a matriz de confusão para análise de eficácia do algoritmo a partir dos parâmetros de `scaleFactor` e `minNeighbors`. Seguida pela matriz de dados estatísticos utilizados no estudo.

scaleFactor	minN	tPos	fPos	fNeg	tNeg
1,03	1	129	170	10	98
1,03	3	114	86	25	182
1,03	5	99	52	40	216
1,03	10	74	28	65	240
1,03	15	61	19	78	249
1,03	20	57	13	82	255
1,03	30	42	8	97	260
1,03	40	29	7	110	261
1,03	50	22	6	117	262
1,05	1	119	114	20	154
1,05	3	100	53	39	215
1,05	5	79	36	60	232
1,05	10	59	20	80	248
1,05	15	47	11	92	257
1,05	20	34	9	105	259
1,05	30	22	8	117	260
1,05	40	8	1	131	267
1,05	50	8	1	131	267
1,08	1	112	93	27	175
1,08	3	78	35	61	233
1,08	5	66	22	73	246
1,08	10	35	9	104	259
1,08	15	27	6	112	262
1,08	20	17	6	122	262
1,08	30	11	1	128	267
1,08	40	3	1	136	267
1,08	50	3	1	136	267
2,00	1	47	33	92	235
2,00	3	13	5	126	263
2,00	5	6	1	133	267
2,00	10	1	0	138	268
2,00	15	0	0	139	268
5,00	1	11	11	128	257
5,00	3	3	3	136	265
5,00	5	1	1	138	267

scaleFactor	minN	Prevalence	ACC	PPV	FDR	FOR	NPV	TPR	FNR	FPR	TNR	MCC
1,03	1	0,342	0,558	0,431	0,569	0,093	0,907	0,928	0,072	0,634	0,366	0,315
1,03	3	0,342	0,727	0,570	0,430	0,121	0,879	0,820	0,180	0,321	0,679	0,474
1,03	5	0,342	0,774	0,656	0,344	0,156	0,844	0,712	0,288	0,194	0,806	0,509
1,03	10	0,342	0,771	0,725	0,275	0,213	0,787	0,532	0,468	0,104	0,896	0,468
1,03	15	0,342	0,762	0,763	0,238	0,239	0,761	0,439	0,561	0,071	0,929	0,439
1,03	20	0,342	0,767	0,814	0,186	0,243	0,757	0,410	0,590	0,049	0,951	0,454
1,03	30	0,342	0,742	0,840	0,160	0,272	0,728	0,302	0,698	0,030	0,970	0,393
1,03	40	0,342	0,713	0,806	0,194	0,296	0,704	0,209	0,791	0,026	0,974	0,305
1,03	50	0,342	0,698	0,786	0,214	0,309	0,691	0,158	0,842	0,022	0,978	0,255
1,05	1	0,342	0,671	0,511	0,489	0,115	0,885	0,856	0,144	0,425	0,575	0,413
1,05	3	0,342	0,774	0,654	0,346	0,154	0,846	0,719	0,281	0,198	0,802	0,511
1,05	5	0,342	0,764	0,687	0,313	0,205	0,795	0,568	0,432	0,134	0,866	0,457
1,05	10	0,342	0,754	0,747	0,253	0,244	0,756	0,424	0,576	0,075	0,925	0,419
1,05	15	0,342	0,747	0,810	0,190	0,264	0,736	0,338	0,662	0,041	0,959	0,403
1,05	20	0,342	0,720	0,791	0,209	0,288	0,712	0,245	0,755	0,034	0,966	0,326
1,05	30	0,342	0,693	0,733	0,267	0,310	0,690	0,158	0,842	0,030	0,970	0,233
1,05	40	0,342	0,676	0,889	0,111	0,329	0,671	0,058	0,942	0,004	0,996	0,174
1,05	50	0,342	0,676	0,889	0,111	0,329	0,671	0,058	0,942	0,004	0,996	0,174
1,08	1	0,342	0,705	0,546	0,454	0,134	0,866	0,806	0,194	0,347	0,653	0,435
1,08	3	0,342	0,764	0,690	0,310	0,207	0,793	0,561	0,439	0,131	0,869	0,456
1,08	5	0,342	0,767	0,750	0,250	0,229	0,771	0,475	0,525	0,082	0,918	0,452
1,08	10	0,342	0,722	0,795	0,205	0,287	0,713	0,252	0,748	0,034	0,966	0,333
1,08	15	0,342	0,710	0,818	0,182	0,299	0,701	0,194	0,806	0,022	0,978	0,299
1,08	20	0,342	0,686	0,739	0,261	0,318	0,682	0,122	0,878	0,022	0,978	0,205
1,08	30	0,342	0,683	0,917	0,083	0,324	0,676	0,079	0,921	0,004	0,996	0,211
1,08	40	0,342	0,663	0,750	0,250	0,337	0,663	0,022	0,978	0,004	0,996	0,086
1,08	50	0,342	0,663	0,750	0,250	0,337	0,663	0,022	0,978	0,004	0,996	0,086
2,00	1	0,342	0,693	0,588	0,413	0,281	0,719	0,338	0,662	0,123	0,877	0,257
2,00	3	0,342	0,678	0,722	0,278	0,324	0,676	0,094	0,906	0,019	0,981	0,173
2,00	5	0,342	0,671	0,857	0,143	0,333	0,668	0,043	0,957	0,004	0,996	0,144
2,00	10	0,342	0,661	1,000	0,000	0,340	0,660	0,007	0,993	0,000	1,000	0,069
2,00	15	0,342	0,658	NaN	NaN	0,342	0,658	0,000	1,000	0,000	1,000	NaN
5,00	1	0,342	0,658	0,500	0,500	0,332	0,668	0,079	0,921	0,041	0,959	0,080
5,00	3	0,342	0,658	0,500	0,500	0,339	0,661	0,022	0,978	0,011	0,989	0,041
5,00	5	0,342	0,658	0,500	0,500	0,341	0,659	0,007	0,993	0,004	0,996	0,023



## Apêndice C

# Imagens Utilizadas

Todas imagens utilizadas para análise dos dados se encontram no link : Anexo de Imagens Carlos Araki TG2