UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

ESCOLA DE INFORMÁTICA APLICADA

PresentEye: Sistema de Controle de Presença por Reconhecimento Facial

Guilherme Caeiro de Mattos

Orientadora

Geiza Maria Hamazaki da Silva

RIO DE JANEIRO, RJ – BRASIL

DEZEMBRO DE 2017

Catalogação informatizada pelo autor

M435

Mattos, Guilherme Caeiro de
PresentEye: Sistema de Controle de Presença por Reconhecimento Facial / Guilherme Caeiro de Mattos. -- Rio de Janeiro, 2017.
71 f

Orientadora: Geiza Maria Hamazaki da Silva.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2017.

1. Registro de Presença. 2. Reconhecimento Facial. 3. Desenvolvimento de Sistemas. 4. Escalabilidade de Software. I. Silva, Geiza Maria Hamazaki da, orient. II. Título.

PresentEye: Sistema de Controle de Presença por Reconhecimento Facial

Guilherme Caeiro de Mattos

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada por:

_____

Dra. Geiza Maria Hamazaki da Silva (UNIRIO)

_____

Dra. Morganna Carmem Diniz (UNIRIO)

Rio de Janeiro, RJ – Brasil.

Dezembro de 2017

AGRADECIMENTOS

Primeiramente, agradeço à UNIRIO e ao Governo Federal, que proporcionaram o ambiente acadêmico no qual cursei, gratuitamente, o nível superior, e a oportunidade de estudar no exterior, com financiamento da CAPES, como bolsista do programa Ciência Sem Fronteiras.

Em seguida, agradeço àqueles que me acompanharam nessa jornada, como os docentes e discentes da universidade. Dentre eles, especialmente, a orientadora deste trabalho, professora Geiza Maria Hamazaki da Silva, que teve paciência para me aturar e orientar em várias disciplinas e projetos no decorrer do curso, os alunos William Brum, Karina Martinez, César Luís Barbosa, Lucas Alves, Antônio França da Guia e Bernardo Gouvêa, com quem sempre conversa e colaborava em trabalhos de disciplinas, e minha família, que sempre apoiou minhas escolhas durante esse tempo.

Por fim, mas não menos importante, agradeço também ao Shibaura Institute of Technology, que me aceitou no âmbito do Ciência Sem Fronteiras, aos membros dos grupos Shibaura International Students' Association (SISA) e International Communication Project (ICP), como Shouko Kusayama, Rei Koyama, Seiya Inadera, Masako Ohara, Taihei Koyama, Hanae Tanizawa e Yuutarou Kureyama, e às muitas outras pessoas que pude conhecer no Japão, dentre professores e amigos, como Harukazu Igarashi, Naoko Ooba, Arika Takahashi, Xie Yuhua, Yuu Oono e Shouko Inoue.

RESUMO

Existem diversos métodos para verificação de presença de alunos em sala de aula, como a chamada oral da lista de presença, a assinatura da lista de presença, o registro por cartões magnéticos, a verificação biométrica, etc. Apesar de todos esses mecanismos terem suas vantagens e desvantagens, costumam compartilhar uma característica que pode fazer com que o registro do comparecimento seja feito de maneira errônea ou nem mesmo realizado: eles dependem da ação voluntária do professor ou do aluno para serem executados.

Nesse contexto, este projeto buscou desenvolver um sistema de registro de presença (com baixa necessidade de intervenção humana) através do uso de reconhecimento facial e analisar o desempenho do emprego desse tipo de biometria na tarefa de evitar erros no controle de frequência. Tal reconhecimento foi efetuado sobre imagens do interior de uma sala de aula, capturadas nos horários das aulas, através de câmeras instaladas no recinto.

Palavras-chave: Registro de Presença, Reconhecimento Facial, Desenvolvimento de Sistemas

ABSTRACT

There are many methods to check student attendance to classes, like calling students' names, signing an attendance list, using magnetic cards, checking biometry, etc. Despite having their advantages and disadvantages, those methods generally share a characteristic that might make the attendance registration be executed wrongly or even not executed at all: they rely on student's or teacher's voluntary actions.

Given that context, this project intended to develop a system to register class attendance (with a low need for human intervention) through the use of facial recognition and analyze the use of that type of biometry to avoid errors in the attendance registration. That recognition was executed on images from videos recorded during classes, using cameras installed in a classroom.

Keywords: Class Attendance, Facial Recognition, System Development

Índice

Índice de Tabelas

Índice de Figuras

1. Introdução

1.1 Motivação

Historicamente, em salas de aula de todo o mundo, os professores têm a obrigação de registrar a presença de alunos em suas aulas. Para atender a essa necessidade, diversos métodos foram desenvolvidos, sendo o mais comum a chamada oral da lista de presença. Desde seu surgimento, essa prática possuiu poucas alternativas, como por exemplo a "assinatura da lista de presença", onde os alunos assinam uma lista em um determinado momento da aula. Nas últimas décadas, no entanto, formas mais eficientes surgiram, como, por exemplo, o uso de cartões (dispositivos) magnéticos com etiquetas RFID (***R**adio-**F**requency **ID**entification*, ou Identificação por Radiofrequência), identificadores biométricos (scanners de impressões digitais, de retina, de face, etc), dentre outros.

Apesar de eficazes, essas práticas possuem uma característica que pode levar a falhas: elas dependem da ação voluntária do professor e (ou) do aluno para serem executadas. Por exemplo, durante uma chamada, um professor pode vir a "pular" um nome, ou, até mesmo, um aluno distraído pode não perceber que foi chamado. No caso de uma lista de presença, algumas pessoas podem acabar não tendo contato com ela, caso circule de maneira irregular. No uso de cartões magnético ou de identificação por biometria, um estudante pode esquecer de registrar sua presença junto ao dispositivo responsável por essa tarefa.

Diante dessa situação, percebeu-se que a redução da intervenção humana nesse processo pode diminuir a ocorrência de erros. Assim, a detecção da presença do aluno sem que ele ou professor executem alguma ação voluntária (sem que precisassem agir conscientemente) talvez possa ser uma forma mais eficiente de registrar presença.

Dentre as alternativas atualmente existentes, a com menor necessidade de intervenção costuma a ser a identificação por reconhecimento facial. Nesse contexto, este trabalho apresenta uma solução de registro de presença por reconhecimento facial autônoma (utilizando imagens providas através de câmeras instaladas nas salas de aula) e verifica o desempenho dessa aplicação em uma situação real.

1.2 Objetivos

Este trabalho busca desenvolver uma aplicação capaz de realizar o controle de presença de maneira automática, através do reconhecimento facial executado sobre imagens providas por câmeras em sala de aula, e analisar seu desempenho e aplicabilidade na solução do problema. A fim de alcançar esses objetivos, o sistema, chamado de PresentEye, foi subdividido em duas partes:

I. PresentEye Tunnel: aplicação desktop responsável por concentrar a conexão de câmeras, salvar os vídeos capturados, efetuar o reconhecimento facial e prover certo grau de escalabilidade; e

II. PresentEye Server: aplicação web responsável por gerir os dados do sistema, como alunos, turmas e etc, permitindo a execução de operações CRUD (***C**reate*, ***R**ead*, ***U**pdate* e ***D**elete*) sobre essas entidades e a geração de um relatório de presença.

Quanto à análise, ela consiste em:

I. Avaliar o desempenho da aplicação quanto às suas funções, como transmissão de dados e execução de reconhecimento facial; e

II. Verificar sua performance com dados reais de sala de aula.

Com base nesses resultados, será discutida a aplicabilidade desse sistema no dia-a-dia das salas de aula e que vantagens e desvantagens terá em relação a outros métodos existentes.

1.3 Organização do Texto

Este texto foi dividido em 5 capítulos, os quais são descritos a seguir:

- Capítulo 1: Introdução - O presente capítulo, onde é exibida a motivação para este trabalho, seus objetivos e a estrutura do texto;
- Capítulo 2: Trabalhos Relacionados - Apresenta outros sistemas destinados ao registro de presença por reconhecimento facial e faz uma breve introdução à técnica de reconhecimento facial (redes neurais convolucionais) utilizada pelas bibliotecas empregadas no software desenvolvido neste trabalho;

- Capítulo 3: Desenvolvimento do PresentEye - Apresenta a especificação do PresentEye e descreve sua divisão e seu funcionamento em certo grau de detalhamento;
- Capítulo 4: Testes - Descreve os testes efetuados para verificar o desempenho da aplicação e apresenta seus resultados. É dividido em duas seções destinadas a, respectivamente, verificar a performance da técnica e o comportamento da aplicação desenvolvida ao receber dados reais; e
- Capítulo 5: Conclusões - Apresenta os resultados alcançados e trabalhos futuros.

2. Trabalhos Relacionados

Este capítulo busca prover uma breve introdução sobre o funcionamento do reconhecimento facial e apresentar ferramentas que o utilizam para o problema do registro de presença.

2.1 Sistemas Eletrônicos Para Registro de Presença

Ao redor do mundo, o reconhecimento facial é utilizado de diferentes maneiras para solucionar o problema do registro de presença. Nas seções a seguir, são citados alguns sistemas e suas respectivas abordagens. É importante citar que, por se tratarem de soluções proprietárias, não é possível entrar em detalhes sobre as técnicas de identificação facial utilizadas e outros pormenores.

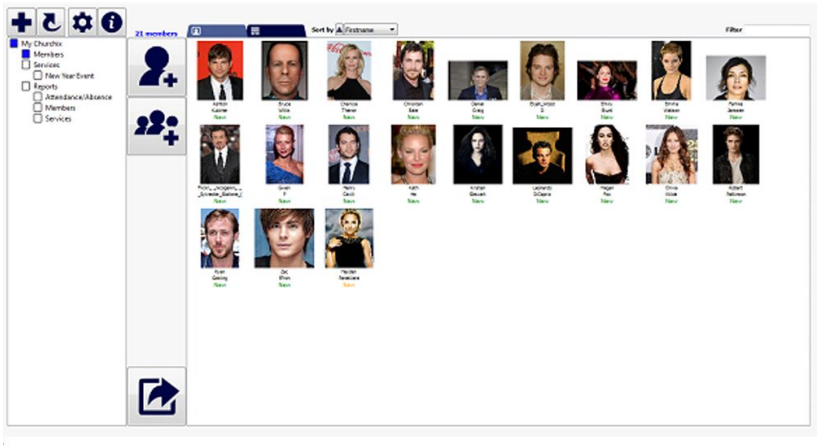
2.1.1 Churchix

O Churchix[1] é um *software* comercial de reconhecimento facial que visa identificar pessoas em vídeos e fotos, podendo registrar eventos de presença de estudantes, funcionários, suspeitos (pessoas sem acesso autorizado) etc. Segundo as informações presentes em seu website, o "Churchix é voltado para igrejas, salas de aula, hotéis, T&A (*Time and Attendance*, ou Tempo e Comparecimento, como em ambiente de trabalho) e registro de comparecimento em eventos, e para acompanhar suspeitos e criminosos". A entrada das imagens pode ser feita através de vídeos e fotos, ou através de conexão direta com câmeras conectadas via USB ou câmeras IP.

Na Figura 1 é apresentada uma imagem da tela de fotos de referência cadastradas no sistema. A técnica de reconhecimento utilizada por ele (não divulgada) extrai informações dessas fotos e as compara às extraídas de faces desconhecidas, para verificar se pertencem a alguém presente no banco de dados.

[1] "Churchix." http://churchix.com/. Acessado em 12 nov. 2017.

Figura 1 - Tela de fotos de referência cadastradas



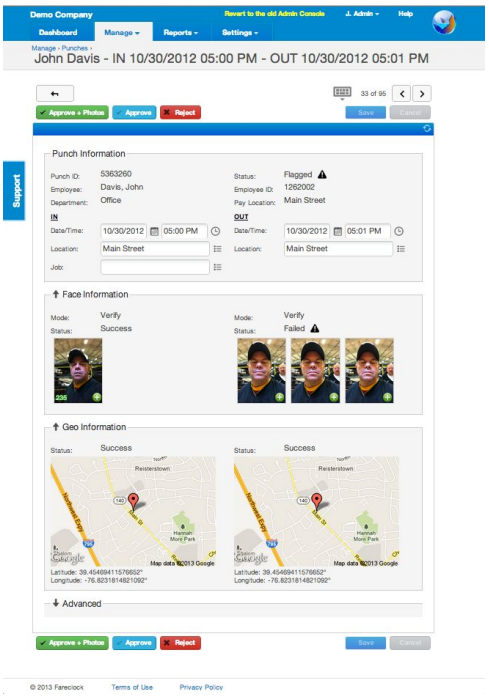
Fonte: churchix.com

2.1.2 Fareclock

O Fareclock[2] é um sistema pago de controle de presença e de horário de funcionários, que é administrado através de seu website. Um dos métodos que ele provê para registrar presença é o por reconhecimento facial, executado através de um aplicativo para smartphones que captura uma foto do funcionário e salva online a informação sobre o reconhecimento da face (incluindo a foto utilizada), o horário em que o procedimento ocorreu e o local onde foi executado (obtido através de GPS). A Figura 2 exibe uma tela com o resultado de um registro de presença.

[2] "Fareclock." https://www.fareclock.com/. Acessado em 12 nov. 2017.

Figura 2 - Tela de dados de um registro de presença



Fonte: fareclock.zendesk.com

2.1.3 Sistemas dedicados de controle de presença por biometria

Existe o segmento empresarial de máquinas dedicadas para o controle de acesso e/ou de presença por meio de biometria, o que pode envolver reconhecimento de impressão digital, reconhecimento de íris e o reconhecimento facial. A indiana NavKar Systems[3], por exemplo, oferece o "Face T71F" (ilustrado na Figura 3), um equipamento capaz de efetuar controle de acesso e de presença por reconhecimento de impressão digital, reconhecimento facial, uso de senha e uso de cartão com tag RFID. A FingerTec[4], por sua vez, possui toda uma linha de dispositivos similares chamada de "Face ID", que se diferenciam pelo tamanho do equipamento e funcionalidades disponíveis.

[3] "NavkarSys.com." http://navkarsys.com/. Acessado em 12 nov. 2017.
[4] "FingerTec." http://www.fingertec.com/. Acessado em 12 nov. 2017.

Figura 3 - Imagem ilustrativa do Face T71F da Navkar Systems

Fonte: biometricattendanceindia.com

2.2 Reconhecimento Facial

O problema do reconhecimento facial é tão antigo quanto o campo da visão computacional[1], devido a sua possível aplicação em diversos contextos (segurança, categorização de imagens, personalização de serviços, etc). Por abrir um leque de possibilidades se alcançar o nível da capacidade de reconhecimento humana, diversos autores tentaram abordá-lo ao longo das últimas décadas, o que levou a diversas soluções, com variados níveis de sucesso[2].

Para este trabalho, utiliza-se uma técnica considerada "estado de arte" no momento, baseada em redes neurais convolucionais. Como o objetivo deste projeto não é implementar métodos de reconhecimento facial, foram utilizadas as bibliotecas, a Dlib[5] e a *Face Recognition*[6], que, quando combinadas, implementam todo o ciclo de reconhecimento, que, segundo Adam Geitgey[3], consiste em, basicamente, quatro etapas principais:

1. Encontrar e extrair faces em uma imagem;
2. Aplicar transformações sobre as faces encontradas, de modo a tentar centralizá-las o máximo possível na imagem extraída;
3. Extrair "medidas" dessas faces desconhecidas; e
4. Calcular a diferença entre essas medidas e as medidas de faces conhecidas.

[5] "Dlib" http://dlib.net/. Acessado em 12 nov. 2017.
[6] "Face Recognition" https://github.com/ageitgey/face_recognition. Acessado em 12 nov. 2017.

A seguir, na seção “Detecção de Faces com Histograma de Gradientes Orientados”, será abordada a etapa 1. A etapa 2 ficará sob responsabilidade da 2.2.2, “Centralização da face”. Na seção 2.2.3, “Reconhecimento facial com Redes Neurais Convolucionais”, serão cobertas as etapas remanescentes.

2.2.1 Detecção de Faces com Histograma de Gradientes Orientados

Dada uma imagem qualquer, para detectar a eventual presença de faces, uma abordagem tradicionalmente utilizada é a de deslocar sobre ela retângulos de tamanhos variados e, para cada segmento delimitado por esses retângulos, executar um classificador (*support vector machine*, rede neural artificial, etc) treinado para reconhecer se o elemento contido nesse fragmento é uma face ou não. No caso da Dlib, a metodologia utilizada é semelhante, porém, ao invés de utilizar a imagem original como entrada para o classificador, utiliza uma gerada através da aplicação de uma variação[4] da técnica HOG (*histogram of oriented gradients*)[5].

A técnica HOG (*histogram of oriented gradients*), por sua vez, consiste em, dada uma imagem, dividi-la em células e, para os pixels em cada célula, gerar um histograma da direção dos gradientes. Após efetuado esse procedimento, a imagem gerada com essas células (contendo os histogramas) passa a descrever a imagem original sem que haja grandes variações decorrentes de diferenças de luminosidade. Um exemplo pode ser visto na figura 4.

Atualmente, já existem alternativas, teoricamente, mais eficazes que a HOG na detecção de faces[6]. No caso da biblioteca Dlib, essa alternativa é uma combinação de redes neurais convolucionais com *Max-Margin Object Detection*[7] [7] . Porém, tendo em vista que, em testes efetuados neste trabalho, o seu tempo de execução foi muito superior ao do HOG (ver seção 4.1), seu uso foi desconsiderado.

[7] "Dlib’s MMOD loss layer”, http://dlib.net/ml.html#loss_mmod_. Acessado em 19 nov. 2017.

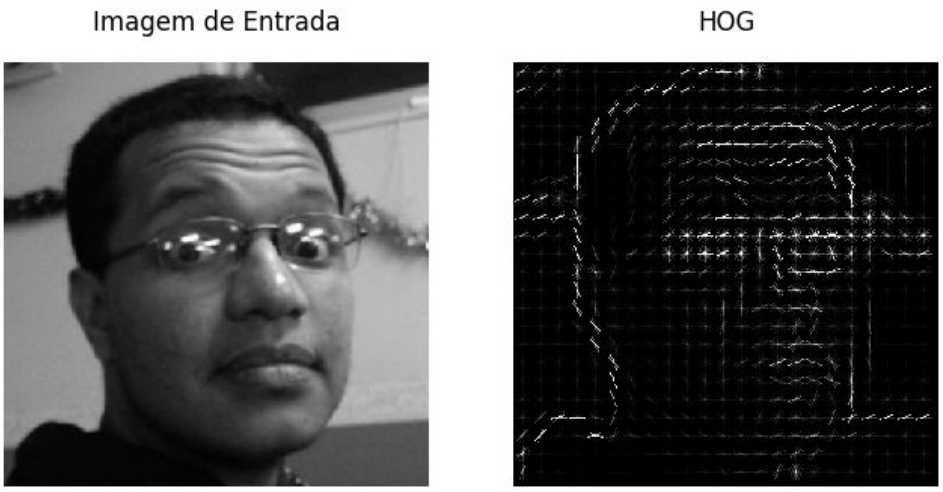


Figura 4: Histograma de Gradientes Orientados aplicado à imagem de um rosto utilizando células de 16 x 16 pixels

2.2.2 Centralização da face

Uma vez que uma face tenha sido encontrada em uma imagem, ela pode estar disposta em diversas formas possíveis, como de lado, olhando para cima, inclinada e etc. Para um ser humano, isso não implica em muitos problemas na hora da identificação, mas, para uma técnica de reconhecimento facial, isso poderá afetar negativamente a precisão, fazendo com que uma pessoa não seja reconhecida ou seja identificada como sendo outra pessoa.

Para contornar esse problema, primeiramente, é necessário encontrar pontos de referência nessas faces, como a localização das sobrancelhas, dos olhos, da boca, o queixo e etc. Para isso, uma das opções, é o uso de técnicas de estimativa de pontos de referência em faces (basicamente, um classificador que percorre a imagem em busca desses pontos), como a de Vahid Kazemi e Josephine Sullivan[8]. No caso da biblioteca *Face Recognition*, as referências são o queixo, as sobrancelhas, o nariz (sua extensão e sua base inferior), os olhos e os lábios. Um exemplo da detecção desses pontos pode ser visto na Figura 5 (à esquerda, imagem original, à direita, imagem com os pontos de referência desenhados sobre ela).

Figura 5 - Exemplo de detecção de pontos de referência em uma face

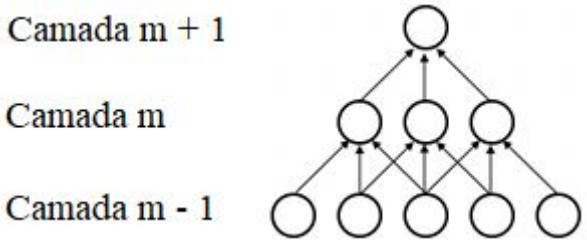
A partir da obtenção desse referencial, é efetuada uma série de transformações (redimensionamento, rotação e etc) em 2D para fazer com que a face encontrada seja disposta em uma imagem o mais centralizada possível.

2.2.3 Reconhecimento facial com Redes Neurais Convolucionais

Na segunda metade do século XX, através de pesquisas sobre o córtex visual de animais, descobriu-se que as células que o compõe são sensíveis apenas a pequenas sub-regiões do campo visual[9]. Ou seja, ao invés de todas lidarem com todo o *input* visual, cada uma é responsável por uma pequena área do que se vê, também conhecida como "campo receptivo". Isso, teoricamente, permite ao córtex filtrar uma imagem complexa e nela detectar diversos objetos, independente de onde e como eles estejam dispostos.

No campo da Ciência da Computação, as redes neurais convolucionais (RNC)[10] tentam reproduzir esse comportamento e, através dele, alcançar um desempenho próximo ao que se observa em humanos. Para isso, ao invés de possuírem uma estrutura igual a das redes neurais artificiais (RNA) convencionais, as RNCs, nas camadas de convolução, possuem neurônios artificiais (ou nós) que se conectam apenas a um grupo de nós (ou campo receptivo) da camada anterior (ou do *input*, se a que precede a camada de convolução for a entrada da rede neural). Esse comportamento é análogo a ter uma pequena janela se deslocando por uma imagem, da esquerda para a direita e de cima para baixo, tentando detectar (filtrar) informações como curvas, linhas horizontais e etc em ao longo dessa figura.

Figura 6 - Exemplo de conexão entre nós em camadas de convolução



Fonte: deeplearning.net

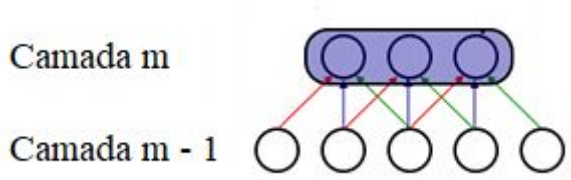
Como pode ser visto no exemplo da Figura 6, cada um dos neurônios da camada **m** está conectado a um campo receptivo de apenas três nós da camada **m - 1**. A camada **m + 1**, por sua vez, assim como a **m**, tem seus neurônios adjacentes à apenas 3 da camada anterior, mas, em relação à camada **m - 1**, ela processa, indiretamente, a saída oriunda de 5 neurônios diferentes. Isso significa que camadas mais ao fundo da rede neural são capazes de combinar as informações extraídas em camadas anteriores e "entenderem" e detectarem formas mais complexas. Por exemplo, se uma camada convolucional no ínicio de uma rede for capaz de detectar linhas horizontais, verticais e diagonais, uma outra camada convolucional posterior a ela pode ser capaz juntar os dados das camadas predecessoras e extrair formas como quadrados, retângulos, triângulos e etc.

Para serem capazes de filtrar as mesmas características em cada campo receptivo, todos os neurônios em uma camada convolucional possuem exatamente os mesmos pesos (e bias) em suas conexões com a camada anterior, o que significa que, se um nó A em uma camada possui conexões a_1, a_2 e a_3, com os nós P, Q e R na camada anterior, e um nó B, na mesma camada que A, possui conexões b_1, b_2 e b_3, com os nós Q, R e S na camada anterior, os pesos de a_1, a_2 e a_3 serão, respectivamente, iguais aos de b_1, b_2 e b_3. Isso permite que todos os neurônios de uma camada convolucional possam reagir aos mesmos estímulos, independente da área da camada anterior com a qual estejam lidando. Isso quer dizer que, se uma rede neural é treinada para encontrar um objeto específico, ela será capaz de localizá-lo em uma imagem não importando se está centralizado, posicionado no campo inferior esquerdo ou disposto à direita.

Ainda sobre a filtragem de elementos, tendo em vista que um único filtro em um neurônio não seria capaz de detectar todos os tipos de informações, há também o conceito de "mapa de características". Nele, considera-se que os nós de uma camada convolucional são,

na verdade, compostos por um ou mais filtros, cada um com suas próprias conexões (e, consequentemente, pesos) com o campo receptivo do neurônio na camada anterior. Isso pode ser também interpretado como se uma camada convolucional tivesse múltiplas subcamadas paralelas, cada uma tendo seus neurônios responsáveis por filtrar uma ou mais "características" de sua entrada. Ou seja, um mapa de características pode ter seus neurônios (ou filtros) capazes de "extrair" curvas, outro mapa pode ser capaz de detectar linhas verticais e diagonais, outro pode ser capaz de encontrar linhas horizontais e etc.

Figura 7 - Exemplo de pesos compartilhados entre nós em um mesmo mapa de características

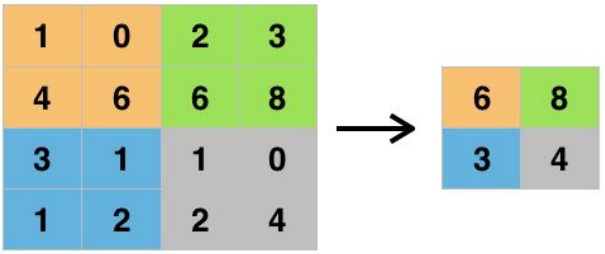


Fonte: deeplearning.net

A Figura 7 tenta ilustrar o compartilhamento de pesos através do uso de cores para as arestas entre as camadas, nas quais cores iguais significam pesos iguais. A área roxa envolvendo a camada de convolução, por sua vez, serve para denotar que esses neurônios representam apenas um mapa de características. Caso haja mais de um mapa em uma camada, o aspecto de que os pesos são compartilhados entre os neurônios só será válido dentro de um mesmo mapa, pois cada mapa de características tem suas próprias conexões (e pesos por conexão) com a camada anterior.

O último conceito diferente presente em uma rede neural convolucional é uma técnica chamada *max pooling* (ilustrada na Figura 8). Ao ser aplicada sobre a saída de uma camada, ela a divide em seções (normalmente de mesmo tamanho) não sobrepostas e, dentre os sinais (valores) em cada seção, extrai apenas aquele de maior valor, gerando como saída a concatenação desses sinais mais relevantes. Esse procedimento é feito para reduzir a dimensão da entrada que recebe e se faz necessário em diversas situações, como, por exemplo, em RNCs que processam imagens grandes, visto que, se ela receber como entrada uma imagem em *full HD*, essa entrada terá um tamanho de 2073600 pixels (1920 x 1080), o que requererá uma quantidade excessiva de recursos para ser processada.

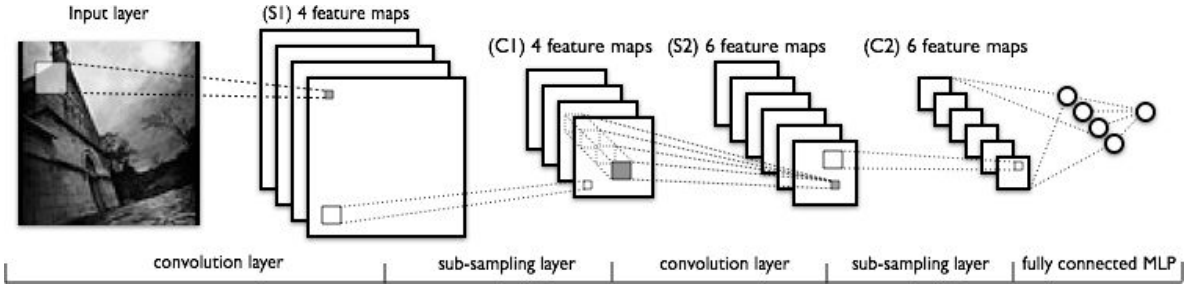
Figura 8 - Exemplo de max pooling



Fonte - wikipedia.org

Dadas essas camadas de convolução e de *max pooling*, uma rede neural convolucional é composta de múltiplas combinações entre camadas desses tipos e as convencionais de redes neurais artificiais ("densas"), onde cada um de seus nós é conectado a todos os nós da anterior. A arquitetura mais apropriada dependente do problema a ser resolvido. A Figura 9 exibe um exemplo de uma rede neural convolucional. Note que, nessa imagem, "*feature map*" refere-se ao mapa de características citado nesta seção, "*sub-sampling*" remete ao *max pooling* e "*fully connected MLP*" (*Multilayer Perceptron*) são as camadas densas.

Figura 9 - Diagrama exemplificando uma rede neural convolucional



Fonte: deeplearning.net

Quanto à saída de uma rede neural convolucional, quando utilizada para o processamento de imagens, podem ser valores representando classificação ou regressão. Para o caso de reconhecimento facial com a Dlib, a rede utilizada é um modelo de regressão, onde é gerado um *encode* ("codificação") de 128 valores que denotam uma face. Com esse vetor em mãos, para efetuar o reconhecimento, basta calcular a distância (norma de um vetor) entre

ele e outros vetores representando outras faces e, se o resultado para um dado par estiver dentro de uma distância tolerável, declarar as duas faces como pertencentes à mesma pessoa.

2.3 Considerações Finais

Neste capítulo foram apresentadas algumas soluções comerciais para o registro de presença por reconhecimento facial e foi feita uma breve introdução às tecnologias envolvidas no ciclo de identificação de faces, a qual teve a exclusiva intenção de dar uma ideia sobre como esse processo funciona. No próximo capítulo será abordado o desenvolvimento do PresentEye, o levantamento de requisitos, o modelo de dados, a arquitetura do sistema e as tecnologias empregadas.

3. Desenvolvimento do PresentEye

Considerando os métodos mais utilizados para registro de presença, como a chamada oral da lista de presença e o uso de cartões magnéticos, várias possibilidades de falhas podem ser identificadas, como um professor esquecer de verificar a presença, efetuar a chamada antes de alguns alunos chegarem, um estudante esquecer de passar seu cartão magnético no dispositivo responsável por registrar seu comparecimento, etc. Para tentar resolver esses problemas, o PresentEye foi desenvolvido pensando em transformar a checagem de presença em um procedimento automático e involuntário.

As seções a seguir apresentam a especificação do PresentEye, descrevem sua arquitetura e comentam sobre as tecnologias empregadas no desenvolvimento do sistema.

3.1 Requisitos

Afim de prover um conjunto mínimo de funcionalidades para a criação de um MVP (*Minimum Viable Product*) capaz de alcançar os objetivos deste trabalho, foi levantado um conjunto mínimo de requisitos. As seções a seguir listam esses requisitos separados por componente do PresentEye.

3.1.1 PresentEye Tunnel Client

Requisitos Funcionais

O sistema deverá:

- Permitir a definição dos dados para conexão com PresentEye Tunnel Server (IP e porta); e
- Prover uma opção para que o usuário defina o número máximo de câmeras com as quais deseja se conectar.

Requisitos não-funcionais

O sistema deverá:

- Disponibilizar suas opções de configuração através de um arquivo de texto;
- Ser capaz de funcionar em Linux e em Windows; e
- Dar suporte a webcams.

3.1.2 PresentEye Tunnel Server

Requisitos Funcionais

O sistema deverá:

- Permitir a definição dos dados para a conexão com o PresentEye Server (IP, porta, usuário e senha);
- Possibilitar a definição dos diretórios para o armazenamento de arquivos de vídeo processados e não-processados;
- Ser capaz de efetuar reconhecimento facial sobre os frames recebidos; e
- Ser apto a salvar os frames recebidos.

Requisitos não-funcionais

O sistema deverá:

- Disponibilizar suas opções de configuração através de um arquivo de texto;
- Utilizar múltiplos núcleos do processador durante a execução do reconhecimento facial;
- Ser executado em Linux;
- Salvar os *frames* recebidos como vídeo; e
- Armazenar os vídeos em arquivos de tamanho inferior a 1MB (um megabyte).

3.1.3 PresentEye Server

Requisitos Funcionais

O sistema deverá:

- Permitir a execução de operações CRUD (*Create, Read, Update, Delete*) para câmeras, Tunnel Clients, Tunnel Servers, estudantes, usuários, salas de aula, disciplinas e aulas;
- Possibilitar o registro de detecções de estudantes; e
- Ser capaz de gerar relatório de presença de estudantes ao longo das aulas de uma disciplina.

Requisitos não-funcionais

O sistema deverá:

- Prover a funcionalidade de registro de detecções apenas via API;
- Ser compatível com o servidor HTTP Apache, mesmo que através do uso de *plug-ins*; e
- Funcionar em ambiente Linux.

3.2 Modelo de Dados

O modelo de dados da PresentEye, conforme exibido na Figura 10, foi modelado através um diagrama de classes da UML e mostra quais são os dados armazenados e como eles se relacionam. Ele é aplicável apenas ao PresentEye Server, pois as outras aplicações não possuem banco de dados.

Quanto à lógica por trás desse esquema, ele foi construído em volta de um relacionamento fundamental para o caso do registro de presença por reconhecimento facial: "um estudante é detectado em uma aula" (em outras palavras, ele é reconhecido pelo sistema como presente em uma aula). A partir desse relacionamento o modelo foi expandido.

O primeiro passo para essa melhoria foi pensar em como e onde um **estudante** é **detectado**. Se o reconhecimento facial ocorre através de **vídeo** capturado dentro de uma **sala de aula**, então, é possível considerar a existência de uma **câmera** nesse local. Além disso, surgiram questões sobre como essa câmera se conectaria ao sistema e como ela seria identificada. Para respondê-las, foi necessário considerar como seria a arquitetura necessária para permitir que a aplicação suportasse uma variação na demanda, e isso levou à figura do **PresentEye Tunnel** e seus componentes, **Server** e **Client**. Como uma câmera fica conectada

a apenas um Client, sua identificação pode ser dependente dele, e, como um Client só se conecta a um Server, sua identificação pode depender de seu Server.

Após definida uma estrutura inicial que permitiu saber como e onde um **aluno** foi detectado, o desafio seguinte foi definir o que é uma **aula** e quando ela ocorre. Nesse caso, uma aula é sempre relacionada a alguma **aplicação semestral** de uma **matéria** (na qual um **aluno** pode estar matriculado). Também é válido considerar que aulas de uma matéria em um **semestre** podem acontecer em múltiplos **dias da semana**, várias vezes ao dia e em **horários** diferentes.

Por fim, tendo em vista que o PresentEye Server precisaria controle de usuário, foi criada a classe System User. Mesmo que, em um primeiro momento, o PresentEye seja destinado a professores, ela recebeu esse nome para generalizar os usuários, pois, no futuro, pode ser que não-professores também venham a utilizar o sistema.

Com esse contexto base definido, o passo seguinte foi verificar o que poderia vir a se tornar uma classe. A Tabela 1 mostra o resultado. Dentre os elementos com grandes chances, um deles, o "vídeo", acabou sendo desconsiderado, pois o sistema desenvolvido, no estado de MVP (*Minimum Viable Product*), não necessitaria manter dados sobre os vídeos capturados. Horário não foi utilizado como classe por ter sido mapeado como atributo.

Tabela 1 - Mapeamento de elementos candidatos a se tornarem classes

Candidato	Classe Resultante	Candidato	Classe Resultante
Estudante / Aluno	*Student*	Matéria	*Subject*
Aula	*Subject Class*	Aplic. Semestral	*Subject Semester*
Detecção	*Detection*	Semestre	*Semester*
Vídeo	-----	Dia da Semana	*Week Day*
Sala de Aula	*Room*	Horário	-----
Camera	*Camera*	Usuário	*System User*
PEye Tunnel Client	*Tunnel Client*		
PEye Tunnel Server	*Tunnel Server*		

Neste texto não foi feita menção à maioria dos atributos das entidades. Porém, esses podem ser facilmente vistos na Figura 10. Foram utilizados termos em inglês para que o código fique acessível ao maior número de pessoas possível, caso venha a ser disponibilizado publicamente.

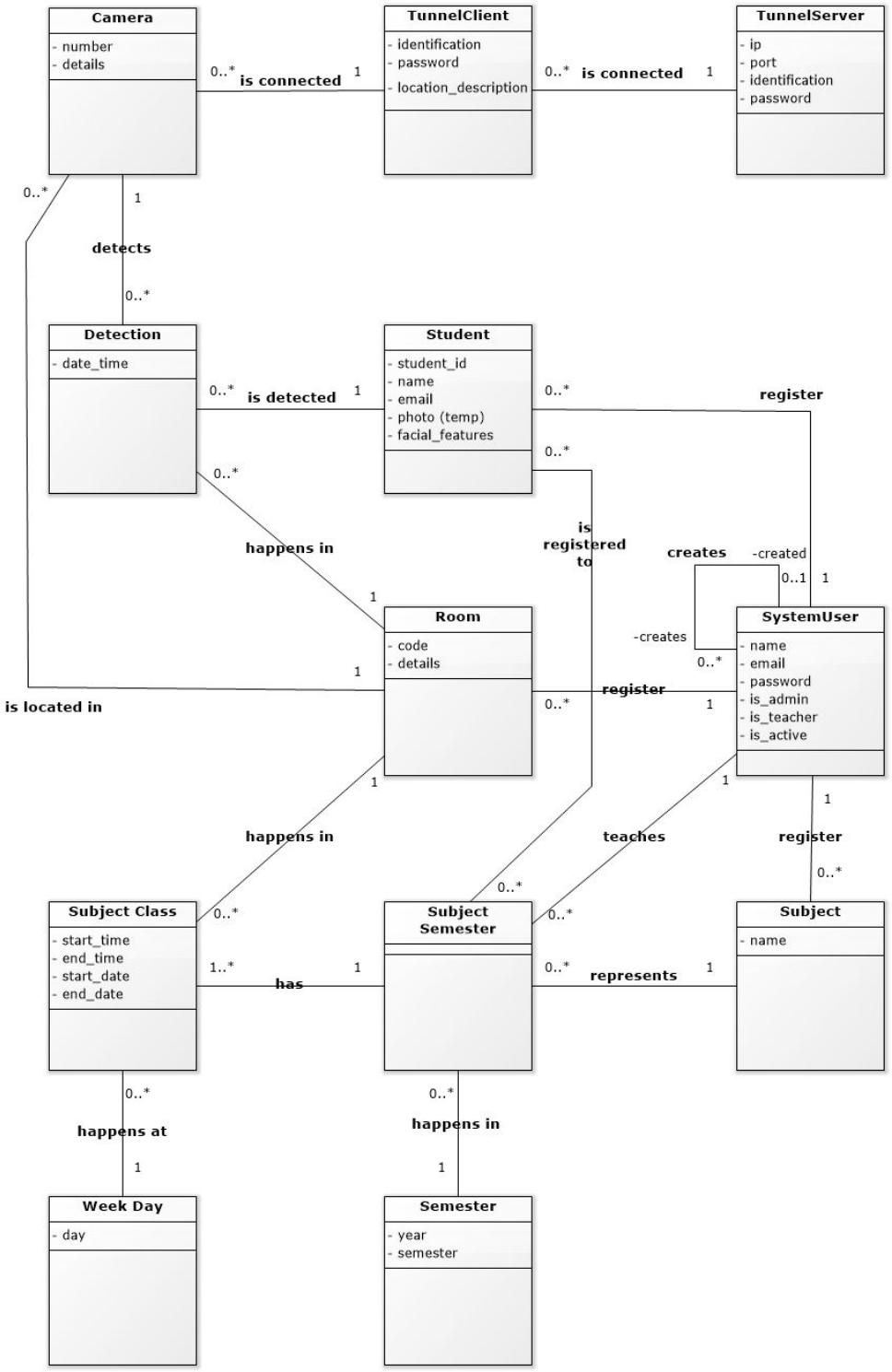


Figura 10 - Modelo de dados do PresentEye Server

3.3 Tecnologias Empregadas

Para desenvolver o sistema, optou-se pelo uso de tecnologias e ferramentas simples e populares, visando simplificar o processo de desenvolvimento e facilitar a obtenção de informações em caso de problemas. Foram utilizadas apenas bibliotecas e ferramentas *open-source* ou com uso gratuito ilimitado.

3.3.1 Python

A linguagem Python[8] é uma linguagem de programação interpretada, que se popularizou na última década por, dentre outros motivos, possuir uma sintaxe bastante simples e legível. Além desses fatores, ela foi escolhida para o desenvolvimento do PresentEye pelo fato de ser interpretada (evitando perda de tempo com compilação) e possuir bibliotecas que proveem a funcionalidade de reconhecimento facial. São utilizadas as versões 2.7[9] (no PresentEye Tunnel) e 3[10] (no PresentEye Server).

3.3.2 Sublime Text

O Sublime Text[11] é um editor de texto multiplataforma. É pago, mas oferece uma versão de testes sem restrições quanto ao tempo de uso. Foi escolhido por prover a funcionalidade de destacamento de elementos de sintaxe para diversas linguagens de programação, marcação e etc, o que inclui Python, HTML e CSS, as quais foram utilizadas neste projeto. Além disso, por não ter todos os recursos providos por uma IDE, é extremamente leve.

[8] "Python" https://www.python.org/. Acessado em 12 nov. 2017.
[9] "Python 2.7.14 documentation." https://docs.python.org/2/index.html. Acessado em 12 nov. 2017.
[10] "Python 3.6.3 Documentation." https://docs.python.org/. Acessado em 12 nov. 2017.
[11] "Sublime Text." https://www.sublimetext.com/. Acessado em 12 nov. 2017.

3.3.3 Django

O Django[12] é um framework para a criação de aplicações web na linguagem Python. Foi escolhido para o PresentEye Server por ter uma boa documentação e, principalmente, por prover um componente de ORM (*Object-Relational Mapping*), através do qual é possível criar o modelo de dados do sistema e executar operações CRUD sobre ele sem a necessidade do uso explícito de SQL.

3.3.4 SQLite

O SQLite[13] é um banco de dados bem simples e "embarcado", baseado em um único arquivo. Para utilizá-lo, normalmente, basta importar uma biblioteca e informar o nome do arquivo do banco de dados.

O motivo para seu uso se resume à simplicidade da atual implementação do PresentEye, que se apresenta como um MVP (*Minimum Viable Product*), e pelo fato de o Django prover suporte para ele. No entanto, se o PresentEye Server vier a ser utilizado para atender dezenas de PresentEye Tunner Servers, a utilização de um SGBD (Sistema de Gerenciamento de Banco de Dados) propriamente dito poderá ser mais apropriada.

3.3.5 Biblioteca Dlib

A Dlib[14] é uma biblioteca de *machine learning*, escrita em C++, que implementa diversos algoritmos e ferramentas para lidar com problemas dessa área, dentre eles, reconhecimento facial. Possui uma interface em Python, que é utilizada neste projeto indiretamente, através da biblioteca *Face Recognition*[15], também em Python.

Foi escolhida por ser mais precisa e eficiente que a biblioteca OpenCV[16] no reconhecimento facial ao utilizar a técnica HOG (*Histogram of Oriented Gradients*), em conjunto com um classificador treinado para detectar faces (especificamente, *support vector*

[12] "Django" https://www.djangoproject.com/. Acessado em 12 nov. 2017.
[13] "SQLite" https://www.sqlite.org/. Acessado em 12 nov. 2017.
[14] "Dlib" http://dlib.net/. Acessado em 12 nov. 2017.
[15] "Face Recognition" https://github.com/ageitgey/face_recognition. Acessado em 14 nov. 2017.
[16] "OpenCV" https://opencv.org/. Acessado em 12 nov. 2017.

machine), e utilizar um modelo pré-treinado de uma Rede Neural Convolucional na hora de gerar o vetor descritor da face (a partir do qual o reconhecimento propriamente dito pode ser alcançado através do simples cálculo de distância entre vetores descritores). Para um descritor de 128 dimensões, alcançou uma precisão de 99.38%[11] no dataset Labeled Faces in the Wild[12].

3.3.6 Biblioteca Face Recognition

A *Face Recognition*[17] é uma biblioteca de reconhecimento facial (na verdade, uma API) baseada nos recursos providos pela biblioteca Dlib para essa função. Foi escolhida por, além de prover uma interface simples para acesso aos recursos da Dlib, implementar também o cálculo de distância euclidiana entre os vetores descritores de faces.

3.3.7 Biblioteca OpenCV

O OpenCV[18] é uma popular biblioteca de *computer vision* disponível para diversas linguagens de programação, inclusive Python. Possui, dentre outras funcionalidades, a capacidade de se conectar a câmeras e, até mesmo, executar reconhecimento facial. Porém, tendo em vista que as técnicas que disponibiliza para essa segunda função não são tão eficientes quanto à implementada pela biblioteca Dlib, seu trabalho neste projeto se resume a extrair os quadros de vídeo das câmeras.

3.3.8 Biblioteca Sckit-Video

A biblioteca Scikit-Video[19] é uma biblioteca criada para prover funcionalidades de processamento de vídeos para a linguagem Python. Vale ressaltar que a OpenCV também oferece esta funcionalidade, mas a Scikit-Video foi empregada neste projeto pois problemas foram encontrados ao tentar utilizar a biblioteca OpenCV para salvar vídeos no Linux.

[17] "Face Recognition" https://github.com/ageitgey/face_recognition. Acessado em 12 nov. 2017.
[18] "OpenCV" https://opencv.org/. Acessado em 12 nov. 2017.
[19] "Scikit-Video" http://www.scikit-video.org/. Acessado em 12 nov. 2017.

3.3.9 Computação em Nuvem

Para a execução dos testes, foi utilizado um servidor na nuvem visando testar o funcionamento da arquitetura da aplicação. O provedor desse serviço foi o *Google Cloud Platform*[20], no qual foi utilizada uma instância (basicamente, uma máquina virtual) de seu componente "*Google Compute Engine*", com 15 GB de memória RAM e uma CPU com quatro núcleos virtuais. O uso desse serviço foi integralmente financiado pelos créditos de teste oferecidos gratuitamente pela própria Google para qualquer usuário novo.

3.3.10 Apache Webserver

O Apache[21] é um servidor web gratuito e de código aberto. Foi empregado neste projeto para executar o PresentEye Server durante a fase de teste, pois é uma solução mais confiável que o servidor web simples que a linguagem Python possui em sua biblioteca padrão, que é mais voltar para testes e debug durante o desenvolvimento de aplicações.

3.4 Arquitetura da Aplicação Implementada

O PresentEye é composto por duas aplicações totalmente interdependentes. São elas o PresentEye Tunnel e o PresentEye Server. O PresentEye Server é uma aplicação web responsável por manter os dados e prover uma interface para interação com o usuário. O PresentEye Tunnel, por sua vez, é o responsável por gerir a conexão com as câmeras, salvar os vídeos capturados e executar o reconhecimento facial propriamente dito.

Dada a natureza das tarefas realizadas pelo Tunnel, ele também se subdivide em duas partes, "Client" e "Server" (disponibilizadas juntas), buscando viabilizar certo grau de escalabilidade. O "Client" tem a exclusiva função de servir de *hub* para câmeras e enviar o *streaming* obtido para um "Server". O "Server", por sua vez, recebe os dados de um ou mais "Clients" e efetua o reconhecimento facial sobre os quadros de vídeo recebidos, caso haja a necessidade. Um esquema dessa arquitetura pode ser visto na Figura 11.

[20] "Google Cloud Platform." https://cloud.google.com/. Acessado em 12 nov. 2017.
[21] "Apache" https://www.apache.org/. Acessado em 14 nov. 2017.

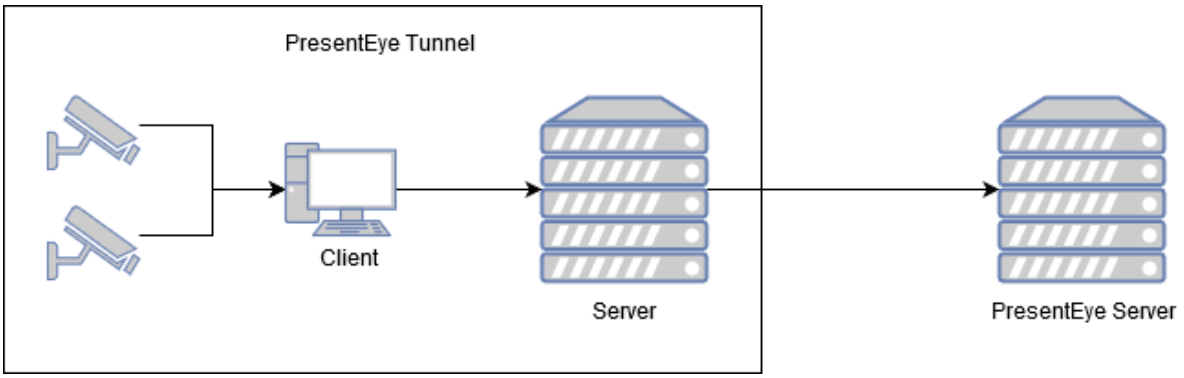


Figura 11 - Arquitetura em alto nível do PresentEye

As seções a seguir entrarão nos pormenores de cada um desses componentes.

3.4.1 PresentEye Tunnel Client

Trata-se de uma aplicação de linha de comando (desenvolvida em Python), responsável por acessar as câmeras disponíveis no computador em que estiver sendo executada (dispositivo com capacidade de processamento de dados em geral, como um PC, um Raspberry Pi, um Beaglebone, etc) e enviar o stream de vídeo para o PresentEye Tunnel Server. Para a primeira tarefa, utiliza a biblioteca OpenCV, que implementa código para lidar com essa função. Como o PresentEye se apresenta como um MVP (*Minimum Viable Product*), apenas a comunicação com câmeras web USB foi implementada. A conexão com câmeras IP poderá ser adicionada no futuro, tendo em vista que a biblioteca utilizada disponibiliza essa funcionalidade.

Quanto ao protocolo utilizado para comunicação do Client com o PresentEye Tunnel Server, também se apresenta de modo simplificado. Considerando que o sistema já tenha sido inicializado, o arquivo de configurações lido e, consequentemente, os dados do Tunnel Server (IP e porta) e do Tunnel Client (usuário, senha) tenham sido obtidos, esse Client estabelece com seu Tunnel Server uma conexão (TCP) para cada câmera e passa a enviar continuamente os quadros de vídeo obtidos. A frequência padrão de captura dos quadros de uma câmera é de, aproximadamente, 2 quadros por segundo (2 fps, ou *frames per second*), porém, como a obtenção e envio de um quadro depende do envio do anterior (da mesma câmera), essa frequência real pode cair para 1 ou menos quadros por segundo se a velocidade da conexão

não for adequada para o tamanho do quadro trafegado ou se a latência na comunicação estiver muito alta. O tamanho de cada um desses quadros, por sua vez, depende da resolução de captura definida no arquivo de configurações. O formato da mensagem responsável por enviar um quadro ao Tunnel Server é apresentado na Figura 12.

[TAM][USUARIO][TAM][SENHA][TAM][CAMERA][TAM][QUADRO]

Figura 12 - Formato da mensagem trafegada entre o PresentEye Tunnel Client e o PresentEye Tunnel Server

Como pode ser visto, cada componente da mensagem é precedido por um valor (especificamente, um long) que indica seu tamanho. Assim, quando um Tunnel Server recebe uma mensagem de um Client, ele precisará saber o número de campos esperado e ler o primeiro número recebido para ser capaz de interpretar a mensagem por completo.

Caso o Client seja incapaz de estabelecer conexões com o Server, as tentativas serão suspensas por 30 segundos antes de serem restabelecidas. Isso se repetirá até que ele consiga se conectar ou seja finalizado.

Apesar de funcional, o protocolo utilizado é ineficiente, já que estabelece um streaming permanente sobre o protocolo TCP. Caso o desenvolvimento do PresentEye seja continuado, essa se tornará uma partes a serem melhoradas, caso contrário, o uso em um uma situação real poderá se tornar muito custoso ou, até mesmo, inviável, pois poderá necessitar de uma rede com altíssima velocidade e baixíssima latência entre o Tunnel Client e o Tunnel Server.

3.4.2 PresentEye Tunnel Server

Responsável por concentrar conexões de Clients, o Tunnel Server também tem como função, se necessário, efetuar o reconhecimento facial e salvar vídeos. Para descrever seu funcionamento, considere a seguinte situação hipotética, a ser referenciada como Exemplo 1, onde:

1. Em uma sala, existem duas webcams conectadas, via cabo USB, a um Tunnel Client localizado em um lugar próximo qualquer (dentro dessa mesma sala, em uma sala próxima etc);
2. O Client encontra-se conectado, através da internet, a um Tunnel Server sendo executado em uma máquina virtual num serviço de computação em nuvem, e;
3. O Tunnel Server se comunica com um PresentEye Server disponibilizado em uma outra máquina virtual na mesma rede em que se encontra.

Na Figura 13 tal situação é ilustrada.

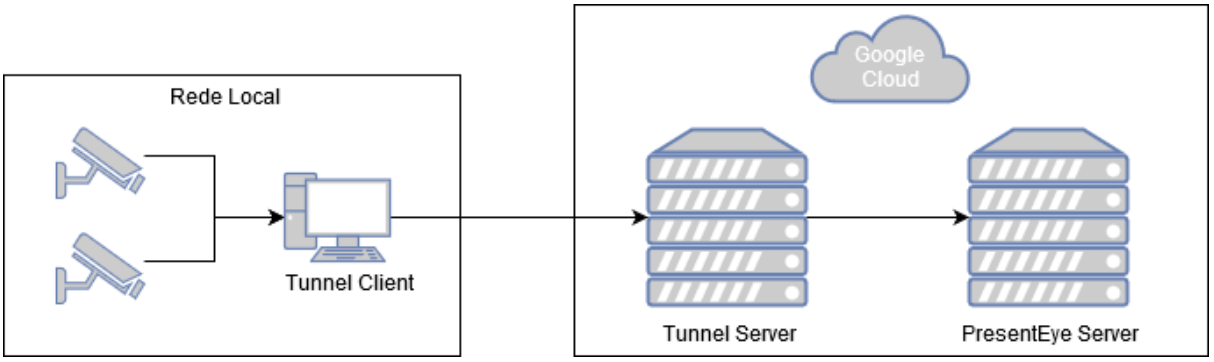


Figura 13 - Arquitetura do PresentEye segundo o Exemplo 1

Dado esse contexto, o Tunnel Server, ao ser inicializado, lerá informações do arquivo de configurações e aguardará por conexões de Tunnel Clients (por padrão, através da porta 3256). Quando um Client se conectar, o Server começará a receber um fluxo contínuo de frames da câmera conectada a esse Client. No Exemplo 1, como o número de câmeras é dois, o Client irá estabelecer uma conexão para cada câmera e, consequentemente, enviará os frames oriundos das duas em separado. O Server, por sua vez, salvará em seu estado (uma variável) a informação que as câmeras estão disponíveis e descartará todos os quadros recebidos, visto que não há a necessidade de salvar dados se não estiver ocorrendo uma aula no local onde essas câmeras se encontram.

Paralelamente a isso e sem depender da existência de Clients conectados, a cada 30 segundos, uma thread do Tunnel Server tentará enviar ao PresentEye Server (a ser referido a partir daqui como “webserver”), através do método “reportAvailableCameras” da API deste

último, um JSON contendo dados de autenticação e uma lista com cada par de Client e câmera conectados. A chamada a esse método poderá resultar em três possíveis respostas:

1. Um JSON contendo, para cada par de Client e câmera, uma lista com os estudantes de uma disciplina e suas "características faciais" (o *encoding* citado na seção 2.2.3), se uma aula da disciplina estiver acontecendo onde a câmera se encontra.
2. Um JSON contendo uma lista vazia de pares de Client e câmera, se nenhuma aula estiver ocorrendo onde as câmeras se localizam, ou se, na chamada ao método, tiver sido enviada uma lista vazia de pares de Client e câmera.
3. Um erro 404 (página não encontrada), se as informações de autenticação do servidor estiverem incorretas.

Se o retorno for como nas respostas 2 e 3, nada será feito e a execução da aplicação não sofrerá alteração. Por outro lado, se for a alternativa 1, os dados dos estudantes e de seu *encoding* facial serão salvos em memória e, para cada câmera em um local onde estiver ocorrendo uma aula, o sistema começará a também salvar em memória os frames recebidos. No Exemplo 1, a existência de duas câmeras numa mesma sala resultaria na replicação dos dados de estudante para cada câmera.

Para evitar que o armazenamento de imagens esgote a memória RAM, a cada 20 segundos, uma thread verificará se existem frames mantidos pela aplicação e, em caso afirmativo, para cada par de Client e câmera, criará um arquivo de vídeo contendo os quadros recebidos. Cada vídeo será salvo em um diretório dedicado aos arquivos ainda não processados (nos quais ainda não foi feito o reconhecimento facial) e será nomeado conforme o padrão da Figura 14.

nproc_record_[CLIENT_ID]_[CAMERA]_[TEMPO_EM_MILISSEGUNDOS].mp4

Figura 14 - Padrão de nomenclatura dos arquivos de vídeo salvos

Se o Client do Exemplo 1 possuir ID "CCC222" e, a ele conectada, existir uma câmera de numeração "0", localizada em uma sala onde esteja acontecendo uma aula, um

vídeo salvo em “1507149387155” (tempo em milissegundos) teria a sua nomenclatura como na Figura 15.

nproc_record_CCC222_0_1507149387155.mp4

Figura 15 - Exemplo de nomenclatura de arquivos de vídeo ainda não processados

Essa ação de receber frames continuamente e, com eles, gerar pequenos vídeos em curtos intervalos de tempo continuará até que se verifique o término da aula onde as câmeras se encontram (até que passe o horário de término da aula, também recebido via JSON).

Até este ponto, as tarefas citadas têm o objetivo de salvar o vídeo recebido das câmeras, mas em nada cobrem o de efetuar o reconhecimento facial. Para alcançar esse propósito, existe um outra thread sendo executada em paralelo que tentará, a cada segundo, efetuar as seguintes tarefas:

1. Obter uma cópia dos dados dos pares de Client e câmera cujas câmeras ficam em locais onde estejam acontecendo aulas no momento;
2. Coordenar a execução do reconhecimento facial em vídeos ainda não “processados” (nos quais tal tarefa ainda não foi executada); e
3. Enviar ao PresentEye Server os dados dos alunos detectados.

A primeira tarefa executada não requer detalhamento, visto que não passa de um caso de cópia de referência de variável. A segunda, por outro lado, é a que consome mais recursos (processamento) em toda a aplicação. Descrevendo-a com base no Exemplo 1, os dois pares de Client e câmera produzirão um arquivo de vídeo cada, a cada 20 segundos. A thread responsável por lidar com as tarefas relacionadas ao reconhecimento facial, por sua vez, tentará executar o segundo passo da lista a cada segundo e, uma vez que encontre um arquivo ainda não processado, irá percorrê-lo quadro por quadro. Para cada quadro, tentará criar um processo para executar o reconhecimento facial (foi definido um limite máximo de 5 processos por padrão, mas esse valor pode ser alterado no arquivo de configurações). Caso não seja possível, a thread aguardará até os processos utilizados para processar frames anteriores terminarem sua execução e dará continuidade a essa tarefa.

Uma vez que um arquivo tenha sido totalmente processado, ele será movido para um diretório dedicado aos arquivo já processados e terá o "nproc" no início de seu nome substituído por "proc", como na Figura 16.

proc_record_CCC222_0_1507149387155.mp4

Figura 16 - Exemplo de nomenclatura de arquivos de vídeo processados

Após a execução do reconhecimento facial, se, em um frame, for detectado um estudante, essa informação será mantida em memória e, quando o terceiro passo for executado, ela será enviada ao PresentEye Server, em um JSON, através do método "reportRecogtion", que retornará um outro JSON informando se houve erro ou não (erros podem ocorrer, por exemplo, no caso de falha de autenticação). No JSON enviado pelo Tunnel Server será enviada uma lista com os pares de Client e câmeras e, para cada par, uma lista com os alunos cuja presença foi detectada e os horários nos quais as detecções ocorreram. Da parte do PresentEye Server, é possível processar múltiplas detecções para um determinado aluno em um dado intervalo de tempo, porém, na implementação atual do Tunnel Server, cada aluno só tem reportada uma detecção por intervalo, cujo horário é ajustado para o momento do término do intervalo (momento em que o arquivo de vídeo foi salvo, o que causa uma diferença de algo entre 0 e 25 segundos entre o horário real e o horário de "detecção" salvo). Essa abordagem é imprecisa, mas reduz a quantidade de dados enviados.

3.4.3 PresentEye Server

Webserver responsável por gerir os dados do PresentEye e prover o relatório de presença nas aulas. Foi desenvolvido em Python e construído sobre o framework web Django, que possui um ótimo componente de ORM (*Object-Relational Mapping*). Por essa razão, todo o seu modelo de dados é gerado no banco de dados automaticamente conforme a especificação das classes que o definem (Figura 10). Por questões de simplicidade, o banco de dados escolhido foi o SQLite.

O PresentEye Server possui uma API simples para ser consumida pelo PresentEye Tunnel Server, constituída por dois métodos: “reportAvailableCameras” e “reportRecogtion”, os quais são detalhados na seção 3.4.2. Também é capaz de efetuar operações CRUD sobre os dados, tanto através de suas próprias páginas, quanto pela interface de administração do framework (Django Admin) que, com base nos metadados do modelo de dados, gera automaticamente um website secundário que permite efetuar diretamente a criação, leitura, alteração e exclusão de instâncias do modelo. Capturas de tela dessas duas interfaces podem ser vistas nas Figuras 17 e 18.

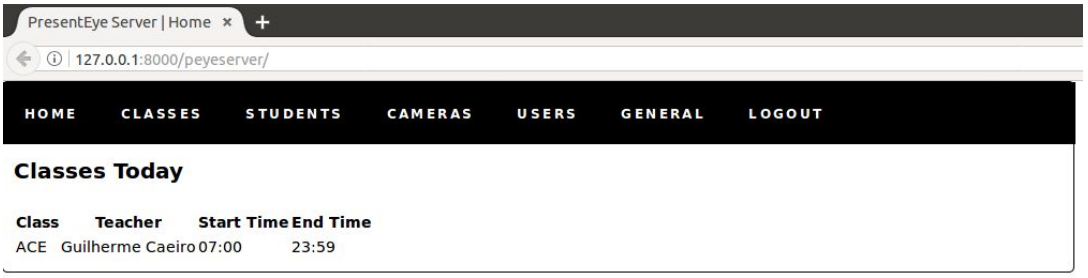


Figura 17 - Tela inicial do PresentEye Server

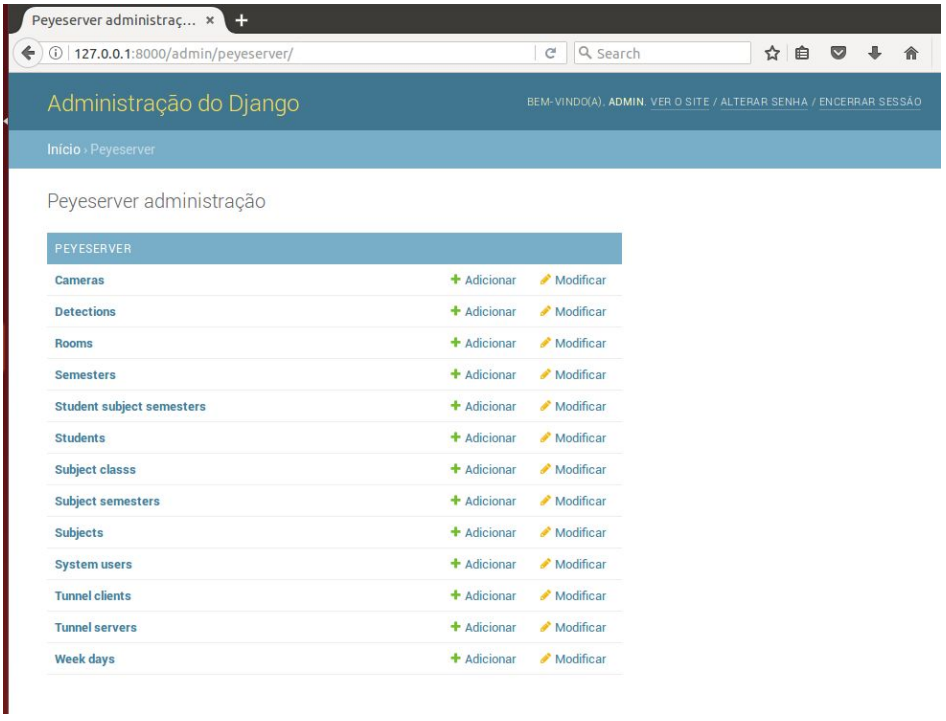


Figura 18 - Tela inicial do site Django Admin

O relatório de presença (Figura 19) apresenta para cada aula uma lista de alunos, e, para cada aluno, além de seu id de estudante e seu nome, um indicador de detecções ao longo da aula e uma classificação informando se estava realmente presente ou não. Esse indicador e essa classificação funcionam do seguinte modo:

1. O tempo da aula é dividido em dez intervalos iguais. Ou seja, se uma aula possui uma hora de duração, terá esse tempo dividido em dez intervalos de seis minutos;
2. Para cada intervalo, é verificado se existe alguma detecção desse aluno. Caso exista, no relatório, é marcado um "O" verde. Se não existir, é marcado um "X" vermelho.
3. Se o aluno tiver sido detectado em 40% dos intervalos dos intervalos (número alcançado através dos testes reportados na seção 4.2) ou mais, será classificado presente. Se ele for detectado em 3 ou menos, será classificado como ausente.

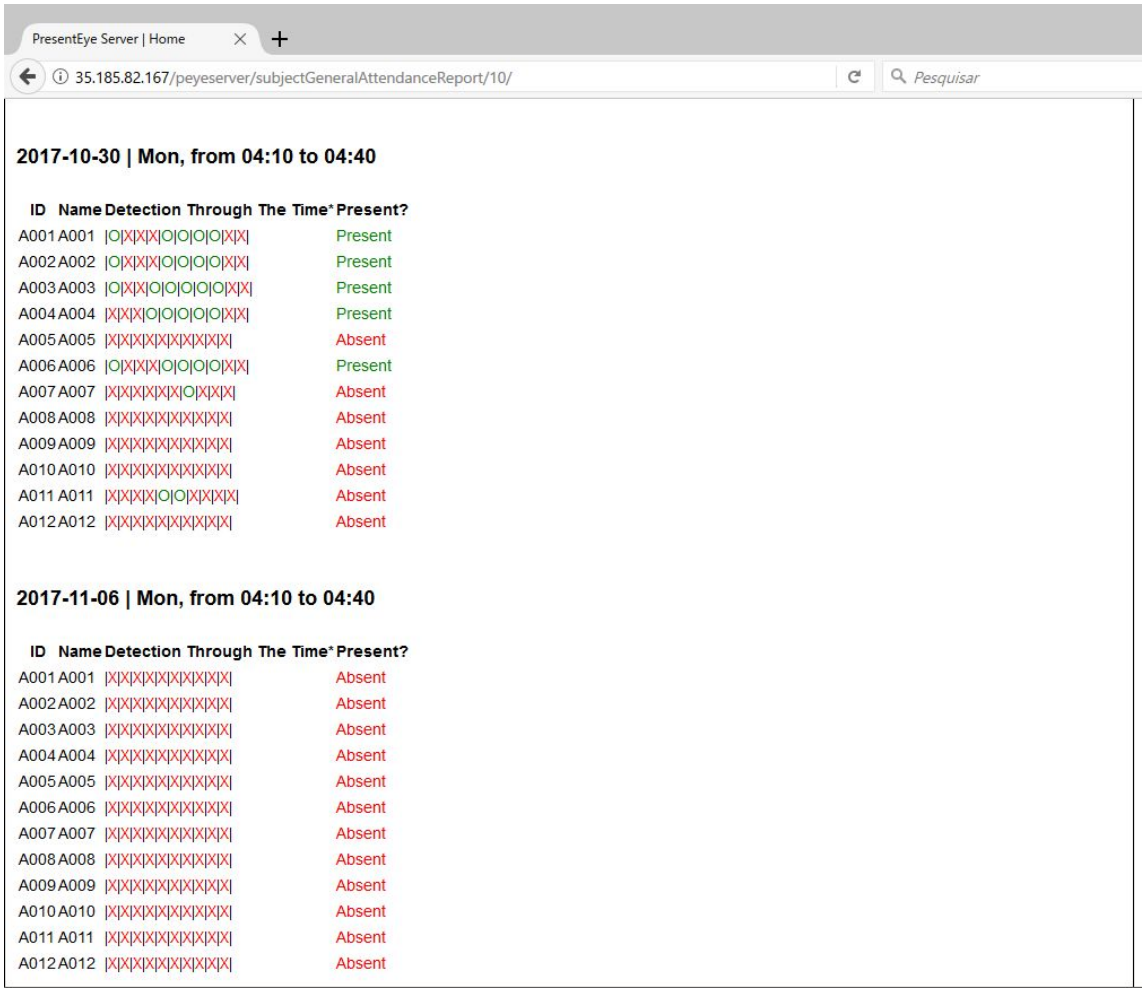


Figura 19 - Tela do relatório de presença

Optou-se por não considerar o aluno como presente com apenas uma detecção para reduzir o impacto causado por falsas detecções positivas. Por exemplo, mesmo que a técnica de reconhecimento facial tenha um percentual de sucesso próximo a 100%, ela não é perfeita, detectando algumas pessoas erroneamente em alguns momentos. Ao requerer que alguém seja detectado múltiplas vezes ao longo de uma aula, sendo pelo menos 4 delas em 4 diferentes intervalos que constituem 40% do tempo de uma classe (cada intervalo é equivalente a 10% do tempo de uma aula), reduz-se a probabilidade de falsos positivos fazerem com que alguém que faltou seja dado como presente.

3.4.4 Observações

Como o sistema foi desenvolvido com a ideia de prover um MVP, é importante citar que ele se encontra longe da perfeição, sendo propenso a apresentar problemas. Algumas observações relevantes a serem feitas são:

- Nas imagens dos arquivo de vídeo, não são adicionadas marcações (que facilitariam a identificação de quem foi detectado) durante ou após o reconhecimento, pois isso iria consumir recursos desnecessariamente;
- Os JSONs trafegados entre o PresentEye Tunnel Server e o PresentEye Server possuem chaves não utilizadas e chaves redundantes (resultantes de alterações durante o desenvolvimento), o que causa um pequeno, porém indesejado, aumento na quantidade de dados enviados e recebidos através da rede; e
- A interface da aplicação web é simples (é muito ruim dos pontos de vista de usabilidade, acessibilidade e estética), pois o objetvo do trabalho foi prover uma funcionalidade mínima para testar se a aplicação conseguiria atender a proposta deste trabalho (utilizar reconhecimento facial para registrar presença em sala de aula).

3.5 Considerações finais

Neste capítulo foram abordados os requisitos, o modelo de dados e a arquitetura do sistema para cada um de seus componentes: PresentEye Server, PresentEye Tunnel Server e PresentEye Tunnel Client. Também foram apresentadas as tecnologias utilizadas para seu desenvolvimento. No próximo capítulo, serão exibidos os testes efetuados com a técnica de reconhecimento facial utilizada e os testes do software desenvolvido.

4. Testes

Com o objetivo de verificar a viabilidade do uso de reconhecimento facial para o registro de presença, duas abordagens se fazem necessárias:

1. Analisar a performance da técnica de reconhecimento facial escolhida, buscando descobrir parâmetros ideias para seu funcionamento, e
2. Testar a aplicação devidamente calibrada e com dados reais, de modo a avaliar seu desempenho.

As próximas seções tratarão desses dois casos. Como ambiente de execução para esses testes, foi utilizada uma máquina virtual com as configurações apresentadas na Tabela 2, em funcionamento na *Google Cloud Platform*. No caso dos testes da seção 4.1, não foi feito uso de multithreading, nem de processamento em GPU. Ou seja, tudo foi executado, de modo sequencial, em um único núcleo de uma CPU, pois buscou-se prover uma base comum e acessível para todos os testes, visto que, no caso de multithreading, poderia haver interferência entre os testes (no acesso ao disco, uso do processador e etc) e, no caso de processamento em GPU, placas gráficas integradas não costumam suportar esse recurso. Na análise da seção 4.2, multithreading e múltiplos processos foram utilizados, seguindo oque está descrito no Capítulo 3, mas desconsiderando o processamento via GPU, pois a configuração do ambiente para suportá-lo não é simples (a biblioteca Dlib precisaria ser recompilada para oferecer suporte a isso, o que requer a instalação de todo um conjunto de dependências adicionais).

Tabela 2 - Configuração da máquina virtual utilizada nos testes

Processador	4 vCPUs (Intel Xeon E5 v3 / 2,3 GHz)
Memória RAM	15GB
Armazenamento (SSD)	30GB
Localização	EUA, costa leste (us-east1-c), SC
Latência a partir de Rio de Janeiro, Brasil	132 milissegundos (média)
Sistema Operacional	Ubuntu Linux 16.04 (ver. para servidores)

O vídeo utilizado como base para os testes possui resolução de 1280 x 720 pixels e duração de, aproximadamente, 29 minutos e 40 segundos, com um *framerate* de 2 quadros por segundo, totalizando 3559 quadros. Este foi gravado em um dos laboratórios de informática do CCET (Centro de Ciências Exatas e Tecnologia) da Universidade Federal do Estado do Rio de Janeiro (UNIRIO), durante uma aula real e com a colaboração de um grupo de estudantes voluntários.

No total, a gravação contou com a presença de 8 participantes, distribuídos em 3 fileiras horizontais, que receberam como identificação os códigos, A001, A002, A003, A004, A005, A006, A007 e A008, sendo o A001 o estudante mais à esquerda na primeira fileira, e o A008 o mais à direita na terceira fileira. O ambiente e a distribuição dos alunos podem ser vistos na Figura 20.

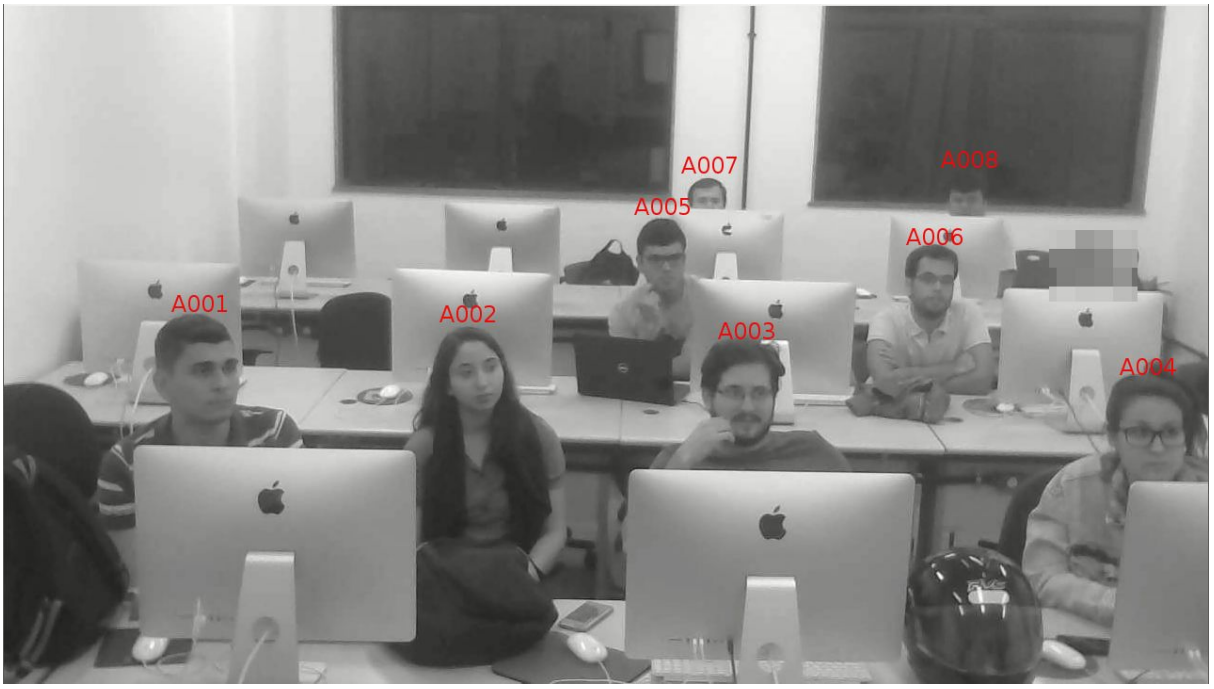


Figura 20 - Quadro do vídeo gravado em um laboratório de informática do CCET/UNIRIO, com os IDs associados a cada participante sendo exibidos sobre suas cabeças

Para abranger a possibilidade de alunos ausentes e permitir que falsos positivos fossem detectados, além das fotos dos participantes, foram adicionados à lista de alunos, além dos oito voluntários, quatro fotos de estudantes de faixa etária similar, obtidas do *dataset*

“faces94” [13], disponibilizado publicamente pelo Dr Libor Spacek, da Universidade de Essex. Esse dataset é de uso gratuito para propósitos de pesquisa, mas infelizmente, não permite a publicação das imagens em eventuais publicações. Assim, a seguir, são listados os dados das fotos dessa fonte que foram utilizados:

- phughe.4: jovem caucasiana do sexo feminino, com idade entre 18 e 25 anos e cabelo longo de cor castanho-clara;
- slbirc.8: jovem caucasiana do sexo feminino, com idade entre 20 e 30 anos e cabelo longo e loiro;
- 9338446.15: jovem caucasiano do sexo masculino, com idade entre 18 e 27 anos e cabelo curto de cor preta; e
- ambarw.17: jovem afro-americano do sexo masculino, com idade entre 18 e 25 anos e cabelo curto de cor preta.

Essas pessoas receberam, respectivamente, como identificação, A009, A010, A011 e A12.

Quanto aos não-participantes (apenas uma pessoa) e pesquisadores (o autor deste texto e sua orientadora) presentes no recinto, não foram considerados como alunos e tiveram suas imagens censuradas nos quadros de vídeo publicados neste trabalho. Eventuais reconhecimentos ocorridos em suas faces, por configurarem falsos positivos, foram tratados como tal.

4.1 Testes de Performance do Reconhecimento Facial

Ao lidar com reconhecimento facial, é possível se deparar com os problemas de: detecção de faces, precisão no reconhecimento e de uso de recursos (tempo de execução, consumo de memória, etc). Para os testes a seguir, que abordarão cada um desses casos, os parâmetros avaliados serão:

1. Técnicas de detecção de faces - Serão testadas a HOG e o uso de redes neurais convolucionais na tarefa de encontrar faces em uma imagem;

2. Número de *upsamples* - Quantas vezes uma imagem tem seu tamanho aumentado durante a tentativa de detectar faces (o que pode proporcionar a detecção de faces menores);
3. Módulo (vetorial) máximo para se considerar em um reconhecimento - Dados dois vetores de características faciais, qual seria a distância (norma) máxima entre eles para que sejam considerados pertencentes à mesma pessoa;
4. Tratamento de casos onde uma face é reconhecida como pertencente a múltiplas pessoas - Dada uma face, ao compará-la às faces de outras pessoas, duas ou mais têm distância inferior à distância máxima para se ter um reconhecimento.

4.1.1 Detecções de faces

Para a tarefa de detectar faces, a biblioteca *Face Recognition* provê duas possibilidades: HOG e Redes Neurais Convolucionais (RNC). Para compará-las, foram considerados dois critérios: número de faces detectadas, duração da execução e número de *upsamples*. Além disso, foi utilizada apenas uma amostra composta pelos primeiros 120 *frames* do vídeo.

Avaliando primeiro a diferença entre o número de faces detectadas, onde se pressupõe que RNC seja melhor, o HOG demonstrou um desempenho inferior ao longo dos *frames*. Nessa amostra, foram totalizadas 429 detecções para o HOG e 489 para o outro. Uma diferença de 60 detecções, ou 13.9%. Esse resultado pode ser observado (não muito facilmente, visto que é uma discrepância total de cerca de 14%) *frame* por *frame* na Figura 21, onde o uso de RNC proporcionou, em muitos dos quadros (especialmente entre o 15 e o 60), um maior número de faces encontradas.

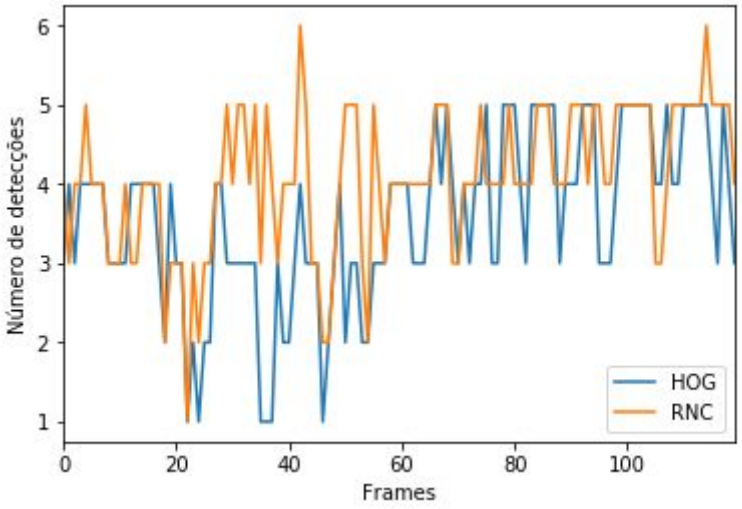


Figura 21 - Diferença entre o número de faces detectadas por *frame* através do uso de HOG (em azul) e RNC (em laranja)

Apesar de não-negligível, dada a natureza do problema que se tenta resolver (registro de presença), o que define se essa diferença é pouco relevante ou não é se o uso de RNC permite a detecção de faces em situações desfavoráveis (que aparecem parcialmente obstruídas ou que se encontram mais distantes). Para essa verificação, foi efetuada uma checagem manual dos *frames* após aplicadas ambas as técnicas e constatou-se que, no intervalo observado (120 primeiros frames do vídeo), com a aplicação do HOG, os alunos A005, A007 e A008 não tiveram seus rostos detectados sequer uma vez, enquanto que o uso de RNC proporcionou algumas poucas detecções para o A007 e o A008. O A005, por estar com o rosto aparecendo apenas parcialmente e não estar olhando para a frente, não foi detectado por nenhuma das técnicas.

Esses resultados demonstram que a aplicação de RNC parece ser mais favorável, porém, é preciso verificar primeiro um fator crucial: o tempo gasto para analisar uma imagem em busca de faces. Na Figura 22 essa comparação é feita e, de acordo com o que pode ser visto, RNC é muitas vezes mais lento. Enquanto o tempo médio que a HOG leva para processar uma imagem é de 2.09 segundos, RNC leva 80.98 segundos. Cerca de 40 vezes mais.

Apesar de o tempo de execução da RNC poder ser reduzido se a mesma for processada em uma GPU, não em CPU, a máquina virtual utilizada não oferecia tal possibilidade, logo, o uso de RNC passa a ser inviável para uso na aplicação desenvolvida,

pois a demora verificada não justifica o ganho obtido no número de detecções. Se fosse utilizada uma combinação de processamento em GPU e multiprocessamento, essa técnica poderia se tornar mais favorável.

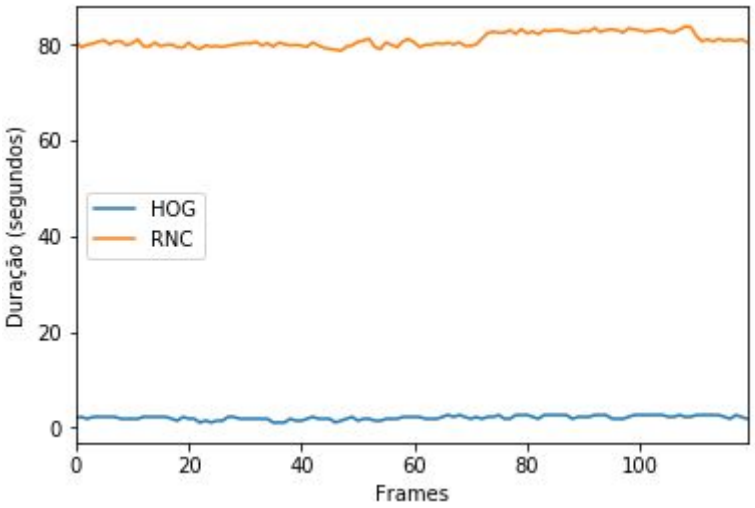


Figura 22 - Diferença entre HOG e RNC quanto ao tempo levado para processar quadros de 1280 x 720 pixels

Para finalizar, o último aspecto a ser avaliado no contexto da detecção de faces é o impacto do aumento no número de *upsamples*. Por padrão, a biblioteca "Face Recognition" efetua um *upsample*. Na Figura 23 é demonstrada a diferença de duração do processamento de um frame ao efetuar esse procedimento uma, duas e três vezes em conjunto com a técnica HOG.

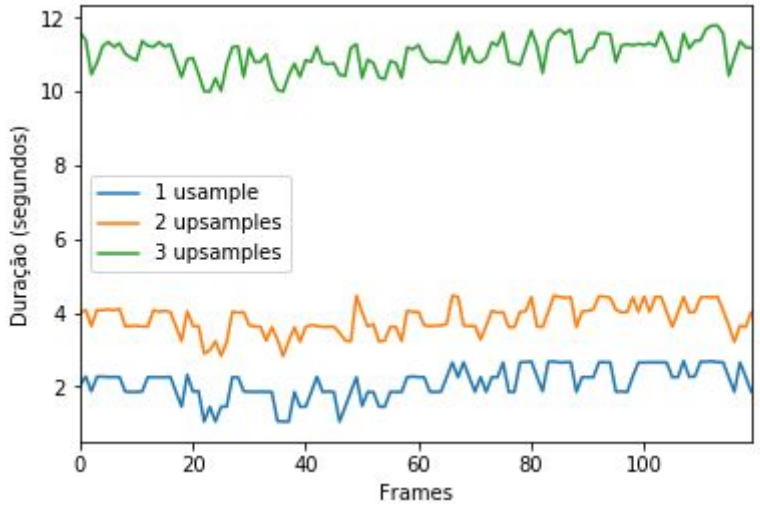


Figura 23 - Diferença no tempo de execução da técnica HOG quando efetuados 1, 2 e 3 *upsamples*

Para 1 *upsample*, o tempo médio é o que havia sido verificado em testes anteriores, 2.09 segundos. Porém, para 2 *upsamples*, esse tempo quase duplica, sendo de 3.82 segundos, e, para 3 *upsamples*, é cerca de 5 vezes maior, alcançando 10.99 segundos. Esses números já são suficientes para descartar a possibilidade de efetuar mais que dois *upsamples*.

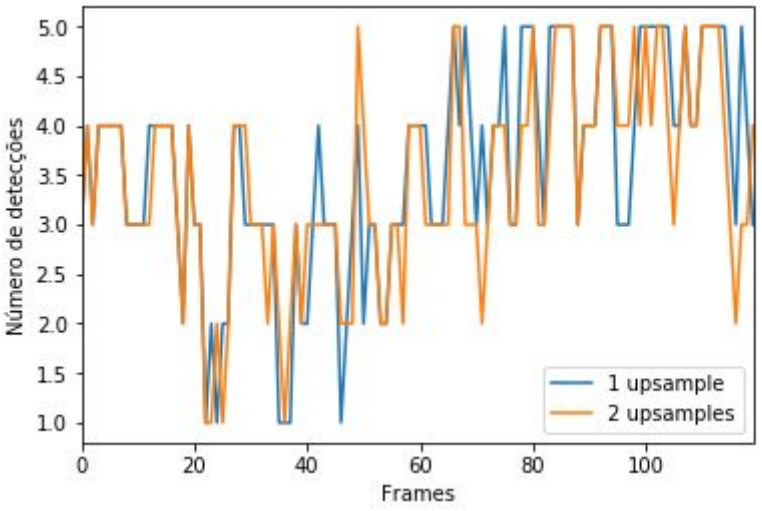


Figura 24 - Diferença no número de faces detectadas utilizando a técnica HOG quando efetuados 1 e 2 *upsamples*

Quanto a diferença no número de detecções para um e dois *upsamples*, pode ser observada na Figura 24. Visualmente, o desempenho em ambos os casos parece ser bastante similar. No entanto, se o número total de detecções na amostra for considerado, obteve-se mais resultados com um *upsample* (429) que com dois (415). Isso mostra que a abordagem padrão da biblioteca parece ser a mais apropriada.

4.1.2 Reconhecimento de faces

O reconhecimento facial consiste em, basicamente, prover uma imagem de uma face para uma RNC, efetuar a norma entre o vetor obtido como saída e vetores de faces conhecidas e, caso alguma distância obtida seja menor que um determinado limite de tolerância, declarar que uma face foi reconhecida. Porém, esse procedimento tem potencial para acabar em três situações problemáticas: uma face pode não ser reconhecida por causa de

uma tolerância baixa demais, uma face pode ser reconhecida como pertencente a mais de uma pessoa, e uma face pode ser reconhecida de maneira incorreta (em função do problema anterior ou não).

Para analisar o impacto da variação da tolerância, o reconhecimento facial foi executado em uma amostra igual a da seção 4.1.1, com 120 frames, e foi observado o número de faces reconhecidas para tolerâncias de 0.6 (valor padrão recomendado pela biblioteca utilizada) a 0.1. Segundo os resultados exibidos no gráfico da Figura 25, é possível verificar que, para valores inferiores a 0.4, nenhum reconhecimento ocorreu e que, para 0.4, as identificações são excessivamente escassas (26). Apenas tolerância de 0.5 ou 0.6 mostram desempenho aceitável (220 e 410 reconhecidos, respectivamente)..

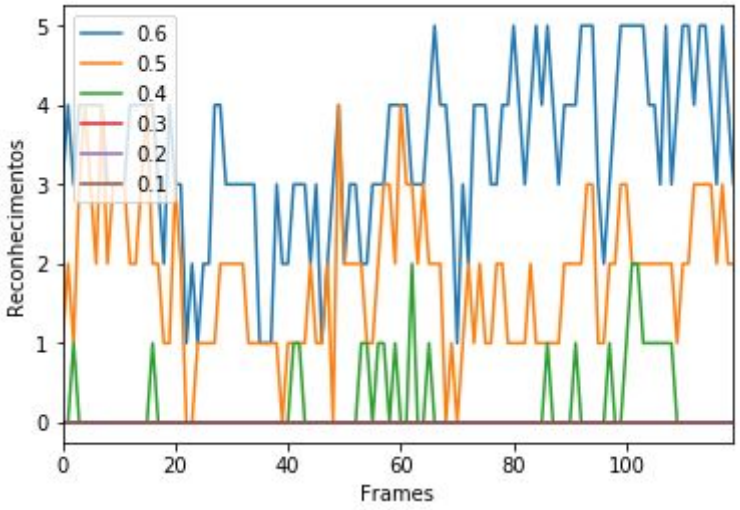


Figura 25 - Número de faces reconhecidas ao longo dos frames para tolerâncias de 0.1 a 0.6

Quanto ao problema de uma face ser reconhecida como pertencente a mais de uma pessoa, a abordagem natural para lidar com isso é considerar apenas o candidato cujo vetor de "características faciais" apresenta a menor norma em relação a imagem sendo verificada. Outra possibilidade é descartar completamente os casos onde isso ocorre. Porém, para avaliar o impacto dessas alternativas é preciso observar a diferença no número de reconhecimentos e a ocorrência de falsos positivos. Na Tabela 3, é possível verificar a variação nessas variáveis em uma amostra de 120 frames (primeiro 1 minuto de vídeo) quando se altera a tolerância (de 0.5 para 0.6) e a abordagem em relação ao reconhecimento de uma face como sendo

pertencente a mais de uma pessoa. A verificação de falsos positivos foi feita manualmente. Foi considerado também o tempo médio de processamento de uma imagem apenas para verificar se as abordagens apresentam alguma diferença nesse quesito.

Tabela 3 - Variação no número de falsos positivos em uma amostra de 120 quadros de acordo com a tolerância e a abordagem para o tratamento de reconhecimentos múltiplos para uma face

Abordagem	Tolerância	Reconhecimentos	Falsos positivos	Tempo médio (s)
Considerar apenas a pessoa mais similar	0.6	410	2	2,093
	0.5	220	0	2,096
Desconsiderar faces reconhecidas como sendo de mais de uma pessoa	0.6	327	1	2,108
	0.5	220	0	2,100

Segundo esses resultados, o desempenho na amostra foi plenamente satisfatório para ambos os níveis de tolerância. Porém, como os alunos A005, A007 e A008 não foram detectados nem mesmo uma única vez, não é possível, com base nessa quantidade limitada de dados, afirmar que norma máxima seria mais apropriada. Os alunos A009, A010, A011 e A012 (grupo de controle, adicionado para tornar evidente a ocorrência de identificações errôneas) também não foram detectados.

Quanto aos falsos positivos, ocorreram com o aluno A006, que foi detectado como A001, uma vez em ambas as abordagens (Figura 26), e como A005 uma vez na primeira. O tempo médio de execução de todos os passos para efetuar o reconhecimento (detecção de face + reconhecimento facial) não apresentou uma variação relevante.

Como última tentativa de decidir sobre qual valor de tolerância é mais adequado, efetuou-se a mesma avaliação da Tabela 3 para todos os 3559 frames do vídeo, porém, tendo em vista que não seria possível efetuar uma verificação de identificações incorretas manualmente com uma alto grau de confidência para essa quantidade de imagens (até porque essa verificação ocorreria um vez para cada caso de teste) essa checagem foi ignorada, passando a ser considerada para a observação de falsos positivos apenas a existência de detecção de alunos não presentes (de A009 a A012). O tempo de execução por frame também foi desconsiderado.

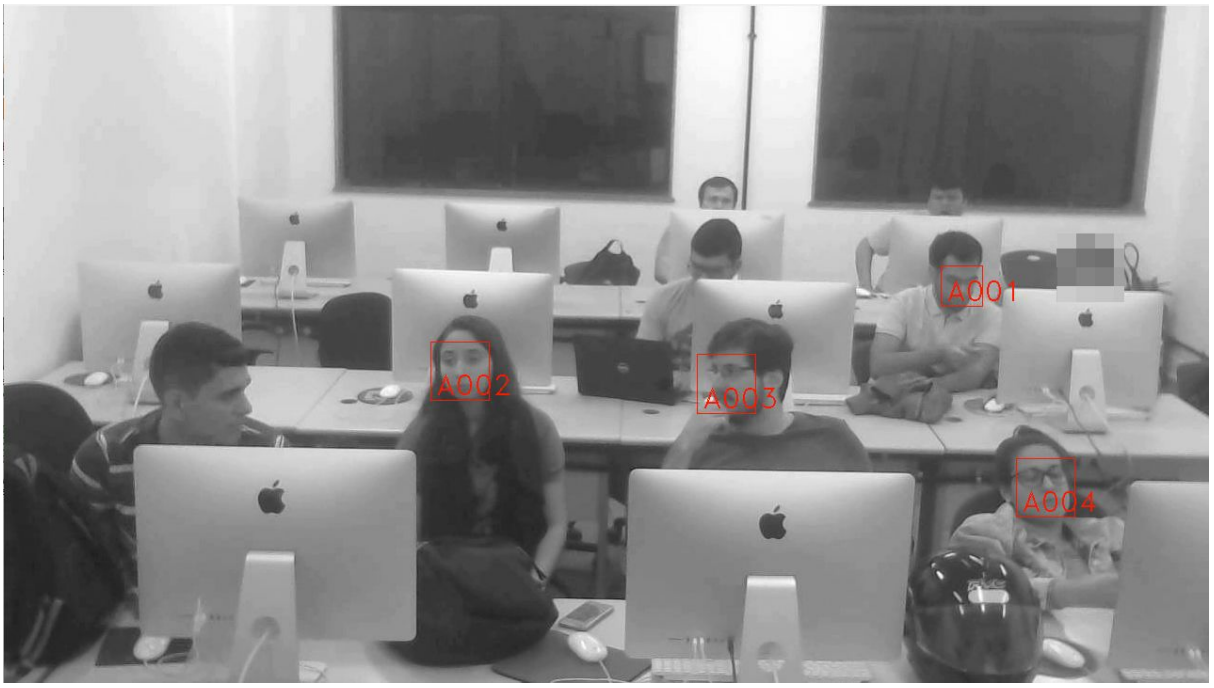


Figura 26 - Aluno A006 sendo identificado erroneamente como A001

Na Tabela 4, é apresentado o número de identificações para cada aluno, ao longo de quase 30 minutos de aula, ao se variar a abordagem para reconhecimentos múltiplos para uma face e a tolerância. De acordo com esses resultados, não é possível determinar os casos de falso positivo entre os alunos presentes, porém, é possível perceber que, para os casos de tolerância 0,6, dois alunos ausentes foram detectados algumas vezes. Por outro lado, caso a tolerância seja de 0,5, isso não ocorre, mas os estudantes A007 e A008 deixam de ser detectados.

Tabela 4 - Variação no número de falsos positivos em um conjunto de 3559 quadros de acordo com a tolerância e a abordagem para o tratamento de reconhecimentos múltiplos para uma face

Estudante	Considerar apenas a pessoa mais similar		Desconsiderar faces reconhecidas como sendo de mais de uma pessoa	
	Tolerância 0,6	Tolerância 0,5	Tolerância 0,6	Tolerância 0,5
A001	1122	657	1092	657
A002	2052	1544	1676	1544
A003	2380	1935	2019	1935
A004	1129	444	1065	444
A005	256	47	100	44
A006	2136	1491	458	1460
A007	51	0	40	0
A008	24	0	19	0
A009	0	0	0	0
A010	0	0	0	0
A011	1	0	1	0
A012	5	0	3	0

Esses resultados sugerem que a decisão final sobre a tolerância dependerá de qual a prioridade no registro da presença: evitar detectar alunos ausentes como presentes ou se certificar de detectar todos os que compareceram. Como deixar de registrar a presença para alguém que estava em uma aula pode ser danoso à situação dessa pessoa na matéria à qual essa classe pertence, neste trabalho, o valor 0,6 foi considerado mais apropriado.

Quanto à abordagem em relação às ocorrências onde uma face é reconhecida como pertencente a mais de uma pessoa, descartar completamente esses casos parece ter um resultado ligeiramente melhor. Por exemplo, no caso de A012, reduz-se em 2 o número de falsos positivos e A007 e A008 não deixam de ser detectados.

4.1.3 Parâmetros ideais

Com base nos resultados das seções anteriores, os parâmetros ideais para o PresentEye são os apresentados na Tabela 5. Esses dados permitem que todos os alunos presentes sejam detectados, mas, caso o ideal seja reduzir ao máximo os falsos positivos, reduzir a tolerância para 0.5 também pode ser pertinente, visto que a dificuldade em detectar faces na última fileira ocorre mais em função de monitores as bloqueando, não se tratando de um problema de "reconhecimento".

Tabela 5 - Parâmetros ideais para o funcionamento do PresentEye

Técnica de detecção de faces	HOG
Número de *upsamples*	1
Tolerância no reconhecimento	0,6 (ou 0,5)
Abordagem em relação múltiplos reconhecimento para uma face	Descartar reconhecimento obtido para a face detectada

4.2 Testes da Aplicação

Definida como um dos objetivos deste trabalho, a criação de um sistema capaz de efetuar o registro de presença por reconhecimento facial resultou no PresentEye, um conjunto de aplicações com funções definidas que colaboram para alcançar esse propósito. Para verificar o comportamento do sistema para sua função, foi montado um ambiente como demonstrado na Figura 13, com as definições da Tabela 2, mas sem câmeras conectadas ao Client, sendo os dados de vídeo capturados diretamente do arquivo de vídeo utilizado como base para os testes deste capítulo. Na seção 4.2.1 serão discutidos a metodologia e os resultados referentes ao registro de presença por reconhecimento facial, enquanto a 4.2.2 abordará a capacidade do sistema de lidar com um número crescente de câmeras.

4.2.1 Registro de Presença Através de Reconhecimento Facial

Usando como base o arquivo de vídeo utilizado na seção 4.1, o teste inicial de desempenho do PresentEye foi efetuado da seguinte maneira:

1. Modificou-se o PresentEye Tunnel Client para coletar *frames* de arquivos de vídeo, onde, a cada 0,5 segundos (aproximadamente), um frame é lido e disponibilizado para as threads (duas, já que foram simuladas duas câmeras) responsáveis pelo envio da imagem ao servidor (essas threads também são executadas a cada 0,5 segundos);
2. Modificou-se o PresentEye Tunnel Server para contar e exibir o número de frames efetivamente recebidos (aceitos e utilizados) durante sua execução;
3. Cadastrou-se uma turma de "Computer Literacy" no PresentEye Server, com uma aula de 30 minutos e os alunos A001 ao A012;
4. Executou-se o sistema, estando o Tunnel Client em uma máquina virtual (Ubuntu Linux 14.04, com conexão banda larga com 35 Mbps de *downstream* e 20 Mbps *upstream*) em um computador no Rio de Janeiro (Brasil) e o Tunnel Server e o PresentEye Server em uma máquina virtual no *Google Cloud Platform*, na costa leste dos EUA (na Carolina do Sul), cada um em uma máquina virtual (conforme a Tabela 2).

Para essas condições e com os parâmetros ideias da seção 4.1.3 (HOG, tolerância de 0.6 e ignorar reconhecimento múltiplo para uma face), o desempenho obtido na transmissão dos quadros, descrito na Tabela 6, mostra que, dos 3559 *frames* lidos do arquivo de vídeo, cada câmera enviou ao Tunnel Server apenas pouco mais de 260 quadros ao longo de uma aula de 30 minutos. Além disso, para cada câmera, o tempo médio para a execução do procedimento de entrega de um *frame* foi superior a 6.5 segundos.

Se for considerada a latência média de 132 milissegundos (Tabela 2) entre os componentes Client e Server do PresentEye Tunnel, e o fato de haver uma espera de 0,5 segundos entre a entrega dos quadros de uma câmera, cerca de 0,63 segundos (0,5 + 0,13 segundos), dessa demora são facilmente explicáveis, sendo o restante relacionado ao tempo de transferência de uma imagem. Porém, se for levado em conta que cada *frame* possui um tamanho médio inferior a 50 KB e que as velocidades das conexões das máquinas virtuais

envolvidas é mais que suficiente para lidar com o stream requerido para transmití-los (menos de 500 Kbps por câmera), há uma indicação de que ou a rede (local, internet ou a do Google Cloud), ou o uso de múltiplas threads (sendo executadas em um único núcleo do processador) pela aplicação, ou o processador utilizado pela máquina virtual, podem estar influenciando negativamente no desempenho do Tunnel Server.

Tabela 6 - Desempenho no envio de um quadro para o Tunnel Server para duas câmeras transmitindo simultaneamente

Duração da aula	30 min. (1800 segundos)
Número de câmeras	2
Frames enviados com sucesso	Câmera 0: 270 Câmera 1: 262
Média de tempo de envio por câmera	Câmera 0: 6,67 segundos Câmera 1: 6,87 segundos

Para verificar isso, o mesmo teste foi realizado em situações com uma e três câmeras e, segundo os resultados obtidos (exibidos na Tabela 7, junto ao resultado já provido pela Tabela 6), vê-se que, para cada câmera adicionada, aumenta-se em pouco mais de 3 segundos o tempo médio de envio de cada *frame*.

Tabela 7 - Desempenho na entrega de um quadro para o Tunnel Server para variados números de câmeras enviando simultaneamente

N.o de câmeras	1	2	3
Duração da aula	10 minutos	30 minutos	10 minutos
Frames enviados com sucesso	Câmera 0: 167	Câmera 0: 270 Câmera 1: 262	Câmera 0: 55 Câmera 1: 59 Câmera 2: 60
Média de tempo de envio por câmera	Câmera 0: 3,59 s	Câmera 0: 6,67 s Câmera 1: 6,87 s	Câmera 0: 10,91 s Câmera 1: 10,17 s Câmera 2: 10 s

Para uma verificação mais minuciosa para checar se é um problema da aplicação ou da infraestrutura utilizada, executou-se mais um teste, porém, desta vez, o Tunnel Client e o Tunnel Server funcionaram em uma mesma máquina virtual (com 4 núcleos e 8 GB de RAM) em um computador doméstico (com um processador Intel Core i7 4720HQ[22] e 16GB de RAM). Com base nos resultados, disponibilizados na Tabela 8, é possível descartar que o PresentEye Tunnel seja o causador de tamanha lentidão. Também é possível especular que a má performance obtida em outros testes seja em função de os recursos na nuvem serem compartilhados.

Tabela 8 - Desempenho no envio de um quadro para o Tunnel Server para duas câmeras transmitindo simultaneamente

Duração da aula	10 min. (600 segundos)
Número de câmeras	2
Frames enviados com sucesso	Câmera 0: 752 Câmera 1: 745
Média de tempo de envio por câmera	Câmera 0: 0,80 segundos Câmera 1: 0,81 segundos

Em relação ao comparecimento, foram considerados presentes os alunos que constavam em 4 (40%) ou mais intervalos, pois, ao analisar todos os *frames* do vídeo utilizado como entrada para o sistema (resultados na Figura 27, onde a verificação foi feita fora do PresentEye), um dos estudantes do grupo de controle (que não estava presente) foi reconhecido em 3. Essa não é uma referência precisa pois se baseia em apenas uma amostra (um vídeo de uma aula), mas, por ser capaz de desconsiderar os falsos positivos do grupo de controle, passou a ser a considerada como válida.

De volta ao caso onde o Tunnel Server é executado na nuvem, cujos resultados são apresentados na Tabela 6, seu relatório de presença é apresentado na Figura 28. Nele é possível perceber que, apesar do baixo número de quadros recebidos pelo Tunnel Server (532 no total), os estudantes da primeira fileira (A001 a A004) e o aluno A006 foram detectados

[22] "Intel® Core™ i7-4720HQ" https://ark.intel.com/pt-br/products/78934/Intel-Core-i7-4720HQ-Processor-6M-Cache-up-to-3_60-GHz. Acessado em 16 nov. 2017.

em quase todos os intervalos de tempo, enquanto o A005, o A007 e o A008 foram identificados em 3 ou menos. Além disso, também fica evidente a ocorrência de falsos positivos, visto que aconteceram detecções para o estudante A011 no início e próximo ao fim da aula.

A001 OOOOOOOOOO

A002 OOOOOOOOOO

A003 OOOOOOOOOO

A004 OOOOOOOOOO

A005 OOOOOOOXOO

A006 OOOOOOOOOO

A007 XOOOXXXOXO

A008 XOOOOOOOXO

A009 XXXXXXXXXX

A010 XXXXXXXXXX

A011 XXOXXXXXXX

A012 OXXXOXXOXX

Figura 27 - Distribuição dos reconhecimentos dos estudantes ao longo de dez intervalos de tempo de igual tamanho

2017-11-15 | Wed, from 02:05 to 02:35

ID	Name	Detection Through The Time*	Present?
A001	A001	|O|O|O|O|O|O|O|O|O|O|	Present
A002	A002	|O|O|O|O|O|O|O|O|O|O|	Present
A003	A003	|O|O|O|O|O|O|O|O|O|O|	Present
A004	A004	|O|O|O|O|O|O|O|O|X|O|	Present
A005	A005	|X|X|O|X|X|X|X|X|O|O|	Absent
A006	A006	|O|O|O|O|O|O|O|O|O|O|	Present
A007	A007	|X|X|X|X|X|X|X|X|X|O|	Absent
A008	A008	|X|O|X|X|X|X|X|X|X|X|	Absent
A009	A009	|X|X|X|X|X|X|X|X|X|X|	Absent
A010	A010	|X|X|X|X|X|X|X|X|X|X|	Absent
A011	A011	|O|X|X|X|X|X|X|X|X|O|	Absent
A012	A012	|X|X|X|X|X|X|X|X|X|X|	Absent

Figura 28 - Resultado do teste com duas câmeras, seguindo os parâmetros da seção 4.1.3 e a arquitetura da Figura 13

Outro fato importante a ser citado sobre os testes baseados no vídeo da seção 4.1, é que os frames enviados por ambas as "câmeras" podem ser iguais, caso o Tunnel Client tente obtê-los num intervalo de tempo muito próximo (inferior a 0.5 segundos). Mesmo assim, como nem sempre essa requisição por frames ocorre num intervalo tão curto, é possível ocorrer também o envio de frames diferentes em boa parte do tempo.

Para finalizar, foi feito um outro teste para duas câmeras, mas, desta vez, empregando uma tolerância de 0,5, mais restrita que os 0,6 recomendados na seção 4.1.3, para a diferença entre vetores de *encoding* facial. Assim como na seção 4.1.2, essa alteração foi capaz de evitar a ocorrência de detecções para os estudantes do grupo de controle, mas também resultou em nenhuma identificação para os alunos A007 e A008. O relatório de presença resultante pode ser visto na Figura 29.

2017-11-15 | Wed, from 03:20 to 03:50

ID	Name	Detection Through The Time*	Present?
A001	A001	|O|O|O|O|O|X|O|O|O|O|	Present
A002	A002	|O|O|O|O|O|O|O|O|O|O|	Present
A003	A003	|O|O|O|O|O|O|O|O|O|O|	Present
A004	A004	|O|O|O|O|O|O|O|O|X|O|	Present
A005	A005	|X|X|X|X|X|X|X|X|O|O|	Absent
A006	A006	|O|O|O|O|O|O|O|O|O|O|	Present
A007	A007	|X|X|X|X|X|X|X|X|X|X|	Absent
A008	A008	|X|X|X|X|X|X|X|X|X|X|	Absent
A009	A009	|X|X|X|X|X|X|X|X|X|X|	Absent
A010	A010	|X|X|X|X|X|X|X|X|X|X|	Absent
A011	A011	|X|X|X|X|X|X|X|X|X|X|	Absent
A012	A012	|X|X|X|X|X|X|X|X|X|X|	Absent

Figura 29 - Resultado do teste com duas câmeras, seguindo os parâmetros da seção 4.1.3 e a arquitetura da Figura 13, mas utilizando tolerância de 0,5

4.2.2 Escalabilidade

Segundo André Bondi[14], escalabilidade é a habilidade de um sistema para acomodar ou processar um volume crescente de elementos e trabalho de maneira elegante, e/ou ser susceptível ao aumento. No caso do PresentEye, essa extensão se caracteriza pela possibilidade de atender a um número variável de câmeras, o que permitiria ser utilizado tanto em uma simples sala de aula, como em toda uma universidade.

Para alcançar esse objetivo, assim como explicitado no Capítulo 3, o sistema foi dividido em duas aplicações, PresentEye Server e PresentEye Tunnel, e este último foi também separado em duas outras: Client e Server. A razão para essa abordagem é fazer com que cada parte se preocupe apenas com um grupo limitado de tarefas. São elas:

- PresentEye Server: armazenar e administrar os dados;
- PresentEye Tunnel Client: concentrar a conexão de câmeras próximas e enviar os dados capturados para um Tunnel Server; e
- PresentEye Tunnel Server: receber o fluxo de dados de vídeo, armazená-lo e executar o reconhecimento facial sobre ele.

Essa fragmentação permite que sistema seja escalável no caso da necessidade de adição de novas câmeras, pois, ao invés de replicar toda a aplicação, só haverá a necessidade de adicionar novos Tunnel Clients e Servers, cuja operação, configuração e manutenção é mais simples que a do PresentEye Server. Por exemplo, se, em um prédio de 2 andares, o primeiro têm suas câmeras operando com o PresentEye, ao expandir a cobertura para o segundo piso bastará instalar o Tunnel Client no dispositivo que se conectará às câmeras e criar novos Tunnel Servers conforme necessário. Nesse mesmo contexto, se o Server ficar em um ambiente de computação em nuvem, adicionar um novo se resumirá a criar uma nova máquina virtual com base em uma *snapshot* de um disco virtual (uma imagem de um estado do disco) com o PresentEye Tunnel já instalado, inicializá-la, alterar algumas linhas no arquivo de configurações da aplicação (IP do PresentEye Server e dados de autenticação próprios) e iniciar o script do software. Isso tudo pode ser feito em cerca de 5 minutos e, caso a demanda se reduza, o usuário pode simplesmente excluir a VM (*Virtual Machine*) desnecessária.

Entretanto, quanto ao PresentEye Server, a expansão é um pouco mais complexa. A versão atual do sistema não prevê essa possibilidade, porém, ela poderia ser alcançada através da migração para uma solução de banco de dados que suporte um uso mais intensivo, que permita a replicação do webserver.

Para testar a escalabilidade do PresentEye, focou-se apenas em verificar o comportamento do sistema após a adição de novos componentes PresentEye Tunnel. Esse teste ocorreu do seguinte modo:

- No *Google Cloud Platform*, criou-se uma máquina virtual para executar o PresentEye Server e duas para executar o PresentEye Tunnel Server;
- Para evitar ultrapassar o limite de 8 núcleos do Google Cloud para contas de teste todas as VMs na nuvem foram configuradas com 2 núcleos virtuais e 7.5 GB de memória RAM;
- Instalou-se o PresentEye Tunnel em uma máquina virtual (com 4 núcleos e 8 GB de RAM) em um computador doméstico, que será responsável por executar um Tunnel Server e todos os Tunnel Clients;
- Cadastrou-se uma aula de 10 minutos no PresentEye Server;
- Foram inicializados todos os Tunnel Servers (que se conectaram ao PresentEye Server;
- Conectou-se à cada Tunnel Server, um Tunnel Client com duas pseudo câmeras (Tunnel Clients lendo o arquivo de vídeo da seção 4.1); e
- Observou se ocorre alguma anomalia na execução dos componentes do PresentEye durante o registro de presença para a aula cadastrada.

O ambiente resultante pode ser melhor visualizado através da Figura 30.

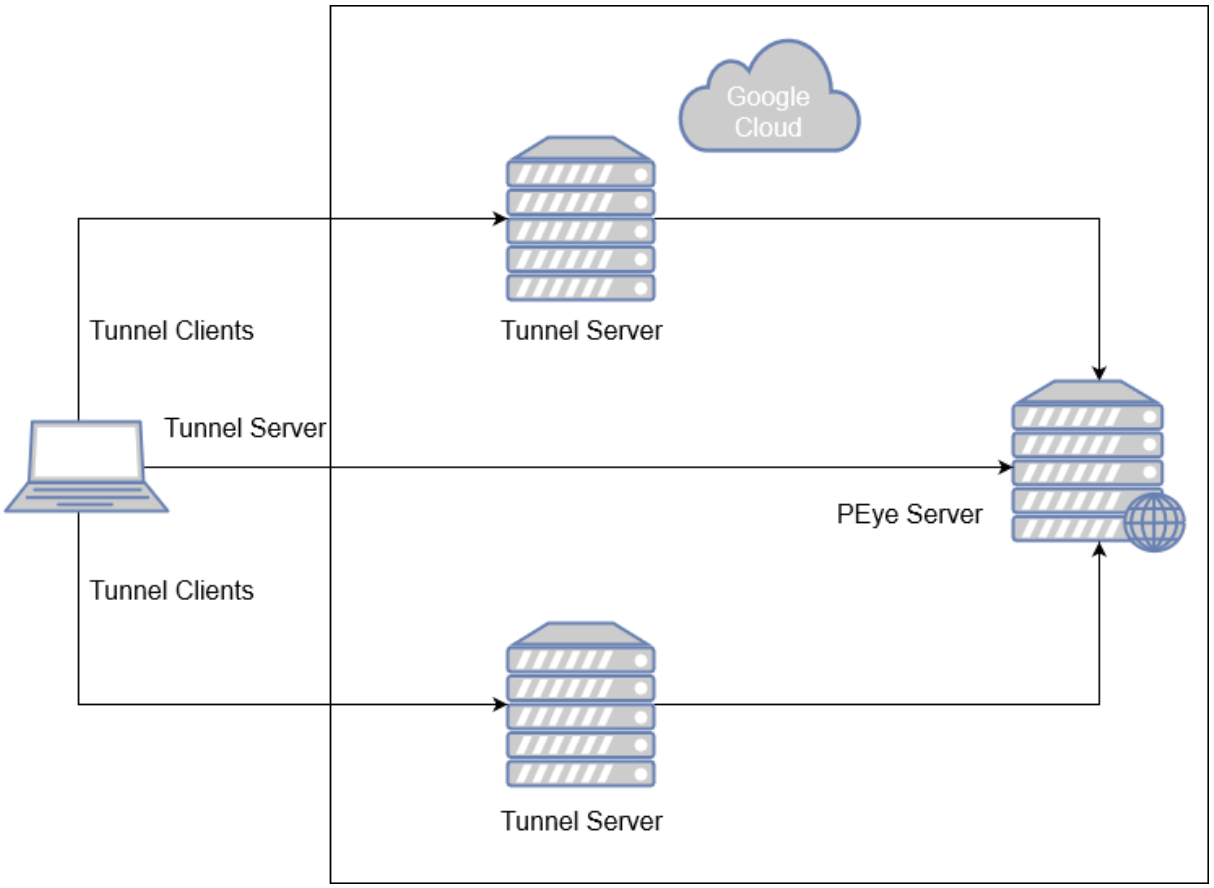


Figura 30 - Ambiente utilizado para testar o aumento no número de Tunnel Servers utilizados

Na Tabela 9, é possível verificar os resultados da execução do teste. Como pode ser visto, o desempenho dos Tunnel Servers em funcionamento na nuvem (1 e 2) foi pior que na seção 4.2.1, devido ao fato de que, nos testes daquela seção, as VMs possuíam 4 núcleos virtuais ao seu dispor, enquanto as empregadas nos testes reportados na presente seção possuíam apenas 2. O Server rodando em um computador doméstico (3), obteve o melhor desempenho.

Quanto ao impacto do aumento de Tunnel Servers no PresentEye Server, nada foi percebido. A aplicação web continuou a responder normalmente aos *requests* recebidos.

Tabela 9 - Resultados dos testes para reconhecimento facial de alunos

Tunnel Server	1	2	3
Duração da aula	10 minutos	10 minutos	10 minutos
Frames enviados com sucesso	Câmera 0: 32 Câmera 1: 58	Câmera 0: 53 Câmera 1: 35	Câmera 0: 674 Câmera 1: 677
Média de tempo de envio de quadro por câmera	Câmera 0: 18,75 s Câmera 1: 10,34 s	Câmera 0: 11,32 s Câmera 1: 17,14 s	Câmera 0: 0,89 s Câmera 1: 0,88 s

4.3 Considerações finais

Neste capítulo foram executados os testes de reconhecimento facial e de aplicação, onde o primeiro foi capaz de servir de base para a decisão sobre quais parâmetros são mais apropriados para a execução do PresentEye. Além disso, a questão da escalabilidade do sistema também foi abordada, apontando que ele é capaz de se expandir, mas sem determinar os limites. No próximo capítulo, serão abordadas as conclusões alcançadas com este trabalho, as limitações enfrentadas e os possíveis trabalhos futuros.

5. Conclusão

Neste trabalho, buscou-se, através do uso de reconhecimento facial, reduzir a ocorrência de erros durante o registro de presença em sala de aula e, para isso, foi desenvolvido um sistema composto de duas aplicações, PresentEye Server e PresentEye Tunnel, para lidar com partes específicas do problema. Com essa divisão, procurou-se também prover certo grau de escalabilidade ao software, permitindo que ele atendesse a um número crescente de câmeras.

Foram também efetuados testes para verificar se a solução atende aos objetivos propostos e, frente aos resultados obtidos, foi possível concluir que, apesar de imperfeito, o registro automático de presença por reconhecimento facial através de câmeras instaladas em salas de aula é possível e, até certo ponto, eficaz. Porém, em ambientes onde os dispositivos de captura não possuem uma clara visão das faces dos estudantes, é necessário fazer uma escolha: priorizar a detecção de quem está no recinto ou evitar a identificação de pessoas ausentes como presentes.

Essa decisão é indispensável, pois, para reconhecer rostos mais distantes da câmera (mais propensos a ruído visual, como distorções causadas por baixa resolução de imagem e etc) ou parcialmente obstruídos, pode ser necessário aumentar a distância máxima tolerável no cálculo da norma na hora de comparar os vetores de *encodings* faciais. Isso, infelizmente, tem um custo: a propensão à ocorrência de falsos positivos se amplia. Reduzir essa tolerância pode tornar os reconhecimentos mais precisos, mas também pode fazer com que pessoas presentes não sejam detectadas, caso sua visibilidade esteja prejudicada.

No caso da solução desenvolvida, optou-se pelo aumento da tolerância, o que causou um crescimento no número de falsos.positivos. A abordagem tomada para tentar amenizar isso (requerer que alguém seja detectado ao longo de múltiplos intervalos de tempo) acabou por fazer com que algumas pessoas legitimamente presentes acabassem consideradas ausentes por terem sido identificadas poucas vezes, gerando uma situação próxima a de se utilizar uma tolerância mais baixa.

Quanto ao desempenho, o PresentEye se mostrou lento no tráfego dos quadros do PresentEye Tunnel Client ao Tunnel Server quando este último foi executado em um serviço de computação em nuvem, mas consegue efetuar o reconhecimento facial de maneira

relativamente eficiente, ao ser capaz de utilizar múltiplas threads da CPU para essa função. A comunicação entre o Tunnel Server e o PresentEye Server, por sua vez, funcionou dentro do esperado e permitiu constatar que o sistema consegue atender a um número crescente de câmeras, desde que a aplicação web não seja sobrecarregada.

Por fim, apesar de ter alcançado os objetivos deste trabalho, o sistema desenvolvido se encontra imaturo, visto que ainda se trata de um protótipo e tem diversos pontos a serem aprimorados.

5.1 Comparativo entre soluções

As soluções apresentadas no Capítulo 2, apesar de poderem ser utilizadas para registrar a presença de pessoas em ambientes, possuem focos distintos. Umas, por exemplo, podem ser mais apropriadas para controle de acesso, como os dispositivos da seção 2.1.3 (sistemas dedicados de controle de presença por biometria), enquanto outra é focada especificamente no controle de horas trabalhadas por funcionários (Fareclock).

Na Tabela 10, é feito um comparativo entre as funcionalidades desses sistemas e as do software desenvolvido neste trabalho, visando mostrar como ele se posiciona em relação a soluções semelhantes. Para a categoria da seção 2.1.3, será utilizado como representante o Face T71F da NavKar Systems.

Quanto ao significado de cada item comparativo:

- Número de câmeras suportadas: número máximo de câmeras suportadas pelo sistema;
- Reconhecimento de múltiplas faces em uma imagem: em uma imagem, mais de uma face podem ser reconhecidas por vez;
- Máximo de faces de cadastradas: número máximo de faces de referência que podem ser cadastradas no sistema;
- Escalável: o sistema pode ser "estendido" para atender a um aumento de demanda em sua utilização (neste tipo de sistema, se o sistema pode atender a um número crescente de câmeras em utilização e faces e usuários cadastrados);
- Funciona independente de rede de computadores: o sistema pode ser executado em um único dispositivo, sem depender de conexão com outros serviços ou componentes rodando em outras máquinas, seja na rede local, seja na internet;

- Dispositivo dedicado: o sistema é ou pode ser composto de um dispositivo específico, dedicado a sua execução; e
- Oferece outros tipos de biometria: é oferecida também a opção de reconhecimento por algum outro tipo de biometria (como impressão digital, íris, etc).

Tabela 10 - Comparação entre as funcionalidades de alguns sistemas de reconhecimento facial

	Churchix	Fareclock	Face T71F	PresentEye
Número de Câmeras suportadas	Desconhecido (ao menos 1)	1 por smartphone / aplicativo	1	1+ (a depender da infra. utilizada)
Reconhecimento de múltiplas faces em uma imagem	Sim	Não	Não	Sim
Máximo de faces de cadastradas	Desconhecido (possivelmente ilimitado)	Desconhecido	1000	Ilimitado
Escalável	Desconhecido	Desconhecido	Não	Sim (de modo manual)
Funciona independente de rede de computadores	Sim	Não (requer conexão com a internet, para o app acessar o servidor do serviço)	Sim	Sim (é possível executar todos os componentes em um único computador
Dispositivo dedicado	Não	Não	Sim	Não
Oferece outros tipos de biometria	Não	Não	Sim	Não

Com base nos resultados da Tabela 10, a Tabela 11 mostra o quanto essas aplicações são apropriadas para as funções de controle de acesso, registro de presença (em um ou múltiplos ambientes) e registro de ponto. O significado dos símbolos utilizados é:

- "+": altamente apropriado;
- "-": pouco apropriado;
- "x": não apropriado.

Tabela 11 - Comparativo entre entre os sistemas estudados neste trabalho quanto a sua adequação a algumas situações

	Churchix	Fareclock	Face T71F	PresentEye
Controle de Acesso	**X**	**X**	+	**X**
Registro de ponto	-	+	+	**X**
Registro de presença em um ambiente (sala de aula etc)	+	-	-	+

Diante dos resultados dessas duas tabelas, é possível concluir que dentre os softwares comparados, o PresentEye é o único garantidamente escalável, sendo capaz de lidar com um número crescente de câmeras. Porém, é importante citar que a avaliação das outras soluções foi baseada em informações públicas, presentes em seus *websites*, logo, existe a possibilidade de suas capacidades estarem além do exposto nesta seção.

5.2 Limitações deste trabalho

Vale ressaltar as seguintes limitações, que influenciaram diretamente nos resultados deste trabalho:

- Todos os testes foram realizados utilizando apenas uma gravação em vídeo com duração de pouco menos de 30 minutos, o que faz com que os resultados obtidos não possam ser declarados como muito abrangentes; e
- O teste relacionado à escalabilidade não envolveu teste de estresse da aplicação web, logo, não é possível afirmar quantas câmeras e Tunnel Servers são suportados.

5.3 Trabalhos futuros

As possibilidades de trabalhos futuros não apenas envolvem melhorias no desempenho, como também uma análise mais aprofundada e prolongada do uso de reconhecimento facial para registro de presença em sala de aula (como qual a tolerância mais apropriada de maneira geral, etc). No futuro, novas verificações podem ser feitas com o sistema sendo executado com dados obtidos em tempo real.

Outra possibilidade é abordar, em separado, o problema da escalabilidade. Neste texto, limitou-se a verificar se as aplicações que compõem o PresentEye conseguiriam atender a um número variado de câmeras, porém, em outros trabalhos, pontos como automatização da criação e remoção de Tunnel Servers, replicação do PresentEye Server e testes de estresse de componentes podem ser visitados.

Referências Bibliográficas

[1] Choudhury, T. (2000) “History of Face Recognition”, http://vismod.media.mit.edu /tech-reports/TR-516/node7.html, Acessado em 16 de Novembro de 2017.

[2] Liu, Y., Ng W. and Liu C. (2009) “A Comparison of Different Face Recognition Algorithms”, National Taiwan University.

[3] Geitgey, A. (2016) “Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning”, https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78, Acessado em 16 de Novembro de 2017.

[4] Felzenszwalb, P., Girshick, R., McAllester, D. and Ramanan, D. (2010) “Object Detection with Discriminatively Trained Part Based Models”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 9.

[5] Dalal, N. and Triggs, B. (2005) “Histograms of Oriented Gradients for Human Detection”, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1.

[6] Chen, Q., Meng, X., Li, W., Fu, X., Deng, X. and Wang J. (2017) “A Multi-Scale Fusion Convolutional Neural Network for Face Detection”, In *2017 IEEE International Conference on Systems, Man, and Cybernetics*.

[7] King, D. E. (2015) “Max-Margin Object Detection”, https://arxiv.org/abs/1502.00046, Acessado em 16 de Novembro de 2017.

[8] Kazemi, V. and Sullivan, J. (2014) “One Millisecond Face Alignment with an Ensemble of Regression Trees”, In *2014 IEEE Conference on Computer Vision and Pattern Recognition*.

[9] Hubel, D. and Wiesel, T. (1968) “Receptive fields and functional architecture of monkey striate cortex”, Journal of Physiology (London), Vol. 195, p. 215–243.

[10] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). “Gradient-based learning applied to document recognition”, Proceedings of the IEEE, Vol. 86 (11), p. 2278–2324.

[11] King, D. (2017) “High Quality Face Recognition with Deep Metric Learning”, http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html, Acessado em 16 de Novembro de 2017.

[12] Huang, G. B., Ramesh, M., Berg, T. and Learned-Miller, E. (2007) “Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments”, University of Massachusetts, Amherst, Technical Report 07-49.

[13] Spacek, L. (2007) “Collection of Facial Images: Faces94”, http://cswww.essex.ac.uk /mv/allfaces/faces94.html, Acessado em 16 de Novembro de 2017.

[14] Bondi, A. B. (2000) “Characteristics of Scalability and Their Impact on Performance”, In *Proceedings of the 2nd international workshop on Software and performance*, p. 195-203