

**UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"**  
FACULDADE DE CIÊNCIAS - CAMPUS BAURU  
DEPARTAMENTO DE COMPUTAÇÃO  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL TATSUKI TAKAGI

**UTILIZAÇÃO DE REDES NEURAIS PARA A CRIAÇÃO DE UM  
GERADOR E EDITOR DE FACES**

BAURU  
Novembro/2019

RAFAEL TATSUKI TAKAGI

**UTILIZAÇÃO DE REDES NEURAIS PARA A CRIAÇÃO DE UM  
GERADOR E EDITOR DE FACES**

Trabalho de Conclusão do Curso de Bacharelado  
em Ciência da Computação da Universidade  
Estadual Paulista “Júlio de Mesquita Filho”,  
Faculdade de Ciências, Campus Bauru.  
Orientador: Prof. Associado Aparecido Nilceu  
Marana

BAURU  
Novembro/2019

Rafael Tatsuki Takagi      Utilização de Redes Neurais para a criação de um gerador e editor de faces/ Rafael Tatsuki Takagi. – Bauru, Novembro/2019-  
**42** p. : il. (algumas color.) ; 30 cm.  
Orientador: Prof. Associado Aparecido Nilceu Marana  
Trabalho de Conclusão de Curso – Universidade Estadual Paulista “Júlio de Mesquita Filho”  
Faculdade de Ciências  
Ciência da Computação, Novembro/2019.  
1. Ciência da Computação 2. Redes Neurais 3. Encoder 4. Decoder 5. Python

Rafael Tatsuki Takagi

## **Utilização de Redes Neurais para a criação de um gerador e editor de faces**

Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

---

**Prof. Associado Aparecido Nilceu Marana**

Orientador

Universidade Estadual Paulista "Júlio de Mesquita Filho"  
Faculdade de Ciências  
Departamento de Ciência da Computação

---

**Profa. Dra. Simone das Graças Domingues Prado**

Universidade Estadual Paulista "Júlio de Mesquita Filho"  
Faculdade de Ciências  
Departamento de Computação

---

**Prof. Dr. Clayton Reginaldo Pereira**

Universidade Estadual Paulista "Júlio de Mesquita Filho"  
Faculdade de Ciências  
Departamento de Computação

Bauru, \_\_\_\_\_ de \_\_\_\_\_ de \_\_\_\_\_.  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

*Dedico este trabalho à minha família, ao Circo e à todos que me apoiaram nesta jornada.*

# Agradecimentos

Agradeço à minha família por todo o apoio dado, físico e mental, ao longo desses quatro anos.

Aos amigos que fiz durante a graduação, especialmente ao pessoal do Circo, sem eles eu não chegaria até aqui com tantas memórias.

Ao meu orientador Prof. Associado Aparecido Nilceu Marana por aceitar ser meu orientador mesmo estando carregado de tarefas e outros orientandos.

Ao pessoal do Carrossel Caipira por me darem a oportunidade de fazer parte do time e participar do projeto.

Ao ex-aluno Gustavo Rosa pela construção do modelo de TCC em Latex.

À UNESP por proporcionar esta oportunidade de cursar um curso superior não tão distante de minha terra natal.

À todos que de alguma forma me ajudaram nessa experiência universitária.

*É meio que uma lei da natureza. O objetivo que alguém almeja raramente pode ser alcançado por um caminho simples.*

Konosuke Matsushita

*Existe um provérbio japonês que literalmente diz 'ice as velas com sua mão dominante', significa que você deve ir atrás das oportunidades que aparecem na sua vida com o que você tiver de melhor para realizá-las.*

Soichiro Honda

# Resumo

A ideia de poder gerar e editar faces humanas é algo interessante pois estas podem ter várias aplicações como o uso em personagens de jogos ou de animações. Dependendo de quantas faces são desejadas, é extenuante para uma pessoa criar faces manualmente. Para facilitar este trabalho, uma aplicação que possa automatizar a geração e edição de faces pode ser muito interessante e útil. A aplicação desenvolvida neste projeto busca ser um gerador e editor de faces de baixo custo computacional, visto que algumas aplicações existentes similares exigem muito poder computacional. Para isso os conceitos de *auto-encoder* e o de PCA foram utilizados para extrair as características importantes de uma face humana que podem ser alteradas para gerar de uma face diferente.

**Palavras-chave:** *Auto-encoder*. PCA. Redes Neurais Convolucionais. Aprendizado de Máquina. Geração e Edição de Faces.

# Abstract

The idea of generating and editing human faces is interesting as they can be used in a plethora of applications such as video game or animation characters. Depending on how many faces are desirable, it is tiresome for someone to create faces manually. Therefore, it is very interesting and useful the existence of an application that automatise such task. The application developed in this work longs to be an face generator and editor that does not require high amounts of computational power, since some similar existing applications require high amounts of computational power. To reach this objective the concept of auto-encoder and the concept of PCA were applied to extract prominent features of a human face that can be altered to generate a different face.

**Keywords:** Auto-encoder. PCA. Convolutional Neural Networks. Machine Learning. Face Generator and Editor.

# Listas de figuras

Figura 1 – O que o ser humano vê e o que a máquina vê.	15
Figura 2 – Diferença entre a posição de detalhes entre duas faces.	16
Figura 3 – Exemplo de uma rede neural.	19
Figura 4 – Exemplo de convolução bidimensional.	21
Figura 5 – Exemplos de <i>Pooling</i> .	22
Figura 6 – Estrutura básica de <i>auto-encoder</i> .	23
Figura 7 – Exemplos de faces geradas pelo site <a href="https://thispersondoesnotexist.com">https://thispersondoesnotexist.com</a> .	24
Figura 8 – Captura de tela do site <i>Yearbook Face Editor</i> .	25
Figura 9 – Representação do <i>auto-encoder</i> .	27
Figura 10 – Imagens originais e a saída pelo modelo.	28
Figura 11 – Esquema de funcionamento da aplicação.	29
Figura 12 – Casos de uso da aplicação.	30
Figura 13 – Estrutura hierárquica do TensorFlow.	31
Figura 14 – Imagem antes (à esquerda) e depois (à direita) do tratamento.	33
Figura 15 – Interface da aplicação com a face média.	34
Figura 16 – Interface da aplicação com uma face pré carregada.	35
Figura 17 – Exemplos de faces geradas por meio do botão “ALEATÓRIO” da aplicação.	39

# **Lista de códigos**

5.1 Implementação da função de saturação . . . . . 33

# Lista de abreviaturas e siglas

IA	Inteligência Artificial
MTCNN	<i>Multi-task Cascaded Convolutional Networks</i>
PCA	<i>Principal Component Analysis</i>
GAN	<i>Generative Adversarial Network</i>
RAM	<i>Random-access memory</i>

# Sumário

<b>Lista de códigos</b>	10	
<b>1</b>	<b>INTRODUÇÃO</b>	14
1.1	<b>Problema</b>	14
1.2	<b>Objetivos</b>	15
1.3	<b>Desafios</b>	15
1.4	<b>Organização da monografia</b>	16
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	17
2.1	<b>Processamento de Imagens</b>	17
2.2	<b>Aprendizado de Máquina</b>	17
2.2.1	Aprendizado de Máquina Profundo	17
2.3	<b>Análise de Componente Principal</b>	18
2.4	<b>Redes Neurais</b>	18
2.4.1	Redes Neurais Convolucionais	19
2.4.1.1	Camada de Convolução	20
2.4.1.2	Camada de <i>Pooling</i>	20
2.4.2	<i>Auto-encoders</i>	21
<b>3</b>	<b>APLICAÇÕES E SOLUÇÕES EXISTENTES</b>	24
3.1	<i>This Person Does Not Exist</i>	24
3.2	<i>Yearbook Face Editor</i>	25
<b>4</b>	<b>MATERIAIS E MÉTODOS</b>	26
4.1	<b>Conjunto de Treinamento</b>	26
4.2	<b><i>Auto-Encoder, Encoder e PCA</i></b>	27
4.3	<b>Gerador e Editor de Faces</b>	28
4.4	<b>Tecnologias Escolhidas</b>	29
4.4.1	TensorFlow	29
4.4.2	<i>Face Recognition</i>	31
<b>5</b>	<b>DESENVOLVIMENTO</b>	32
5.1	<b>Tratamento do conjunto de treinamento</b>	32
5.2	<b>Desenvolvimento e treinamento do <i>Auto-Encoder</i></b>	32
5.3	<b>O <i>Encoder</i> e o PCA</b>	34
5.4	<b>A aplicação final</b>	34
5.4.1	Controles deslizáveis	35

5.4.2	Botões e Caixa de Texto . . . . .	35
<b>5.5</b>	<b>Inserção de novas faces . . . . .</b>	<b>36</b>
<b>5.6</b>	<b>Passo a Passo . . . . .</b>	<b>36</b>
<b>6</b>	<b>TESTE E VALIDAÇÃO . . . . .</b>	<b>38</b>
<b>6.1</b>	<b>Testes . . . . .</b>	<b>38</b>
<b>6.2</b>	<b>Validação . . . . .</b>	<b>38</b>
<b>7</b>	<b>CONCLUSÃO . . . . .</b>	<b>40</b>
<b>7.1</b>	<b>Trabalhos Futuros . . . . .</b>	<b>40</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>41</b>

# 1 Introdução

Atualmente, a ideia de poder criar e editar faces humanas é algo que prende a atenção das pessoas, muitas já tentaram imaginar como elas seriam se certa parte do rosto delas fosse um pouco diferente, tanto que o site <<https://thispersondoesnotexist.com/>> (KARRAS; LAINE; AILA, 2018) causou certa movimentação em sites de notícias sobre tecnologia no Brasil e no mundo. Como é inviável que uma pessoa possa criar uma quantidade enorme de faces humanas realistas, como por exemplo para uma personagem de um jogo ou de uma animação tridimensional, uma aplicação utilizando Inteligência Artificial (IA) foi desenvolvida por Karras, Laine e Aila (2018).

Uma maneira possível de desenvolver uma aplicação utilizando IA que realize feitos similares é com a utilização de *encoders* e *decoders*, mais precisamente no conceito dos *Auto-Encoders* (DOERSCH, 2016).

Um *Encoder* tem a função de traduzir algum arquivo em código, seja imagem, áudio, vídeo ou até mesmo outro código. Seu principal uso é o de padronização e compressão. Um *Decoder* tem a função oposta, reconstruindo o arquivo original a partir do código gerado por um *encoder*.

Russell e Norvig (2009) definem Inteligência Artificial como o campo que estuda “agentes inteligentes”: qualquer dispositivo que observa seu ambiente e realiza ações que maximizam suas chances de sucesso em alcançar determinado objetivo. Russell e Norvig (2009) também definem que coloquialmente IA é utilizado para descrever máquinas ou computadores que imitam funções cognitivas características de humanos, como aprendizado e a capacidade de solucionar problemas.

## 1.1 Problema

A tarefa de mentalmente criar e modificar faces a partir de informações dadas sobre a composição delas é simples para um ser humano, mas para uma máquina é um desafio considerável pois, diferentemente de um ser humano que utiliza vários outros recursos para identificar objetos em uma imagem, a máquina utiliza um conjunto de números como ilustra a Figura 1.

Uma maneira possível de obter os resultados desejados é utilizar *auto-encoders*. Neste caso deve-se obter um conjunto de faces para treinar o *auto-encoder* e utilizar o modelo criado para a geração das faces.

Figura 1 – O que o ser humano vê e o que a máquina vê.



Fonte: Elaborada pelo autor.

## 1.2 Objetivos

O objetivo deste trabalho foi desenvolver uma ferramenta para possibilitar a criação e edição de faces a partir de parâmetros dados pelo usuário, como largura da face, posição dos olhos e sobrancelhas e expressões faciais.

Tal ferramenta não deverá exigir um alto poder computacional, devendo rodar sem travamentos em um computador pessoal com componentes razoáveis e sem placa de vídeo dedicada.

## 1.3 Desafios

Um dos desafios é a obtenção de um banco de imagens onde as pessoas estejam em posições similares para não haver muita discrepância em dados como a posição do nariz em relação ao queixo como ilustra a Figura 2, pois o fato do conjunto de treinamento não ser uniforme irá impactar na qualidade do modelo criado.

Uma maneira de diminuir as discrepâncias entre as posições dos rostos nas imagens é utilizar uma ferramenta de detecção e reconhecimento de faces humanas, como o MTCNN (disponível em <https://github.com/ipazc/mtcnn>) e o *Face Recognition* (disponível em [https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition)).

Outro desafio é a definição dos parâmetros: o que deve ser alterado nos parâmetros para a geração de uma face nova ou para alterar uma face já gerada.

Figura 2 – Diferença entre a posição de detalhes entre duas faces.



Fonte: Elaborada pelo autor.

Para isso a utilização da técnica de Análise de Componente Principal (*Principal Component Analysis, PCA*) é utilizada. Esta técnica determina quais características das imagens de treinamento são as mais importantes e que, portanto, devem ser utilizadas como os parâmetros de geração e edição das faces.

## 1.4 Organização da monografia

Além deste capítulo introdutório, que apresenta a motivação, os objetivos e os desafios do projeto, esta monografia possui outros seis capítulos, apresentados a seguir.

O Capítulo 2 apresenta a fundamentação teórica necessária para o entendimento e organização do trabalho, incluindo os conceitos sobre Aprendizado de Máquina e Redes Neurais Artificiais que foram utilizados.

O Capítulo 3 exibe aplicações similares ao desenvolvido neste trabalho.

O Capítulo 4 descreve os materiais, as ferramentas e os métodos utilizados no desenvolvimento deste trabalho.

O Capítulo 5 relata o processo de desenvolvimento e uso, desde a obtenção e tratamento do conjunto de dados até o desenvolvimento do programa. Ao final do capítulo, breves instruções de como utilizar um conjunto diferente para o treinamento são disponibilizadas.

O Capítulo 6 relata os testes realizados durante o desenvolvimento e as validações finais do trabalho.

Por fim o Capítulo 7 apresenta a conclusão da monografia, sugerindo possíveis trabalhos futuros.

## 2 Fundamentação Teórica

### 2.1 Processamento de Imagens

Uma imagem pode ser definida como uma função bidimensional  $f(x, y)$  onde  $x$  e  $y$  são coordenadas espaciais e a amplitude de  $f$  em qualquer par de coordenadas  $(x, y)$  é chamada de intensidade da imagem naquele ponto. Quando os valores das coordenadas e das amplitudes são todas finitas e discretas, tem-se uma imagem digital. O campo de processamento de imagens digitais se refere ao processamento de imagens digitais por meio de um computador ([GONZALEZ; WOODS, 2008](#)).

### 2.2 Aprendizado de Máquina

No início da Inteligência Artificial (IA) os problemas aplicados eram intelectualmente complicados para humanos mas simples para máquinas pois eram problemas descritos com uma série de regras formais matemáticas. Começou-se a ter um grande desafio quando os problemas nos quais desejava-se aplicar a inteligência artificial eram simples para humanos porém difíceis para serem descritos formalidades matemáticas, como, por exemplo, o reconhecimento de faces ou de letras e símbolos.

A dificuldade dos computadores resolverem tais problemas utilizando códigos puros levou à concepção da ideia de que um sistema de IA deve adquirir seu próprio conhecimento, a partir de padrões nos dados apresentados. A esta capacidade dá-se o nome de Aprendizado de Máquina.

Aprendizado de Máquina é um ramo de pesquisa da Inteligência Artificial que aborda problemas de maneira diferente, treinando um modelo com base em uma série de exemplos, com um algoritmo sendo usado para modificá-lo e otimizá-lo quando preciso ([ALPAYDIN, 2009](#)).

#### 2.2.1 Aprendizado de Máquina Profundo

A hierarquia de conceitos possibilita ao computador aprender conceitos complexos a partir de outros mais simples. Se desenharmos um grafo ilustrando como esses conceitos são construídos a partir de outros ele será profundo, com várias camadas. Por este motivo chamamos essa aproximação de Aprendizado Profundo ([GOODFELLOW; BENGIO; COURVILLE, 2016](#)).

## 2.3 Análise de Componente Principal

A Análise de Componente Principal (PCA - *Principal Component Analysis*) é uma técnica estatística que consiste na utilização de transformações ortogonais para converter um dado conjunto de observações de variáveis possivelmente correlacionados para um conjunto de variáveis não correlacionados chamados de Componentes Principais. O PCA é definido de forma que o o primeiro componente principal possui a maior variância possível no conjunto inicial, e cada componente seguinte possui a maior variância sob a restrição de ser ortogonal a todos os componentes anteriores.

As principais funções do PCA incluem ([WOLD; ESBENSEN; GELADI, 1987](#)): simplificação, redução de dados, modelagem, detecção de ponto fora da curva, seleção de variáveis, classificação, predição, ordenação.

## 2.4 Redes Neurais

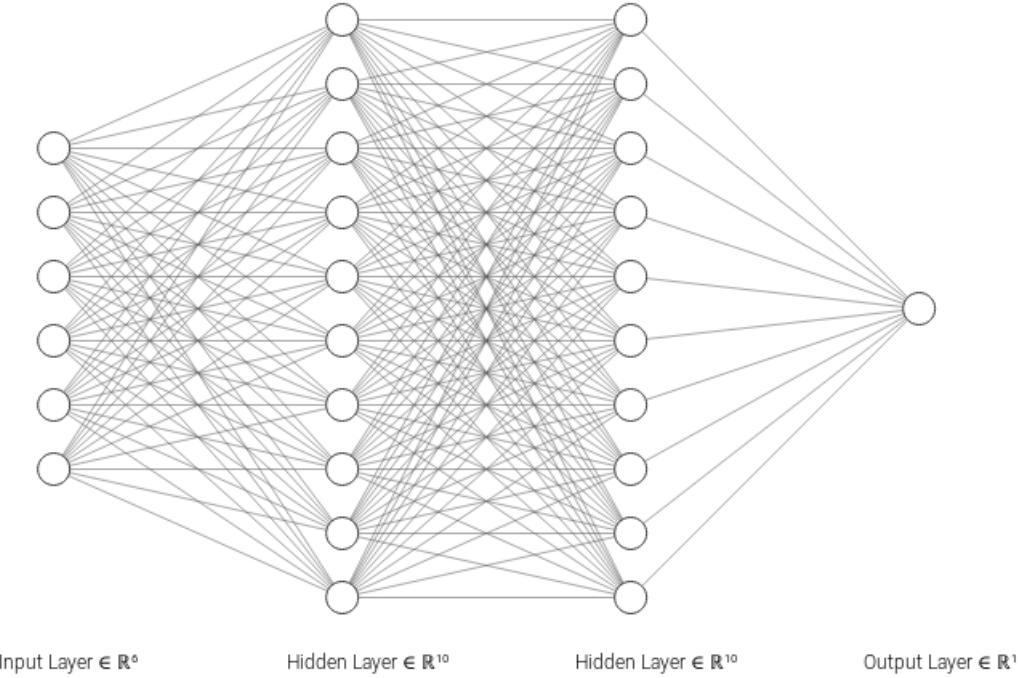
Uma Rede Neural Artificial é um modelo computacional que é baseado em como o cérebro humano processa informação ([KARN, 2016](#)), não sendo um algoritmo em si mas sendo um *framework* onde diversos tipos de algoritmos relacionados ao Aprendizado de Máquina trabalham em conjunto. O aprendizado de máquina se baseia na ideia de aprender com exemplos. Em vez de darmos um conjunto específico de regras para resolver um problema, nós damos um modelo onde se avalia exemplos e uma lista de instruções para modificar o modelo caso hajam falhas. É esperado que após certo tempo um bom modelo deva ser capaz de resolver o problema de forma bem precisa ([BUDUMA, 2017](#)). Redes neurais podem ser constituídas por uma ou mais camadas de neurônios.

A arquitetura de uma rede neural é baseada no funcionamento de neurônios artificiais que, assim como neurônios biológicos, recebem estímulos de entrada e ativam se estes estímulos forem significantes o suficiente, repassando o estímulo processado para outros neurônios conectados.

Um neurônio recebe os estímulos como números reais e retorna como saída o resultado da aplicação de uma função não linear (geralmente a função sigmoide) na soma dos estímulos. Esta saída ainda é submetida a um ajuste causado por pesos existentes nas conexões entre neurônios. Estes pesos variam de acordo com o aprendizado da rede.

Geralmente os neurônios artificiais estão organizados em camadas dentro de uma rede neural e cada camada pode realizar uma operação específica em sua entrada. Os estímulos viajam da primeira camada (a camada de entrada) até a última (a camada de saída), possivelmente após atravessar múltiplas camadas. A Figura 3 ilustra uma rede neural.

Figura 3 – Exemplo de uma rede neural.



Fonte: Elaborada pelo autor.

#### 2.4.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais (CNN - *Convolutional Neural Networks*) são um tipo especializado de redes neurais para processar dados organizados em topologias de grade, por exemplo imagens representadas por uma grade bidimensional de pixels. Como seu nome implica, as CNNs aplicam uma operação matemática linear chamada convolução. CNNs são simplesmente redes neurais que se utilizam de convolução em vez de uma multiplicação geral matricial em ao menos uma de suas camadas ([GOODFELLOW; BENGIO; COURVILLE, 2016](#)).

Convolução é um operador linear que a partir de duas funções gera uma terceira função que expressa como a forma de uma função é modificada pela outra. A operação é representada por um asterisco e é definida pela integral do produto de duas funções após uma delas ter sido invertida e transladada como descrita pela Equação 2.1.

$$(f * g)(t) = \int f(a)g(t - a)da \quad (2.1)$$

Pela terminologia utilizada por CNNs, o primeiro argumento (a função  $f$ ) é denominado entrada e o segundo argumento (a função  $g$ ) é denominado Kernel.

Para intervalos discretos a Equação 2.2 pode ser utilizada.

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \quad (2.2)$$

Em aplicações de CNNs a entrada geralmente é um vetor multidimensional de dados e o kernel um vetor multidimensional de parâmetros adotado pelo algoritmo. Estes vetores multidimensionais podem ser chamados de tensores. Para uma aplicação com entrada bidimensional  $I$  e kernel  $K$  também bidimensional é utilizada a Equação 2.3.

$$(I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \quad (2.3)$$

A Equação 2.3 pode ser escrita como segue na Equação 2.4 pois a operação de convolução é comutativa.

$$(K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \quad (2.4)$$

A utilização da Equação 2.4 é mais simples no campo do aprendizado de máquina pois há menos variação no intervalo de valores válidos de  $m$  e  $n$ .

A Figura 4 ilustra a operação de convolução em uma entrada de duas dimensões e um kernel  $2 \times 2$ .

#### 2.4.1.1 Camada de Convolução

Nesta camada cada neurônio está associado a uma janela de kernel que será aplicada na entrada. Esse kernel é composto pelos valores de pesos das ligações do neurônio. A partir da tensor de entrada original são gerados  $n$  tensores resultantes das operações em cada neurônio.

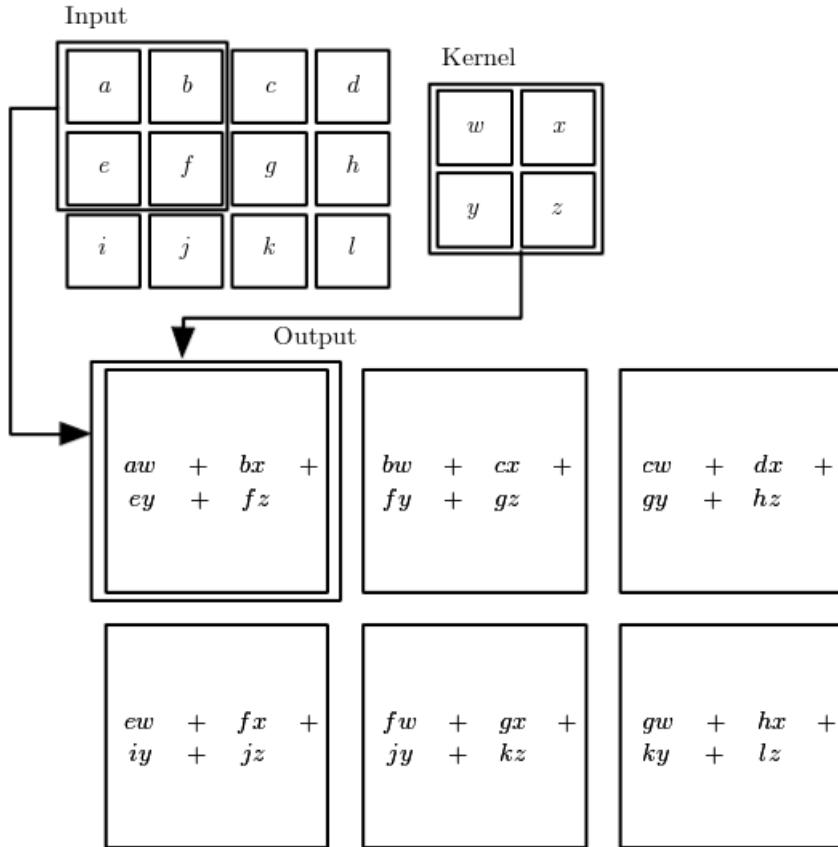
Caso a entrada seja uma imagem, após realizada a convolução é aplicada uma função de ativação ReLU nos tensores. A função é dada por  $\max(0, x)$  e serve para retirar quaisquer valores negativos que possam existir nos tensores, substituindo-os por zero. Isso pois uma imagem não pode ter valores negativos em seus pixels.

#### 2.4.1.2 Camada de *Pooling*

Esta camada é responsável por diminuir as dimensões dos tensores para diminuir o poder computacional necessário para processar os dados.

Existem dois tipos de *pooling*: *Max Pooling* e *Average Pooling*. *Max Pooling* retorna o valor máximo da área coberta pelo kernel. O *Average Pooling* retorna a média da área coberta. A Figura 5 ilustra ambos tipos de *pooling*.

Figura 4 – Exemplo de convolução bidimensional.



Fonte: ([GOODFELLOW; BENGIO; COURVILLE, 2016](#)).

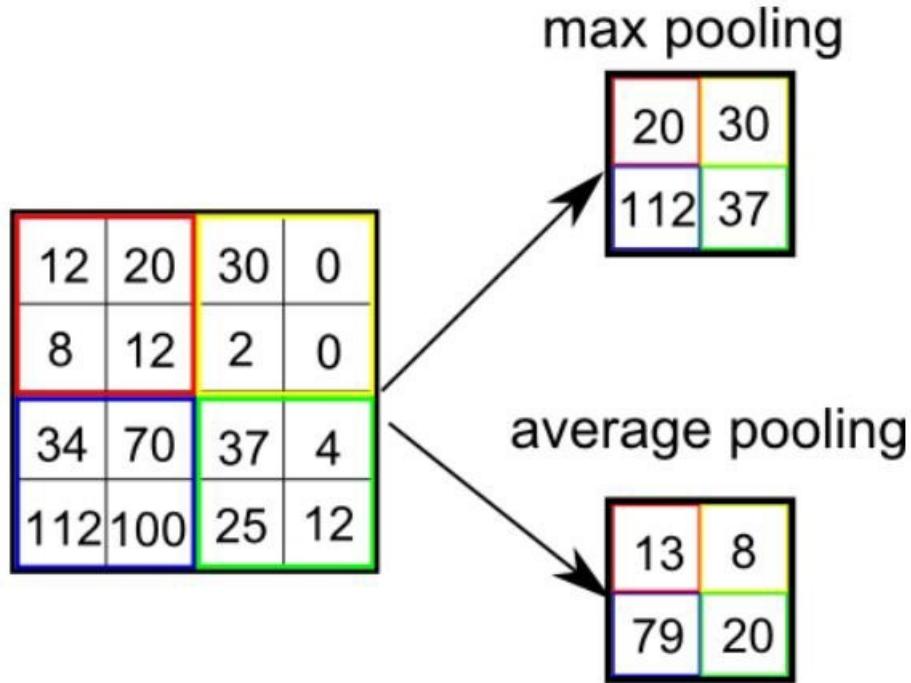
#### 2.4.2 Auto-encoders

*Auto-encoders* são algoritmos de compressão de dados que possuem certas características:

- São específicos para um tipo de dado, um *auto-encoder* treinado com imagens só pode operar com imagens;
- A saída de um *auto-encoder* sempre terá um grau de degradação em relação à entrada;
- Aprendem automaticamente por exemplos, sem precisar da ajuda humana.

Um *auto-encoder* é um tipo de Rede Neural Artificial que é basicamente constituído entre a união entre um *encoder* e um *decoder*. O principal uso de um *auto-encoder* é de aprender como comprimir dados em código e reconstruir os dados originais a partir do código com o mínimo de perda após a passagem dos dados por um gargalo: o espaço latente ([LADJAL; NEWSON; PHAM, 2019](#)).

Figura 5 – Exemplos de *Pooling*.



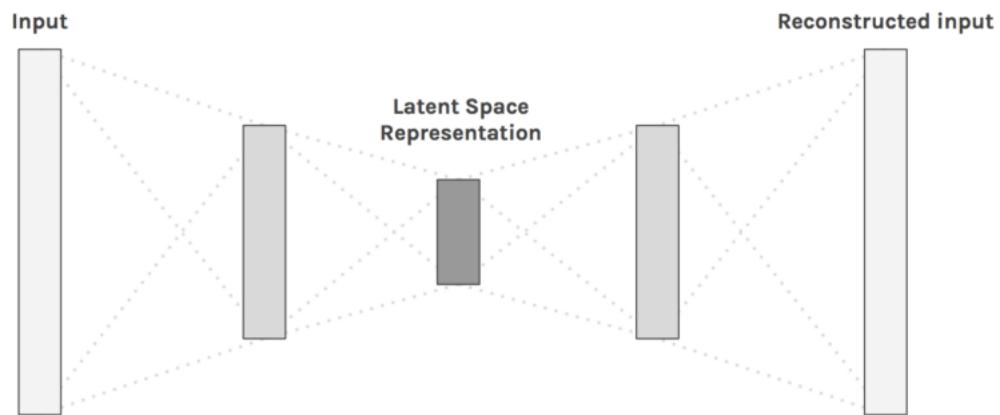
Fonte: <<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>>.

A arquitetura popular de redes neurais em geral indica que antes da utilização de uma rede neural para uma tarefa, características da entrada sejam extraídas por meio de camadas da rede. O espaço latente nada mais é do que a camada onde estão as características da entrada após estas serem extraídas. Um *auto-encoder* realiza tanto o papel de extração de características da entrada quanto a reconstrução da entrada tendo apenas as características da entrada.

A Figura 6 ilustra a estrutura básica de um *auto-encoder*.

Um exemplo de uso de *auto-encoders* é [Li et al. \(2018\)](#) que recentemente desenvolveram *auto-encoders* que, após “aprenderem” a associar palavras escritas com vídeos e imagens, conseguem gerar vídeos curtos a partir de um texto dado.

Figura 6 – Estrutura básica de *auto-encoder*.



Fonte: <<https://ai-odyssey.com/2017/02/07/autoencoders>>.

### 3 Aplicações e soluções existentes

Aplicações que geram faces humanas realistas já existem, algumas inclusive permitem um certo nível de modificação nas faces que são geradas. As próximas seções apresentam duas dessas aplicações.

#### 3.1 *This Person Does Not Exist*

Disponível em <<https://thispersondoesnotexist.com/>>, é provavelmente o exemplo mais famoso de uma aplicação que gera faces humanas, foi implementada por [Karras, Laine e Aila \(2018\)](#) utilizando uma técnica recente de aprendizado profundo: as Redes Adversárias Generativas (GAN - Generative Adversarial Network), onde o conceito básico é que duas redes neurais competem entre si em um jogo (no sentido de Teoria dos Jogos) para aprenderem determinada tarefa. A Figura 7 ilustra exemplos de faces geradas pelo site.

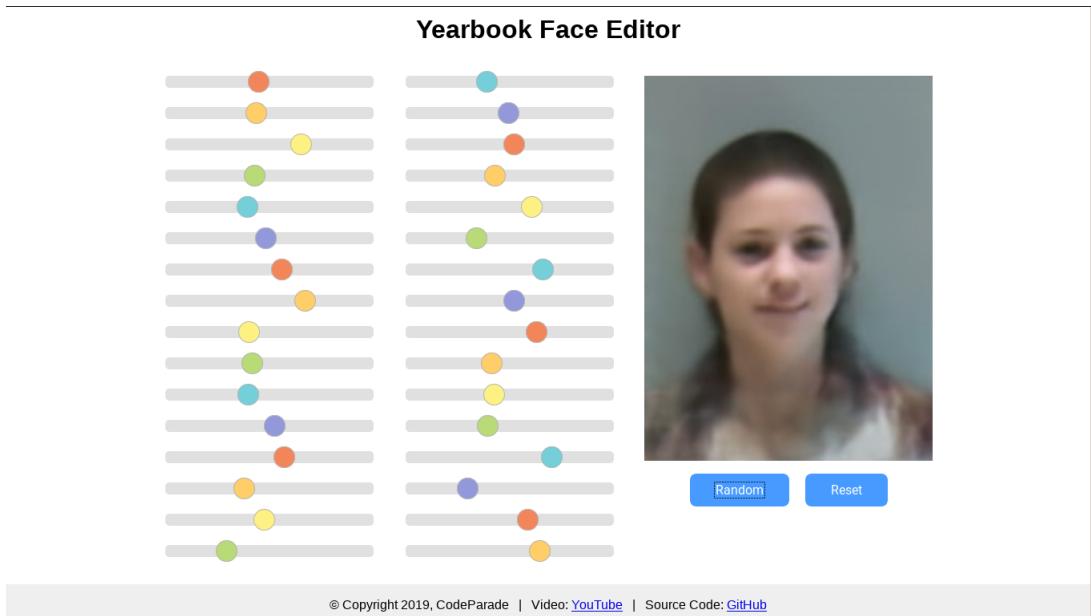
Uma GAN funciona da seguinte forma: uma das redes (a rede geradora) gera candidatos e a outra rede (a rede discriminadora) as avalia. Inicialmente a rede discriminadora é treinada com um conjunto de dados até que alcance uma precisão aceitável e a rede geradora é treinada com um espaço latente pré determinado. A função do gerador é conseguir gerar imagens que consigam enganar o discriminador e uma função de retro propagação é aplicada em ambas as redes para que a rede geradora gere imagens melhores e a rede discriminadora melhore sua habilidade de avaliar se a imagem foi ou não gerada pela rede geradora ([GOODFELLOW et al., 2014](#)).

Figura 7 – Exemplos de faces geradas pelo site <https://thispersondoesnotexist.com>.



Fonte: ([KARRAS; LAINE; AILA, 2018](#)).

Figura 8 – Captura de tela do site *Yearbook Face Editor*.



Fonte:<<http://codeparade.net/faces/>>.

### 3.2 *Yearbook Face Editor*

Uma aplicação desenvolvida pelo indivíduo identificado como CodeParade ou HackerPoet como hobby e disponível em <<http://codeparade.net/faces/>>, utiliza o PCA diretamente no conjunto de dados de treinamento, sem passar pelos processos que foram utilizados neste trabalho, possui um conjunto de dados de treinamento bem menor visto que este foi o anuário de escola do autor. Uma captura de tela da aplicação pode ser vista na Figura 8.

# 4 Materiais e métodos

## 4.1 Conjunto de Treinamento

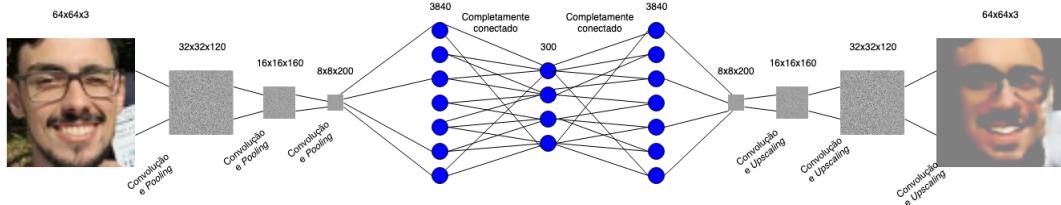
Para a obtenção do conjunto de dados para a realização do treinamento da rede neural foi feita uma análise de algumas bases de dados públicas pela Internet. Os critérios para a escolha foram a variação de pessoas nas bases de dados, o fato do nome da pessoa estar ou não associada à imagem e as imagens serem coloridas.

A seguir estão as bases e o veredito da escolha:

- **Specs on Faces Dataset**([AFIFI; ABDELHAMED, 2019](#)): um conjunto de dados com 42592 imagens de 112 pessoas em diferentes condições de iluminação, posição e oclusão. Possui uma baixa variação de pessoas.
- **Labeled Faces in the Wild Home**([HUANG et al., 2007](#)): este conjunto de dados possui mais de 13000 imagens de 1680 pessoas, todas as imagens possuem o nome da pessoa associada. Possui várias imagens de uma mesma pessoa, diminuindo a variação de faces no conjunto.
- **Famous Birthdays**([famousbirthdays.com](#)): não é um conjunto de dados formal, porém possui imagens de milhares de celebridades. Cada dia do ano mostra as 48 celebridades mais famosas nascidas no dia, para um potencial de  $366 \times 48 = 17520$  imagens. Como cada celebridade aparece apenas uma vez, a variação de faces dentre as imagens é alta.
- **VGGFace2** ([CAO et al., 2018](#)): conjunto com mais de 3,3 milhões de faces de mais de 9000 pessoas, é diferente dos outros conjuntos de dados pois este conjunto possui tanto imagens quanto modelos já treinados disponíveis para o uso. As imagens não possuem identificação da pessoa nela. Este conjunto também é notório por necessitar de uma alta capacidade computacional para ser utilizado.

A base de dados escolhida foi o site *Famous Birthdays* pela alta variação de pessoas nas imagens. Mesmo assim uma parte das imagens teve que ser descartada por não possuir uma face reconhecível pelo algoritmo de reconhecimento facial utilizado ou por ser em escala de cinzas.

Figura 9 – Representação do *auto-encoder*.



Fonte: Elaborada pelo autor.

## 4.2 Auto-Encoder, Encoder e PCA

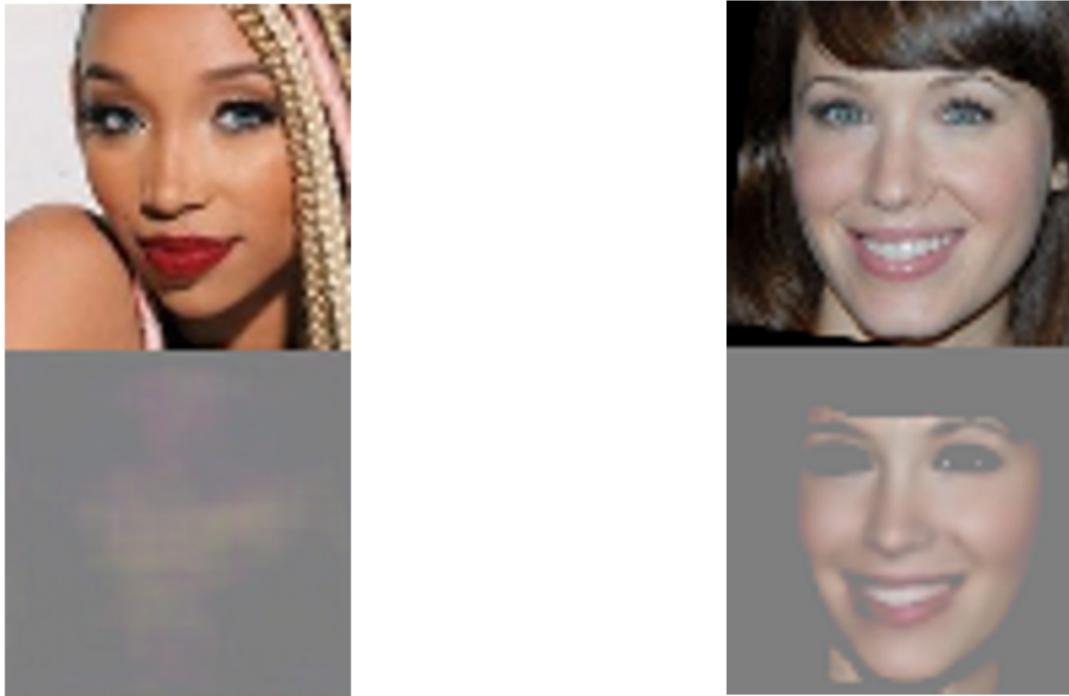
O uso de um *auto-encoder* não é necessariamente indispensável porém sua utilização é interessante por dois motivos principais: o primeiro é que o *auto-encoder* realiza a redução da quantidade de dimensões da imagem para que o PCA não tenha que descartar muitas características e também para diminuir o custo de processamento do PCA (inclusive este é o motivo do porquê não foi utilizado o VGGFace2, notório por exigir muito poder computacional). O segundo motivo é que ao passar pelas camadas de convolução, a rede neural passa a ter um conhecimento espacial de como os pixels se relacionam um com os outros, adquirindo um senso de “vizinhança” para cada pixel. O segundo motivo torna possível a rede neural entender mais facilmente características como a rotação da cabeça ou o tamanho do nariz, traços intrinsecamente espaciais e físicos da imagem.

O *auto-encoder* recebe como entrada uma imagem com 64 pixels de largura, 64 pixels de altura e 3 canais de cores - o que resulta em 12288 características - e as diminui para  $4 \times 4 \times 240 = 3840$  características com pouca perda, mesclando as características originais, e dessas 3840 características são escolhidas 300 para a representação densa. Caso o PCA fosse aplicado diretamente nas imagens originais, quase 12000 características seriam descartadas, ao contrário de 3540, uma perda bem menor. A Figura 9 ilustra de forma simplificada o caminho percorrido pela imagem.

Após o treinamento, o *auto-encoder* gera um modelo das faces que será utilizado por um encoder para gerar um vetor denso das imagens a partir do modelo. A Figura 10 mostra exemplos de imagens do conjunto de treinamento e sua saída gerada pelo modelo treinado pelo *auto-encoder*, observa-se a evolução do modelo em gerar saídas que melhor representam faces humanas.

Depois de gerado os vetores densos, o PCA é utilizado para obter os autovetores e autovalores.

Figura 10 – Imagens originais e a saída pelo modelo.



Modelo gerado na época 2

Modelo gerado na época 182000

Fonte: Elaborada pelo autor.

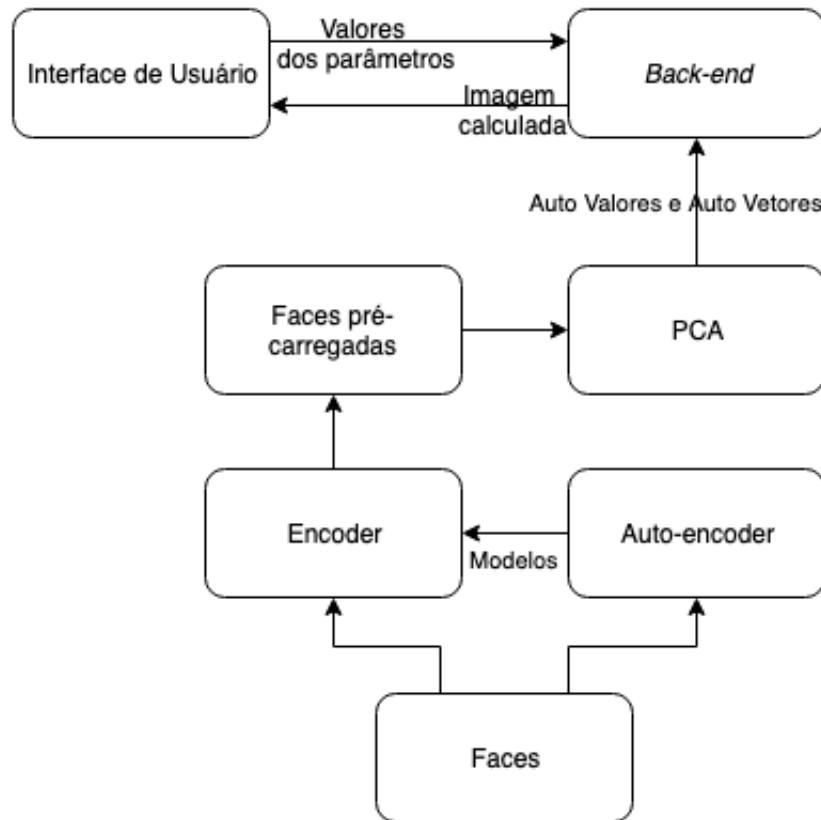
### 4.3 Gerador e Editor de Faces

A aplicação projetada e desenvolvida neste trabalho consiste em uma interface de usuário simples onde o usuário pode alterar posições em controles deslizáveis para alterar os valores que serão utilizados para o cálculo da imagem. Esses valores serão repassados para o *back-end* que realizará os cálculos em tempo real e retornará uma imagem reconstruída a partir dos valores.

A Figura 11 ilustra o funcionamento da aplicação. Primeiro o *back-end* da aplicação deve ser alimentado com os auto-valores e auto-vetores obtidos pela aplicação do PCA em todas as faces dado um modelo gerado pelo *auto-encoder*. Após a alimentação do *back-end* o uso se faz primariamente pela interface, onde o usuário pode alterar os parâmetros que serão repassados ao *back-end* que realizará uma multiplicação de cada parâmetro com seu respectivo auto-vetor e auto-valor e retornará à interface a imagem calculada.

A Figura 12 ilustra ações que o usuário pode realizar. As ações mais simples e que possivelmente serão as mais realizadas são: a de alterar os parâmetros para o cálculo da nova face, a de ir diretamente para uma face pré-carregada e a de salvar a imagem atual. Se o usuário quiser ele pode inserir novas faces ou até mesmo utilizar um conjunto de treinamento

Figura 11 – Esquema de funcionamento da aplicação.



Fonte: Elaborada pelo autor.

totalmente diferente e retreinar o *auto-encoder*, gerando um novo modelo e aplicar o PCA nesse novo modelo criado.

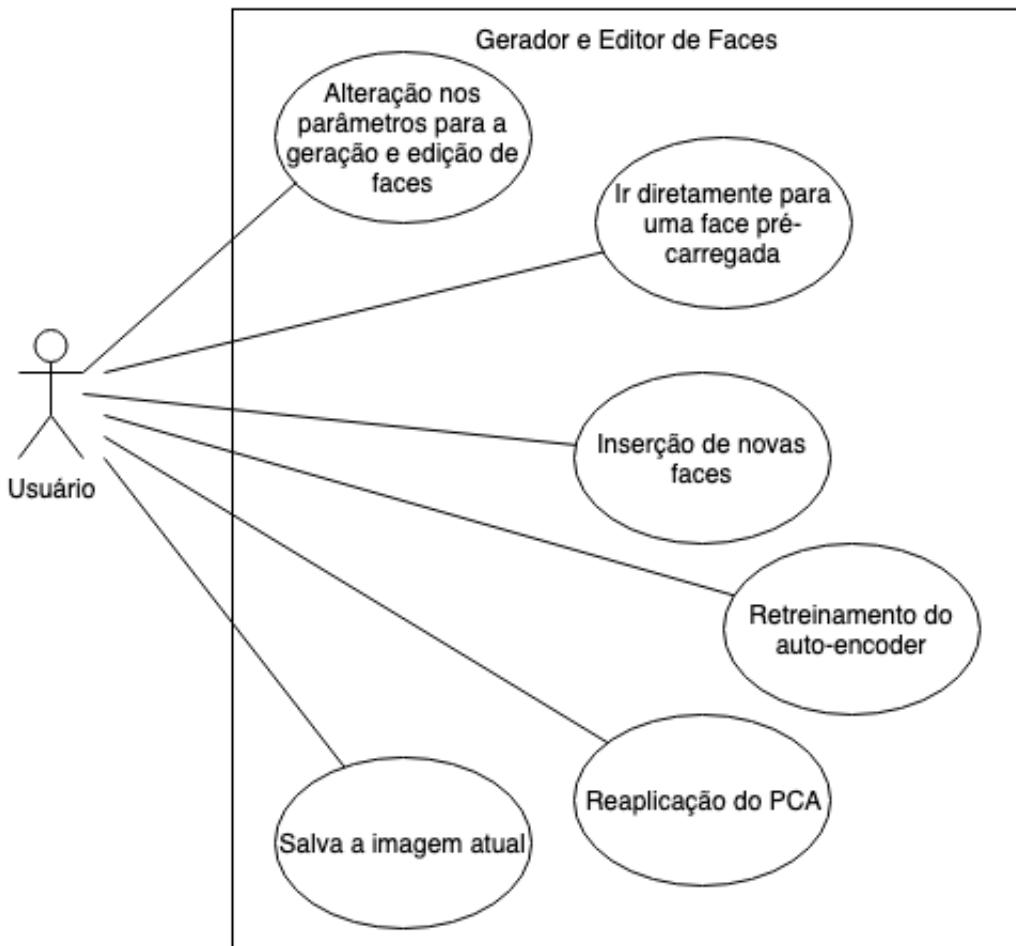
## 4.4 Tecnologias Escolhidas

Para a implementação do trabalho optou-se pelo uso da linguagem Python3, a biblioteca *Face Recognition* para o reconhecimento de faces nas imagens, o TensorFlow como a plataforma para o treinamento do *auto-encoder* e a biblioteca PyGame para a criação da interface de usuário.

### 4.4.1 TensorFlow

TensorFlow ([ABADI et al., 2015](#)) é uma biblioteca de código aberto multiplataforma para computação numérica e aprendizado de máquina utilizando grafos de fluxo de dados. Ele reúne uma gama de modelos e algoritmos de aprendizado de máquina e aprendizado profundo e foi desenvolvido inicialmente pelo *Google Brain Team* da Google para uso interno, sendo lançado com a licença Apache 2.0 em novembro de 2015.

Figura 12 – Casos de uso da aplicação.



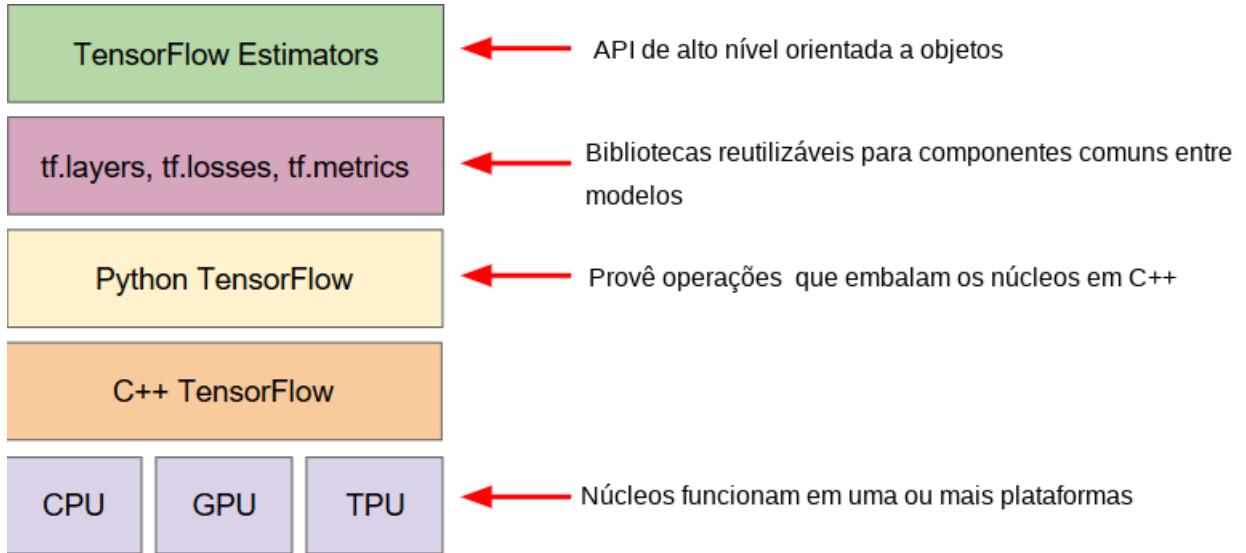
Fonte: Elaborada pelo autor.

O TensorFlow é composto por uma base implementada na linguagem C++ por causa de seu desempenho e possui uma interface em Python por ser uma linguagem mais simples. Em cima dessa interface Python situam-se bibliotecas de alto nível, como bibliotecas de modelos comumente utilizados e seus componentes e sobre essas bibliotecas existem as interfaces de mais alto nível para simplificar a criação de modelos e redes neurais como o Keras. A Figura 13 ilustra a estrutura hierárquica do TensorFlow.

O funcionamento do TensorFlow se baseia na ideia de grafos, onde o usuário utiliza o TensorFlow para construir grafos onde em cada nó uma operação matemática é realizada e as arestas são tensores comunicados entre os nós.

O maior benefício de utilizá-lo é a abstração que ele oferece para o processo de desenvolvimento de aplicações em aprendizado de máquina, simplificando a implementação. Outro benefício é sua ampla comunidade e curva de aprendizado não-íngreme, alguns concorrentes como o *CNTK - Microsoft Cognitive Toolkit* da Microsoft possui uma curva de aprendizado mais acentuada.

Figura 13 – Estrutura hierárquica do TensorFlow.



Fonte: Baseado em <<https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit>>.

#### 4.4.2 Face Recognition

*Face Recognition* é uma biblioteca em código aberto de reconhecimento facial que não necessita do uso do TensorFlow, o que o torna mais leve e menos exigente de poder computacional. A biblioteca se encontra disponível em <[https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition)>.

Possui várias funcionalidades, desde a simples detecção de faces em uma imagem e suas posições até o reconhecimento de faces de uma mesma pessoa em diferentes imagens. A funcionalidade mais interessante a este trabalho é a de detecção de faces e suas posições na imagem.

# 5 Desenvolvimento

Neste capítulo são apresentadas as etapas realizadas neste trabalho: obtenção e tratamento da base de dados, a biblioteca de aprendizado de máquina em Python utilizada, a rede neural implementada e o programa em si.

## 5.1 Tratamento do conjunto de treinamento

Por não ser um conjunto de dados formal, não existe uma uniformização nas imagens do *Famous Birthdays*, logo as imagens não são alinhadas com o rosto na vertical, necessitando uma normalização de todas as imagens recolhidas do site.

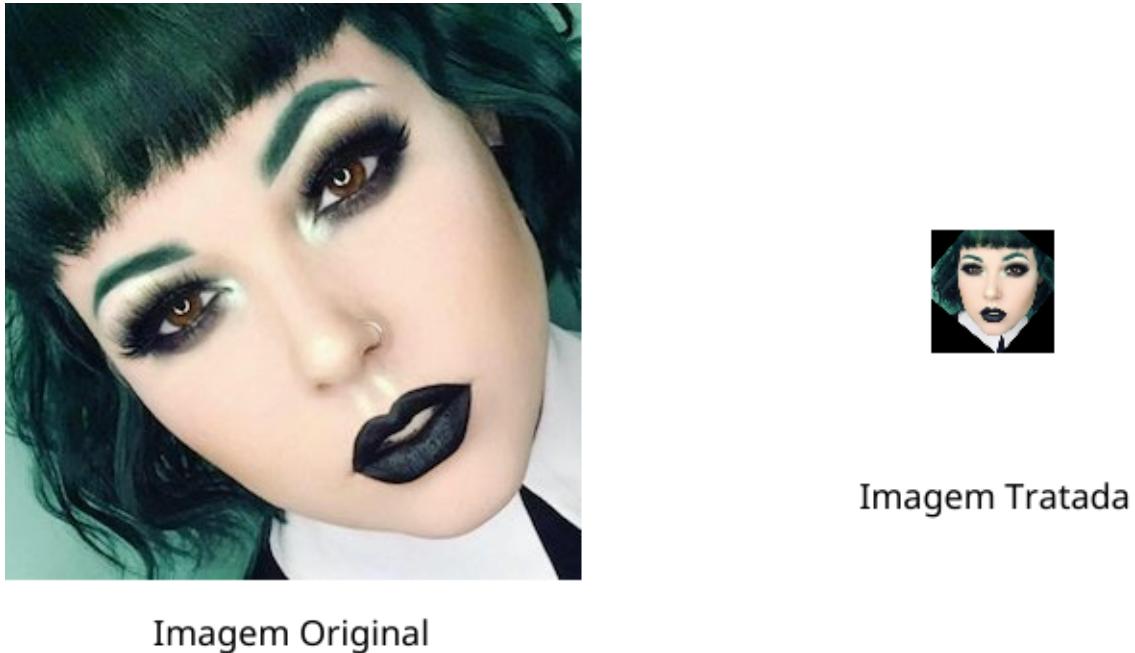
A obtenção da base de dados foi feita por meio de um rastreador de páginas que realizava o download apenas das imagens desejadas e já realizava o alinhamento das faces. Para definir se as imagens estavam ou não em escala de cinza foi utilizada uma função baseada nas equações apresentadas em 5.1 para calcular a saturação da imagem, onde  $R_{norm}$ ,  $G_{norm}$  e  $B_{norm}$  são os valores normalizados dos canais R, G e B respectivamente,  $\mu_R$ ,  $\mu_G$  e  $\mu_B$  são as médias dos valores de cada canal,  $\delta_R$ ,  $\delta_G$  e  $\delta_B$  são os desvios padrões de cada canal e  $I_{sat}$  é o escore de saturação da imagem cujo caso valor for menor que 0,04 a imagem é considerada em escala de cinza e descartada. O Código 5.1 mostra a implementação na linguagem Python da função de saturação.

$$\begin{aligned} R_{norm} &= \frac{R - \mu_R}{\delta_R} \\ G_{norm} &= \frac{G - \mu_G}{\delta_G} \\ B_{norm} &= \frac{B - \mu_B}{\delta_B} \\ I_{sat} &= (R_{norm} - G_{norm})^2 + (R_{norm} - B_{norm})^2 + (G_{norm} - B_{norm})^2 \end{aligned} \tag{5.1}$$

No final 12095 imagens foram selecionadas. O tratamento realizado nas imagens foi a verticalização das faces: a imagem foi rotacionada para que o centro dos lábios e o centro do nariz estejam na mesma posição horizontal, os olhos em posições fixas determinadas no código e o redimensionamento das imagens de 300x300 para 64x64. A Figura 14 ilustra como a imagem antes e depois do tratamento.

## 5.2 Desenvolvimento e treinamento do *Auto-Encoder*

Figura 14 – Imagem antes (à esquerda) e depois (à direita) do tratamento.



Fonte: Elaborada pelo autor.

Código 5.1 – Implementação da função de saturação

```
import numpy as np
import imageio

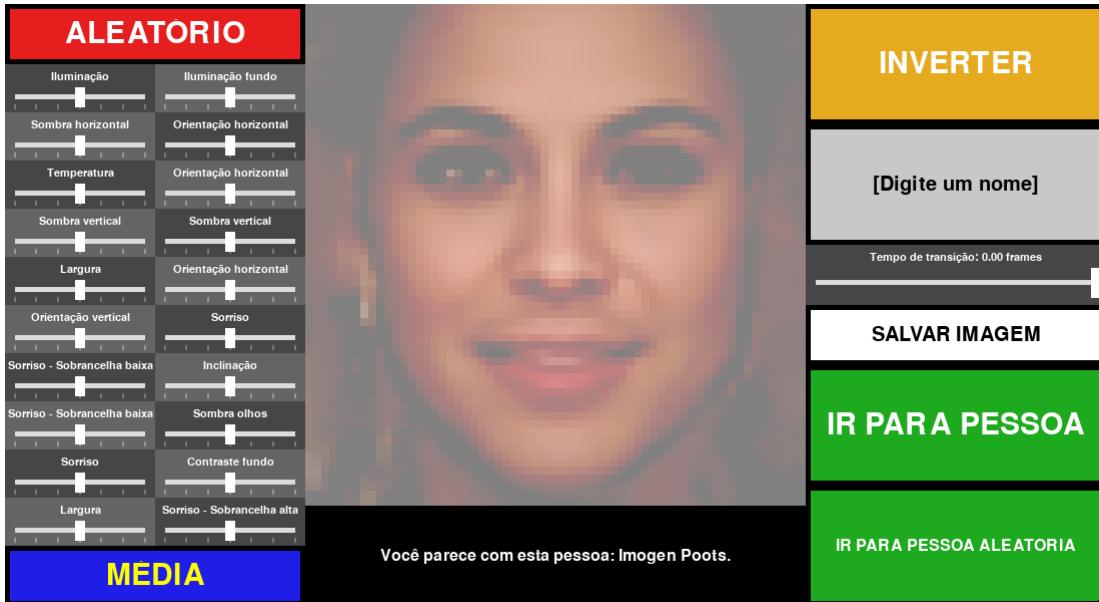
def getNorm(a):
    return (a-np.mean(a))/np.std(a)

imgNumpy = imageio.imread("temp.png")
imgNumpy = np.asarray(imgNumpy)
cores = 0

nR = getNorm(imgNumpy[:, :, 0])
nG = getNorm(imgNumpy[:, :, 1])
nB = getNorm(imgNumpy[:, :, 2])
cores = np.mean(np.square(nR-nG))+np.mean(np.square(nR-nB))+
       np.mean(np.square(nG-nB))
```

Como dito nas Seções 4.1 e 4.2, não é necessário o uso de um *auto-encoder*, principalmente com a existência de modelos já treinados e bem utilizados disponibilizados pelo VGGFaces. Porém é interessante a rede neural ter conhecimento da “vizinhança” entre os pixels e também é interessante a possibilidade de utilizar máquinas menos potentes (como a utilizada

Figura 15 – Interface da aplicação com a face média.



Fonte: Elaborada pelo autor.

neste trabalho) pois nem todas as pessoas têm condições de adquirir computadores com alta capacidade de processamento devido aos elevados custos.

O *auto-encoder* foi treinado com lotes de duzentas imagens selecionadas aleatoriamente do conjunto de treinamento e cada lote representa uma época de treinamento. Inicialmente um modelo foi salvo a cada três épocas como *backup* porém a utilização de armazenamento se mostrou elevada, por isso optou-se por salvar um modelo como *backup* a cada 100 épocas. O *auto-encoder* foi treinado por 182000 épocas. Durante todo o processo de treinamento a taxa de perda flutuou entre 0.62 e 0.69.

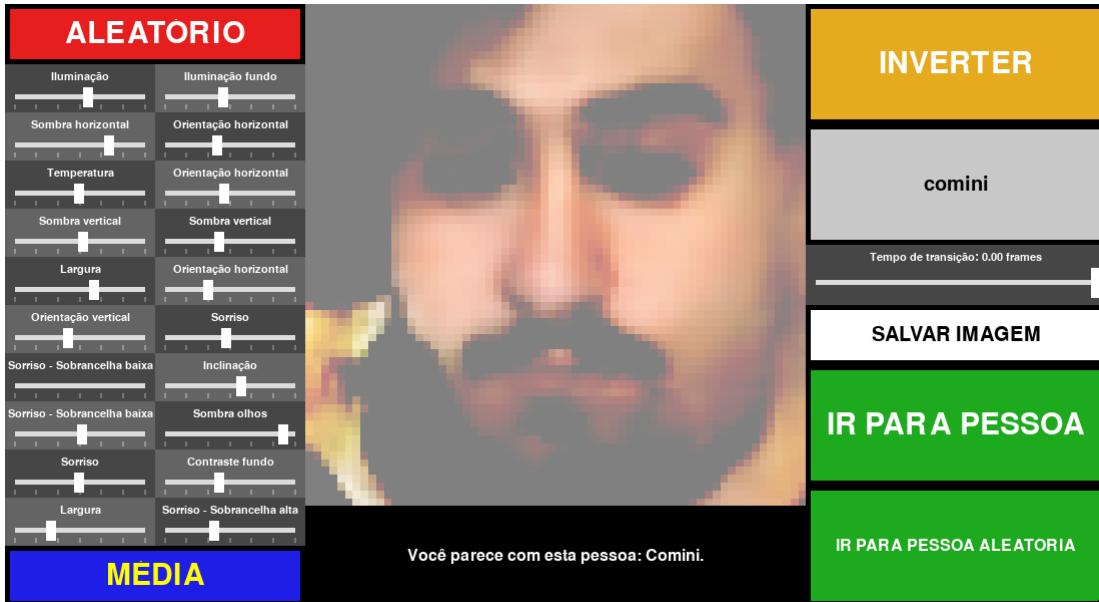
### 5.3 O *Encoder* e o PCA

Após o *auto-encoder* gerar o modelo após 182000 épocas, o encoder utilizou-se deste para codificar todo o conjunto de treinamento em um vetor denso, é neste vetor denso que se encontram os dados das faces pré-carregadas. O vetor denso foi submetido ao PCA que obteve os auto-vetores e auto-valores.

### 5.4 A aplicação final

As Figuras 15 e 16 mostram a interface da aplicação sendo utilizada para gerar faces. A aplicação possui uma interface de usuário simples,

Figura 16 – Interface da aplicação com uma face pré carregada.



Fonte: Elaborada pelo autor.

#### 5.4.1 Controles deslizáveis

Os controles deslizáveis à esquerda são a parte da aplicação responsável por modificar a imagem mostrada na interface. Cada controle deslizável corresponde a um dos trezentos componentes principais obtido pelo PCA, foram rotulados possíveis características até o trigésimo quinto componente. A imagem é recalculada a qualquer alteração da posição em qualquer um dos controles deslizáveis, passando uma sensação de “edição em tempo real” da imagem exibida.

O controle deslizável situado à direita é responsável por regular a quantidades de quadros que levará para uma face se transformar em outra após o botão “IR PARA PESSOA” ou “IR PARA PESSOA ALEATÓRIA” ser pressionado. Supõe-se que um segundo corresponda a sessenta quadros.

#### 5.4.2 Botões e Caixa de Texto

Existem alguns botões na interface que realizam diferentes operações se pressionados:

- **ALEATÓRIO:** Atribui valores aleatórios baseados em uma distribuição normal a cada um dos controles deslizáveis;
- **MÉDIA:** Atribui zero para cada um dos controles deslizáveis;
- **INVERTER:** Multiplica cada controle deslizável por  $-1$ ;

- **Digite um nome:** Ativa uma caixa de texto onde é possível inserir o nome de uma pessoa;
- **SALVAR:** Salva a imagem exibida atualmente no diretório “/imagensSalvas/imgSalva.png”;
- **IR PARA PESSOA:** Caso o nome da pessoa inserida na caixa de texto “Digite um nome” exista no conjunto de faces pré-carregadas, os controles deslizáveis se ajustaram até terem os valores de cada componente referentes à pessoa;
- **IR PARA PESSOA ALEATÓRIA:** Similar ao botão anterior porém uma pessoa do conjunto de faces pré-carregadas é aleatoriamente selecionada.

A caixa de texto situada logo abaixo à imagem exibida mostra a pessoa entre as faces pré-carregadas mais “parecida” com a disposição atual dos controles deslizáveis. Essa pessoa mais parecida é calculada a partir de uma simples distância euclidiana dos valores atuais com os valores das faces pré-carregadas, retornando a pessoa a menor distância.

## 5.5 Inserção de novas faces

A inserção de novas faces ao conjunto de faces pré-carregadas é realizada por um código a parte para simplificar e modularizar melhor cada tarefa deste trabalho. A imagem deve inicialmente ser tratada para atender os requisitos da aplicação (a exemplo da Figura 14) pelo código em “AlinhadorQualquer.py” e depois codificado junto ao conjunto de faces pré-carregadas pelo código em “EncoderQualquer.py”. Este último código retornará um novo vetor denso com as novas faces adicionadas que deverá ser carregado pela aplicação principal.

## 5.6 Passo a Passo

Para facilitar o uso de algum usuário que desejar utilizar a aplicação do zero, um breve passo-a-passo é disponibilizado a seguir:

1. **Obtenção do banco de dados:** Fica à critério do usuário. Caso utilize o mesmo banco utilizado neste trabalho (o site *Famous Birthdays*) o código em “PegaFaces.py” já realiza a obtenção e o tratamento das imagens;
2. **Tratamento do banco de dados:** O código em “AlinhadorQualquer.py” realiza esta tarefa. Caso utilize o mesmo banco utilizado neste trabalho o código em “PegaFaces.py” já realiza o tratamento nas imagens;
3. **Geração do modelo:** O código em “AutoEncoder.py” é responsável por esta tarefa, inicialmente ele salva modelos como *backups* a cada três épocas;

4. **Codificação:** A etapa que gera o vetor denso com todas as imagens é realizada pelo “Encoder.py”;
5. **PCA:** Etapa que obtém os auto-valores e auto-vetores é realizada pelo adequadamente denominado “PCA.py”.

# 6 Teste e Validação

## 6.1 Testes

Os primeiros testes realizados foram relacionados às ferramentas utilizadas para habituação com elas. Testes simples com a biblioteca *Face Recognition* e com a plataforma TensorFlow.

O teste subsequente foi o teste realizado para definir qual a profundidade do *auto-encoder*, quantas camadas ele deve ter de codificação e de decodificação.

Inicialmente foi testado um *auto-encoder* simples, com apenas uma camada oculta, isto foi devido ao uso da mentalidade da “Navalha de Occam”: geralmente as ideias mais simples são as mais adequadas como solução de problemas. Depois de cerca de cinco mil épocas de treinamento a qualidade dos modelos não foi satisfatória, o que levou à adição de mais uma camada de codificação e uma camada de decodificação ao *auto-encoder*. Este processo se repetiu até que quatro camadas de codificação e quatro de decodificação resultaram em modelos satisfatórios.

O teste final foi já na aplicação quase finalizada, onde foi necessária a utilização da aplicação para poder rotular os controles deslizáveis com as possíveis características que elas alteram.

## 6.2 Validação

A validação deste trabalho foi mais qualitativa: a aplicação foi mostrada a colegas de turma e amigos que puderam utilizá-la.

Foi observado que ao atribuir valores próximos às extremidades em vários controles deslizáveis as faces se tornavam muito distorcidas, algumas vezes não era possível reconhecer uma face na imagem, porém caso a atribuição seja de valores que se assemelham à uma distribuição normal, as faces geradas são mais reconhecíveis e realistas. A Figura 17 apresenta algumas faces geradas pela aplicação por meio do botão “ALEATÓRIO”.

Figura 17 – Exemplos de faces geradas por meio do botão “ALEATÓRIO” da aplicação.



Fonte: Elaborada pelo autor.

# 7 Conclusão

Este trabalho teve como objetivo o desenvolvimento de uma aplicação leve que pudesse gerar e editar faces humanas. Para realizar tal feito foram utilizadas técnicas de aprendizado de máquina e de estatística, necessitando um estudo sobre os campos envolvidos e alguns campos além para a certificação de que foi estudado o necessário. Para isso foram utilizados livros, artigos e documentação de ferramentas para o desenvolvimento, como o TensorFlow e a linguagem Python.

A aplicação final obteve sucesso em gerar e editar faces a partir do conjunto de treinamento e com algumas observações interessantes, como o fato de que vários componentes obtidos pelo PCA são responsáveis por alterar o sorriso nas faces, alguns alterando a sobrancelha, outros não. Ressalta-se o fato de que a máquina “aprendeu” isso sozinha, que existe uma relação entre o sorriso de uma pessoa com a posição de suas sobrancelhas, a partir de números.

Também foi alcançado o objetivo da aplicação ser leve, visto que todo o processo de desenvolvimento, treinamento e utilização foi realizado em um computador comum, com um processador Intel i5, sem placa de vídeo dedicada e com oito gigabytes de memória RAM.

No entanto é possível perceber uma certa quantidade de ruído nas imagens reconstruídas causados pela redução de características para apenas 300, além das baixas dimensões das imagens (64x64 pixels) pelo fato de o treinamento do *auto-encoder* com imagens maiores exigiria um maior poder computacional, certamente as imagens seriam mais interessantes para o usuário se estivessem em uma resolução maior.

## 7.1 Trabalhos Futuros

Para trabalhos futuros pode-se sugerir o uso do VGGFace que por motivos destacados na Seção 4.2 não pôde ser utilizado neste trabalho, a implementação deste mesmo trabalho porém sem a utilização de um *auto-encoder* também pode vir a ser interessante, possivelmente com resultados parecidos só que com mais ruído no momento da reconstrução das faces. Também pode ser interessante realizar trabalhos similares a este utilizando uma GAN, visto que no momento as GANs apenas geram as imagens sem poder editá-las depois.

Outra sugestão é utilizar imagens maiores para o treinamento do *auto-encoder* juntamente com um aumento de características no espaço latente, melhorando a resolução e qualidade das imagens reconstruídas.

# Referências

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIÉGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <<https://www.tensorflow.org/>>. Acesso em: 28 out. 2019.

AFIFI, M.; ABDELHAMED, A. Afif4: Deep gender classification based on adaboost-based fusion of isolated facial features and foggy faces. *Journal of Visual Communication and Image Representation*, Elsevier, 2019. Disponível em: <<https://arxiv.org/abs/1706.04277>>. Acesso em: 12 out. 2019.

ALPAYDIN, E. *Introduction to Machine Learning*. [s.n.], 2009. Disponível em: <[https://books.google.com.br/books?hl=pt-BR&lr=&id=TtrxCwAAQBAJ&oi=fnd&pg=PR7&dq=machine+learning+introduction&ots=T5elKJ-7rR&sig=PSu1GHskl1jQwvBmce\\_4mdecQc4#v=onepage&q=introduction&f=false](https://books.google.com.br/books?hl=pt-BR&lr=&id=TtrxCwAAQBAJ&oi=fnd&pg=PR7&dq=machine+learning+introduction&ots=T5elKJ-7rR&sig=PSu1GHskl1jQwvBmce_4mdecQc4#v=onepage&q=introduction&f=false)>. Acesso em: 02 out. 2019.

BUDUMA, N. *Fundamentals of Deep Learning*. [S.I.]: O'Riley Media, Inc, 2017. ISBN 978-1-491-92561-4.

CAO, Q.; SHEN, L.; XIE, W.; PARKHI, O. M.; ZISSERMAN, A. Vggface2: A dataset for recognising face across pose and age. *International Conference on Automatic Face and Gesture Recognition*, 2018.

DOERSCH, C. Tutorial on variational autoencoders. 2016. Disponível em: <<https://arxiv.org/abs/1606.05908>>. Acesso em: 20 mar. 2019.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. [S.I.]: Pearson Education Inc., 2008.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Disponível em: <<http://www.deeplearningbook.org>>. Acesso em: 12 out. 2019.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S. Generative adversarial nets. *Neural Information Processing Systems Conference*, 2014. Disponível em: <<https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>>. Acesso em: 02 out. 2019.

HUANG, G. B.; RAMESH, M.; BERG, T.; LEARNED-MILLER, E. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. [S.I.], 2007. Disponível em: <<http://vis-www.cs.umass.edu/lfw/>>. Acesso em: 12 out. 2019.

KARN, U. *A Quick Introduction to Neural Networks*. 2016. Disponível em: <<https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks>>. Acesso em: 11 mar. 2019.

- KARRAS, T.; LAINE, S.; AILA, T. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. Disponível em: <<http://arxiv.org/abs/1812.04948>>. Acesso em: 11 mar. 2019.
- LADJAL, S.; NEWSON, A.; PHAM, C. A pca-like autoencoder. *CoRR*, abs/1904.01277, 2019. Disponível em: <<http://arxiv.org/abs/1904.01277>>. Acesso em: 12 out. 2019.
- LI, Y.; MIN, M. R.; SHEN, D.; CARLSON, D.; CARIN, L. Video generation from text. 2018. Disponível em: <<https://www.aaai.org/GuideBook2018/16152-72279-GB.pdf>>. Acesso em: 11 mar. 2019.
- RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.I.]: Prentice Hall, 2009.
- WOLD, S.; ESBENSEN, K.; GELADI, P. Principal component analysis. *Chemometrics and intelligent laboratory systems*, Elsevier, v. 2, n. 1-3, p. 37–52, 1987. Disponível em: <<http://pzs.dstu.dp.ua/DataMining/pca/bibl/Principal\%20components%20analysis.pdf>>. Acesso em: 12 out. 2019.