

Tatiana Holanda Silva

Reconhecimento Facial com Identificação de Indivíduos

São Paulo – Brasil

2020, v-1.0

Jatiano Holand Silva

Tatiana Holanda Silva

Reconhecimento Facial com Identificação de Indivíduos

Relatório final de pesquisa de Iniciação Científica – Programa PIBIC-CNPq.

Pontifícia Universidade Católica de São Paulo – PUC/SP

Faculdade de Ciências Exatas e Tecnologia

Programa Institucional de Bolsas de Iniciação Científica

Orientador: Prof. Dr. Alexandre Barbosa de Lima

São Paulo – Brasil

2020, v-1.0

Agradecimentos

Agradeço ao Programa Institucional de Bolsas de Iniciação Científica (PIBIC) do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por ter financiado esta pesquisa por meio da concessão de bolsa de estudos.

Resumo

Nas últimas décadas, o interesse por tecnologias de biometria que utilizam reconhecimento facial, reconhecimento de voz, escaneamento de retina e impressão digital têm crescido rapidamente, tendo em vista o desenvolvimento de uma miríade de técnicas de reconhecimento, as quais foram viabilizadas pelo aumento exponencial do poder computacional desde a década de 1960.

De forma geral, reconhecimento facial refere-se ao problema de identificar ou verificar pessoas em fotografias ou vídeos.

A área de pesquisa em reconhecimento facial integra o escopo de um campo do conhecimento denominado visão computacional. Apesar de a visão ser algo trivial para seres humanos, a área de pesquisa em visão computacional continua a demandar um grande esforço da comunidade científica há várias décadas, porque implementar um sistema computacional que simule a percepção visual humana ainda é um problema a ser resolvido.

As redes neurais convolucionais constituem uma tecnologia disruptiva em inteligência artificial, pois quebraram recordes de desempenho em vários domínios tais como texto, voz e vídeo/imagem desde o início da década de 2010.

O objetivo principal deste trabalho é investigar a aplicação das redes convolucionais para reconhecimento facial com identificação de indivíduos em imagens.

Os demais resultados obtidos, tais como reconhecer dígitos manuscritos, um problema clássico em visão computacional, confirmam que as redes convolucionais realmente são o estado da arte nesta área de pesquisa. **PIBIC-CNPq**

Palavras-chaves: inteligência artificial. aprendizado de máquina. *deep learning*. redes convolucionais.

Lista de ilustrações

Figura 1 – Sistema de reconhecimento facial.	17
Figura 2 – Arquitetura da rede DeepFace.	20
Figura 3 – Visualização das curvas de aprendizado pelo TensorBoard.	24
Figura 4 – Anaconda <i>navigator</i>	25
Figura 5 – Ambiente de programação Spyder.	26
Figura 6 – Repositório GitHub.	26
Figura 7 – Ambiente de programação Colab.	27
Figura 8 – Visualização do código no GitHub.	27
Figura 9 – Paradigmas de programação.	31
Figura 10 – Modelo de McCulloch e Pitts	35
Figura 11 – função Relu.	36
Figura 12 – Exemplo de uma rede neural densamente conectada com uma camada oculta (<i>hidden layer</i>).	37
Figura 13 – <i>Deep Learning</i> (DL) com seis camadas.	39
Figura 14 – relação entre Inteligência Artificial (IA), <i>Machine Learning</i> (ML) e DL.	39
Figura 15 – Como funciona o DL.	40
Figura 16 – componentes de uma camada convolucional.	42
Figura 17 – o fenômeno do <i>overfitting</i>	46
Figura 18 – Rede MNIST2: curva de aprendizado – erro.	49
Figura 19 – Rede MNIST2: curva de aprendizado – acurácia.	49
Figura 20 – Rede MNIST4: curva de aprendizado – erro.	49
Figura 21 – Rede MNIST4: curva de aprendizado – acurácia.	50
Figura 22 – Rede MNIST6a: curva de aprendizado – erro.	50
Figura 23 – Rede MNIST6a: curva de aprendizado – acurácia.	50
Figura 24 – Rede MNIST6b: curva de aprendizado – erro.	51
Figura 25 – Rede MNIST6b: curva de aprendizado – acurácia.	51
Figura 26 – Rede MNIST8: curva de aprendizado – erro.	51
Figura 27 – Rede MNIST8: curva de aprendizado – acurácia.	52
Figura 28 – Rede MNISTCNN: curva de aprendizado – erro.	52
Figura 29 – Rede MNISTCNN: curva de aprendizado – acurácia.	52
Figura 30 – arquitetura FaceNet.	53
Figura 31 – função de perda tríplice.	54
Figura 32 – Fotos cadastradas: Ozzy Osbourne e Brad Pitt.	54
Figura 33 – Reconhecimento facial.	55

Lista de tabelas

Tabela 1 – Cronograma de atividades.	28
Tabela 2 – Redes neurais treinadas para reconhecimento de dígitos manuscritos. A época de treinamento é notada por t	45
Tabela 3 – Desempenho das redes neurais.	48

LISTA DE ABREVIATURAS

Adam	<i>Adaptive Moment Estimation</i>
API	<i>Application Programming Interface</i>
BN	<i>Batch Normalization</i>
CIA	<i>Central Intelligence Agency</i>
CNN	<i>Convolutional Neural Network</i>
CNTK	Microsoft Cognitive Toolkit
CPU	<i>Central Processing Unit</i>
CV	<i>Computer Vision</i>
DCNN	<i>Deep Convolutional Neural Network</i>
DeepID	<i>Deep hidden IDentity features</i>
DL	<i>Deep Learning</i>
DNN	<i>Deep Neural Network</i>
ENIAC	<i>Electronic Numerical Integrator and Computer</i>
FR	<i>Face Recognition</i>
GPU	<i>Graphics Processing Unit</i>
IA	Inteligência Artificial
IDE	<i>Integrated Development Environment</i>
ILSVRC	<i>ImageNet Large Scale Visual Recognition Challenge</i>
LBP	<i>Local Binary Pattern</i>
LE	<i>Learning-Based Descriptor</i>
LFW	<i>Labeled Faces in the Wild</i>
MCP	Modelo de McCulloch e Pitts
ML	<i>Machine Learning</i>
MSE	<i>Mean Square Error</i>

MNIST *The MNIST database of handwritten digits*

NIST United States National Institute of Science and Technology

PCA *Principal Component Analysis*

RLS Regressão Linear Simples

RLM Regressão Linear Múltipla

RNA Rede Neural Artificial

RNPC Rede Neural Profunda Convolutacional

SGD *Stochastic Gradient Descent*

SVM *Support Vector Machines*

TPU *Tensor Processing Unit*

Sumário

1	INTRODUÇÃO	15
	<i>Este capítulo apresenta a motivação, os objetivos e os resultados deste trabalho.</i>	
1.1	Motivação	15
1.2	Escopo e Resultados	21
1.3	Método do Trabalho	23
1.4	Relatório de Atividades	25
1.5	Estrutura do Relatório	25
2	APRENDIZADO DE MÁQUINA	29
	<i>Este capítulo apresenta o machine learning e qual é a sua relação com o campo da inteligência artificial.</i>	
2.1	Machine Learning	29
2.2	A Noção de Redes Neurais	34
2.3	Deep Learning	36
2.4	Redes Convolucionais	41
3	EXPERIMENTOS COMPUTACIONAIS	45
	<i>Este capítulo apresenta os resultados obtidos em Computer Vision (CV) e Face Recognition (FR) por meio de simulações.</i>	
3.1	Reconhecimento de Dígitos Manuscritos	45
3.2	Reconhecimento Facial	48
4	CONCLUSÕES E TRABALHOS FUTUROS	57
	<i>Este capítulo sumariza os resultados obtidos e lista algumas sugestões para trabalhos futuros.</i>	
	REFERÊNCIAS	59
	APÊNDICES	65
	APÊNDICE A – CÓDIGOS PYTHON	67
A.1	Reconhecimento de Dígitos Manuscritos	67
A.2	Implementação do FaceNet em Python Usando TensorFlow 2 e Keras	90

1 Introdução

Este capítulo apresenta a motivação, os objetivos e os resultados deste trabalho.

1.1 Motivação

Nas últimas décadas, o interesse por tecnologias de biometria¹ que utilizam reconhecimento facial (FR), reconhecimento de voz, escaneamento de retina e impressão digital (CHHAOUI et al., 2016), (KORTLI et al., 2020) têm crescido rapidamente, tendo em vista o desenvolvimento de uma miríade de técnicas de reconhecimento, as quais foram viabilizadas pelo aumento exponencial do poder computacional desde a década de 1960 (MACK, 2011), a conhecida “Lei de Moore”².

De forma geral, FR refere-se ao problema de identificar ou verificar pessoas em fotografias ou vídeos (BROWNLEE, 2020) [p. 455].

A tecnologia de FR é não intrusiva (a biometria por escaneamento de retina e a impressão digital são consideradas intrusivas) e tem sido utilizada e/ou testada em aplicações como vigilância por vídeo, identificação criminal, controle de acesso e veículos autônomos³ (KORTLI et al., 2020).

A área de pesquisa em FR integra o escopo de um campo do conhecimento denominado visão computacional (CV). Segundo (BROWNLEE, 2020)[p. 3, 4],

“A Visão Computacional, frequentemente abreviada como CV, é definida como um campo de estudo que busca desenvolver técnicas para ajudar os computadores a ver e entender o conteúdo de imagens digitais, como fotos e vídeos”. (...) É um campo multidisciplinar que poderia ser amplamente chamado de subcampo da inteligência artificial e aprendizado de máquina, que pode envolver o uso de métodos especializados e fazer uso de algoritmos gerais de aprendizado.” (tradução livre)

Nilsson (NILSSON, 2010) [p. 169]⁴ afirma que:

¹ O dicionário Houaiss da língua portuguesa define biometria como o “estudo das medidas e de estruturas e órgãos de seres vivos”.

² Gordon Moore, um dos fundadores da Intel, fabricante de circuitos integrados, enunciou, em 1965, que o desempenho dos microchips produzidos em massa dobraria a cada doze meses (MOORE, 1965). A indústria constatou que o número de componentes por chip dobra a aproximadamente cada dezoito meses (MACK, 2011).

³ Por exemplo, um veículo poderia entrar em modo autônomo se o sistema de reconhecimento facial detectasse que o motorista dormiu durante a condução em uma estrada (NAGENDRAN; KOLHE, 2018).

⁴ Versão web do livro.

“As pessoas não cegas obtêm muitas informações através da visão. Essa parte da IA [Inteligência Artificial] chamada “visão computacional” (ou, às vezes, “visão de máquina”) trata de dar aos computadores essa capacidade. A maioria dos trabalhos de visão computacional é baseada no processamento de imagens bidimensionais coletadas de um mundo tridimensional - imagens coletadas por uma ou mais câmeras de televisão, por exemplo (...).” (tradução livre)

De acordo com (BROWNLEE, 2020), CV e processamento de imagens são áreas distintas de investigação, uma vez que esta última tem como objetivo a criação de uma nova imagem a partir de uma imagem existente (BROWNLEE, 2020). O processamento de sinais tem como foco a representação, a transformação e a manipulação de sinais (OPPENHEIM; SCHAFER, 2019), (SOLOMON; BRECKON, 2011). Sendo o processamento de imagens uma sub-área de processamento digital de sinais, “entender” o conteúdo de uma imagem não faz parte do escopo dessa área do conhecimento.

Apesar de a visão ser algo trivial para seres humanos, a área de pesquisa em CV continua a demandar um grande esforço da comunidade científica há várias décadas, porque implementar um sistema computacional que simule a percepção visual humana ainda é um problema a ser resolvido (BROWNLEE, 2020).

Um sistema automático de FR pode ser dividido em quatro estágios (Fig. 1) (LI; JAIN, 2011 apud BROWNLEE, 2020, p. 449):

1. **Detecção de face:** localização de uma ou mais faces na imagem e marcação com caixa delimitadora (*bounding box*).
2. **Alinhamento de face:** normalização da face de forma a ser consistente com o banco de dados em termos de geometria e fotometria.
3. **Extração de características:** extrair características ou *features* da face que podem ser usadas para a tarefa de reconhecimento.
4. **Reconhecimento de face:** realizar a correspondência da face com uma ou mais faces conhecidas em um banco de dados preparado.

O livro (LI; JAIN, 2011 apud BROWNLEE, 2020, p. 451) descreve dois modos principais de FR:

- **Verificação de face** ou **Autenticação:** mapeamento de um-para-um de uma dada face contra uma identidade conhecida (ex.: esta é a pessoa?)
- **Identificação de face** ou **Reconhecimento:** mapeamento de um-para-vários de uma dada face contra um banco de dados de faces (ex.: quem é esta pessoa?)

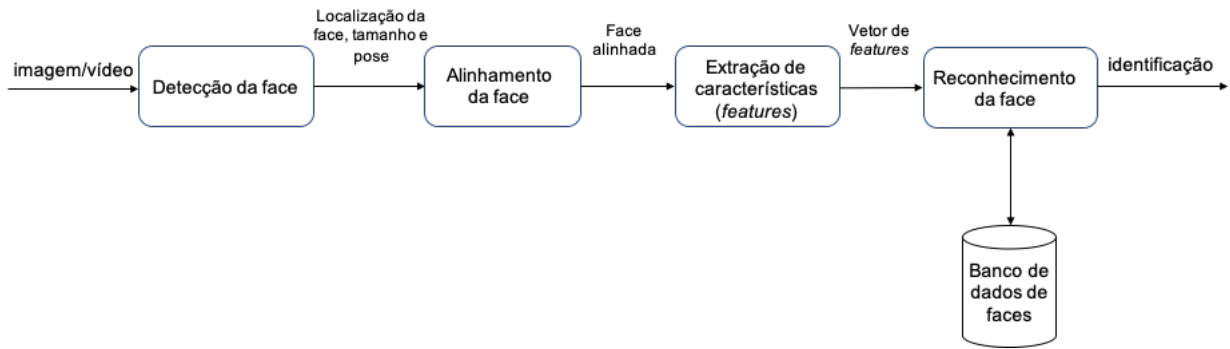


Figura 1 – Sistema de reconhecimento facial.

O artigo de Wang e Deng (WANG; DENG, 2018) apresenta de forma bastante detalhada a evolução das técnicas empregadas em FR, assim como (CHHAOUI et al., 2016) e (KORTLI et al., 2020). Segue-se uma revisão extremamente resumida e não exaustiva da literatura e da evolução das técnicas relevantes para a área de FR, uma vez que, conforme será justificado mais adiante, este trabalho tem como foco o uso do DL em FR, pois é o estado da arte em CV.

No início dos anos 1960 (NILSSON, 2010) [p. 9], Woodrow W. Bledsoe, Charles Bisson e Helen Chan Wolf, da empresa Panoramic Research, localizada em Palo Alto (Califórnia, EUA), desenvolveram técnicas de FR financiados por projetos da *Central Intelligence Agency* (CIA).

Lawrence G. Roberts⁵ escreveu um programa de computador no início da década de 1960 capaz de identificar objetos em imagens digitais em escala de cinza⁶ e de determinar as respectivas orientação e posição espaciais. A tese de doutorado de Roberts introduziu ideias-chave como detecção de aresta e correspondência baseada em modelo (ROBERTS, 1963), (RUSSEL; NORVIG, 2013) [p. 841].

Em 1970, Michael D. Kelly⁷ escreveu um programa que era capaz de detectar de forma automática características faciais em fotos de forma a identificar pessoas (KELLY, 1970).

Outro pioneiro em CV é o prof. Takeo Kanade, da Carnegie Mellon University, um

⁵ Roberts era aluno de doutorado do Lincoln Laboratory do MIT nesta época.

⁶ Segundo (SOLOMON; BRECKON, 2011) [p. 1, 2], “Uma imagem digital pode ser considerada uma representação discreta de dados que possuem informações espaciais (*layout*) e intensidade (cores). (...) A imagem bidimensional (2D) $I(m, n)$ representa a resposta de algum sensor (ou simplesmente um valor de algum interesse) em uma série de posições fixas ($m = 1, 2, \dots, M; n = 1, 2, \dots, N$) em coordenadas cartesianas 2-D e é derivada do sinal espacial contínuo 2-D $I(x, y)$ através de um processo de amostragem frequentemente chamado de discretização. (...) Os índices m e n designam, respectivamente, as linhas e colunas da imagem. Os elementos individuais da imagem ou pixels são referidos pelos seus índices 2-D (m, n) . (...) Uma imagem contém um ou mais canais de cores que definem a intensidade ou a cor em um local de pixel específico $I(m, n)$ (...) O mapa de cores mais comum é a escala cinza (*greyscale*), que atribui todos os tons de cinza desde preto (zero) a branco (máximo) de acordo com o nível do sinal.” (tradução livre)

⁷ À época um aluno de doutorado em Stanford.

pesquisador que ainda é bastante ativo e que tem dado contribuições fundamentais nesta área desde a defesa de sua tese de doutorado em CV em 1973 (KANADE, 1973).

No início dos anos 1990, a abordagem *Eingeface* (TURK; PENTLAND, 1991), que é baseada na Análise de Componentes Principais⁸ (PCA) revolucionou a área de FR. Contudo, o *Eingeface*, bem como os métodos ditos holísticos ou globais⁹, é sensível a variações de pose, iluminação, expressão facial e orientação (CHHAOUI et al., 2016). O método *Eingeface* possui uma acurácia¹⁰ de aproximadamente 60% quando testado contra a base de dados *Labeled Faces in the Wild* (LFW) (HUANG et al., 2008), que é considerada a base de dados de referência para testes de FR pela comunidade de visão computacional. Note-se que o desempenho humano é de 97,53% para a LFW segundo (WANG; DENG, 2018).

Em meados da década de 1990, Yann LeCun e outros pesquisadores demonstraram a importância das redes neurais convolucionais (*Convolutional Neural Network* (CNN)) em aplicações como reconhecimento de dígitos manuscritos (LECUN et al., 1995). Algumas das conclusões deste artigo são “proféticas” e foram comprovadas em 2012, quando a *Deep Convolutional Neural Network* (DCNN) (Rede Neural Profunda Convolucional (RNPC)) AlexNet¹¹ (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) ganhou a competição *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), a “olimpíada” da CV, reduzindo o erro de 26% para 15%, inaugurando a era da supremacia do DL¹² em CV e outras aplicações (RUSSAKOVSKY et al., 2015)¹³. Segue-se um excerto da “profecia” de (LECUN et al., 1995):

“As redes convolucionais são particularmente adequadas para reconhecer ou rejeitar formas do mundo real com tamanho, posição e orientação bastante variadas, como as que são tipicamente produzidas por segmentadores heurísticos em sistemas de reconhecimento do mundo real. (...) Quando há muitos dados disponíveis, vários métodos podem atingir uma precisão respeitável.

⁸ O *Principal Component Analysis* (PCA) é considerado uma transformação linear ótima que pode ser usada em processamento de imagens.

⁹ Segundo Chihaoui et al (CHHAOUI et al., 2016), “Nestas abordagens, também chamadas de métodos baseados em aparência, as imagens da face são tratadas globalmente, ou seja, não há necessidade de extrair pontos característicos ou regiões faciais (boca, olhos etc.). Assim, uma imagem de rosto é representada por uma matriz de pixels, e essa matriz é frequentemente transformada em vetores de pixels para facilitar sua manipulação” (tradução livre).

¹⁰ A definição desta figura de mérito será vista no Cap. 3.

¹¹ A AlexNet foi projetada por dois alunos de pós-graduação da Universidade de Toronto (Alex Krizhevsky e Ilya Sutskever) sob a orientação professor Geoffrey E. Hinton.

¹² Os “padrinhos” do DL, Yoshua Bengio, Geoffrey Hinton e Yann LeCun, foram agraciados com o prêmio Turing de 2018, que é considerado o “prêmio Nobel” da Ciência da Computação (<<https://amturing.acm.org/byyear.cfm>>).

¹³ Segundo (RUSSAKOVSKY et al., 2015), “O Desafio de reconhecimento visual de grande escala do ImageNet é uma referência na classificação e detecção de categorias de objetos em centenas de categorias de objetos e milhões de imagens. O desafio foi realizado anualmente a partir de 2010 até o presente momento, atraindo a participação de mais de cinquenta instituições.”

Embora os métodos que usam redes neurais exijam um tempo de treinamento considerável, redes treinadas são muito mais rápidas e requerem muito menos espaço em técnicas baseadas em memória. **A vantagem das redes neurais se tornará mais impressionante à medida que os bancos de dados de treinamento continuam a aumentar de tamanho**”. (tradução livre - grifo nosso)

De fato, a literatura tem demonstrado desde 2012 que os métodos de aprendizado profundo (DL) superam as técnicas de aprendizado de máquina de última geração em vários campos, com a CV sendo um dos casos mais importantes (VOULODIMOS et al., 2018).

No final da década de 1990 e início dos anos 2000, destacam-se os trabalhos de Rowley, Baluja e Kanade (Rowley; Baluja; Kanade, 1998), que propôs um algoritmo baseado em redes neurais para detecção de vistas frontais e verticais dos rostos em imagens em escala de cinza, e o algoritmo de detecção de faces proposto por Viola e Jones (VIOLA; JONES, 2001), que é baseado em três conceitos: integral de imagem, treinamento de classificadores usando *boosting* e seu posterior uso em cascata.

Na década de 2000, surgiram os métodos de FR Gabor (LIU; WECHSLER, 2002) e *Local Binary Pattern* (LBP) (AHONEN; HADID; PIETIKAINEN, 2006) baseados em características locais da imagem. Tais métodos apresentam um desempenho na faixa de 70% quando testados contra a base de dados LFW.

Note-se que as *Support Vector Machines* (SVM)¹⁴ (BOSER; GUYON; VAPNIK, 1992) dominaram a área de CV nas décadas de 1990 e 2000 até o surgimento da AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) no início dos anos 2010.

Segundo (VOULODIMOS et al., 2018),

“Um dos avanços mais substanciais na aprendizagem profunda ocorreu em 2006, quando Hinton et al. (HINTON; OSINDERO; TEH, 2006) introduziram a Deep Belief Network, com várias camadas de máquinas restritas de Boltzmann, treinando avidamente uma camada de cada vez, de maneira não-supervisionada. Orientar o treinamento de níveis intermediários de representação usando aprendizado não supervisionado, realizado localmente em cada nível, foi o principal princípio por trás de uma série de desenvolvimentos que provocaram o aumento da última década em arquiteturas profundas e algoritmos de aprendizado profundo.

Entre os fatores mais importantes que contribuíram para o grande impulso

¹⁴ O SVM é um algoritmo de classificação de padrões desenvolvido por V. Vapnik e equipe no AT&T Bell Labs no início dos anos 1990.

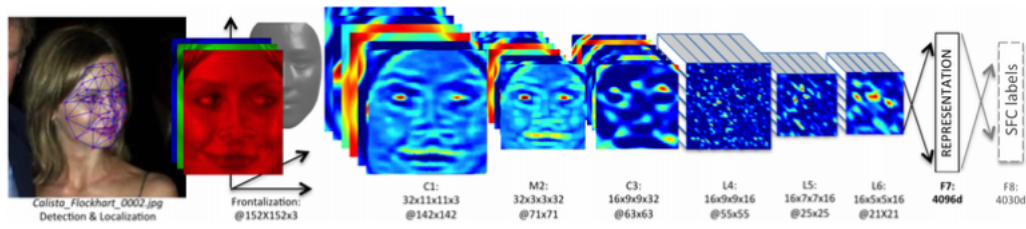


Figura 2 – Arquitetura da rede DeepFace.

do aprendizado profundo estão o aparecimento de grandes conjuntos de dados rotulados de alta qualidade, disponíveis ao público, juntamente com o empoderamento da computação paralela usando *Graphics Processing Unit* (GPU), que permitiu a transição da *Central Processing Unit* (CPU) para a GPU permitindo assim uma aceleração significativa no treinamento de modelos de aprendizado profundo.” (tradução livre)

No início dos anos 2010, foram propostos métodos que usam descritores locais baseados em aprendizado (ou *Learning-Based Descriptor* (LE)) (CAO et al., 2010), cujo desempenho é de aproximadamente 82% para a LFW.

Em 2014, a arquitetura de DL DeepFace (TAIGMAN et al., 2014), baseada DCNN, atingiu a acurácia de 97,35%, aproximando-se do desempenho humano (97,53%), por meio do treinamento de um modelo de DCNN de nove (9) camadas com mais de 120 milhões de parâmetros utilizando quatro (4) milhões de imagens faciais para o aprendizado de máquina (ML) supervisionado (o conceito de ML será visto no Cap. 2). A Fig. 2 (retirada de (TAIGMAN et al., 2014)) mostra a arquitetura da rede DeepFace. As camadas da rede serão abordadas no Capítulo 2.

O advento do DeepFace alterou o foco da pesquisa na área de FR para abordagens baseadas em DCNN, de forma que a acurácia em FR foi dramaticamente aumentada para cerca de 99,80% (desempenho sobre-humano) em apenas três anos (WANG; DENG, 2018).

Segundo (BROWNLEE, 2020) [p. 452], existem, além do DeepFace, três outros sistemas baseados em DL que impulsionaram o campo de FR com suas inovações, a saber: a série de sistemas *Deep hidden IDentity features* (DeepID) (Sun; Wang; Tang, 2014), VGGFace (PARKHI; VEDALDI; ZISSERMAN, 2015) e FaceNet (SCHROFF; KALENICHENKO; PHILBIN, 2015).

“Os sistemas DeepID estavam entre os primeiros modelos de aprendizado profundo a obter um desempenho melhor que o humano na tarefa, por exemplo, O DeepID2 alcançou 99,15% no conjunto de dados Labeled Faces in the Wild (LFW), que é um desempenho melhor que o humano de 97,53%. Sistemas

subsequentes como FaceNet e VGGFace obtiveram resultados ainda melhores” (BROWNLIE, 2020)[p. 453]. (tradução livre)

Por fim, é importante ressaltar algumas limitações do DL (RAHIMI, 2017), (ADCOCK; DEXTER, 2020):

- Ainda não existe uma teoria matemática bem estabelecida das redes neurais profundas. Por que funcionam bem?
- Não há garantia de que o mínimo global da função custo seja atingido; e
- Não há critérios de projeto.

De acordo com (CHOLLET, 2018) [p. 329],

“(...) até agora, o único sucesso real da aprendizagem profunda foi a capacidade de mapear um espaço X para um espaço Y usando uma transformação geométrica contínua, dada a utilização de grandes quantidades de dados anotados por humanos. Fazer isso bem é um divisor de águas para essencialmente todos os setores, mas ainda está muito longe da IA em nível humano.

Para superar algumas das limitações que discutimos e criar a IA que possa competir com o cérebro humano, precisamos nos afastar dos mapeamentos diretos de entrada para saída e adotar uma abordagem que utilize raciocínio e abstração.” (tradução livre)

1.2 Escopo e Resultados

O objetivo principal deste trabalho é investigar a aplicação de sistemas de DCNN para FR em imagens.

Os objetivos específicos são os seguintes:

- reproduzir alguns resultados clássicos em CV relacionados ao problema do reconhecimento de dígitos manuscritos (CUN et al., 1990) usando técnicas de DL;
- familiarização com as técnicas e métodos de pesquisa em FR em imagens usando sistemas de DCNN; e
- implementar algoritmos de FR usando sistemas de DCNN pré-treinados em bases de dados de grande escala¹⁵. Segundo (CHOLLET, 2018) [p. 143], a representação aprendida pela base convolucional do modelo é reutilizável porque as DCNN possuem um alto poder de generalização em CV.

¹⁵ Estas bases de dados pertencem a grandes empresas de tecnologia como Google e Facebook.

As seguintes questões foram examinadas no decorrer da pesquisa:

- linguagem de programação a ser utilizada (Python? MATLAB/OCTAVE? R?);
- seleção de biblioteca de [DL](#) e *Application Programming Interface* ([API](#));
- escolha do ambiente de programação;
- otimização para o treinamento de modelos profundos baseada no algoritmo da descida pelo gradiente estocástico ou *Stochastic Gradient Descent* ([SGD](#))¹⁶; e
- seleção de arquiteturas de [DCNN](#) existentes adequadas para o tema da pesquisa.

Os principais resultados deste trabalho são os seguintes:

- aprendizado do modelo não linear de um neurônio artificial, o qual possui sinais de entrada, pesos sinápticos, somador, função de ativação não linear e sinal de saída;
- aprendizado da linguagem de programação Python em conjunto com as bibliotecas TensorFlow 1.0/2.0 e [API Keras](#) para a codificação de modelos de [DL](#);
- aprendizado de alguns conceitos matemáticos básicos em [DL](#): tensores, operações com tensores e otimização com algoritmos de gradiente estocástico;
- aprendizado das arquiteturas de redes neurais *feedforward* (sem laços ou “loops” internos) relevantes para o tema: camada única, múltiplas camadas (camadas totalmente conectadas) e convolucionais (possui camadas convolucionais, camadas de *pooling* ou sub-amostragem e camadas totalmente conectadas)¹⁷;
- aprendizado dos algoritmos de otimização mais usados em [DL](#): [SGD](#), AdaGrad, RMSprop, Adam, etc. ([GOODFELLOW](#); [BENGIO](#); [COURVILLE](#), 2016) [Cap. 8];
- aprendizado do algoritmo de retropropagação dos erros (*backpropagation*) para estimação do gradiente da função custo da rede;
- aprendizado da metodologia de avaliação de modelos de [ML](#), que envolve a divisão dos dados em três conjunto (treinamento, validação e teste) com o objetivo de obter modelos com poder de generalização, o que requer o uso de técnicas de mitigação de *overfitting* (sobre-parametrização) do modelo;
- reconhecimento de dígitos manuscritos usando redes densas e convolucionais;
- seleção de arquiteturas de [DCNN](#) existentes adequadas para [FR](#); e

¹⁶ É comum a literatura referir-se ao [SGD](#) como **gradiente estocástico**.

¹⁷ Essas arquiteturas serão vistas no Cap. 2.

- portabilidade do código fonte Python de FR de (BUSSON; et al, 2018), que usa o modelo pré-treinado FaceNet (SCHROFF; KALENICHENKO; PHILBIN, 2015), da plataforma TensorFlow 1.14 para TensorFlow 2 (GOOGLE, 2020b). O código encontra-se listado no Apêndice A.

1.3 Método do Trabalho

O cerne do trabalho envolve a implementação de modelos de DL estado da arte para FR. O resultado esperado é a confirmação dos resultados relatados pela literatura.

A revisão da literatura indicou que a linguagem de programação Python é a mais indicada para a pesquisa em FR usando a abordagem de DL, uma vez que possui vários *frameworks* gratuitos e de código aberto (*opensource*) para DL e que essa linguagem tem sido muito usada pela comunidade de ML (CHOLLET, 2018), (GULLI; KAPOOR; PAL, 2019), (SKANSI, 2018). Outras linguagens e ambientes também possuem bibliotecas de DL como MATLAB (MATHWORKS, 2020) e R (CHOLLET; ALLAIRE, 2017). O MATLAB possui como grande desvantagem o fato de não ser gratuito, não obstante ser amplamente usado no ensino, pesquisa e desenvolvimento em Engenharia. A PUC-SP, por exemplo, possui licenças do MATLAB, mas elas não incluem o *toolbox* de DL. O ambiente gratuito de programação científica GNU Octave (Octave Forge, 2020) não possui pacotes de DL.

Frameworks open source para ML como TensorFlow (GOOGLE, 2020b), PyTorch (PASZKE et al., 2020), MXNet (APACHE, 2020) e Microsoft Cognitive Toolkit (CNTK) (MICROSOFT, 2020)¹⁸ têm se destacado nos últimos anos. Contudo, o TensorFlow e a API Keras¹⁹ oferecem alguns diferenciais, tais como a possibilidade de execução dos códigos no Google Colaboratory, ou “Colab” (GOOGLE, 2020a), usando o navegador (*browser*) e com acesso gratuito a GPU²⁰. Além disso, os códigos Python armazenados na plataforma GitHub (GitHub, 2020) com extensão `.ipynb` podem ser executados no Colab, o que sem dúvida facilita sobremaneira o compartilhamento dos resultados do trabalho. A extensão `.ipynb` é usada pelo Colab e pelo *Integrated Development Environment* (IDE) Jupyter Notebook (Project Jupyter, 2020), pois ambos utilizam o interpretador interativo IPython como *kernel* (PÉREZ; GRANGER, 2007). Note-se que todo código Python (extensão `.py`) é executável no *shell* IPython.

Além disso, o TensorFlow oferece uma ferramenta de visualização baseada em *browser* denominada TensorBoard, cujo principal objetivo é ajudar o usuário a monitorar visualmente tudo o que acontece dentro do seu modelo durante o treinamento (CHOLLET,

¹⁸ A biblioteca Caffe2 (FACEBOOK, 2020) foi absorvida pelo PyTorch.

¹⁹ Segundo (GULLI; KAPOOR; PAL, 2019), são os *frameworks* mais utilizados pela comunidade de DL. O Keras faz parte do TensorFlow 2.0.

²⁰ Por até doze horas.

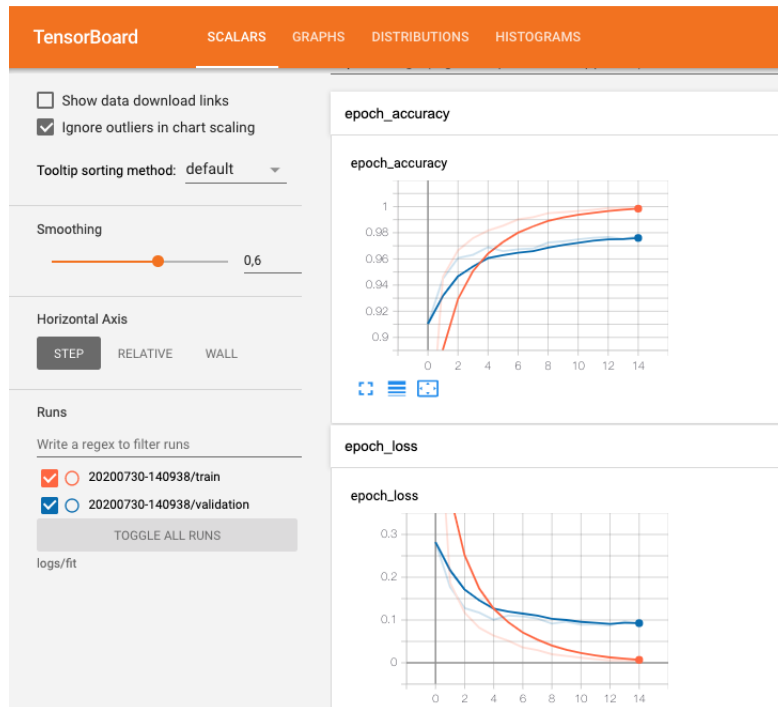


Figura 3 – Visualização das curvas de aprendizado pelo TensorBoard.

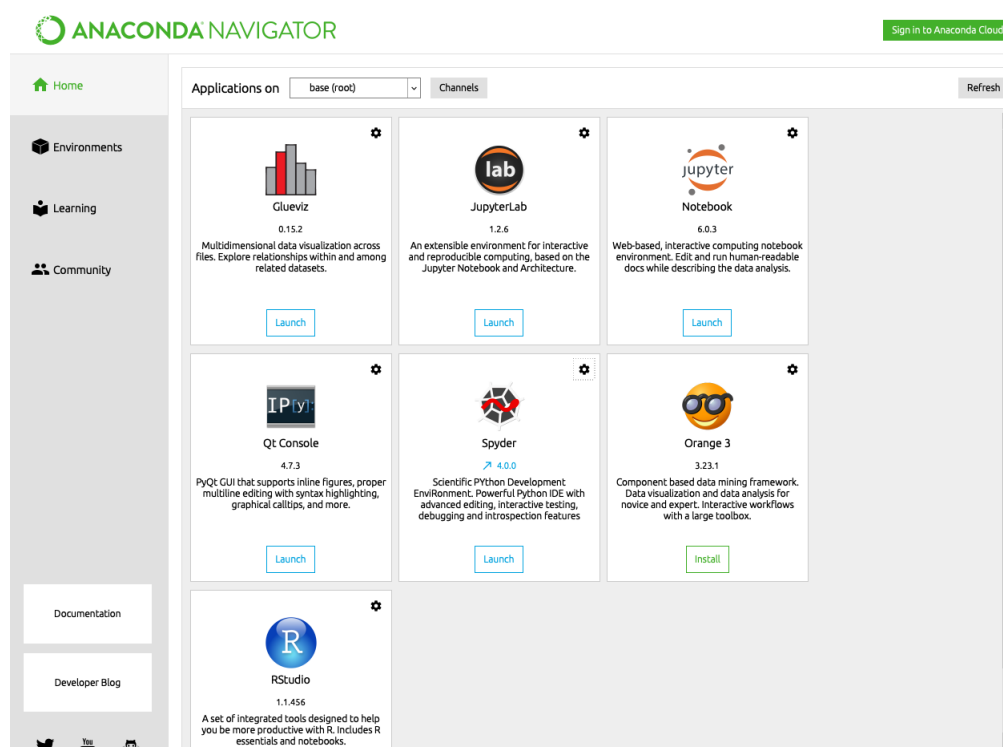
2018) [p. 252]. O TensorBoard automatiza algumas funcionalidades como a visualização da curva de aprendizado das redes neurais (Fig. 3).

Assim, optou-se pela utilização da linguagem Python e do *framework* TensorFlow 2 em conjunto com a API Keras.

Segundo (CHOLLET, 2018) [p. 61],

“Keras possui os seguintes recursos principais:

- Permite que o mesmo código seja executado perfeitamente na CPU ou GPU.
- Possui uma API amigável que facilita a prototipagem rápida de modelos de aprendizado profundo.
- Possui suporte embutido para redes convolucionais (para visão computacional), redes recorrentes (para processamento de sequências) e qualquer combinação de ambas.
- Suporta arquiteturas de rede arbitrárias: modelos de múltiplas entradas ou saídas, compartilhamento de camadas, compartilhamento de modelos e assim por diante. Isso significa que o Keras é apropriado para criar essencialmente qualquer modelo de aprendizado profundo, de redes adversárias generativas (GOODFELLOW et al., 2014) a máquinas neurais de Turing (GRAVES; WAYNE; DANIHELKA, 2014).” (tradução livre)

Figura 4 – Anaconda *navigator*.

Os modelos de Rede Neural Artificial (RNA) listados no Apêndice A foram desenvolvidos por meio da ferramenta computacional Anaconda²¹ (Fig. 4) utilizando-se o IDE Spyder 4.0 (Fig. 5) e a linguagem Python 3.7.7 (Anaconda, 2020).

Os códigos-fonte Python (extensão .py) desenvolvidos por esta pesquisa estão listados no Apêndice A.

1.4 Relatório de Atividades

Todas as etapas propostas no cronograma original (vide tabela 1) foram cumpridas. A atividade da etapa 10 encontra-se em andamento. Pretende-se publicar o resultado desta pesquisa em congresso de iniciação científica e/ou no arXiv (Cornell University, 2020).

1.5 Estrutura do Relatório

A fundamentação teórica desta pesquisa é discutida no capítulo 2. O capítulo 3 apresenta os resultados obtidos em CV e FR por meio de simulações. Finalmente, o

²¹ O Anaconda é uma ferramenta computacional gratuita e de fácil instalação que permite gerenciar distribuições de Python, ambientes de trabalho e módulos nos sistemas operacionais Windows, Mac OSX e Linux. Alguns dos gráficos aplicativos Python mais populares já estão instalados ou estão disponíveis para instalação no Anaconda, como os IDE Spyder e Jupyter Notebook (em que o IDE é montado no *browser*).

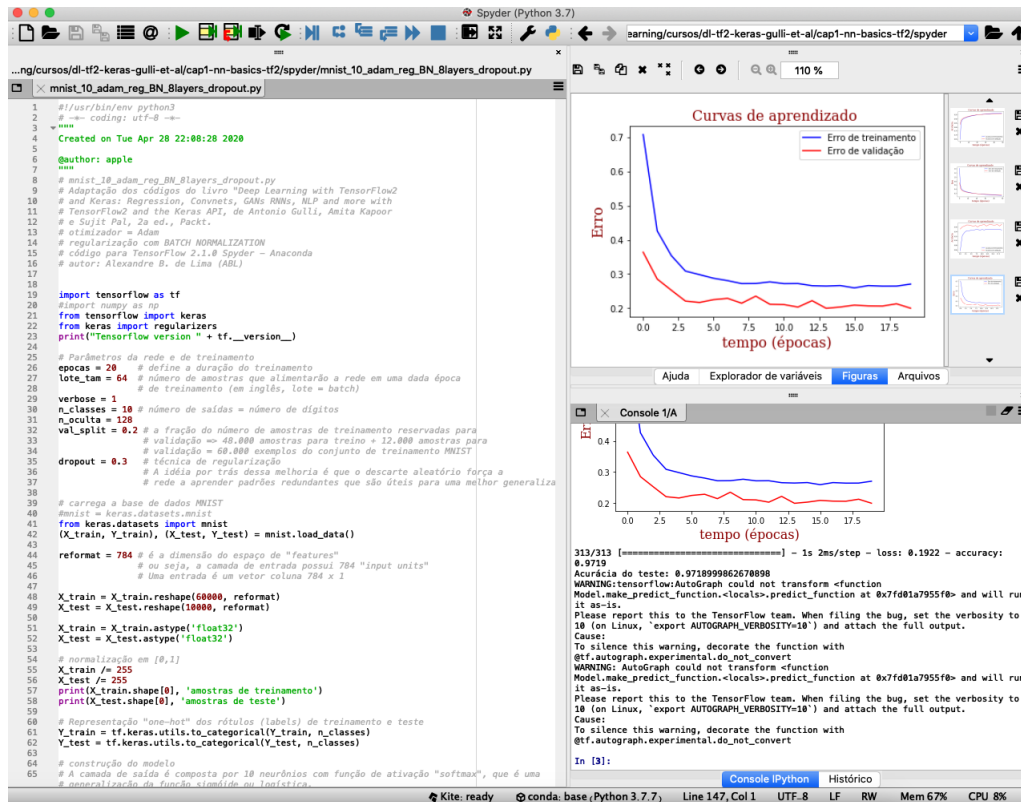


Figura 5 – Ambiente de programação Spyder.

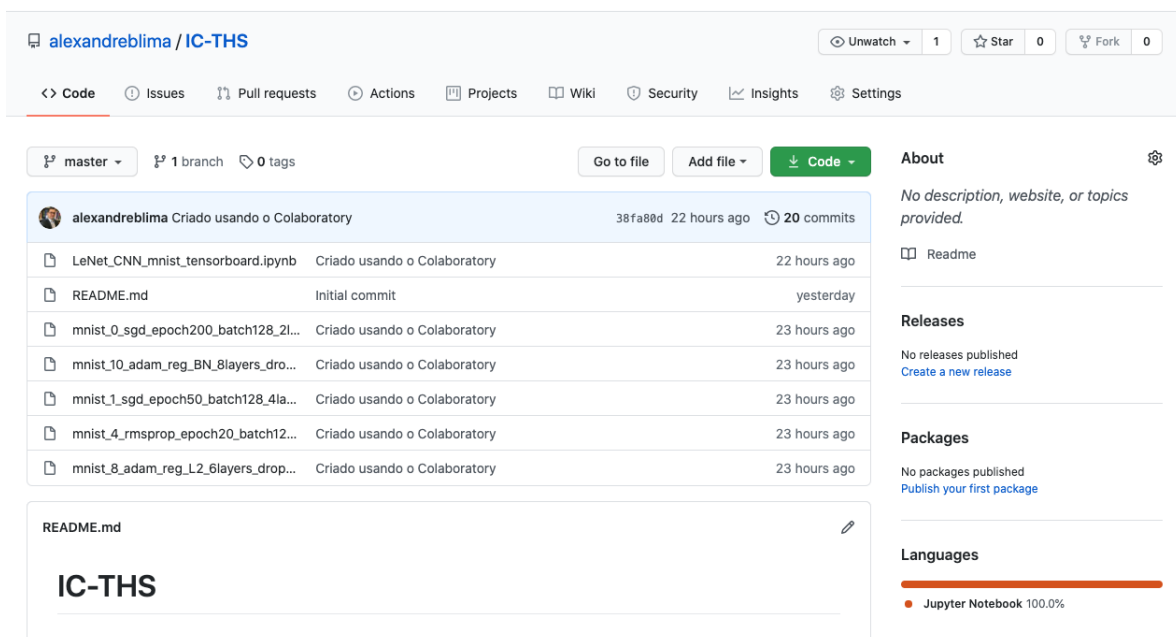
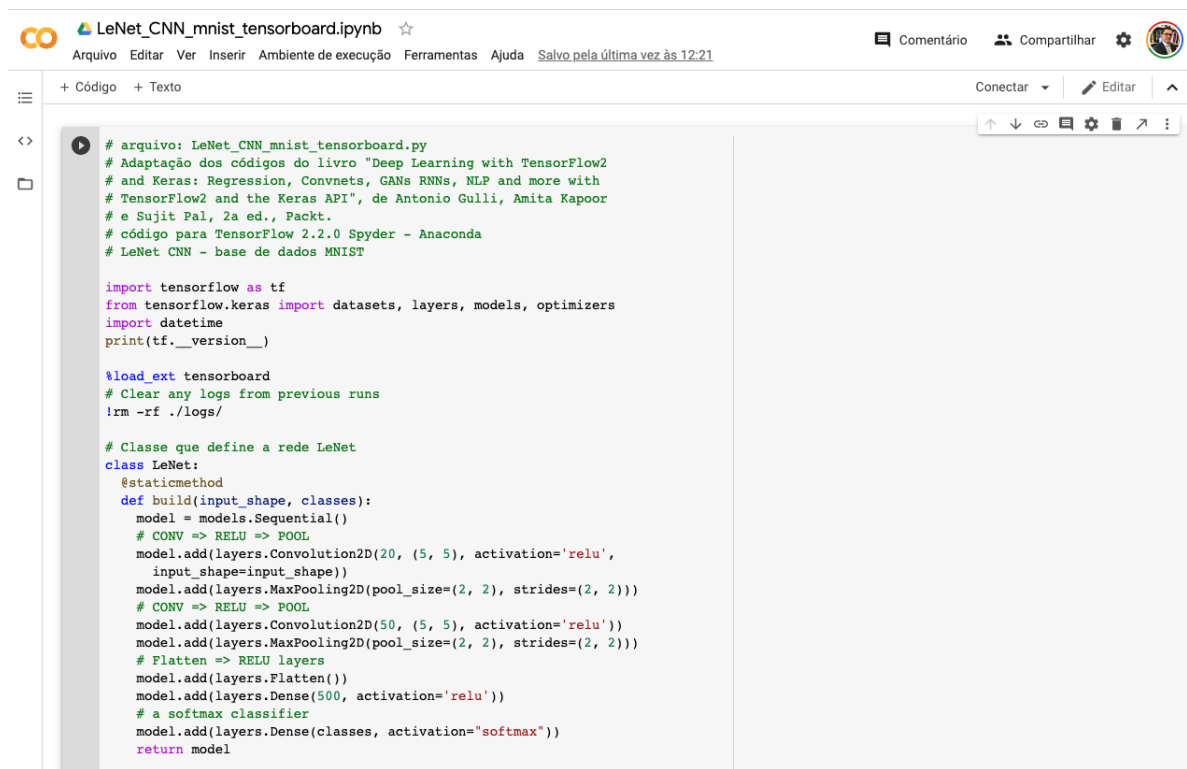


Figura 6 – Repositório GitHub.



```

# arquivo: LeNet_CNN_mnist_tensorboard.py
# Adaptação dos códigos do livro "Deep Learning with TensorFlow2
# and Keras: Regression, Convnets, GANs RNNs, NLP and more with
# TensorFlow2 and the Keras API", de Antonio Gulli, Amita Kapoor
# e Sujit Pal, 2a ed., Packt.
# código para TensorFlow 2.2.0 Spyder - Anaconda
# LeNet CNN - base de dados MNIST

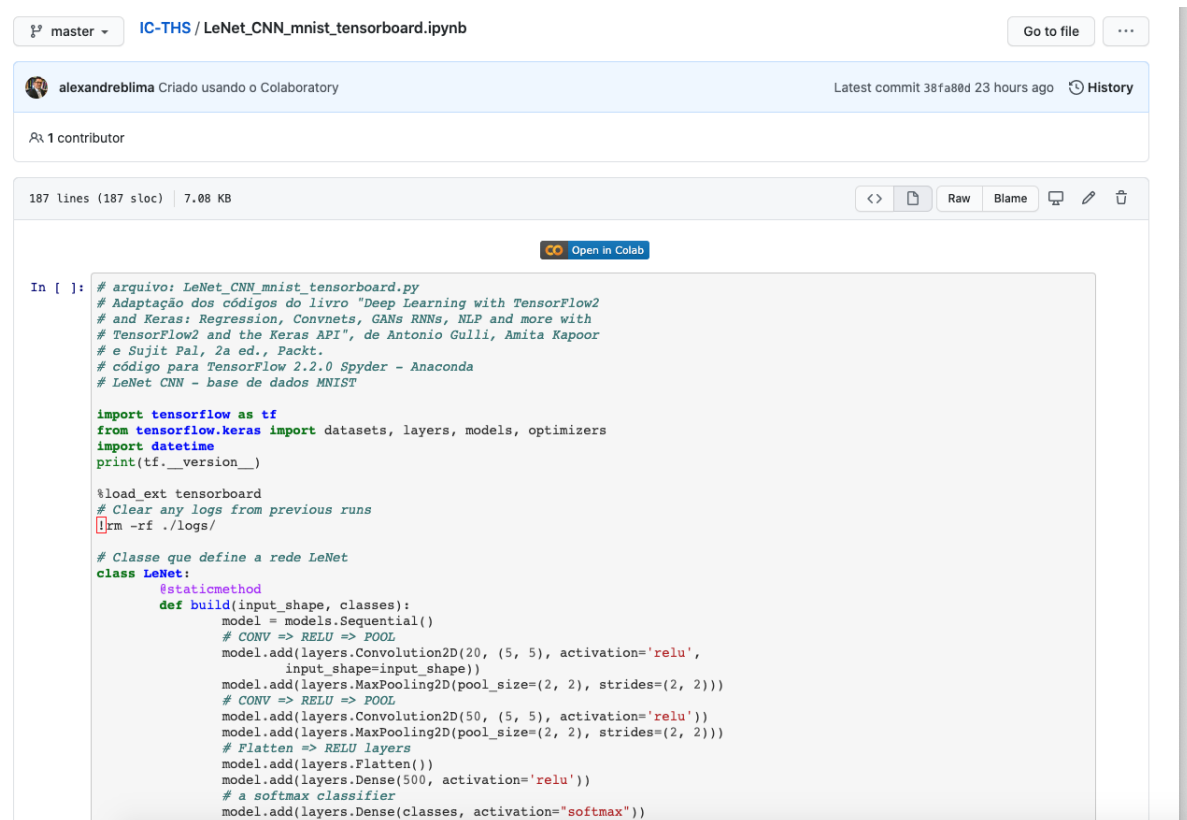
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, optimizers
import datetime
print(tf.__version__)

%load_ext tensorboard
# Clear any logs from previous runs
!rm -rf ./logs/

# Classe que define a rede LeNet
class LeNet:
    @staticmethod
    def build(input_shape, classes):
        model = models.Sequential()
        # CONV => RELU => POOL
        model.add(layers.Convolution2D(20, (5, 5), activation='relu',
            input_shape=input_shape))
        model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # CONV => RELU => POOL
        model.add(layers.Convolution2D(50, (5, 5), activation='relu'))
        model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # Flatten => RELU layers
        model.add(layers.Flatten())
        model.add(layers.Dense(500, activation='relu'))
        # a softmax classifier
        model.add(layers.Dense(classes, activation="softmax"))
        return model

```

Figura 7 – Ambiente de programação Colab.



```

In [ ]: # arquivo: LeNet_CNN_mnist_tensorboard.py
# Adaptação dos códigos do livro "Deep Learning with TensorFlow2
# and Keras: Regression, Convnets, GANs RNNs, NLP and more with
# TensorFlow2 and the Keras API", de Antonio Gulli, Amita Kapoor
# e Sujit Pal, 2a ed., Packt.
# código para TensorFlow 2.2.0 Spyder - Anaconda
# LeNet CNN - base de dados MNIST

import tensorflow as tf
from tensorflow.keras import datasets, layers, models, optimizers
import datetime
print(tf.__version__)

%load_ext tensorboard
# Clear any logs from previous runs
!rm -rf ./logs/

# Classe que define a rede LeNet
class LeNet:
    @staticmethod
    def build(input_shape, classes):
        model = models.Sequential()
        # CONV => RELU => POOL
        model.add(layers.Convolution2D(20, (5, 5), activation='relu',
            input_shape=input_shape))
        model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # CONV => RELU => POOL
        model.add(layers.Convolution2D(50, (5, 5), activation='relu'))
        model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # Flatten => RELU layers
        model.add(layers.Flatten())
        model.add(layers.Dense(500, activation='relu'))
        # a softmax classifier
        model.add(layers.Dense(classes, activation="softmax"))
        return model

```

Figura 8 – Visualização do código no GitHub.

Etapa	Atividades	Mês(meses)
1	Estudo do Cálculo Diferencial	1 – 3
2	Estudo do Cálculo Diferencial em Campos Escalares	4 – 6
3	Revisão da literatura	1 – 2
4	Curso on line de <i>machine learning</i>	1 – 2
5	Estudo do TensorFlow	1 – 6
6	Estudo das Linguagens Python e MATLAB/Octave	1 – 6
7	Seleção de algoritmos de <i>deep learning</i> e de conjuntos de treinamento	5 – 6
8	Elaboração do relatório parcial	6 – 7
9	Testes e comparação de algoritmos de <i>deep learning</i> (simulações)	6 – 11
10	Elaboração de artigo científico	11 – 12
11	Elaboração do relatório final	12

Tabela 1 – Cronograma de atividades.

capítulo 4 sumariza os resultados obtidos e lista algumas sugestões para trabalhos futuros.

2 Aprendizado de Máquina

Este capítulo apresenta o machine learning e qual é a sua relação com o campo da inteligência artificial.

2.1 Machine Learning

Augusta Ada Lovelace¹ (1815 – 1852) especulou sobre a possibilidade de IA aproximadamente um século antes de o primeiro computador eletrônico digital ser desenvolvido pelos engenheiro John P. Eckert e físico John Mauchly, o *Electronic Numerical Integrator and Computer* (ENIAC)², em 1946, nos EUA, a pedido do Exército Norte-Americano (GOODFELLOW; BENGIO; COURVILLE, 2016).

Alan Turing sentiu-se obrigado a refutar uma afirmação de Lady Lovelace³, que consta no seu famoso artigo “Notas de A. A. L.” (“*Notes by A. A. L.*”) (LOVELACE, 1843), a famosa “objeção de Lady Lovelace” (TURING, 1950)[p. 14]:

“Our most detailed information of Babbage’s Analytical Engine comes from a memoir by Lady Lovelace (1842). In it she states, “The Analytical Engine has no pretensions to *originate* anything. It can do *whatever we know how to order it to perform.*” (her italics)”

“Nossas informações mais detalhadas da Máquina Analítica de Babbage vêm das Notas de Lady Lovelace (1842). Nelas, ela afirma: “A Máquina Analítica não tem pretensões para *originar* qualquer coisa. Pode fazer *qualquer coisa que nós saibamos como ordená-la* a executar.” (itálico dela)

A refutação de Turing à objeção de Lady Lovelace fundamenta-se na proposta de se implementar uma *learning machine* (máquina aprendiz): um programa que simule o cérebro de uma criança recém-nascida e que seja capaz de aprender, de forma que consiga simular o cérebro de um humano adulto ao fim de um processo de educação (TURING, 1950).

Para maiores detalhes da contribuição que Lady Lovelace deu para a Computação, recomenda-se a leitura do artigo (Fuegi; Francis, 2003), que aborda a história da descrição

¹ Seu nome de solteira era Augusta Ada Byron, pois era filha do ilustre poeta inglês Lord Byron.

² No ENIAC, a programação era feita por meio da fiação do painel. Até então não havia o conceito de programa armazenado. John von Neumann propôs uma arquitetura que transformou os computadores eletrônicos da década de 1940 (como o ENIAC) nos computadores da era contemporânea. Tal arquitetura possui três componentes principais: uma unidade central de processamento (CPU) para processamento das instruções codificadas usando 0’s e 1’s (codificação binária), memória (para armazenamento das instruções e de dados) e sistemas de Entrada/Saída (*Input/Output*).

³ Nota G, segundo parágrafo de (LOVELACE, 1843).

da Máquina Analítica de Charles Babbage de autoria de Lady Lovelace, o famoso “*Notes by A. A. L.*”.

Nos primórdios da IA (segunda metade da década de 1950), tarefas abstratas e que podem ser bem descritas pela Matemática, mas que não requerem conhecimento sobre o mundo real, como “jogar xadrez”, impulsionaram o desenvolvimento na área.

Contudo, tarefas “fáceis” para seres humanos, tais como reconhecer emoções faciais e análise de sentimentos em sinais de voz, são “difíceis” para os computadores, pois demandam que eles tenham algum conhecimento do mundo físico (LIMA; BARRETO; AMAZONAS, 2018).

A solução para os problemas “difíceis” de IA consiste em **permitir que os computadores aprendam com a experiência e que consigam formular algum tipo de modelo do mundo exterior a partir de uma hierarquia de conceitos**. Esta abordagem, denominada **Aprendizado de Máquina** ou ML, tem como grande vantagem evitar que todo o conhecimento a ser adquirido pela máquina tenha que ser formalmente especificado pelo homem (RUSSEL; NORVIG, 2013). Segundo Arthur Samuel, um dos pais da IA (NG, 2019),

“*Machine Learning* é a área do conhecimento que dá aos computadores a habilidade de aprender sem serem explicitamente programados”.

Tom Mitchell oferece uma definição formal de ML (NG, 2019):

“Um computador APRENDE a partir da sua EXPERIÊNCIA (E) com relação a alguma TAREFA (T) e alguma MEDIDA DE DESEMPENHO (P), se o seu desempenho em T, medido por P, melhora com a sua experiência E”.

O jogo de damas é um problema que pode ser resolvido por meio de algoritmos de ML. De acordo com a definição, tem-se que:

- E = “treinamento” do programa, que consiste em o programa jogar um número suficientemente grande⁴ de partidas de jogos de damas.
- T = jogar uma nova partida.
- P = probabilidade de ganhar a próxima partida contra um novo oponente.

Para a grande maioria dos algoritmos de ML, o homem fornece os dados de entrada (conjunto de treinamento) bem como as respostas esperadas a partir dos dados. A saída

⁴ Por exemplo, centenas de milhares de jogos.

do processamento é um conjunto de regras, que serão aplicadas sobre novos dados de entrada, de forma a produzir respostas originais. Um sistema de ML é treinado ao invés de ser explicitamente programado para uma tarefa. A Fig. 9 ilustra os dois paradigmas de programação, o clássico (diagrama superior), usado pela inteligência artificial simbólica, e o de ML (diagrama inferior).

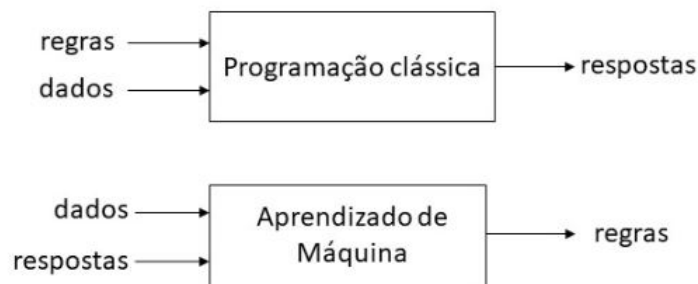


Figura 9 – Paradigmas de programação.

Segue-se uma lista com alguns algoritmos importantes de ML ([LANTZ, 2013](#)):

- *nearest neighbor* (vizinho mais próximo);
- *decision trees and rules* (árvores de decisão e regras);
- regressão;
- DL (usando RNA);
- *support vector machines*;
- regras de associação; e
- *clustering with k-means*

De acordo com ([CHEN; MEDINI; SHRIVASTAVA, 2019](#)),

“os algoritmos de aprendizagem profunda (DL) são o foco central dos sistemas modernos de aprendizado de máquina. À medida que os volumes de dados continuam crescendo, tornou-se habitual treinar grandes redes neurais com centenas de milhões de parâmetros com capacidade suficiente para memorizar esses volumes e obter precisão estado da arte.” (tradução livre)

Os algoritmos de ML podem ser divididos em quatro grupos ([CHOLLET, 2018](#)):

- **Aprendizado Supervisionado** (“Supervised Learning”), em que o homem fornece os dados de entrada (conjunto de treinamento), bem como as respostas esperadas a partir dos dados (associa-se um rótulo para cada dado). A saída do processamento é um conjunto de regras, que serão aplicadas sobre novos dados de entrada, de forma a produzir respostas originais (inferências do tipo classificação ou regressão).
- **Aprendizado Não Supervisionado** (“Unsupervised Learning”), para treinamento de um **modelo descritivo** capaz de **reconhecer padrões**. Não há um professor. Aplicações: descobrir padrões de compra em supermercados e tarefas de agrupamento (“clustering”) de dados (ex.: identificação de grupos de pessoas que compartilham interesses comuns tais como esportes, religião ou música).
- **Aprendizado Auto-Supervisionado** (“Self-supervised Learning”), é um tipo de aprendizagem supervisionada, mas os rótulos são gerados a partir dos dados de entrada por meio de um algoritmo.
- **Aprendizado por Reforço** (“Reinforcement Learning”), em que um **programa** é treinado a interagir em um **ambiente** por meio de ações para atingir um objetivo. O aprendizado usa um mecanismo de **recompensas** e/ou **punições**. O algoritmo AlphaZero⁵, desenvolvido pelo Google DeepMind, utiliza esta técnica (SILVER; AL, 2017).

Por exemplo, considere o problema da previsão do preço de um imóvel em reais (variável dependente y) dada a área construída em m^2 (variável independente, explanatória ou regressor x). Neste caso, o conjunto de treinamento poderia consistir em uma base de dados com os preços de venda de milhares de imóveis e respectivas áreas construídas coletados na cidade de São Paulo nos últimos doze meses. Este problema pode ser resolvido utilizando-se uma Regressão Linear Simples (RLS). Note-se que a previsão (predição ou saída) é uma variável contínua, ou seja y é um número real.

Há casos em que a saída (variável y) é uma variável discreta (ex.: 0 ou 1). Considere o problema de **classificar** um câncer de mama dado o tamanho do tumor. A resposta (saída) é “sim” ou “não”, que pode ser associada aos bits 1 ou 0, respectivamente. Em ML, diz-se que o tamanho do tumor (variável x) é uma *feature* (atributo ou característica).

Outros atributos do câncer de mama seriam: espessura dos grupos, uniformidade do tipo de célula, uniformidade do tipo de célula, etc.

Em suma, a **regressão** prevê o valor de uma variável contínua, enquanto que a **classificação** prediz o valor de uma variável discreta.

⁵ O AlphaZero derrotou o campeão mundial do jogo Go, Lee Sedol, em um desafio patrocinado pelo Google em 2016, na Coreia do Sul.

Suponha uma Regressão Linear Múltipla (RLM) em que as variáveis de entrada são x_1 e x_2 e a variável de saída é y . Associe a variável y com o preço previsto de um imóvel em reais dadas a idade da construção (variável x_1) e a área construída em m^2 (variável x_2). Admita ainda que se possua um banco de dados com 50 preços em reais de imóveis em função da idade e da área construída. Em ML, adota-se a seguinte notação para este problema:

- $m = 50$ corresponde ao número de exemplos de treinamento;
- x_1 e x_2 denotam as variáveis de entrada ou *features*;
- y representa a variável de saída;
- (x_1, x_2, y) denota um exemplo de treinamento;
- $(x_1^{(i)}, x_2^{(i)}, y^{(i)})$ é o i -ésimo exemplo de treinamento;
- $h_{\Theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = y(x)$ é a hipótese (ou modelo) e $\Theta = [\theta_0, \theta_1, \theta_2]^T$ é o vetor de parâmetros a serem estimados.

A RLM acima pode ser resolvida por um método numérico de otimização denominado **algoritmo do gradiente**, cujo objetivo é estimar os valores de θ_0 , θ_1 e θ_2 que minimizam as distâncias entre os exemplos de treinamento $y^{(i)}$ e os valores previstos pelo modelo $\hat{y}^{(i)} = h_{\Theta}^{(i)}(x)$, $i = 1, 2, \dots, m$:

$$e^{(i)} = y^{(i)} - \hat{y}^{(i)} \quad (2.1)$$

A variável $e^{(i)}$ também é designada por *erro de estimação*.

O algoritmo do gradiente encontra o mínimo de uma função, designada por **função custo** J (ou **função de perda** - *loss function*), usando um esquema iterativo, onde em cada passo de iteração se toma a direção negativa do gradiente da função, a qual corresponde à direção de declive máximo. A água que desce morro abaixo, por efeito da gravidade, segue o método da descida pelo gradiente, uma vez que o curso d'água segue pelo caminho de declividade máxima. Em inglês, o algoritmo do gradiente é comumente chamado de *gradient descent* (SGD).

É usual que o ajuste dos exemplos de treinamento seja realizado pelo **método dos mínimos quadrados**, segundo o qual a hipótese a ser adotada é aquela que torna mínimo o **erro quadrático médio** (*Mean Square Error* (MSE)) de estimação (função custo). Ou seja, o algoritmo do gradiente busca minimizar a média quadrática $E(e^2)$:

$$J(\Theta) = E(e^2) = \frac{1}{2m} \sum_{i=1}^m [h_{\Theta}(x^{(i)}) - y^{(i)}]^2 \quad (2.2)$$

Note-se que função custo $J(\Theta)$ é dividida por 2 por convenção.

repeat

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \theta_2)$$

until *atingir a convergência*;

Algoritmo 1: Algoritmo do gradiente em [RLM](#).

O algoritmo do gradiente é dado acima, em que η denota o passo de adaptação ou taxa de aprendizado (*learning rate*).

As [RNA](#) são ferramentas indicadas para implementação de regressão não linear multivariada ([RUSSEL; NORVIG, 2013](#)) [p. 638]. Além disso, note-se que [FR](#) é um problema de classificação não linear ([NG, 2019](#)). Assim, justifica-se a abordagem utilizada neste trabalho, baseada em [DL](#).

O aprendizado supervisionado utilizado neste trabalho envolve o treinamento de uma rede neural profunda. Segundo ([GOODFELLOW; BENGIO; COURVILLE, 2016](#), p.267), de todos os problemas de otimização em [DL](#), este é o problema mais difícil, pois, às vezes, o treinamento de uma rede neural pode demandar o uso de vários computadores por dias, semanas ou meses.

2.2 A Noção de Redes Neurais

O *The Handbook of Artificial Intelligence* apresenta a seguinte definição operacional⁶ para Inteligência Artificial (IA) ([R. Anderson et al, 1981](#)):

“Artificial Intelligence (AI) is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior - understanding language, learning, reasoning, solving problems, and so on.”

“Inteligência artificial (IA) é a parte da ciência da computação preocupada em projetar sistemas computacionais inteligentes, isto é, sistemas que exibem as características que associamos à inteligência no comportamento humano - compreensão da linguagem, aprendizagem, raciocínio, resolução de problemas e assim por diante.”

Existem duas linhas principais de pesquisa em [IA](#): a conexionista e a simbólica. De acordo com Boden ([BODEN, 2016](#)), ambas foram inspiradas no artigo seminal intitulado *A Logical Calculus of the Ideas Immanent in Nervous Activity* (1943), de Warren Sturgis McCulloch e Walter Pitts ([MCCULLOCH; PITTS, 1943](#)), a primeira teoria computacional

⁶ Não há consenso sobre o conceito de inteligência.

moderna⁷. A literatura reconhece que a pesquisa realizada por McCulloch e Pitts é o trabalho pioneiro em IA (RUSSEL; NORVIG, 2013).

A abordagem conexionista usa RNA. A linha simbólica, às vezes chamada de IA simbólica (BODEN, 2016)) segue a tradição lógica e teve John McCarthy e Allen Newell, entre outros, como alguns de seus grandes expoentes (NEWELL, 1980).

A Fig. 10 ilustra o Modelo de McCulloch e Pitts (MCP) (MCCULLOCH; PITTS, 1943).

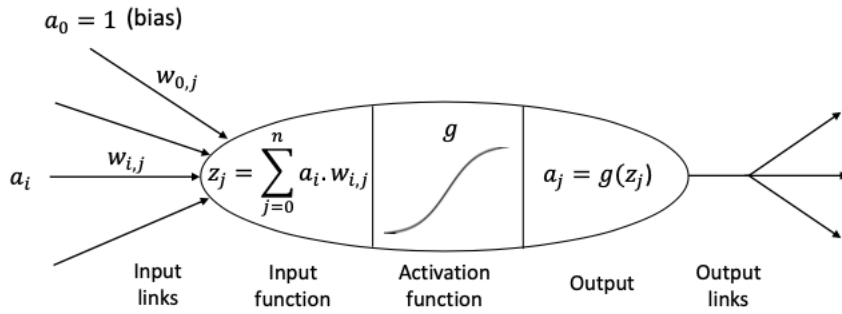


Figura 10 – Modelo de McCulloch e Pitts

O neurônio (ou unidade) artificial da Fig. 10 possui as seguintes características:

- sinais (ativações) de entrada (a_1, a_2, \dots, a_n) são bits 0 ou 1. O sinal externo $a_0 = 1$ é o *bias*;
- pesos sinápticos do j-ésimo neurônio: w_0, w_1, \dots, w_n ; e
- a_j denota o sinal de saída (ou ativação de saída) do j-ésimo neurônio, dado por:

$$z_j = \sum_{i=0}^n w_{i,j} a_i \quad (2.3)$$

$$a_j = g(z_j) = \{0, 1\} \quad (2.4)$$

em que $g(z)$ é uma função de ativação não linear. A saída é binária (bit 0 ou bit 1); por conseguinte, diz-se que o MCP tem a propriedade “tudo ou nada”.

⁷ A teoria é considerada moderna porque emprega a noção matemática de computação estabelecida por Turing em 1936 (TURING, 1936).

A função de ativação do **MCP** é a função de Heaviside (função degrau unitário)

$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \quad (2.5)$$

Atualmente, as redes neurais usam outras funções de ativação, que viabilizam o treinamento da rede por meio do algoritmo de retropropagação do erro (*backpropagation* ou “*backprop*”) (KOVÁCS, 2006), tais como a função de ativação designada por REctified Linear Unit (Relu), dada por

$$g(z) = \max\{0, z\}. \quad (2.6)$$

A Fig. 11 ilustra a função Relu.

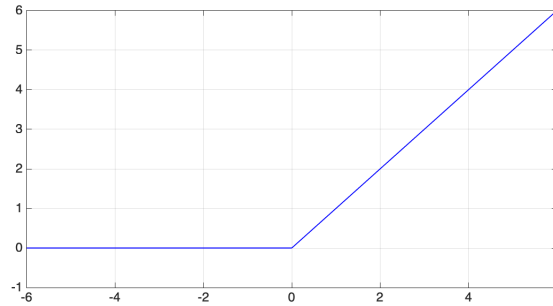


Figura 11 – função Relu.

É importante ressaltar que o papel do algoritmo *backpropagation* é calcular o gradiente em redes neurais com múltiplas camadas (GOODFELLOW; BENGIO; COURVILLE, 2016, p. 198), enquanto que outro algoritmo, como o **SGD**, é o responsável pela atualização dos parâmetros da rede.

Uma **RNA** é um sistema de processamento paralelo distribuído, inspirado na estrutura de processamento do cérebro humano (Fig. 12). A técnica de **RNA** é uma forma de computação não algorítmica de funções.

Na Fig. 12, o vetor $\mathbf{x} = [x_0 \ x_1 \ x_2 \ x_3]^T$, em que $x_0 = 1$ é o *bias*, contém as *features* ou sinais de entrada (camada de entrada) x_1, x_2, x_3 . A camada intermediária, designada por **oculta**, possui três unidades ($a_0^{(2)}, a_1^{(2)}$ e $a_2^{(2)}$), em que $a_0^{(2)} = 1$ é o *bias*. A camada de saída possui somente a unidade $a_1^{(3)} = h_w(x)$.

2.3 Deep Learning

Muitos pensam que **DL** é uma nova tecnologia. Contudo, isto não corresponde à realidade dos fatos. O **DL** surgiu na década de 1940 do século passado, mas não com esse nome.

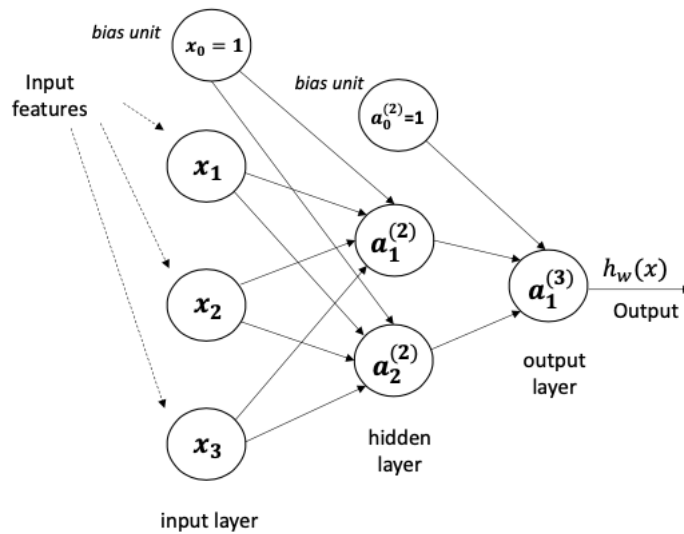


Figura 12 – Exemplo de uma rede neural densamente conectada com uma camada oculta (*hidden layer*).

Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), houve três ondas de pesquisa e desenvolvimento em DL:

1. O DL com o nome de cibernética nos anos 1940-1960.
2. O DL conhecido como connexionismo (*connectionism*) nas décadas de 1980 e 1990.
3. O DL atual, que ressurgiu em 2006, com o trabalho de (HINTON; OSINDERO; TEH, 2006).

Como visto anteriormente, os primeiros algoritmos de DL (os da primeira onda) eram inspirados em modelos computacionais de aprendizado biológico, ou seja, modelos de como o aprendizado ocorre em um cérebro biológico. Tais algoritmos foram designados por RNA.

Goodfellow, Bengio e Courville destacam que (GOODFELLOW; BENGIO; COURVILLE, 2016, p. 14),

“The modern term deep learning goes beyond the neuroscientific perspective on the current breed of machine learning models. It appeals to a more general principle of learning multiple levels of composition, which can be applied in machine learning frameworks that are not necessarily neurally inspired”.

“A terminologia moderna aprendizagem profunda vai além da perspectiva neurocientífica da atual geração de modelos de aprendizado de máquina. Apela para um princípio mais geral de aprendizagem de múltiplos níveis de composição,

que pode ser aplicado em estruturas de aprendizado de máquina que não são necessariamente inspiradas em estruturas neuronais biológicas.” (tradução livre)

Na década de 1980, a segunda onda das redes neurais emergiu de um movimento científico denominado Conexionismo (paradigma conexionista) ou processamento paralelo distribuído, que surgiu no contexto das ciências cognitivas.

Segundo o Conexionismo, um número grande de unidades computacionais simples pode atingir um comportamento inteligente quando interligadas em rede.

A segunda onda das RNA durou até meados dos anos 1990. Naquela época, o treinamento de uma RNA era extremamente custoso do ponto de vista computacional. Por conseguinte, o mercado perdeu o interesse pela tecnologia, pois era inviável do ponto de vista econômico.

A terceira onda das RNA (a atual) é oriunda de uma contribuição seminal dada pelo grupo de pesquisa liderado por Geoffrey Hinton, da Universidade de Toronto, em 2006. Hinton, Osindero e Teh demonstraram que um tipo de rede neural denominada *deep belief* poderia ser treinada com alta eficiência por meio da estratégia *greedy layer-wise pretraining* (pré-treinamento voraz orientado à camada) (HINTON; OSINDERO; TEH, 2006). O artigo de Hinton, Osindero e Teh disparou uma nova onda de pesquisas em RNA, a qual popularizou o termo *Deep Learning*.

O termo *deep* (profundo) em DL denota a ideia de um modelo em rede que possui a capacidade de particionar a representação de uma entrada em múltiplas camadas. O número de camadas de um modelo corresponde à profundidade da rede (ALLAIRE, 2017).

A Fig. 13 mostra um modelo de DL com seis (6) camadas para classificação de dígitos. Reconhecer dígitos escritos à mão é um problema importante em muitas aplicações de visão computacional, tais como classificação automática de correspondências por código postal e leitura automática de cheques (RUSSEL; NORVIG, 2013). O Dr. Yan LeCun, prêmio Turing de 2018, é um dos pioneiros na área (LECUN, 1989), (JACKEL, 1990), (HAFFNER, 1998).

A primeira camada (entrada) da Fig. 13 é denominada **camada visível** e contém os *pixels* de uma imagem, a saber, o dígito 4 escrito à mão. Note-se que a camada visível não possui neurônios artificiais, mas nós de entrada.

A imagem com o dígito manuscrito 4 à esquerda da rede da Fig. 15 contém uma matriz de $20 \times 20 = 400$ *pixels* com valores de 8 bits em tons de escala de cinza (LECUN; CORTES; BURGESS, 2019). A foto faz parte do bando de dados do United States National Institute of Science and Technology (NIST) de dígitos escritos à mão.

Seguem-se quatro (4) **camadas ocultas** e uma **camada de saída**, a qual identifica o dígito. A camada de saída é composta por dez (10) neurônios artificiais, que são os

preditores (classificadores) dos dígitos. Cada neurônio da camada de saída pode, por exemplo, utilizar uma função do tipo logística (ou sigmóide) para fazer a classificação do dígito: $g(z) = \frac{1}{1+e^{-z}}$.

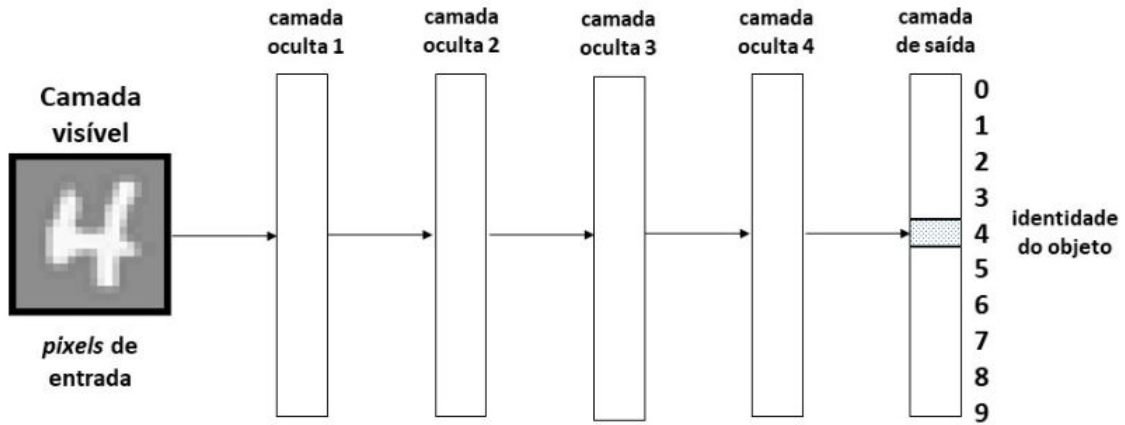


Figura 13 – DL com seis camadas.

A função (modelo ou hipótese) que mapeia o conjunto de *pixels* de entrada em um dado objeto (no caso o dígito 4) é composta pelas últimas cinco (5) camadas. Tal modelo, além de ser **não linear**, é extremamente complicado. A estratégia do DL é desdobrar esse modelo não linear/complexo em uma cascata de transformações sucessivas de menor complexidade, de modo que cada camada do DL execute uma operação de transformação ou filtragem mais simples. Por exemplo, a primeira camada oculta da Fig. 15 poderia identificar bordas na imagem, ao passo que à segunda camada oculta caberia identificar cantos e contornos. Os algoritmos de classificação dos dígitos são executados pelos neurônios da camada de saída.

A Fig. 14 mostra a relação entre IA, ML e DL.

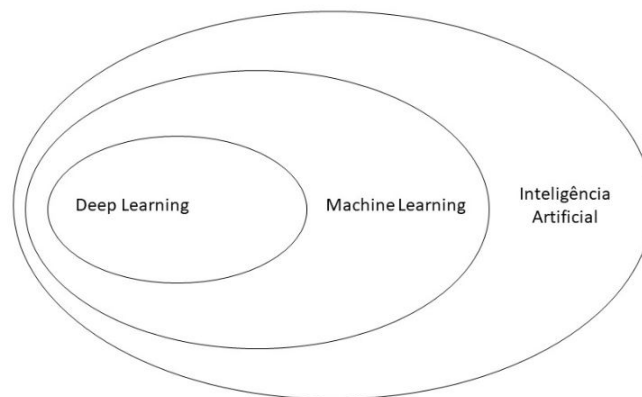


Figura 14 – relação entre IA, ML e DL.

A Fig. 15 ilustra como funciona o DL (CHOLLET, 2018). Na inicialização, valores aleatórios são atribuídos aos pesos w da rede; logo, o valor inicial do resíduo é alto. No entanto, no decorrer do treinamento da rede, os pesos são ajustados incrementalmente na direção correta; ao mesmo tempo, o valor da função custo diminui. Este é o *loop* de treinamento, que, sendo repetido o suficiente, normalmente dezenas de iterações (mini-bateladas ou *mini-batches*) em milhares de exemplos, produz pesos que minimizam a função de custo. Uma rede é considerada treinada quando o mínimo da função custo é atingido.

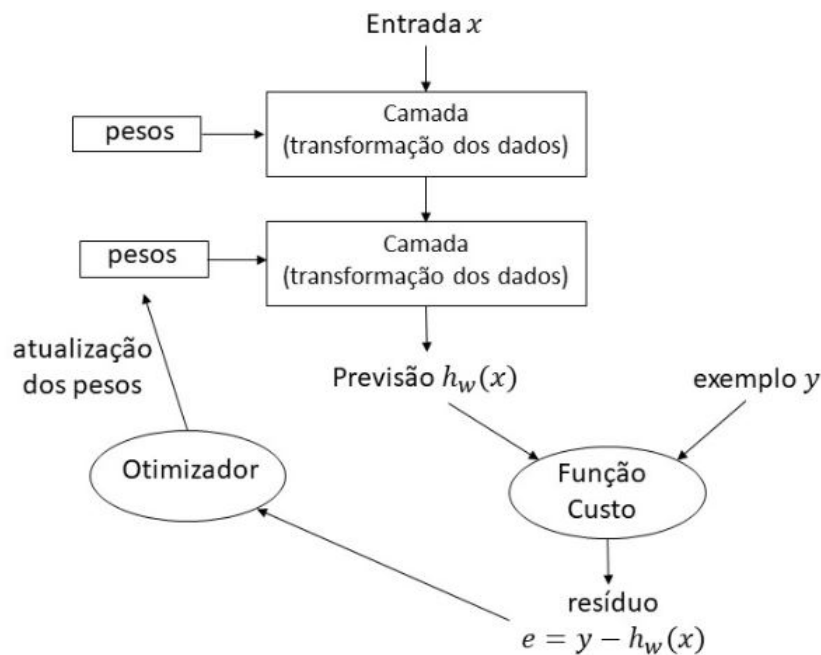


Figura 15 – Como funciona o DL.

O DL alcançou os seguintes avanços, todos em áreas historicamente difíceis de ML (ALLAIRE, 2017), (SILVER; AL., 2018):

- classificação de imagem super-humana;
- reconhecimento de voz em nível quase humano;
- transcrição de escrita à mão quase humana;
- tradução automática;
- conversão de texto em fala;
- assistentes digitais como a Alexa (Amazon);

- condução autônoma de veículo ao nível quase humano;
- segmentação de anúncios (usada por empresas como Google, Baidu e Bing); e
- capacidade de responder perguntas em linguagem natural e desempenho super-humano em jogos como xadrez, shogi e Go.

No entanto, o DL possui limitações. As arquiteturas atuais de RNA não têm o poder de rastrear flutuações randômicas dos dados de treinamento em tempo real. Em outras palavras, o aprendizado profundo ainda não é adaptativo. Observe que o treinamento de um modelo de DL em uma estação de trabalho dedicada pode levar dias ou semanas. Isso se deve à sua complexidade computacional.

Além disso, o DL não é como a matemática ou a física, nas quais avanços teóricos podem ser alcançados com giz e quadro-negro. O DL, assim como a IA, é uma ciência empírica (SIMON, 1996), pois a nossa compreensão de por quê e quando o DL funciona permanece limitada. Por exemplo, não há critério de projeto para o número de camadas na rede, muito menos para o número de neurônios em uma camada oculta. O campo é impulsionado por descobertas experimentais. Mas é claro, existem melhores práticas a serem seguidas .

2.4 Redes Convolucionais

A convolução entre as funções $x(t)$ e $w(t)$ é definida como

$$s(t) = \int x(a)w(t-a) da. \quad (2.7)$$

A convolução é denotada por um asterisco

$$s(t) = x(t) * w(t). \quad (2.8)$$

O primeiro argumento da convolução é a **entrada** x , enquanto que o segundo é o **kernel** ou filtro w . O resultado da operação é o **mapa de atributos** (*feature map*). A convolução é uma operação linear e comutativa.

No caso de uma imagem I , a sua convolução bidimensional com um *kernel* K e dada por (GOODFELLOW; BENGIO; COURVILLE, 2016)

$$S(i, j) = (I * K)(i, j) \sum_m \sum_n I(m, n)K(i-m, j-n) \quad (2.9)$$

ou

$$S(i, j) = (K * I)(i, j) \sum_m \sum_n K(m, n)I(i-m, j-n) \quad (2.10)$$

O resultado da convolução é menor que a imagem original porque as bordas são descartadas. Além disso, cada filtro pode representar um atributo específico da imagem: reta, curva, etc.

O **campo receptivo** de uma unidade $a^{(i)}$ da i -ésima camada de uma rede é composto por todas as unidades das camadas $i - 1, i - 2, \dots, 1$ que afetam a unidade $a^{(i)}$.

Sucessivas convoluções conseguem detectar atributos de mais alto nível, tais como semicírculos, quadrados. etc.

Uma camada convolucional é tipicamente composta por (GOODFELLOW; BENGIO; COURVILLE, 2016) (em ordem), conforme ilustrado pela Fig. 16:

- um estágio convolucional;
- uma não linearidade (como a Relu); e
- um estágio de *pooling* (agrupamento).

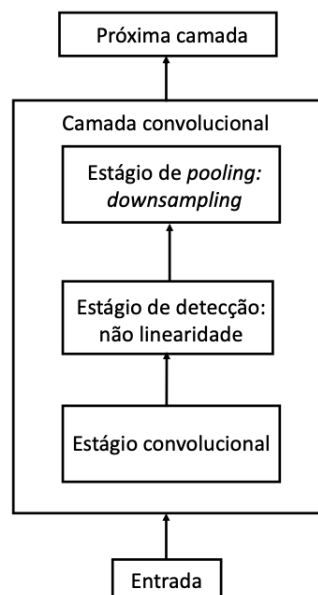


Figura 16 – componentes de uma camada convolucional.

O primeiro estágio é constituído por uma série de operações de convolução em paralelo, a qual produz um conjunto de ativações lineares.

No segundo estágio (estágio de detecção), cada ativação linear sofre uma transformação não linear por meio da aplicação da função de ativação não linear.

O estágio de *pooling* agrupa as ativações de uma dada região. O agrupamento pode ser aplicado, por exemplo, escolhendo-se as ativações de valor máximo por meio

de uma janela deslizante de 2×2 pixels. Por conseguinte, o *pooling* é uma operação de subamostragem (*downsampling*).

3 Experimentos Computacionais

Este capítulo apresenta os resultados obtidos em CV e FR por meio de simulações.

3.1 Reconhecimento de Dígitos Manuscritos

Esta seção apresenta os resultados dos experimentos com várias arquiteturas de RNA (Tabela 2) que têm a capacidade de reconhecer dígitos manuscritos, um problema clássico em CV (CUN et al., 1990). Note-se que a variável L na Tabela 2 denota o número total de camadas (incluindo as de entrada e de saída). O número total de pesos da rede é dado pela coluna “Parâmetros” (inclui pesos não ajustáveis).

Rede	Tipo	L	Otimizador	Parâmetros	Regularização	Overfitting
MNIST2	Densa	2	SGD	7.850	nenhuma	não
MNIST4	Densa	4	SGD	118.282	nenhuma	sim, $t > 20$
MNIST6a	Densa	6	RMSProp	118.282	dropout	sim, $t > 9$
MNIST6b	Densa	6	Adam	118.282	L2 e dropout	não
MNIST8	Densa	8	Adam	119.306	L2, dropout e Batch Normalization (BN)	não
MNISTCNN	CNN	8	Adam	431.080	nenhuma	sim, $t > 1$

Tabela 2 – Redes neurais treinadas para reconhecimento de dígitos manuscritos. A época de treinamento é notada por t .

Os códigos-fonte das redes encontram-se listados no Apêndice A em formato de arquivo .py.

O *The MNIST database of handwritten digits* (MNIST) é um banco de dados de dígitos manuscritos compostos de um conjunto de treinamento de 60.000 exemplos (amostras) e um conjunto de testes de 10.000 exemplos (LECUN; CORTES; BURGESS, 2020). Os exemplos de treinamento são anotados por humanos com a resposta correta. Por exemplo, se o dígito manuscrito for o número “3”, “3” será o rótulo (*label*) associado a esse exemplo. Cada imagem MNIST está codificada em escala de cinza e consiste em uma matriz de 28×28 pixels, em que cada pixel é um número inteiro na faixa $[0, 255]$.

O desafio central em DL é projetar uma rede neural que apresente um bom desempenho para novas entradas, ou seja, entradas para as quais o algoritmo não foi treinado. Essa habilidade é chamada de generalização. O que distingue a otimização em aprendizado de máquina da otimização tradicional é a questão do **erro de generalização**, ou **erro de teste**, que é a figura de mérito a ser minimizada (GOODFELLOW; BENGIO; COURVILLE, 2016). Para tal, precisa-se de um conjunto de teste. Este trabalho segue a melhor prática de dividir o conjunto de teste em dois conjuntos separados: conjuntos de validação e teste (CHOLLET, 2018). Ou seja, o poder de generalização dos modelos foi

medido em relação aos conjuntos de validação e teste. O conjunto de validação é usado para ajustar os hiperparâmetros da rede: número de camadas, número de unidades por camada.

Como corolário do acima exposto, segue-se que o problema central em aprendizado de máquina é a sobre-parametrização do modelo, ou *overfitting*, que ocorre quando o *gap* entre o erro de treinamento e o erro de generalização é muito grande. O combate ao *overfitting* é designado por **regularização**, que pode ser definida como qualquer modificação que se faz em um algoritmo de aprendizagem com o objetivo de reduzir seu erro de generalização, mas não seu erro de treinamento. Para ver este fenômeno, é necessário traçar as curvas de aprendizado de treinamento e generalização, conforme ilustrado pela Fig. 17. Note que os erros de treinamento e de generalização se comportam de maneira diferente. O eixo horizontal representa o número de épocas. O eixo vertical é a função custo (*loss function*). Observe o *overfitting* do modelo a partir da quinta época de treinamento¹.

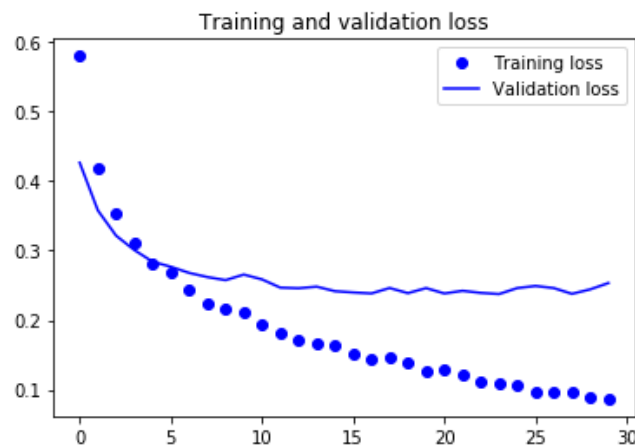


Figura 17 – o fenômeno do *overfitting*.

Segundo (CHOLLET, 2018, p. 107, 260):

“(...) Assim, uma maneira comum de mitigar a sobre-parametrização é impor restrições na complexidade de uma rede, forçando seus pesos [sinápticos] a assumirem apenas valores pequenos, o que torna a distribuição dos valores de peso mais regular. Isso é chamado de regularização de pesos e é feito adicionando à função de custo da rede um custo associado a ter grandes pesos. Em geral, esse custo pode ser do tipo:

- **Regularização L1** - O custo adicionado é proporcional ao valor absoluto dos peso (norma L1 dos pesos).

¹ *Training loss* significa a função custo de treinamento, enquanto que *validation loss* a função custo de validação.

- **Regularização L2** — O custo adicionado é proporcional ao quadrado do valor dos coeficientes de peso (a norma L2 dos pesos). A regularização L2 também é chamada de redução de pesos no contexto das redes neurais.

O **descarte** (*dropout*) é uma das técnicas de regularização mais eficazes e mais comumente usadas para redes neurais, desenvolvida por Geoff Hinton e seus alunos na Universidade de Toronto. O *dropout*, aplicada a uma camada, consiste em eliminar aleatoriamente (“zerar”) uma série de atributos (*features*) de saída da camada durante o treinamento. Digamos que uma determinada camada normalmente retornaria um vetor $[0,2 \ 0,5 \ 1,3 \ 0,8 \ 1,1]$ para um dado exemplo de treinamento. Após aplicar o *dropout*, este vetor terá algumas entradas “zeradas” distribuídas aleatoriamente: por exemplo, $[0 \ 0,5 \ 1,3 \ 0 \ 1,1]$. A taxa de *dropout* é a fração dos atributos que são “zerados”; geralmente é definido um valor entre 0,2 e 0,5. No momento do teste, nenhuma unidade é descartada; em vez disso, os valores de saída da camada são reduzidos por um fator igual à taxa de *dropout*, para equilibrar o fato de que mais unidades estão ativas do que no tempo de treinamento.

A **normalização em batelada** (*batch normalization*) é um tipo de camada (...) introduzida em 2015 por Ioffe e Szegedy (IOFFE; SZEGEDY, 2015); ela pode normalizar dados de forma adaptativa, mesmo quando a média e a variância mudam ao longo do tempo durante o treinamento. Ela funciona mantendo internamente uma média móvel exponencial da média e da variância dos dados vistos durante o treinamento em mini-bateladas. O principal efeito da *batch normalization* é que ela ajuda na propagação do gradiente (...) e, portanto, permite redes mais profundas. Algumas redes muito profundas só podem ser treinadas se incluírem várias camadas *batch normalization*. Por exemplo, *batch normalization* é usado em muitas das arquiteturas convolucionais avançadas que fazem parte das bibliotecas do Keras, como ResNet50, Inception V3 e Xception. ” (tradução livre)

Voltemos à Tabela 2. O otimizador RMSProp é dado pelas seguintes equações:

$$\theta_k = \theta_{k-1} - \frac{\eta}{\sqrt{E[g^2]_k + \epsilon}} \cdot g_k \quad (3.1)$$

em que

$$E[g^2]_k = 0.9E[g^2]_{k-1} + 0.1g_k^2 \quad (3.2)$$

e

$$g_k = \nabla_{\theta_k} J(\theta_k). \quad (3.3)$$

O otimizador *Adaptive Moment Estimation* ([Adam](#)) possui a seguinte regra de atualização:

$$\theta_k = \theta_{k-1} - \frac{\eta}{\sqrt{\hat{v}_k} + \epsilon} \cdot \hat{m}_k \quad (3.4)$$

em que

$$\hat{m}_k = \frac{m_k}{1 - \beta_1^k} \quad (3.5)$$

e

$$\hat{v}_k = \frac{v_k}{1 - \beta_w^k}. \quad (3.6)$$

As variáveis m_k e v_k são estimativas dos momentos de primeira e segunda ordem do gradiente, respectivamente:

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k \quad (3.7)$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2. \quad (3.8)$$

Os proponentes do Adam indicam o uso dos seguintes valores *default*: 0,9 para β_1 , 0,999 para β_2 e 10^{-7} para ϵ ([CHOLLET, 2020](#)).

A Tabela 3 mostra que é possível atingir acurácia de previsão do dígito manuscrito superior à 99,0% utilizando a rede convolucional MNISTCNN, que foi projetada por este trabalho. Este resultado corrobora que, de fato, [DL](#) é o estado da arte em [CV](#) ([GOODFELLOW; BENGIO; COURVILLE, 2016](#)).

Rede	Acurácia
MNIST2	92,18%
MNIST4	96,59%
MNIST6a	97,79%
MNIST6b	97,26%
MNIST8	97,10%
MNISTCNN	99,10%

Tabela 3 – Desempenho das redes neurais.

As Figs. [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#) e [29](#) mostram as curvas de aprendizado das redes neurais da Tabela 3.

3.2 Reconhecimento Facial

O principal objetivo deste trabalho foi alcançado: o código fonte Python de [FR](#) de ([BUSSON; et al, 2018](#)), que usa o modelo estado da arte pré-treinado FaceNet ([SCHROFF; KALENICHENKO; PHILBIN, 2015](#)), foi mapeado da plataforma TensorFlow 1.14 para TensorFlow 2 ([GOOGLE, 2020b](#)). O código encontra-se listado no Apêndice [A](#).

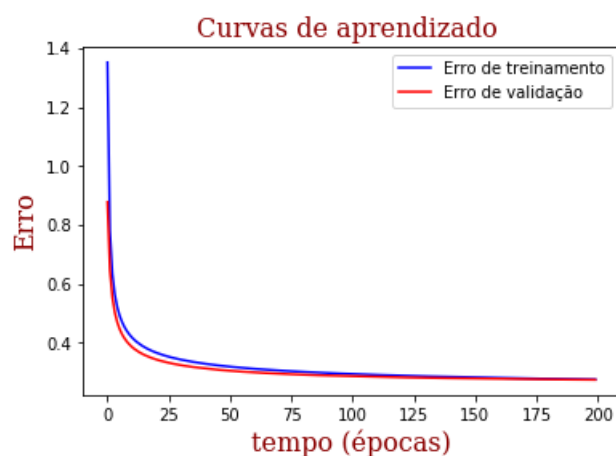


Figura 18 – Rede MNIST2: curva de aprendizado – erro.

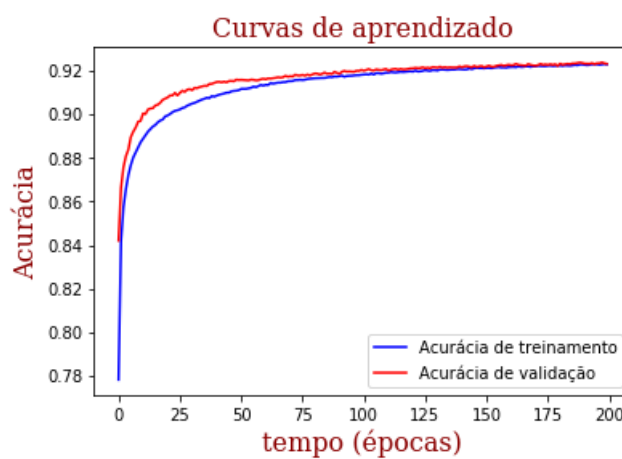


Figura 19 – Rede MNIST2: curva de aprendizado – acurácia.

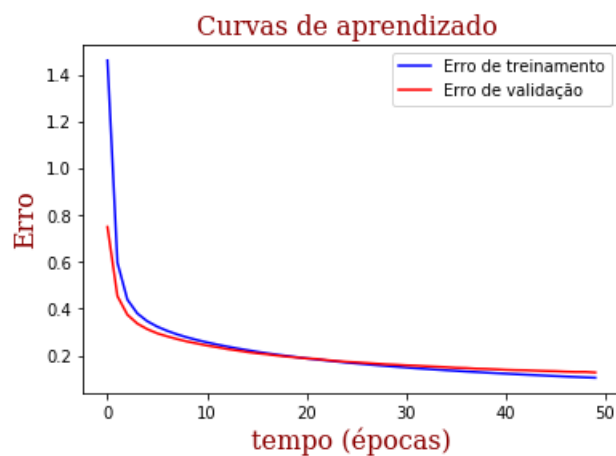


Figura 20 – Rede MNIST4: curva de aprendizado – erro.

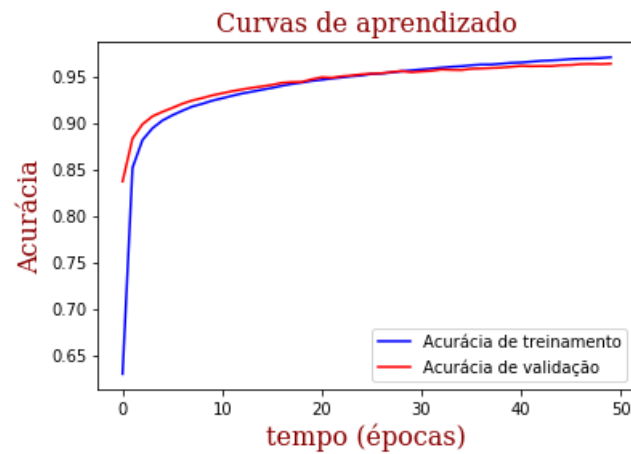


Figura 21 – Rede MNIST4: curva de aprendizado – acurácia.

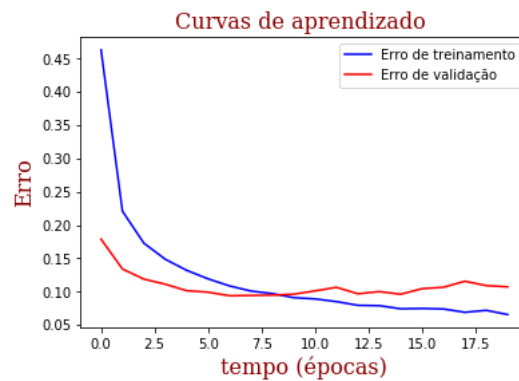


Figura 22 – Rede MNIST6a: curva de aprendizado – erro.

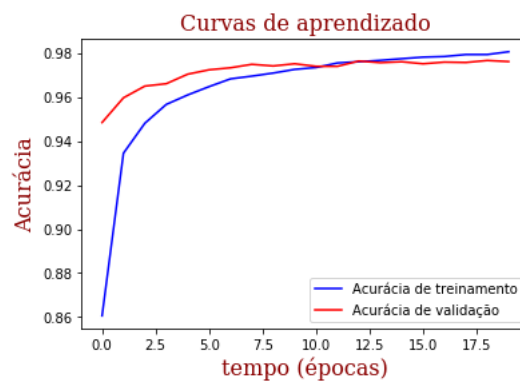


Figura 23 – Rede MNIST6a: curva de aprendizado – acurácia.

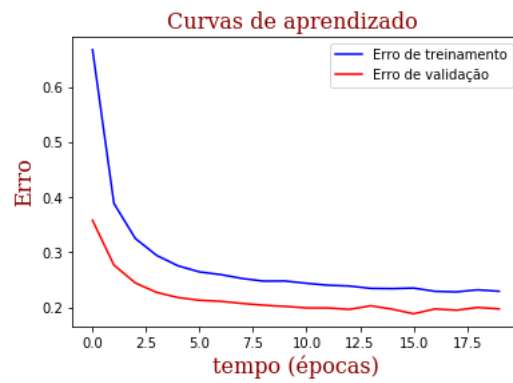


Figura 24 – Rede MNIST6b: curva de aprendizado – erro.

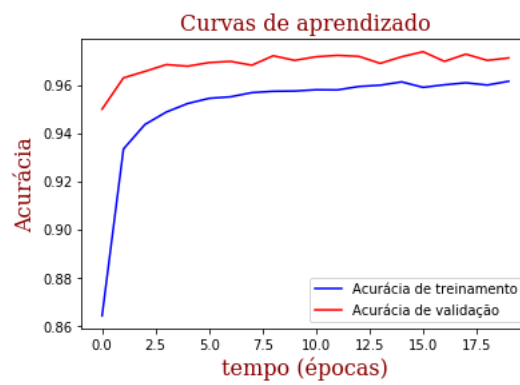


Figura 25 – Rede MNIST6b: curva de aprendizado – acurácia.

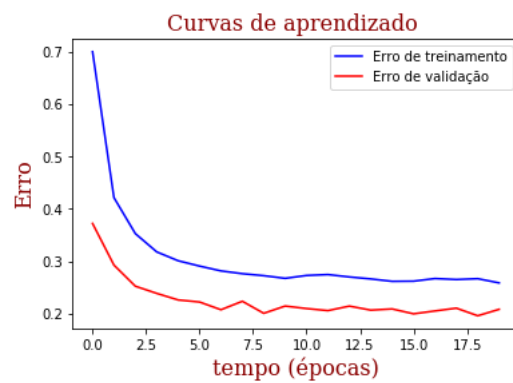


Figura 26 – Rede MNIST8: curva de aprendizado – erro.

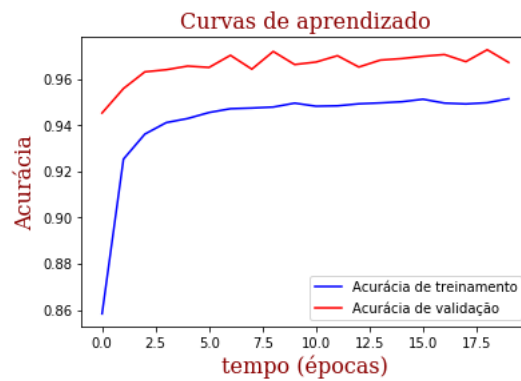


Figura 27 – Rede MNIST8: curva de aprendizado – acurácia.

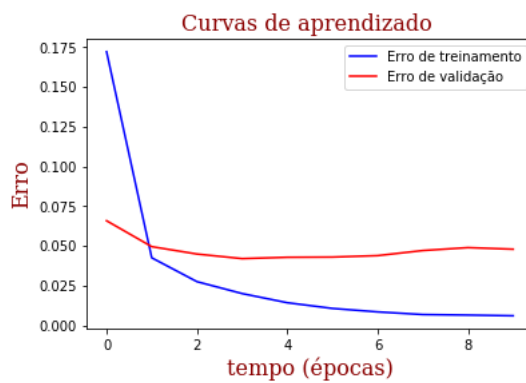


Figura 28 – Rede MNISTCNN: curva de aprendizado – erro.

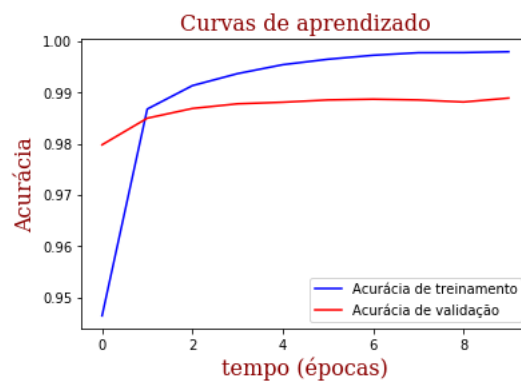


Figura 29 – Rede MNISTCNN: curva de aprendizado – acurácia.

Note-se que há diferenças relevantes entre o TensorFlow 1 e o TensorFlow 2. Os códigos escritos para as versões 1 e 2 são incompatíveis.

O Keras é a [API](#) oficial do TensorFlow 2 ([GULLI; KAPOOR; PAL, 2019](#)). A estrutura de programação em TensorFlow 1 é baseada na noção de grafo computacional (consulte ([GULLI; KAPOOR; PAL, 2019](#)) para maiores informações). O TensorFlow 2 suporta *eager execution*, o que quer dizer que as definições do modelo são dinâmicas e a execução é imediata. Há outras diferenças. Uma outra boa referência é o livro ([CHOLLET, 2018](#)).

O FaceNet verifica a face de uma pessoa por meio da comparação da imagem atual com a imagem cadastrada no banco de dados (trata-se do mesmo indivíduo?), reconhece a face (quem é esta pessoa?) e discrimina a face em um conjunto de faces cadastradas.

A Fig. 30 mostra a estrutura da FaceNet. O método é baseado na extração de um vetor x de atributos (*face embedding*) de uma dada imagem. Na sequência, a similitude entre as imagens é calculada pela Norma-2 (distância euclidiana), dada por

$$||x||_2 = \sqrt{\sum x_i^2}, \quad (3.9)$$

entre a imagem sob análise e todas as imagens cadastradas. A menor distância corresponde à identidade do indivíduo; se a distância for maior que um dado limiar, isto quer dizer que a pessoa não está registrada na base de dados.

A Fig. 31 ilustra a **função de perda tríplice** (A, P, N) usada pelo *framework* FaceNet em que:

- A (âncora) denota uma imagem de uma face;
- P (positiva) é a imagem de face da mesma pessoa de A ; e
- N (negativa) é a imagem de face de um indivíduo diferente de A .

O objetivo é fazer com que A fique próximo da sua P e mais distante de N . A formulação matemática da função de perda tríplice é dada em detalhes pelos proponentes do FaceNet em ([SCHROFF; KALENICHENKO; PHILBIN, 2015](#)).

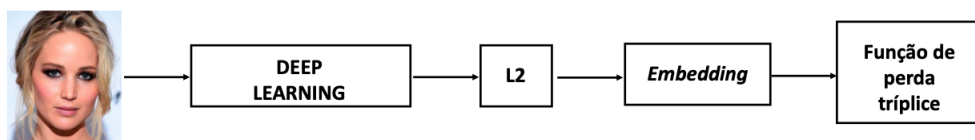


Figura 30 – arquitetura FaceNet.

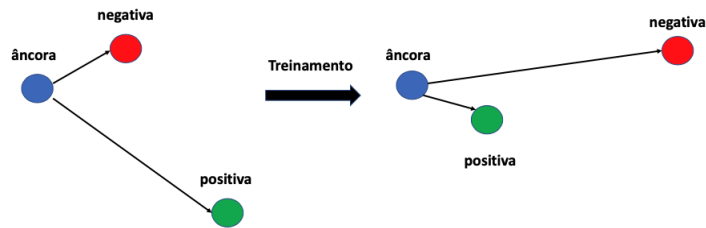


Figura 31 – função de perda tríplice.

As Figs. 32 e 33 mostram as fotos cadastradas e o reconhecimento facial das celebridades Ozzy Osbourne e Brad Pitt, respectivamente.

Ressalta-se que o objetivo primário desta pesquisa não é mensurar o desempenho do FaceNet, mas tão somente migrar o código para TensorFlow 2 e testar a sua funcionalidade, o que efetivamente foi realizado.

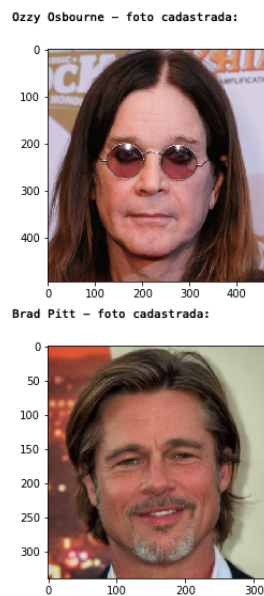


Figura 32 – Fotos cadastradas: Ozzy Osbourne e Brad Pitt.



Figura 33 – Reconhecimento facial.

4 Conclusões e Trabalhos Futuros

Este capítulo sumariza os resultados obtidos e lista algumas sugestões para trabalhos futuros.

O crescente interesse pela área de [FR](#) e a necessidade de tecnologias não intrusivas de biometria impulsiona a pesquisa em [IA](#), mais especificamente em [DL](#).

A aplicação bem sucedida das redes convolucionais em reconhecimento de dígitos manuscritos no final dos anos 1980 abriu as portas para a supremacia do [DL](#) em [CV](#) a partir da década de 2010.

O principal resultado desta pesquisa consiste na reescrita (ou portabilidade) do código fonte Python de reconhecimento facial de ([BUSSON; et al, 2018](#)), que usa o modelo pré-treinado FaceNet ([SCHROFF; KALENICHENKO; PHILBIN, 2015](#)) e a biblioteca TensorFlow 1.14, para o *framework* TensorFlow 2 com [API](#) Keras. Os demais resultados obtidos, tais como reconhecer dígitos manuscritos, um problema clássico em visão computacional, confirmam que as redes convolucionais realmente são o estado da arte nesta área de pesquisa.

Finalmente, segue-se um breve lista (não exaustiva) com sugestões para trabalhos futuros:

- Mensurar o desempenho do FaceNet.
- Testar o Facenet na presença de ruído.
- Investigar outros *frameworks* de [FR](#) e comparar os respectivos desempenhos com o do FaceNet.

Referências

ADCOCK, B.; DEXTER, N. *The gap between theory and practice in function approximation with deep neural networks*. 2020. Citado na página 21.

AHONEN, T.; HADID, A.; PIETIKAINEN, M. Face Description with Local Binary Patterns: Application to Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 28, n. 12, p. 2037–2041, Dec. 2006. Citado na página 19.

ALLAIRE, F. C. J. J. *Deep Learning with R*. 1st ed.. ed. [S.l.]: Manning Publications, 2017. Citado 2 vezes nas páginas 38 e 40.

Anaconda. *Anaconda Open-Source Distribution*. 2020. Disponível em: <<https://www.anaconda.com/products/individual>>. Acesso em: 07/07/ 2020. Citado na página 25.

APACHE. *MXNet*. 2020. Disponível em: <<https://mxnet.apache.org>>. Acesso em: 07/07/ 2020. Citado na página 23.

BODEN, M. A. *AI Its Nature and Future*. 1st. ed. [S.l.]: Oxford University Press, 2016. Citado 2 vezes nas páginas 34 e 35.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: *Proceedings of the fifth annual workshop on Computational learning theory: COLT'92*. [S.l.: s.n.], 1992. p. 144–152. Citado na página 19.

BROWNLEE, J. *Deep Learning for Computer Vision: Image Classification, Object Detection and Face Recognition in Python – e-book*. v1.7. Jason Brownlee, 2020. Disponível em: <<https://machinelearningmastery.com>>. Citado 4 vezes nas páginas 15, 16, 20 e 21.

BUSSON, A. J. G.; et al. Desenvolvendo Modelos de Deep Learning para Aplicação multimídia no tensorflow. In: *Anais do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web: Minicursos*. [S.l.: s.n.], 2018. Citado 3 vezes nas páginas 23, 48 e 57.

CAO, Z. et al. Face Recognition with Learning-Based Descriptor. In: IEEE. *CVPR*. [S.l.], 2010. p. 2707–2714. Citado na página 20.

CHEN, B.; MEDINI, T.; SHRIVASTAVA, A. SLIDE: In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems. *arXiv*, march 2019. Citado na página 31.

CHIHAOUI, M. et al. A Survey of 2D Face Recognition Techniques. *Computers*, v. 5, n. 21, p. 1–28, 2016. Citado 3 vezes nas páginas 15, 17 e 18.

CHOLLET, F. *Deep Learning with Python*. 1st. ed. [S.l.]: Manning Publications, 2018. Citado 8 vezes nas páginas 21, 23, 24, 31, 40, 45, 46 e 53.

CHOLLET, F. *Keras*. 2020. Disponível em: <<https://keras.io/>>. Acesso em: 07/07/ 2020. Citado na página 48.

- CHOLLET, F.; ALLAIRE, J. J. *Deep Learning with R*. 1st. ed. [S.l.]: Manning Publications, 2017. Citado na página 23.
- Cornell University. *arXiv*. 2020. Disponível em: <<https://arxiv.org/about>>. Citado na página 25.
- CUN, L. et al. Handwritten Digit Recognition with a Back-Propagation Network. In: *Advances in Neural Information Processing Systems*. [S.l.]: Morgan Kaufmann, 1990. p. 396–404. Citado 2 vezes nas páginas 21 e 45.
- FACEBOOK. *Caffe2*. 2020. Disponível em: <<https://caffe2.ai>>. Acesso em: 07/07/ 2020. Citado na página 23.
- Fuegi, J.; Francis, J. Lovelace babbage and the creation of the 1843 notes. *IEEE Annals of the History of Computing*, v. 25, n. 4, p. 16–26, 2003. Citado na página 29.
- GitHub. *GitHub Code Repository*. 2020. Disponível em: <<https://github.com>>. Acesso em: 01/06/ 2020. Citado na página 23.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1st. ed. [S.l.]: MIT Press, 2016. Citado 9 vezes nas páginas 22, 29, 34, 36, 37, 41, 42, 45 e 48.
- GOODFELLOW, I. J. et al. *Generative Adversarial Networks*. 2014. Disponível em: <<https://arxiv.org/abs/1406.2661>>. Citado na página 24.
- GOOGLE. *Colaboratory*. 2020. Disponível em: <<https://colab.research.google.com/notebooks/intro.ipynb>>. Acesso em: 07/07/ 2020. Citado na página 23.
- GOOGLE. *TensorFlow*. 2020. Disponível em: <<https://www.tensorflow.org/?hl=pt-br>>. Acesso em: 07/07/ 2020. Citado 2 vezes nas páginas 23 e 48.
- GRAVES, A.; WAYNE, G.; DANIHELKA, I. Neural Turing Machines. *CoRR*, abs/1410.5401, 2014. Disponível em: <<https://arxiv.org/abs/1410.5401>>. Citado na página 24.
- GULLI, A.; KAPOOR, A.; PAL, S. *Deep Learning with TensorFlow 2 and Keras*. 2nd. ed. [S.l.]: Packt>, 2019. Citado 2 vezes nas páginas 23 e 53.
- HAFFNER, Y. L. L. B. Y. B. P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, November 1998. Citado na página 38.
- HINTON, G. E.; OSINDERO, S.; TEH, Y. A fast learning algorithm for deep belief nets. *Neural Computation*, v. 18, p. 1527–1554, 2006. Citado 3 vezes nas páginas 19, 37 e 38.
- HUANG, G. B. et al. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. In: Erik Learned-Miller and Andras Ferencz and Frédéric Jurie. *Workshop on Faces in Real-Life Images: Detection, Alignment, and Recognition*. Marseille, France, 2008. Disponível em: <<https://hal.inria.fr/inria-00321923>>. Citado na página 18.
- IOFFE, S.; SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *Proceedings of the 32nd International Conference on Machine Learning*. [S.l.: s.n.], 2015. Citado na página 47.

- JACKEL, Y. L. B. E. B. J. S. D. D. H. R. E. H. W. E. H. L. D. Handwritten digit recognition with a back-propagation network. In: TOURETZKY, D. S. (Ed.). *Advances in Neural Information Processing Systems 2*. Morgan-Kaufmann, 1990. p. 396–404. Disponível em: <<http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf>>. Citado na página 38.
- KANADE, T. *Picture Processing System by Computer Complex and Recognition of Human Faces*. Doctoral thesis — University of Kyoto, 1973. Citado na página 18.
- KELLY, M. D. Technical Report, *Visual Identification of People by Computer, Stanford AI Project*. 1970. Citado na página 17.
- KORTLI, Y. et al. Face Recognition Systems: A Survey. *Sensors*, v. 20, n. 342, 2020. Disponível em: <<https://www.mdpi.com/1424-8220/20/2/342>>. Citado 2 vezes nas páginas 15 e 17.
- KOVÁCS, Z. L. *Redes Neurais Artificiais*. 4a. ed. [S.l.]: Livraria da Física, 2006. Citado na página 36.
- KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, p. 1097–1105, 2012. Citado 2 vezes nas páginas 18 e 19.
- LANTZ, B. *Machine Learning with R*. 1st. ed. [S.l.]: Packt Publishing, 2013. Citado na página 31.
- LECUN, Y. Generalization and network design strategies. In: STEELS, R. P. Z. S. F. F. L. (Ed.). *Connectionism in Perspective*. Zurich, Switzerland: Elsevier, 1989. Citado na página 38.
- LECUN, Y.; CORTES, C.; BURGESS, C. J. C. *The MNIST database*. 2019. Disponível em: <<http://yann.lecun.com/exdb/mnist/>>. Citado na página 38.
- LECUN, Y.; CORTES, C.; BURGESS, C. J. C. *The MNIST Database of handwritten digits*. 2020. Disponível em: <<http://yann.lecun.com/exdb/mnist/>>. Citado na página 45.
- LECUN, Y. et al. Comparison of Learning Algorithms for Handwritten Digit Recognition. In: *INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS*. [S.l.: s.n.], 1995. p. 53–60. Citado na página 18.
- LI, S. Z.; JAIN, A. K. *Handbook of Face Recognition*. 2nd. ed. [S.l.]: Springer, 2011. Citado na página 16.
- LIMA, A. B. de; BARRETO, M. R. P.; AMAZONAS, J. R. de A. Spectra-based sentiment analysis. *INFOCOMP*, v. 17, n. 2, p. 01–06, december 2018. Citado na página 30.
- LIU, C.; WECHSLER, H. Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition. *IEEE Transactions on Image Processing*, v. 11, n. 4, p. 467–476, april 2002. Citado na página 19.
- LOVELACE, A. A. Notes by A. A. L. *Taylor's Scientific Memoirs*, III, p. 666–731, 1843. Disponível em: <<http://www.fourmilab.ch/babbage/sketch.html>>. Citado na página 29.

- MACK, C. A. Fifty Years of Moore's Law. *IEEE Transactions on Semiconductor Manufacturing*, v. 24, n. 2, p. 202–207, 2011. Citado na página 15.
- MATHWORKS. *MATLAB for Deep Learning*. 2020. Disponível em: <<https://www.mathworks.com/solutions/deep-learning.html>>. Acesso em: 08/07/ 2020. Citado na página 23.
- MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 115–133, 1943. Disponível em: <<http://www.cs.cmu.edu/~epxing/Class/10715/reading/McCulloch.and.Pitts.pdf>>. Citado 2 vezes nas páginas 34 e 35.
- MICROSOFT. *Microsoft Cognitive Toolkit*. 2020. Disponível em: <<https://docs.microsoft.com/en-us/cognitive-toolkit/>>. Acesso em: 07/07/ 2020. Citado na página 23.
- MOORE, G. E. Cramming More Components onto Integrated Circuits. *Electronics*, v. 38, n. 8, p. 114–117, 1965. Citado na página 15.
- NAGENDRAN, N.; KOLHE, A. Security and Safety with Facial Recognition Feature for Next Generation Automobiles. *International Journal of Recent Technology and Engineering*, v. 7, n. 4, p. 289–294, 2018. Citado na página 15.
- NEWELL, A. Physical symbol systems. *Cognitive Science*, v. 4, p. 135–183, April–June 1980. Citado na página 35.
- NG, A. *Machine Learning – Coursera*. 2019. Disponível em: <<https://www.coursera.org/>>. Citado 2 vezes nas páginas 30 e 34.
- NILSSON, N. J. *The Quest for Artificial Intelligence: a History of Ideas and Achievements*. 1st. ed. Cambridge University Press, 2010. Disponível em: <<http://ai.stanford.edu/~nilsson/QAI/qai.pdf>>. Citado 2 vezes nas páginas 15 e 17.
- Octave Forge. *Octave*. 2020. Disponível em: <<https://octave.sourceforge.io/packages.php>>. Acesso em: 10/07/ 2020. Citado na página 23.
- OPPENHEIM, A. V.; SCHAFER, R. W. *Processamento de Sinais em Tempo Discreto*. 3a. ed. [S.l.]: Pearson, 2019. Citado na página 16.
- PARKHI, O. M.; VEDALDI, A.; ZISSERMAN, A. Deep Face Recognition. In: XIE, X.; JONES, M. W.; TAM, G. K. L. (Ed.). *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2015. p. 41.1–41.12. ISBN 1-901725-53-7. Disponível em: <<https://dx.doi.org/10.5244/C.29.41>>. Citado na página 20.
- PASZKE, A. et al. *PyTorch*. 2020. Disponível em: <<https://pytorch.org>>. Acesso em: 07/07/ 2020. Citado na página 23.
- PÉREZ, F.; GRANGER, B. E. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*, IEEE Computer Society, v. 9, n. 3, p. 21–29, may 2007. ISSN 1521-9615. Disponível em: <<https://ipython.org>>. Citado na página 23.
- Project Jupyter. *The Jupyter Notebook*. 2020. Disponível em: <<https://jupyter.org/>>. Acesso em: 07/07/ 2020. Citado na página 23.

- R. Anderson et al. Chapter I – Introduction. In: BARR, A.; FEIGENBAUM, E. A. (Ed.). *The Handbook of Artificial Intelligence*. [S.l.]: Elsevier, 1981. v. 1. Citado na página 34.
- RAHIMI, A. *Ali Rahimi's talk at NIPS Test-of-Time award presentation*. 2017. Disponível em: <<https://youtu.be/Qi1Yry33TQE>>. Citado na página 21.
- ROBERTS, L. G. *Machine Perception of Three-Dimensional Solids*. Doctoral thesis — MIT, 1963. Citado na página 17.
- Rowley, H. A.; Baluja, S.; Kanade, T. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 20, n. 1, p. 23–38, 1998. Citado na página 19.
- RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015. Citado na página 18.
- RUSSEL, S.; NORVIG, P. *Inteligência Artificial*. terceira ed. Rio de Janeiro: Elsevier, 2013. Disponível em: <<http://aima.cs.berkeley.edu/>>. Acesso em: 15/07/ 2020. Citado 5 vezes nas páginas 17, 30, 34, 35 e 38.
- SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. Citado 5 vezes nas páginas 20, 23, 48, 53 e 57.
- SILVER, D.; AL et. General Reinforcement Learning Algorithm. *arXiv*, 2017. Disponível em: <https://arxiv.org/pdf/1712.01815.pdf?utm_campaign=nathan.ai%20newsletter&utm_medium=email&utm_source=Revue%20newsletter>. Citado na página 32.
- SILVER, D.; AL. et. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *Science*, v. 362, p. 1140–1144, dec. 2018. Citado na página 40.
- SIMON, H. A. Artificial intelligence: an empirical science. *Artificial Intelligence - Elsevier*, v. 77, p. 95–127, 1996. Citado na página 41.
- SKANSI, S. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. 1st. ed. [S.l.]: Springer, 2018. Citado na página 23.
- SOLOMON, C.; BRECKON, T. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. 1st. ed. [S.l.]: Wiley-Blackwell, 2011. Citado 2 vezes nas páginas 16 e 17.
- Sun, Y.; Wang, X.; Tang, X. Deep Learning Face Representation from Predicting 10,000 Classes. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 1891–1898. Citado na página 20.
- TAIGMAN, Y. et al. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2014. Citado na página 20.
- TURING, A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, v. 42, p. 230–265, 1936. Citado na página 35.

TURING, A. M. Computing Machinery and Intelligence. *Mind*, v. 49, p. 433–460, 1950. Citado na página 29.

TURK, M.; PENTLAND, A. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, v. 3, p. 71–86, 1991. Citado na página 18.

VIOLA, P.; JONES, M. J. Technical Report, *Robust Real-Time Object Detection*. 2001. Citado na página 19.

VOULODIMOS, A. et al. Deep Learning for Computer Vision: A Brief Review. *Hindawi Computational Intelligence and Neuroscience*, 2018. Disponível em: <https://www.hindawi.com/journals/cin/2018/7068349/>. Citado na página 19.

WANG, M.; DENG, W. Deep Face Recognition: A survey. *CoRR*, abs/1804.06655, 2018. Disponível em: <http://arxiv.org/abs/1804.06655>. Citado 3 vezes nas páginas 17, 18 e 20.

Apêndices

APÊNDICE A – Códigos Python

A.1 Reconhecimento de Dígitos Manuscritos

```

# REDE MNIST2
2 # otimizador = Descida pelo Gradiente Estocástico
# Stochastic Gradient Descent (SGD)
4 # código para TensorFlow 2
# IDE Spyder – Anaconda
6
# Uso do TensorFlow 2 para definir uma rede que reconhece dígitos
8 # manuscritos MNIST (http://yann.lecun.com/exdb/mnist/):

10 # O MNIST é um banco de dados de dígitos manuscritos compostos de um
# conjunto de treinamento de 60.000 exemplos (amostras) e um conjunto de
12 # testes de 10.000 exemplos. Os exemplos de treinamento são anotados por
# humanos com a resposta correta.
14 # Por exemplo, se o dígito manuscrito for o número "3", "3" será o rótulo
# (label) associado a esse exemplo. Cada imagem MNIST está codificada
16 # em escala de cinza e consiste em uma matriz de 28 x 28 pixels, em que
# cada pixel é um número inteiro na faixa [0, 255]
18
# Seguindo o estilo Keras, o TensorFlow 2 fornece bibliotecas adequadas
20 # (https://www.tensorflow.org/api\_docs/python/tf/keras/datasets)
# para carregar o conjunto de dados e dividi-lo em conjuntos de treinamento
22 # X_train, usado para treinar a rede, e conjuntos de testes, X_test,
# usados para avaliar o desempenho. Os dados são convertidos
24 # para o formato ‘float32’ (32 bits em ponto flutuante) e normalizados
# para o intervalo [0, 1]. Além disso, carregamos
26 # os rótulos verdadeiros em Y_train e Y_test, respectivamente
# e executamos uma codificação "one-hot" neles.
28 # Exemplo de codificação one-hot do dígito 3 (feature categórica
# ou não-numérica) em um vetor com 10 elementos:
30 # v = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

32 import tensorflow as tf
from tensorflow import keras
34 print("Tensorflow version " + tf.__version__)

36 # Parâmetros da rede e de treinamento:

38 epocas = 200 # define a duração do treinamento
lote_tam = 128 # número de amostras que alimentarão a rede
40 # em uma dada época de treinamento

```

```
# (em inglês, lote = batch)
42 verbose = 1
   n_classes = 10 # número de saídas = número de dígitos
44 val_split = 0.2
   # fração do número de amostras de treinamento reservadas para
46 # validação => 48.000 amostras para treino + 12.000 amostras para
   # validação = 60.000 exemplos do conjunto de treinamento MNIST
48
   # carrega a base de dados MNIST
50 mnist = keras.datasets.mnist
   (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
52
   # Em Python, uma lista é uma sequência de objetos separados por vírgulas.
54 # Os objetos podem ser de qualquer tipo: números, strings
   # e até mesmo outras listas.
56 # Por exemplo, digite a linha seguinte no Console:
   # animais = ['peixe', 'gato', 'cão']
58 # A variável "animais" é uma lista de objetos de string
   # Indexação dos itens da lista "animais":
60 # Digite as seguintes linhas no Console:
   # animais[0]
62 # animais[1]
   # animais[2]
64 # animais[-3]
   # animais[-2]
66 # animais[-1]

68 # X_train, Y_train, X_test e Y_test são listas numéricas designadas
   # por "array".
70 # X_train possui 60.000 imagens de dígitos manuscritos. Cada imagem
   # está em escala de cinza e consiste em uma matriz de 28 x 28 pixels,
72 # um total de 784 pixels/imagem, em que cada pixel é um número
   # inteiro na faixa [0, 255] (representação com 8 bits).
74 # O mesmo se aplica a X_test.

76 # Cada imagem de X_train e X_test é vetorizada para o formato de
   # vetor coluna com dimensões 784 x 1
78
   # Assim, X_train e X_test serão reformatados para matrizes
80 # com dimensões 60.000 x 784

82 reformat = 784
   # É a dimensão do espaço de "features"
84 # ou seja, a camada de entrada possui 784 "input units"
   # Uma entrada é um vetor coluna 784 x 1
86
   X_train = X_train.reshape(60000, reformat)
```

```
88 X_test = X_test.reshape(10000, reformat)
# "reshape" é um método de "numpy.ndarray"
90 # que retorna um "array" que contém os mesmos
# dados, só que em um outro formato
92
# A representação dos números contidos nos "arrays" X_train e X_test
94 # será convertida para representação em ponto flutuante com 32 bits.
# Para maiores detalhes, consulte:
96 # http://www.ic.uff.br/~simone/scminter/contaulas/6\_FLOAT.pdf

98 X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
100
# normalização em [0,1]
102 X_train /= 255
X_test /= 255
104 print(X_train.shape[0], 'amostras de treinamento')
print(X_test.shape[0], 'amostras de teste')
106
# Representação "one-hot" dos rótulos (labels) de treinamento e teste
108 Y_train = tf.keras.utils.to_categorical(Y_train, n_classes)
Y_test = tf.keras.utils.to_categorical(Y_test, n_classes)
110
# Construção do modelo
112 # a camada de saída é composta por 10 neurônios com função
# de ativação "softmax", que é uma generalização da função
114 # sigmóide ou logística.

116 model = tf.keras.models.Sequential()
model.add(keras.layers.Dense(n_classes, input_shape=(reformat,),
    kernel_initializer='zeros', name='dense_layer', activation='softmax'))
118
# sumário do modelo
120 model.summary()

122 # Depois de definir o modelo, é preciso compilá-lo para que ele possa
# ser executado pelo TensorFlow 2
124
# Opções a serem feitas durante a compilação.
126
# 1) Selecionar um otimizador, que é o algoritmo específico usado para
128 # atualizar os pesos da rede enquanto o modelo é treinado
# vide: https://www.tensorflow.org/api\_docs/python/tf/keras/optimizers
130
# 2) Selecionar uma função custo a ser minimizada pelo otimizador
132 # Nota: frequentemente, a função custo é designada por função
# objetivo ou função de perda
```

```
134 # vide: https://www.tensorflow.org/api\_docs/python/tf/keras/losses
136 # 3) Avaliar o modelo treinado.
138 # compilando o modelo
139 model.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['
    accuracy'])
140
141 # Otimizador SGD:
142 # Na prática, utiliza-se algum algoritmo de otimização que
143 # aproxime o algoritmo do gradiente. Por exemplo, em cada
144 # passo (época) do algoritmo, calcula-se uma estimativa do
145 # gradiente por meio de uma pequena amostra aleatória
146 # ("minibatch") do conjunto de treinamento. A estimação
147 # do gradiente é feita pelo algoritmo de "backpropagation".
148 # O algoritmo de otimização é estocástico porque trabalha
149 # com os "minibatches", em vez de usar todos os exemplos
150 # de treinamento ("batch") em cada passo. Neste caso, seria
151 # denominado algoritmo do gradiente determinístico.
152 # O tamanho do "mini-batch" é especificado pelo argumento
153 # "batch_size" em model.fit().
154 # O otimizador SGD do TensorFlow é do tipo "minibatch
155 # stochastic". A grande vantagem do SGD é que o tempo
156 # de computação da estimativa do gradiente não aumenta
157 # com o tamanho do conjunto de treinamento!
158
159 # Algumas opções comuns para a função de perda são:
160 # Mean Square Error (MSE), que define o erro médio quadrático
161 # entre as previsões e os valores reais.
162 # "categorical_crossentropy", que define a perda logarítmica multiclasse.
163 # "binary_crossentropy", que define a perda logarítmica binária.
164
165 # As métricas são semelhantes às funções de perda, com a única
166 # diferença de que elas não são usadas para treinar um modelo,
167 # mas apenas para avaliar o modelo. Algumas escolhas comuns para a métrica:
168
169 # Acurácia: calcula com que frequência as previsões são iguais aos rótulos.
170 # Precisão: define quantos itens selecionados são relevantes para uma
171 # classificação multi-rótulo.
172 # vide: https://www.tensorflow.org/api\_docs/python/tf/keras/metrics
173
174 # Depois que o modelo é compilado, ele pode ser treinado com o método fit()
175 # ,
176 # que especifica alguns parâmetros:
177
178 # - epochs (época): é o número de vezes que o modelo é exposto
179 # ao conjunto de treinamento. A cada iteração, o otimizador
```



```
# tenta ajustar os pesos para que a função custo seja minimizada.
180
# - batch_size (tamanho do lote): é o número de exemplos de treinamento
182 # observadas antes do otimizador executar uma atualização de peso;
# geralmente há muitos lotes por época.
184
# treinamento do modelo no TensorFlow 2
186 treino = model.fit(X_train, Y_train, batch_size=lote_tam, epochs=epocas,
                    verbose=verbose, validation_split=val_split)
# model.fit retorna o objeto "treino", que é do tipo "History".
188
treino_dic = treino.history
190 # "treino.history" retorna um contêiner de dicionário (uma generalização do
# conceito de lista): tipo "dict"
# Um dicionário contém pares de (chave, valor)
192 treino_dic.keys()
# chaves no tf 2.1.0: ['loss', 'accuracy', 'val_loss', 'val_accuracy']
194
import matplotlib.pyplot as plt
196
font = {'family': 'serif',
198 'color': 'darkred',
'weight': 'normal',
200 'size': 16,
}
202
acc = treino.history['accuracy']
204 val_acc = treino.history['val_accuracy']
loss = treino.history['loss']
206 val_loss = treino.history['val_loss']

208 epochs = range(len(acc))

210 plt.clf()

212 plt.plot(epochs, acc, 'b', label='Acurácia de treinamento')
plt.plot(epochs, val_acc, 'r', label='Acurácia de validação')
214 plt.title('Curvas de aprendizado', fontdict=font)
plt.xlabel('tempo (épocas)', fontdict=font)
216 plt.ylabel('Acurácia', fontdict=font)
plt.legend()
218
plt.figure()
220
plt.plot(epochs, loss, 'b', label='Erro de treinamento')
222 plt.plot(epochs, val_loss, 'r', label='Erro de validação')
plt.title('Curvas de aprendizado ', fontdict=font)
```

```
224 plt.xlabel('tempo (épocas)', fontdict=font)
    plt.ylabel('Erro', fontdict=font)
226 plt.legend()

228 plt.show()

230 # A curva de aprendizado (erro) indica que não ocorre
    # overfitting!
232
    # avaliação do modelo
234 test_loss, test_acc = model.evaluate(X_test, Y_test)
    print('Acurácia do teste:', test_acc)
236 # Acurácia do teste: 92,18%

238 # previsões
    predictions = model.predict(X_test)
```

```
# REDE MNIST4
2 # otimizador = SGD
# código para TensorFlow 2.1.0 Spyder – Anaconda
4
import tensorflow as tf
6 from tensorflow import keras
print("Tensorflow version " + tf.__version__)
8
# Parâmetros da rede e de treinamento
10 epocas = 50 # define a duração do treinamento
lote_tam = 128 # número de amostras que alimentarão a rede em uma dada é
    poca
12 # de treinamento (em inglês, lote = batch)
verbose = 1
14 n_classes = 10 # número de saídas = número de dígitos
n_oculta = 128
16 val_split = 0.2 # a fração do número de amostras de treinamento reservadas
    para
# validação => 48.000 amostras para treino + 12.000 amostras para
18 # validação = 60.000 exemplos do conjunto de treinamento MNIST

20 # carrega a base de dados MNIST
mnist = keras.datasets.mnist
22 (X_train, Y_train), (X_test, Y_test) = mnist.load_data()

24 reformat = 784 # é a dimensão do espaço de "features"
# ou seja, a camada de entrada possui 784 "input units"
26 # Uma entrada é um vetor coluna 784 x 1

28 X_train = X_train.reshape(60000, reformat)
X_test = X_test.reshape(10000, reformat)
30
X_train = X_train.astype('float32')
32 X_test = X_test.astype('float32')

34 # normalização em [0,1]
X_train /= 255
36 X_test /= 255
print(X_train.shape[0], 'amostras de treinamento')
38 print(X_test.shape[0], 'amostras de teste')

40 # Representação "one-hot" dos rótulos (labels) de treinamento e teste
Y_train = tf.keras.utils.to_categorical(Y_train, n_classes)
42 Y_test = tf.keras.utils.to_categorical(Y_test, n_classes)

44 # construção do modelo
# A camada de saída é composta por 10 neurônios com função de ativação "
```

```
softmax", que é uma
46 # generalização da função sigmóide ou logística.

48 model = tf.keras.models.Sequential()
model.add(keras.layers.Dense(n_oculta, input_shape=(reformat,), name='
    dense_layer1', activation='relu'))
50 model.add(keras.layers.Dense(n_oculta, input_shape=(reformat,), name='
    dense_layer2', activation='relu'))
model.add(keras.layers.Dense(n_classes, input_shape=(reformat,), name='
    dense_layer3', activation='softmax'))

52 # sumário do modelo
54 model.summary()

56 # compilando o modelo
model.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['
    accuracy'])

58 # treinamento do modelo no TensorFlow 2
60 treino = model.fit(X_train, Y_train, batch_size=lote_tam, epochs=epocas,
    verbose=verbose, validation_split=val_split)
# model.fit retorna o objeto "treino", que é do tipo "History".

62 treino_dic = treino.history
64 # "treino.history" retorna um contêiner de dicionário (uma generalização do
    conceito de lista): tipo "dict"
# Um dicionário contém pares de (chave, valor)
66 treino_dic.keys()
# chaves no tf 2.1.0: ['loss', 'accuracy', 'val_loss', 'val_accuracy']

68 import matplotlib.pyplot as plt

70 font = {'family': 'serif',
72 'color': 'darkred',
    'weight': 'normal',
74 'size': 16,
    }

76 acc = treino.history['accuracy']
78 val_acc = treino.history['val_accuracy']
loss = treino.history['loss']
80 val_loss = treino.history['val_loss']

82 epochs = range(len(acc))

84 plt.clf()
```

```
86 plt.plot(epochs, acc, 'b', label='Acurácia de treinamento')
plt.plot(epochs, val_acc, 'r', label='Acurácia de validação')
88 plt.title('Curvas de aprendizado', fontdict=font)
plt.xlabel('tempo (épocas)', fontdict=font)
90 plt.ylabel('Acurácia', fontdict=font)
plt.legend()
92
plt.figure()
94
plt.plot(epochs, loss, 'b', label='Erro de treinamento')
96 plt.plot(epochs, val_loss, 'r', label='Erro de validação')
plt.title('Curvas de aprendizado ', fontdict=font)
98 plt.xlabel('tempo (épocas)', fontdict=font)
plt.ylabel('Erro', fontdict=font)
100 plt.legend()

102 plt.show()

104 # A curva de aprendizado (erro) indica overfitting (suave)
# a partir de 20 épocas
106
# avaliação do modelo
108 test_loss, test_acc = model.evaluate(X_test, Y_test)
print('Acurácia do teste:', test_acc)
110 # Acurácia do teste: 96,59%

112 # previsões
predictions = model.predict(X_test)
```

```
# REDE MNIST6a
2 # otimizador = RMSProp
# código para TensorFlow 2.1.0 Spyder – Anaconda
4
import tensorflow as tf
6 from tensorflow import keras
print("Tensorflow version " + tf.__version__)
8
# Parâmetros da rede e de treinamento
10 epocas = 20 # define a duração do treinamento
lote_tam = 128 # número de amostras que alimentarão a rede em uma dada é
    poca
12 # de treinamento (em inglês, lote = batch)
verbose = 1
14 n_classes = 10 # número de saídas = número de dígitos
n_oculta = 128
16 val_split = 0.2 # a fração do número de amostras de treinamento reservadas
    para
# validação => 48.000 amostras para treino + 12.000 amostras para
18 # validação = 60.000 exemplos do conjunto de treinamento MNIST
dropout = 0.3 # técnica de regularização
20 # A idéia por trás dessa melhoria é que o descarte aleatório força a
# rede a aprender padrões redundantes que são úteis para uma melhor
    generalização
22
# carrega a base de dados MNIST
24 mnist = keras.datasets.mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
26
reformat = 784 # é a dimensão do espaço de "features"
28 # ou seja, a camada de entrada possui 784 "input units"
# Uma entrada é um vetor coluna 784 x 1
30
X_train = X_train.reshape(60000, reformat)
32 X_test = X_test.reshape(10000, reformat)

34 X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
36
# normalização em [0,1]
38 X_train /= 255
X_test /= 255
40 print(X_train.shape[0], 'amostras de treinamento')
print(X_test.shape[0], 'amostras de teste')
42
# Representação "one-hot" dos rótulos (labels) de treinamento e teste
44 Y_train = tf.keras.utils.to_categorical(Y_train, n_classes)
```

```
Y_test = tf.keras.utils.to_categorical(Y_test, n_classes)
46
# construção do modelo
48 # A camada de saída é composta por 10 neurônios com função de ativação "
    softmax", que é uma
# generalização da função sigmóide ou logística.
50
model = tf.keras.models.Sequential()
52 model.add(keras.layers.Dense(n_oculta, input_shape=(reformat,), name='
    dense_layer1', activation='relu'))
model.add(keras.layers.Dropout(dropout))
54 model.add(keras.layers.Dense(n_oculta, input_shape=(reformat,), name='
    dense_layer2', activation='relu'))
model.add(keras.layers.Dropout(dropout))
56 model.add(keras.layers.Dense(n_classes, input_shape=(reformat,), name='
    dense_layer3', activation='softmax'))

58 # sumário do modelo
model.summary()
60
# compilando o modelo
62 model.compile(optimizer='RMSProp', loss='categorical_crossentropy', metrics
    =['accuracy'])
# RMSProp tende a convergir mais rápido que o SGD, vide:
64 # https://towardsdatascience.com/understanding-rmsprop-faster-neural-
    network-learning-62e116fcf29a

66 # treinamento do modelo no TensorFlow 2
treino = model.fit(X_train, Y_train, batch_size=lote_tam, epochs=epocas,
    verbose=verbose, validation_split=val_split)
68 # model.fit retorna o objeto "treino", que é do tipo "History".

70 treino_dic = treino.history
# "treino.history" retorna um contêiner de dicionário (uma generalização do
    conceito de lista): tipo "dict"
72 # Um dicionário contém pares de (chave, valor)
treino_dic.keys()
74 # chaves no tf 2.1.0: ['loss', 'accuracy', 'val_loss', 'val_accuracy']

76 import matplotlib.pyplot as plt

78 font = {'family': 'serif',
    'color': 'darkred',
80 'weight': 'normal',
    'size': 16,
82 }
```

```
84 acc = treino.history['accuracy']
    val_acc = treino.history['val_accuracy']
86 loss = treino.history['loss']
    val_loss = treino.history['val_loss']
88
    epochs = range(len(acc))
90
    plt.clf()
92
    plt.plot(epochs, acc, 'b', label='Acurácia de treinamento')
94 plt.plot(epochs, val_acc, 'r', label='Acurácia de validação')
    plt.title('Curvas de aprendizado', fontdict=font)
96 plt.xlabel('tempo (épocas)', fontdict=font)
    plt.ylabel('Acurácia', fontdict=font)
98 plt.legend()

100 plt.figure()

102 plt.plot(epochs, loss, 'b', label='Erro de treinamento')
    plt.plot(epochs, val_loss, 'r', label='Erro de validação')
104 plt.title('Curvas de aprendizado ', fontdict=font)
    plt.xlabel('tempo (épocas)', fontdict=font)
106 plt.ylabel('Erro', fontdict=font)
    plt.legend()
108
    plt.show()
110
    # A curva de aprendizado (erro) indica overfitting a partir de
112 # 9 épocas

114 # avaliação do modelo
    test_loss, test_acc = model.evaluate(X_test, Y_test)
116 print('Acurácia do teste:', test_acc)
    # Acurácia do teste: 97,79%
118
    # previsões
120 predictions = model.predict(X_test)
```



```
# REDE MNIST6b
2 # otimizador = Adam
  # regularização L2
4 # código para TensorFlow 2.1.0 Spyder – Anaconda

6 import tensorflow as tf
  from tensorflow import keras
8 from keras import regularizers

10 # Parâmetros da rede e de treinamento
  epocas = 20      # define a duração do treinamento
12 lote_tam = 64    # número de amostras que alimentarão a rede em uma dada é
                    # pocas
                    # de treinamento (em inglês, lote = batch)
14 verbose = 1
  n_classes = 10  # número de saídas = número de dígitos
16 n_oculta = 128
  val_split = 0.2 # a fração do número de amostras de treinamento reservadas
                    # para
18 # validação => 48.000 amostras para treino + 12.000 amostras para
  # validação = 60.000 exemplos do conjunto de treinamento MNIST
20 dropout = 0.3    # técnica de regularização
  # A idéia por trás dessa melhoria é que o descarte aleatório força a
22 # rede a aprender padrões redundantes que são úteis para uma melhor
  # generalização

24 # carrega a base de dados MNIST
  mnist = keras.datasets.mnist
26 (X_train, Y_train), (X_test, Y_test) = mnist.load_data()

28 reformat = 784 # é a dimensão do espaço de "features"
  # ou seja, a camada de entrada possui 784 "input units"
30 # Uma entrada é um vetor coluna 784 x 1

32 X_train = X_train.reshape(60000, reformat)
  X_test = X_test.reshape(10000, reformat)
34
  X_train = X_train.astype('float32')
36 X_test = X_test.astype('float32')

38 # normalização em [0,1]
  X_train /= 255
40 X_test /= 255
  print(X_train.shape[0], 'amostras de treinamento')
42 print(X_test.shape[0], 'amostras de teste')

44 # Representação "one-hot" dos rótulos (labels) de treinamento e teste
```

```
Y_train = tf.keras.utils.to_categorical(Y_train, n_classes)
56 Y_test = tf.keras.utils.to_categorical(Y_test, n_classes)

48 # construção do modelo
# A camada de saída é composta por 10 neurônios com função de ativação "
    softmax", que é uma
50 # generalização da função sigmóide ou logística.

52 # A complexidade de um modelo pode ser convenientemente representada como o
    número de pesos
# diferentes de zero. Em outras palavras, se tivermos dois modelos M1 e M2
    obtendo praticamente
54 # o mesmo desempenho em termos de função de perda, devemos escolher o
    modelo mais simples, ou seja,
# aquele que tenha o número mínimo de pesos diferentes de zero.
56 # Regularização L2 (também conhecida como Ridge): a complexidade do modelo
    é expressa como a
# soma dos quadrados dos pesos
58
model = tf.keras.models.Sequential()
60 model.add(keras.layers.Dense(n_oculta, input_shape=(reformat, ),
    kernel_regularizer=regularizers.l2(0.001),
    name='dense_layer1', activation='relu'))
62 model.add(keras.layers.Dropout(dropout))
model.add(keras.layers.Dense(n_oculta, input_shape=(reformat, ),
    kernel_regularizer=regularizers.l2(0.001),
64 name='dense_layer2', activation='relu'))
model.add(keras.layers.Dropout(dropout))
66 model.add(keras.layers.Dense(n_classes, input_shape=(reformat, ), name='
    dense_layer3', activation='softmax'))

68 # sumário do modelo
model.summary()
70
# compilando o modelo
72 model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['
    accuracy'])
# RMSProp tende a convergir mais rápido que o SGD, vide:
74 # https://towardsdatascience.com/understanding-rmsprop-faster-neural-
    network-learning-62e116fcf29a

76 # treinamento do modelo no TensorFlow 2
treino = model.fit(X_train, Y_train, batch_size=lote_tam, epochs=epocas,
    verbose=verbose, validation_split=val_split)
78 # model.fit retorna o objeto "treino", que é do tipo "History".

80 treino_dic = treino.history
```

```
# "treino.history" retorna um contêiner de dicionário (uma generalização do
# conceito de lista): tipo "dict"
82 # Um dicionário contém pares de (chave, valor)
treino_dic.keys()
84 # chaves no tf 2.1.0: ['loss', 'accuracy', 'val_loss', 'val_accuracy']

86 import matplotlib.pyplot as plt

88 font = {'family': 'serif',
'color': 'darkred',
90 'weight': 'normal',
'size': 16,
92 }

94 acc = treino.history['accuracy']
val_acc = treino.history['val_accuracy']
96 loss = treino.history['loss']
val_loss = treino.history['val_loss']
98
epochs = range(len(acc))

100
plt.clf()
102
plt.plot(epochs, acc, 'b', label='Acurácia de treinamento')
104 plt.plot(epochs, val_acc, 'r', label='Acurácia de validação')
plt.title('Curvas de aprendizado', fontdict=font)
106 plt.xlabel('tempo (épocas)', fontdict=font)
plt.ylabel('Acurácia', fontdict=font)
108 plt.legend()

110 plt.figure()

112 plt.plot(epochs, loss, 'b', label='Erro de treinamento')
plt.plot(epochs, val_loss, 'r', label='Erro de validação')
114 plt.title('Curvas de aprendizado ', fontdict=font)
plt.xlabel('tempo (épocas)', fontdict=font)
116 plt.ylabel('Erro', fontdict=font)
plt.legend()
118
# A curva de aprendizado indica que não há overfitting,
120 # pois o erro de validação é sempre menor do que o de
# treinamento!
122
# avaliação do modelo
124 test_loss, test_acc = model.evaluate(X_test, Y_test)
print('Acurácia do teste:', test_acc)
126 # Acurácia do teste: 97,26%
```

```
128 # previsões  
    predictions = model.predict(X_test)
```

```
# REDE MNIST8
2 # otimizador = Adam
  # regularização com BATCH NORMALIZATION
4 # código para TensorFlow 2.1.0 Spyder – Anaconda

6 import tensorflow as tf
  from tensorflow import keras
8 from keras import regularizers
  print("Tensorflow version " + tf.__version__)
10

  # Parâmetros da rede e de treinamento
12 epocas = 20      # define a duração do treinamento
  lote_tam = 64    # número de amostras que alimentarão a rede em uma dada é
                   # poca
14 # de treinamento (em inglês, lote = batch)
  verbose = 1
16 n_classes = 10 # número de saídas = número de dígitos
  n_oculta = 128
18 val_split = 0.2 # a fração do número de amostras de treinamento reservadas
                   # para
                   # validação => 48.000 amostras para treino + 12.000 amostras para
20 # validação = 60.000 exemplos do conjunto de treinamento MNIST
  dropout = 0.3    # técnica de regularização
22 # A idéia por trás dessa melhoria é que o descarte aleatório força a
  # rede a aprender padrões redundantes que são úteis para uma melhor
  # generalização
24

  # carrega a base de dados MNIST
26 from keras.datasets import mnist
  (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
28

  reformat = 784 # é a dimensão do espaço de "features"
30 # ou seja, a camada de entrada possui 784 "input units"
  # Uma entrada é um vetor coluna 784 x 1
32

  X_train = X_train.reshape(60000, reformat)
34 X_test = X_test.reshape(10000, reformat)

36 X_train = X_train.astype('float32')
  X_test = X_test.astype('float32')
38

  # normalização em [0,1]
40 X_train /= 255
  X_test /= 255
42 print(X_train.shape[0], 'amostras de treinamento')
  print(X_test.shape[0], 'amostras de teste')
44
```

```

# Representação "one-hot" dos rótulos (labels) de treinamento e teste
46 Y_train = tf.keras.utils.to_categorical(Y_train, n_classes)
   Y_test = tf.keras.utils.to_categorical(Y_test, n_classes)
48
# construção do modelo
50 # A camada de saída é composta por 10 neurônios com função de ativação "
   softmax", que é uma
   # generalização da função sigmóide ou logística.
52
# BATCH NORMALIZATION (BN)
54 # Treinar redes neurais profundas é difícil. E fazê-las convergir em uma
   quantidade razoável de tempo pode ser complicado.
   # A normalização de lotes (BN) é uma técnica popular e eficaz que acelera
   consistentemente a convergência de redes profundas.
56 # Juntamente com os blocos residuais (não foi implementado neste código), o
   BN tornou possível para os profissionais
   # treinar rotineiramente redes com mais de 100 camadas.
58
model = tf.keras.models.Sequential()
60 model.add(keras.layers.Dense(n_oculta, input_shape=(reformat, ),
   kernel_regularizer=regularizers.l2(0.001),
   name='dense_layer1', activation='relu'))
62 model.add(keras.layers.Dropout(dropout))
   model.add(keras.layers.BatchNormalization())
64 model.add(keras.layers.Dense(n_oculta, input_shape=(reformat, ),
   kernel_regularizer=regularizers.l2(0.001),
   name='dense_layer2', activation='relu'))
66 model.add(keras.layers.Dropout(dropout))
   model.add(keras.layers.BatchNormalization())
68 model.add(keras.layers.Dense(n_classes, input_shape=(reformat, ), name='
   dense_layer3', activation='softmax'))

70 # sumário do modelo
   model.summary()
72
# compilando o modelo
74 model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=[
   'accuracy'])
   # RMSProp tende a convergir mais rápido que o SGD, vide:
76 # https://towardsdatascience.com/understanding-rmsprop-faster-neural-
   network-learning-62e116fcf29a

78 # treinamento do modelo no TensorFlow 2
   treino = model.fit(X_train, Y_train, batch_size=lote_tam, epochs=epocas,
   verbose=verbose, validation_split=val_split)
80 # model.fit retorna o objeto "treino", que é do tipo "History".

```

```
82 treino_dic = treino.history
# "treino.history" retorna um contêiner de dicionário (uma generalização do
# conceito de lista): tipo "dict"
84 # Um dicionário contém pares de (chave, valor)
treino_dic.keys()
86 # chaves no tf 2.1.0: ['loss', 'accuracy', 'val_loss', 'val_accuracy']

88 import matplotlib.pyplot as plt

90 font = {'family': 'serif',
# 'color': 'darkred',
92 'weight': 'normal',
# 'size': 16,
94 }

96 acc = treino.history['accuracy']
val_acc = treino.history['val_accuracy']
98 loss = treino.history['loss']
val_loss = treino.history['val_loss']
100
epochs = range(len(acc))
102
plt.clf()
104
plt.plot(epochs, acc, 'b', label='Acurácia de treinamento')
106 plt.plot(epochs, val_acc, 'r', label='Acurácia de validação')
plt.title('Curvas de aprendizado', fontdict=font)
108 plt.xlabel('tempo (épocas)', fontdict=font)
plt.ylabel('Acurácia', fontdict=font)
110 plt.legend()

112 plt.figure()

114 plt.plot(epochs, loss, 'b', label='Erro de treinamento')
plt.plot(epochs, val_loss, 'r', label='Erro de validação')
116 plt.title('Curvas de aprendizado ', fontdict=font)
plt.xlabel('tempo (épocas)', fontdict=font)
118 plt.ylabel('Erro', fontdict=font)
plt.legend()
120
plt.show()
122
# A curva de aprendizado indica que não há overfitting ,
124 # pois o erro de validação é sempre menor do que o de
# treinamento!
126
# avaliação do modelo
```

```
128 test_loss, test_acc = model.evaluate(X_test, Y_test)
    print('Acurácia do teste:', test_acc)
130 # Acurácia do teste: 97,10%

132 # previsões
    predictions = model.predict(X_test)
```



```
# REDE MNISTCNN
2 # código para TensorFlow 2.2.0 Spyder – Anaconda
  # autor: Alexandre B. de Lima
4 # LeNet CNN – base de dados MNIST

6 import tensorflow as tf
  from tensorflow.keras import datasets, layers, models, optimizers
8 print(tf.__version__)

10 # parâmetros da rede e do treinamento
  EPOCHS = 10
12 BATCH_SIZE = 128
  VERBOSE = 1
14 OPTIMIZER = tf.keras.optimizers.Adam()
  VALIDATION_SPLIT = 0.95
16 IMG_ROWS, IMG_COLS = 28, 28
  INPUT_SHAPE = (IMG_ROWS, IMG_COLS, 1)
18 NB_CLASSES = 10

20 # função que define a rede LeNet
  def build(input_shape, classes):
22     model = models.Sequential()
      # CONV => RELU => POOL
24     model.add(layers.Conv2D(20, (5,5), activation='relu', input_shape=
        input_shape))
      model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
26     # CONV => RELU => POOL
      model.add(layers.Conv2D(50, (5,5), activation='relu'))
28     model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
      # camada de vetorização dos canais
30     model.add(layers.Flatten())
      # camada densa com 500 neurônios
32     model.add(layers.Dense(500, activation='relu'))
      # camada de saída densa com classificador softmax
34     model.add(layers.Dense(classes, activation='softmax'))
      return model
36

  # Carrega dados e rótulos
38 (X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()

40 # reformata dados para o formato array do numpy
  X_train = X_train.reshape((60000, 28, 28, 1))
42 X_test = X_test.reshape((10000, 28, 28, 1))
  # "reshape" é um método de "numpy.ndarray" que retorna um "array" que contém
    os mesmos
44 # dados, só que em um outro formato
```

```
46 # normaliza dados
X_train, X_test = X_train/255.0, X_test/255.0
48
# representação numérica dos dados em ponto flutuante - 32 bits
50 X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
52
# one-hot encode (vetorização) dos rótulos
54 y_train = tf.keras.utils.to_categorical(y_train, NB_CLASSES)
y_test = tf.keras.utils.to_categorical(y_test, NB_CLASSES)
56
# construção da rede
58 model = build(input_shape=INPUT_SHAPE, classes = NB_CLASSES)

60 # compilação da rede
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics
              =['accuracy'])
62
# sumário da rede
64 model.summary()

66 # treinamento
treino = model.fit(X_train, y_train,
68 batch_size=BATCH_SIZE, epochs=EPOCHS,
verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
70
# model.fit retorna o objeto "treino", que é do tipo "History".
72
treino_dic = treino.history
74 # "treino.history" retorna um contêiner de dicionário (uma generalização do
conceito de lista): tipo "dict"
# Um dicionário contém pares de (chave, valor)
76 treino_dic.keys()
# chaves no tf 2.1.0: ['loss', 'accuracy', 'val_loss', 'val_accuracy']
78
import matplotlib.pyplot as plt
80
font = {'family': 'serif',
82 'color': 'darkred',
'weight': 'normal',
84 'size': 16,
}

86
acc = treino.history['accuracy']
88 val_acc = treino.history['val_accuracy']
loss = treino.history['loss']
90 val_loss = treino.history['val_loss']
```

```
92 epochs = range(len(acc))
94 plt.clf()
96 plt.plot(epochs, acc, 'b', label='Acurácia de treinamento')
    plt.plot(epochs, val_acc, 'r', label='Acurácia de validação')
98 plt.title('Curvas de aprendizado', fontdict=font)
    plt.xlabel('tempo (épocas)', fontdict=font)
100 plt.ylabel('Acurácia', fontdict=font)
    plt.legend()
102
    plt.figure()
104
    plt.plot(epochs, loss, 'b', label='Erro de treinamento')
106 plt.plot(epochs, val_loss, 'r', label='Erro de validação')
    plt.title('Curvas de aprendizado ', fontdict=font)
108 plt.xlabel('tempo (épocas)', fontdict=font)
    plt.ylabel('Erro', fontdict=font)
110 plt.legend()
112 plt.show()
114 # A curva de aprendizado indica que há overfitting ,
    # após 2 épocas de treinamento
116
    # resultados: custo (loss) e acurácia - TESTE
118 score = model.evaluate(X_test, y_test, verbose=VERBOSE)
    print("\nTest score: ", score[0])
120 print("Test accuracy: ", score[1])
    Acurácia do teste: 96,98%
```

A.2 Implementação do FaceNet em Python Usando TensorFlow 2 e Keras

```

# Reconhecimento facial com a arquitetura FaceNet
2 # adaptado por Tatiana Holanda Silva
# código original de Antonio José G. Busson et al
4 # artigo: "Desenvolvendo Modelos de Deep Learning para Aplicações Multimí-
    dia no Tensorflow"
# XXIV Simpósio Brasileiro de Sistemas Multimídia e Web (2018): Minicursos
6
# Código para TensorFlow 2, Python 3.7, IDE Spyder 4.0 – Anaconda
8
# A seguinte alteração foi realizada na linha 408 do módulo facenet.py
10 # graph_def = tf.GraphDef() ==> linha foi comentada com # e substituída
    pela linha abaixo:
# graph_def = tf.compat.v1.GraphDef() # para viabilizar a migração do có-
    digo TF 1.x para TF 2.x
12
import tensorflow as tf
14 print(tf.__version__)
import matplotlib.pyplot as plt
16 import numpy as np

18 # arquivo local com a arquitetura facenet
# linha de código modificada
20 import facenet.src.facenet

22 import cv2 # opencv biblioteca para processamento de imagem
# Linux Ubuntu 18.04: instalar opencv pelo terminal:
24 # $~ pip install opencv-python==3.4.2.17
# $~ pip install opencv-contrib-python==3.4.2.17
26 # em que $~ representa o prompt ...

28 # zerar variáveis
from IPython import get_ipython
30 get_ipython().magic('reset -sf')

32 # modelo pré-treinado do FaceNet

34 # inicia a sessão no Tensorflow
sess = tf.compat.v1.Session() #
36
# Carregando do modelo pré-treinado
38 # linha de código modificada
facenet.src.facenet.load_model('datasets/facematch/20170512-110547.pb')
40

```

```
# seleção dos tensores necessários para obter os embeddings das imagens
    faciais.
42
#Selecionando os tensores necessarios para execucao
44 image_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("
    input:0")
    embeddings = tf.compat.v1.get_default_graph().get_tensor_by_name("
        embeddings:0")
46 train_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("
    phase_train:0")

48 #imprimindo as informações dos tensores
    print(image_placeholder)
50 print(embeddings)
    print(train_placeholder)
52

# A função abaixo utiliza os tensores do FaceNet carregados para obter os
    embeddings
54 # de uma imagem facial. Primeiro, o arquivo de imagem é aberto e
    redimensionado para o
    # padrão aceito pelo FaceNet 160x160. Em seguida, a imagem é colocada como
    entrada e o
56 # placeholder de treinamento é setado como False. O embedding retornado da
    execução do
    # grafo de computação é retornado, bem como a imagem que foi usada como
    entrada.

58
def get_embedding(img_path):
60 img_size = 160
    img = cv2.imread(img_path)
62 #o opencv abre a imagem em BGR, necessario converter para RGB
    if img is None:
64 print("Imagem não pode ser aberta.")
        return None
66 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #Preparando a imagem de entrada
68 resized = cv2.resize(img, (img_size, img_size), interpolation=cv2.INTER_CUBIC
        )
    reshaped = resized.reshape(-1, img_size, img_size, 3)
70 #Configurando entrada e execucao do FaceNet
    feed_dict = {image_placeholder: reshaped, train_placeholder: False}
72 embedding = sess.run(embeddings, feed_dict=feed_dict)
    return embedding[0], img
74

# Registrando pessoas
76 # A tarefa de reconhecimento facial tenta responder a pergunta "Quem é essa
    pessoa?"
```

```
# Para isso, é necessário registrar os embeddings de imagens faciais, para
# que dessa forma,
78 # seja possível realizar a comparação de similaridade entre as imagens
# faciais. Neste exemplo,
# são usadas imagens de algumas figuras públicas da política brasileira.
80
database = {}
82
database["jennifer"], img = get_embedding("faces/jennifer_0.png")
84 print("Jennifer Aniston - foto cadastrada:")
__ = plt.imshow(img)
86 plt.pause(0.1)

88 database["jolie"], img = get_embedding("faces/jolie_0.png")
print("Angelina Jolie - foto cadastrada:")
90 __ = plt.imshow(img)
plt.pause(0.1)
92
database["ozzy"], img = get_embedding("faces/ozzy_0.png")
94 print("Ozzy Osbourne - foto cadastrada:")
__ = plt.imshow(img)
96 plt.pause(0.1)

98 database["brad"], img = get_embedding("faces/brad_0.png")
print("Brad Pitt - foto cadastrada:")
100 __ = plt.imshow(img)
plt.pause(0.1)
102
#Este é um exemplo de embedding
104 print("Facial Embedding da Jennifer:\n", database["jennifer"])

106 # Reconhecimento facial

108 # Nesta etapa é realizado o processo de reconhecimento facial. Para isso,
# como dito
# anteriormente, é calculada a similaridade entre os embeddings das imagens
# faciais.
110 # Uma forma simples para calcular essa similaridade é usando a equação da
# distancia euclidiana,
# como mostra a função abaixo.
112
# Função que calcula a distancia euclidiana entre dois vetores
114 def distance(vector1, vector2):
return np.sqrt(np.sum((vector1 - vector2)**2))
116

# A função "who_is_it" definida abaixo, identifica uma imagem facial. A fun
ção recebe
```

```
118 # como parâmetro o caminho de um arquivo de imagem e o dicionário de
    # pessoas registradas.
    # Resumidamente, essa função calcula a distancia euclidiana entre os
    # embeddings da imagem de
120 # entrada e das pessoas registradas, o menor distancia é atribuida a
    # identidade da pessoa.

122 def who_is_it(visitor_image_path, database):
    min_dist = 1000
124 identity = -1
    #Calculando o embedding do visitante
126 visitor, img = get_embedding(visitor_image_path)
    #Calculando a distancia do visitante com os demais funcionarios
128 for name, employee in database.items():
    dist = distance(visitor, employee)
130
    if dist < min_dist:
132 min_dist = dist
    identity = name
134 #verificando a identidade
    if min_dist > 0.5:
136 print("Essa pessoa nao esta cadastrada!")
    return None, img
138 else:
    return identity, img
140
    # Realizando testes
142
    # Na última etapa, a função "who_is_it" é utilizada para identificar uma
    # imagem facial.
144 # No código abaixo são apresentados quatro exemplos de identificação.

146 identity, img = who_is_it("faces/jennifer_1.png", database)
    print("Essa pessoa é o(a)", identity, "!")
148 # if identity == "jennifer":
    # print("Hi, I am Jennifer!")
150 _ = plt.imshow(img)
    plt.pause(0.1)
152
    identity, img = who_is_it("faces/jolie_1.png", database)
154 print("Essa pessoa é o(a)", identity, "!")
    _ = plt.imshow(img)
156 plt.pause(0.1)

158 identity, img = who_is_it("faces/ozzy_1.png", database)
    print("Essa pessoa é o(a)", identity, "!")
160 _ = plt.imshow(img)
```

```
plt.pause(0.1)
162 identity, img = who_is_it("faces/brad_1.png", database)
164 print("Essa pessoa é o(a)", identity, "!")
_ = plt.imshow(img)
166 plt.pause(0.1)
```