INSTITUTO FEDERAL DE CIÊNCIA E TECNOLOGIA DE PERNAMBUCO

Campus Recife

Departamento Acadêmico de Sistema, Processos e Controles Eletro-Eletrônicos

Coordenação de Sistemas de Informação

Trabalho de Conclusão do Curso de Análise e Desenvolvimento de Sistemas

ARNALDO CARNEIRO DA SILVA NETO

DIOGO MOURA DIAS

CARACRACHÁ: Investigando eficiência e eficácia de uma rede neural portada para ponto fixo em ambiente embarcado

RECIFE

2018

ARNALDO CARNEIRO DA SILVA NETO
DIOGO MOURA DIAS

CARACRACHÁ: Investigando eficiência e eficácia de uma rede neural portada para ponto fixo em ambiente embarcado

Trabalho de conclusão de curso apresentado Curso de Análise e Desenvolvimento de Sistemas do Instituto Federal de Pernambuco para obtenção do grau de tecnólogo em Análise e Desenvolvimento de Sistemas.
Orientador: M.Sc. Paulo Abadie Guedes.

RECIFE
2018

Ficha elaborada pela bibliotecária Emmely Cristiny Lopes Silva CRB4/1876

S586c
2019 Silva Neto, Arnaldo Carneiro da.
Caracrachá: investigando eficiência e eficácia de uma rede neural portada para ponto fixo em ambiente embarcado / Arnaldo Carneiro da Silva Neto; Diogo Moura Dias. --- Recife: O autor, 2018.
76f. il. Color.

TCC (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Pernambuco, Departamento Acadêmico de Controle de Sistemas Eletrônicos - DASE, 2018.

Inclui Referências e apêndices

Orientador: Professor M.e. Paulo Abadie Guedes.

1. Redes Neurais Feedforward. 2. Reconhecimento facial. 3. Sistemas embarcados. I. Guedes, Paulo Abadie (orientador). II. Instituto Federal de Pernambuco. III. Título.

CDD 004 (21ed.)

Trabalho de Conclusão de Curso apresentado pelos alunos **Arnaldo Carneiro da Silva Neto e Diogo Moura Dias** à coordenação de Análise e Desenvolvimento de Sistemas, do Instituto Federal de Pernambuco, sob o título de "***CARACRACHÁ: INVESTIGANDO EFICIÊNCIA E EFICÁCIA DE UMA REDE NEURAL NO RECONHECIMENTO DE FACES EM AMBIENTE EMBARCADO.***", orientado pelo **Prof. Ms. Paulo Abadie Guedes** e aprovado pela banca examinadora formada pelos professores:

Recife, 26 de novembro de 2018.

Prof. Ms. Paulo Abadie Guedes
CTADS/DASE/IFPE

Prof. Dr. Aida Araújo Ferreira
CTADS/DASE/IFPE

Prof. Ms. José Paulo da Silva Lima
Faculdade Nova Roma

Arnaldo Carneiro da Silva Neto

Diogo Moura Dias

AGRADECIMENTOS

Agradecemos a todos os professores, amigos e familiares que nos apoiaram, acreditaram e de alguma forma colaboraram conosco de forma que nos permitiu chegar até aqui; seja com paciência, com uma palavra de apoio ou meramente levantando nossos ânimos quando estávamos pensando em desistir. Nosso mais sincero muito obrigado.

“Inteligência é a habilidade de evitar fazer o trabalho,
e mesmo assim conseguir ter o trabalho realizado.”
(Linus Torvalds)

RESUMO

O presente trabalho visa avaliar a viabilidade da criação e treinamento de uma rede neural para reconhecimento facial em ponto flutuante num ambiente PC e posteriormente convertê-la em uma rede neural de ponto fixo e portá-la para o ambiente de um sistema embarcado com a mesma função. Foi utilizada neste trabalho uma rede neural estruturada em uma arquitetura de *Deep Feedforward* composta por três camadas escondidas e uma camada de entrada sendo alimentada por imagens de faces em tons de cinza na resolução de 48x48px, onde cada pixel se comunica com cada neurônio da primeira camada. Foi também abordada a questão do comportamento, desempenho, consistência e eficiência dessa rede neural após a sua conversão de ponto flutuante para ponto fixo. Além disso, foi verificado o grau com que essas características da rede se mantêm ao portá-la para o sistema embarcado, e ainda quais sistemas embarcados conseguiram utilizá-la. O treinamento foi realizado em um PC comum e os testes de desempenho e viabilidade se deram em uma *Raspberry Pi*. Foram observados ganhos de 50% no tempo de processamento de uma rede neural usando-se a técnica proposta sem perda considerável de confiabilidade da mesma.

Palavras-chaves: Redes Neurais Feedforward. Reconhecimento Facial. Sistemas Embarcados. Ponto Fixo. Ponto Flutuante. Internet das Coisas.

ABSTRACT

This work aims to present the viability of creating and training a neural network for face recognition in floating point on a PC environment and later to convert it into a fixed point neural network and port it to an embedded system environment with the same behavior. It was used a neural network structured in a Deep Feedforward architecture composed of three hidden layers and an input layer being fed by grayscale images of faces at a 48x48px resolution, where each pixel communicates with each neuron of the first layer. The question of behavior, performance, consistency and efficiency of this neural network after its conversion from floating point to fixed point was also addressed. In addition, it was also verified the degree to which these network characteristics remain after porting the network to an embedded system and which embedded systems have been able to use it. The training was performed on a regular PC and the performance and viability tests were done on a Raspberry Pi. Gains of 50% were observed in the processing time of neural networks when using the proposed technique without considerable loss of reliability.

Key-words: Feedforward Neural Networks. Facial Recognition. Embedded Systems. Fixed Point. Floating Point. Internet of Things.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANNs	*Artificial Neural Networks* (rede neural artificial)
ARM	*Advanced RISC Machine* Família de arquiteturas de processadores com conjunto reduzido de instruções.
ASCII	*American Standard Code for Information Interchange* (Código Padrão Americano para o Intercâmbio de Informação)
BASH	*Bourne-Again SHell* - interpretador de linha de comandos *Unix*
BD	Banco de Dados
Codec	Codificador/Decodificador
CPU	*Central Processing Unit* (Unidade Central de Processamento)
FANN	*Fast Artificial Neural Network* (Biblioteca de Rápida de Redes Neurais Artificiais)
FN	Falso Negativo - a rede acha que está sendo apresentada a alguém a quem ela nunca foi exposta, mas errou
FP	Falso Positivo - a rede acha que identificou alguém, mas errou
IDE	*Integrated Development Enviroment* (Ambiente de Desenvolvimento Integrado)
IEEE	*Institute of Electrical and Electronics Engineers* (Instituto de Engenheiros Eletricistas e Eletrônicos) - organização encarregada de regulamentar as áreas associadas à Eletricidade e Eletrônica
IoT	*Internet of Things* (Internet das Coisas)
MS	*Microsoft*
PC	*Personal Computer* (computador pessoal)
pf	Ponto Flutuante
pfo	Ponto Fixo
PIC	*Peripheral Interface Controller* (Controlador de Intarface Periférica) ou *Programmable Intelligent Computer* (Computador Inteligente Programável)

Pixel	*Pixel - Picture Element* (Elemento de Imagem)
PNG	*Portable Network Graphics* (Gráfico Portátil para Rede) - formato de imagem
px	O mesmo que pixel
RAM	*Random Access Memory* (memória de acesso aleatório)
RN	Rede Neural
RNA	Rede Neural Artificial
USB	*Universal Serial Bus* (barramento universal serial)
VN	Verdadeiro Negativo - a rede acha que está sendo apresentada a alguém a quem ela nunca foi exposta e acertou
VP	Verdadeiro Positivo - a rede acha que identificou alguém e acertou
XOR	*XOR* (Exclusive OR) (Função Lógica "Ou Exclusivo")

LISTA DE SÍMBOLOS

Σ	Letra grega sigma maiúsculo (Somatório)
σ	Letra grega sigma minúsculo
Φ	Letra grega phi
Θ	Letra grega theta
f	Função f
e	Base do logaritmo neperiano

SUMÁRIO

1 INTRODUÇÃO

Sabe-se da importância da segurança em determinados locais, em muitos casos se tratando de restrição de acesso a pessoas autorizadas. Para isso existem várias técnicas e métodos para garantir a autenticidade, seja reconhecimento de digitais, de retina, ou reconhecimento facial. Apesar de funcionar bem no PC, este último método é relativamente pouco usado, devido ao alto custo, seja em relação ao custo financeiro ou ao desempenho do mesmo, já que são necessários um computador (processador) e uma câmera. Se considerarmos uma área com vários ambientes, ter um computador completo por ambiente apenas para controle de acesso pode tornar o sistema proibitivamente caro.

Um sistema embarcado, por outro lado, é bem mais barato que um PC completo. Desta forma, torna-se factível a utilização de um desses sistemas em cada um dos ambientes da área citada anteriormente. Contudo, sistemas embarcados são mais baratos devido ao fato do *hardware* dos mesmos ser limitado, tendo muitas vezes bem menos memória e capacidade de processamento. Ficam então as perguntas: é possível usar, em um sistema embarcado, as mesmas técnicas usadas em um PC para reconhecimento facial? O processo se tornaria muito lento devido às limitações de *hardware*? É possível acelerar o processamento de alguma forma mantendo a confiabilidade do sistema sem introduzir novos custos?

A proposta deste trabalho é mostrar que usando ponto fixo é possível reduzir o tempo de processamento de uma rede neural usada no reconhecimento facial a cerca de metade do tempo de processamento de uma rede neural similar que usa ponto flutuante. Mais ainda: a mudança de ponto flutuante para ponto fixo não acarreta problemas na confiabilidade da rede neural. Uma rede neural nesses moldes pode contrabalançar as desvantagens do uso de sistemas embarcados.

2 OBJETIVOS

2.1 OBJETIVO GERAL

O presente trabalho tem como objetivo geral verificar a viabilidade, a eficiência e a eficácia da utilização de uma rede neural treinada em um PC e portada para um ambiente de sistema embarcado, neste caso o Raspberry Pi, em ponto fixo, no reconhecimento facial.

2.2 OBJETIVOS ESPECÍFICOS

1. Criar um sistema híbrido rápido e barato, onde uma rede em ponto fixo é usada para acelerar a computação no ambiente embarcado.
2. Inserir uma rede neural leve (i.e. uma rede pequena que consome pouca memória - da ordem de uns poucos Kb - e utiliza pouco tempo de CPU) num sistema embarcado.
3. Verificar eficiência de uma rede neural para reconhecimento facial em um PC utilizando imagens em preto e branco em baixa resolução.
4. Verificar viabilidade de portabilidade da rede neural anterior para uma rede neural com mesmo comportamento e eficiência em um sistema de ponto fixo.
5. Averiguar a capacidade mínima de memória necessária em um sistema embarcado para rodar essa rede neural.
6. Baseado na resposta à pergunta anterior verificar quais sistemas embarcados poderiam trabalhar com essa rede neural para reconhecimento facial.
7. Utilizando da resposta à questão anterior, verificar se a portabilidade dessa rede em um sistema embarcado que a comporte, mantém as características que a mesma tinha ao ser executada no PC.
8. Verificar se o comportamento da RN, no sistema embarcado que a comporte, mantém a consistência das saídas em função da conversão de pesos (pf para pfo) conforme as saídas do PC.

O próximo capítulo descreve os trabalhos relacionados. O capítulo 4 aborda a metodologia empregada, apresentando a arquitetura do sistema e o experimento realizado. O capítulo 5 apresenta e discute os resultados obtidos. Finalmente, o último capítulo aborda as conclusões e trabalhos futuros.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 DEFINIÇÕES DE UMA RN

Para que se entenda o presente trabalho é necessário entender os conceitos básicos de uma RN.

3.1.1 O que é uma RN?

Uma Rede Neural Artificial (RNA) ou apenas Rede Neural (RN) é uma estrutura computacional baseada nas redes neurais biológicas que formam os cérebros animais, cujo o objetivo é a resolução de problemas variados como por exemplo comportamento de personagens não jogáveis em jogos eletrônicos, reconhecimento de discursos, faces, etc.

Assim, uma RNA é constituída por uma estrutura básica chamada de neurônio artificial que, de forma similiar ao neurônio biológico, são interligados e transmitem sinais de um a outro, da forma que o neuronio processa um sinal assim que o recebe para depois enviar o resultado desse processamento para outros neurônios conectados a ele seguindo esse procedimento do inicio ao fim da RN.

Comumente as RNAs são implementadas usando-se números reais para a realização das conexões e tem em sua saída uma função não linear responsável pelo calculo do somatório das entradas, sendo essas funções chamadas de funções de ativação. A força do sinal de uma conexão é definida por pesos existentes tanto nos neurônios como nas conexões. São esses pesos que também ajustam os procedimentos de aprendizado. Além disso os neurônios artificiais podem ter um limite de modo que somente se o sinal agregado atravessar esse limite o sinal é enviado (*threshold*).

Uma RN aprende por meio de exemplos de treinamento passados através dela usando um algoritmo de retropropagação onde o erro é utilizado para mudar o peso dos neurônios e assim mitigar gradualmente esse erro ao comparar as saídas obtidas com as saídas esperadas.

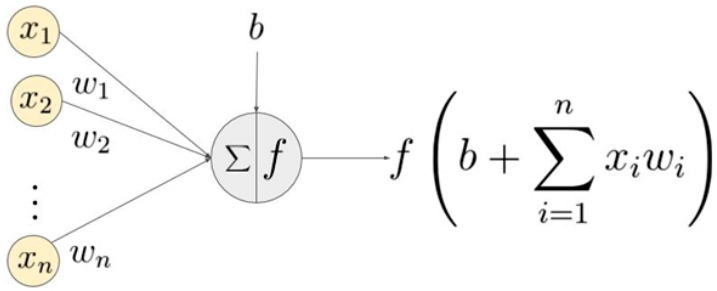
3.1.2 O que é um neurônio artificial?

Um Neurônio Artificial é a unidade mais simples e básica de uma rede neural.

É possível verificar pela figura 1, que o mesmo funciona em duas etapas:

1. calcula a combinação linear das entradas;
2. normaliza a soma por meio da aplicação da função de ativação

Figura 1 – Exemplo de um neurônio mostrando as entradas (x_1 - x_n), os pesos correspondentes (w_1 - w_n), um limiar (b) e uma função de ativação (f) aplicada à combinação afim das esntradas.



Fonte: Gupta (2017)

Além desses parâmetros, necessários na fase de treinamento no aprendizado da RN, também existem pesos associados a cada entrada de um neurônio.

3.1.3 Funções de ativação

A função de ativação tem por finalidade a definição da tomada de decisão na saída de um neurônio, ou seja, delimitar decisões lineares ou não lineares é o papel fundamental das funções de ativação. Outro fator importante delas é o efeito de normalização que trazem para a saída do neurônio prevenindo o excesso na saída dos neurônios em redes com muitas camadas, devido ao efeito cascata.

Devido ao componente não linear introduzido pelas funções de ativação o fator de aprendizado da RN é mais relevante do que quando comparado a componentes lineares em relação as variáveis dependentes e independentes.

Considerando o modelo de uma rede neural clássica com duas camadas escondidas (sem perda de generalidade, omitindo os vieses b_i):

$$y = \phi(\phi(XW_1)W_2)w$$

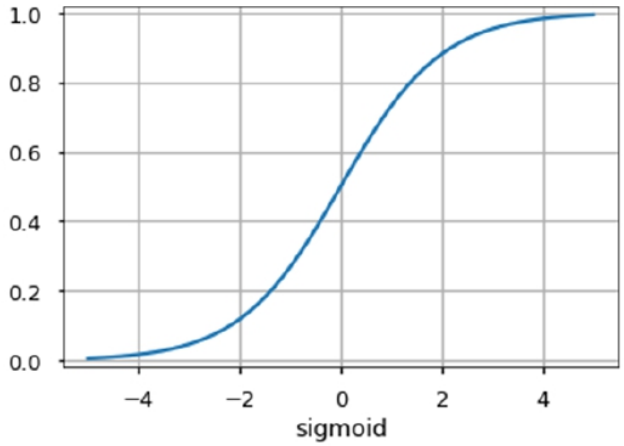
em que X é a matriz de dados, W_i são os pesos das camadas ocultas, w são os pesos da camada de saída e ϕ é uma função não linear qualquer. (FACURE, 2017)

3.1.3.1 Tipos de funções de ativação

A seguir, são apresentadas as três funções de ativação mais largamente usadas:

1. Sigmoide: mapeia a entrada (eixo x) para valores entre 0 e 1. Da mesma forma que neurônios biológicos, funcionam de forma binária (ativando vs. não ativando).

Figura 2 – Sigmoid

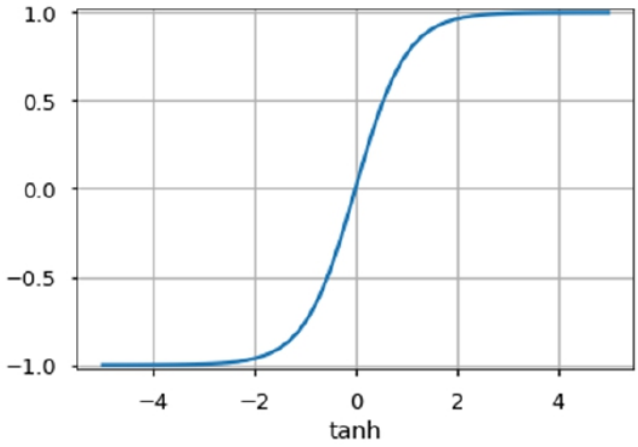


Fonte: Gupta (2017)

É uma boa forma de modelar esse comportamento, já que assume valores apenas entre 0 (não ativação) e 1 (ativação). Matematicamente é representada por:

$$\sigma(x) = \frac{1}{1+e^{x}} \quad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

Figura 3 – $\tanh$

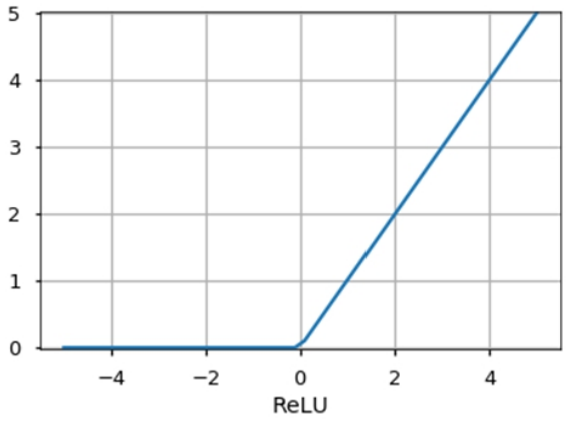


Fonte: Gupta (2017)

2. Tangente Hiperbólica ($\tanh$): similar à função sigmoide, porém mapeia valores de entrada entre -1 e 1. Tem maior proximidade à função identidade, uma função que tem como o valor retornado igual ao valor do argumento, o que a torna uma alternativa mais interessante do que a sigmoide como função de ativação nas camadas ocultas das RNAs. Matematicamente a função $\tanh$ é representada por:

$$\tanh(x) = 2\sigma(2x) - 1 \quad \tanh'(x) = 1 - \tanh^2(x)$$

Figura 4 – ReLU



Fonte: Gupta (2017)

3. Unidade Linear Retificada (ReLU): permite que apenas valores positivos passem por ela. Os valores negativos são mapeados para zero. Redes com a função ReLU são fáceis de otimizar, já que a ReLU é extremamente parecida com a função identidade. A única diferença é que a ReLU produz zero em metade do seu domínio (FACURE, 2017). Matematicamente ela é representada da seguinte forma:

$$ReLU(x) = max\{0, x\} \quad ReLU'(x) = \begin{cases} 1, & \text{se } x \geq 0 \\ 0, & \text{c.c.} \end{cases}$$

Importante citar que há outras funções como *Unit Step function*, *leaky ReLU*, *Noisy ReLU*, *Exponencial ReLU*, entre outras, sendo que cada uma tem suas vantagens e desvantagens; não cabendo ao escopo deste trabalho explanar sobre elas.

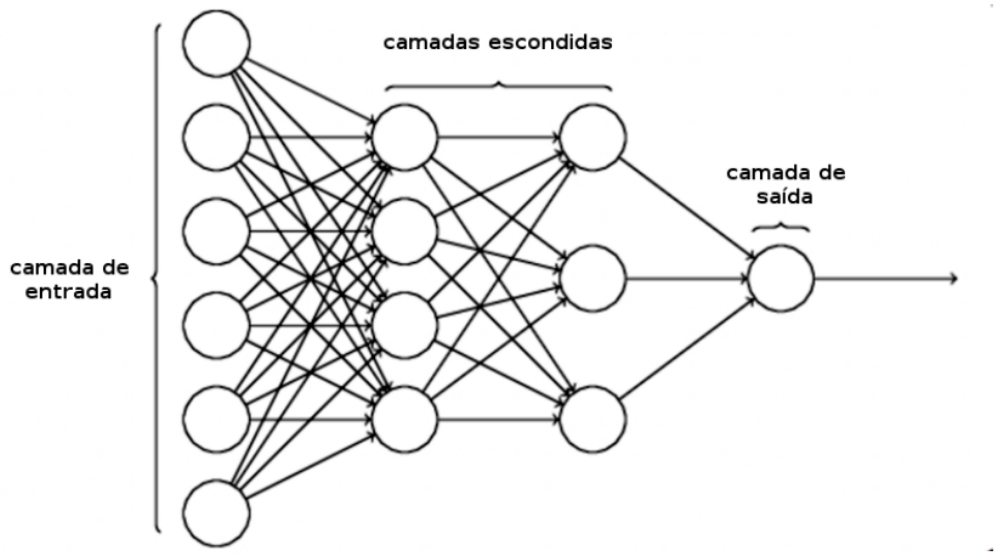
3.1.4 Camada de entrada

Esta é a primeira camada de uma RN. Ela é usada para prover a entrada de dados ou as características da rede.

3.1.5 Camada de saída

É a camada onde se aplica a função de ativação e onde se dá o aprendizado da RN através de classificações ou previsões. A escolha da função de ativação varia conforme o tipo de problema a ser resolvido.

Figura 5 – Exemplo de rede neural artificial com as diferentes camadas



Fonte: Gupta (2015)

3.1.6 Camadas escondidas ou ocultas

Região composta por uma ou mais camadas onde se processa os dados oriundos das camadas anteriores. Quanto maior o número de camadas, problemas mais complexos podem ser resolvidos.

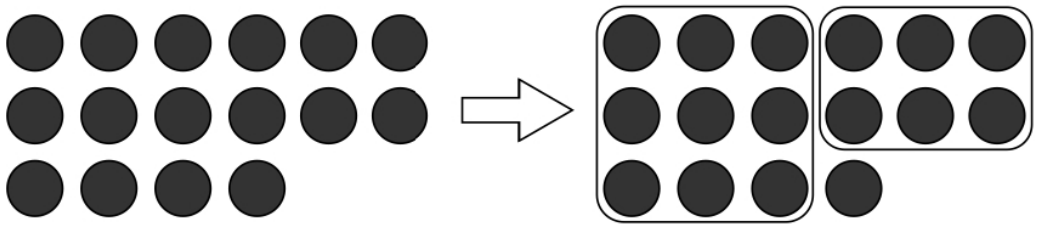
3.1.7 *FeedForward* de camadas múltiplas

Parte essencial nos modelos de *deep learning*, a **Rede *FeedForward* Profunda**, também conhecida por *Perceptrons* Multicamadas (MLPs) tem por objetivo aproximar alguma função f^*.

Por exemplo, para um classificador, $y = f^*(x)$ mapeia uma entrada x para uma categoria y. Uma rede Feedforword define um mapeamento $y = f(x; \theta)$ e aprende os valores dos parâmetros θ que resulta em uma melhor função de aproximação.(GUPTA, 2015)

Esses modelos são chamados *feedforward* porque a informação flui através de neurônios em uma camada escondida mais próxima à camada de entrada, para os neurônios da próxima camada escondida, mais próxima da camada de saída, sem nunca terem as informações retornando para si mesmos. Isso quer dizer que a informação flui de uma função a partir de x, indo através de cálculos que definem f até a saída y sem haver conexões de *feedback*.

Figura 6 – Exemplo de agrupamento em múltiplos de potências de 3



Fonte: Os autores (2017)

3.2 PONTO FIXO E PONTO FLUTUANTE

Para entender a importância deste trabalho, é preciso antes entender o que são números em ponto flutuante e em ponto fixo. Nesta seção são apresentados estes conceitos para aqueles que não os conhecem.

3.2.1 Bases numéricas

Os conceitos abstratos de número e quantidade não necessitam do conceito de base para serem compreendidos. Porém, na hora de representá-los, seja para apresentação dos mesmos a outras entidades ou para a realização de cálculos com os mesmos, se faz necessário a introdução do conceito de base numérica.

Bases numéricas são uma forma de representar um número na forma de agrupamento de quantidades como múltiplos de uma quantidade fixa. Por exemplo, na figura 6, o conjunto de bolas no lado esquerdo pode ser agrupado como múltiplos de três bolas. Desta forma, tem-se um conjunto com três conjuntos de três bolas, dois conjuntos de três bolas e uma bola isolada. A quantidade então pode ser denotada por $121_{(3)}$ (Lê-se, “um, dois, um base três”). Os números são lidos da esquerda para a direita, com os números à esquerda representando potências maiores que os da direita. É fácil verificar que o mesmo conjunto pode ser denotado por $16_{(10)}$, $10_{(16)}$, $100_{(4)}$, $31_{(5)}$ e $10000_{(2)}$.

Apesar de terem existido diversas bases numéricas utilizadas pela humanidade no passado, até recentemente a base mais usual era a base 10 (ou decimal), com outras bases sendo usadas apenas em casos específicos (como as bases 12, 24 e 60 na contagem de tempo). Com o advento dos computadores, as bases 2 (ou binária) e 16 (ou hexadecimal) passaram a ser amplamente utilizadas, ultrapassando o uso das outras bases, pois a imensa maioria dos cálculos hoje são realizados em dispositivos eletrônicos.

3.2.2 Representações numéricas em computadores

Nos computadores e sistemas eletrônicos em geral, procura-se usar sistemas de representações numéricas que tornem os cálculos e armazenamento dos números o mais barato computacionalmente possível, mas ainda mantendo a integridade e a confiabilidade dos dados e cálculos.

Alguns tipos de circuitos, tais como amplificadores operacionais, conseguiriam facilmente realizar operações de soma e subtração com altíssima precisão e muito rapidamente, já que trabalham diretamente com sinais analógicos, porém, é muito difícil armazenar os números nesta representação. Devido a esta dificuldade, as tentativas iniciais de se trabalhar em base dez nos computadores foi abandonada em favor da base dois, já que é mais fácil diferenciar sobre dois estados (há tensão ou não há tensão em um ponto do circuito, por exemplo) do que diferenciar entre dez estados.

Não faz parte do escopo deste trabalho detalhar a parte eletrônica da representação binária em um computador. Em um nível mais abstrato, pode-se dizer que cada dígito 0 ou 1 em binário em um computador é implementado fisicamente como uma chave que pode estar desligada ou ligada, respectivamente. Este dígito é chamado *bit*. Convencionou-se que um grupo de 8 *bits* formaria um *byte*, a unidade padrão para cálculo e armazenamento de dados nos primeiros computadores digitais. Visando-se aumentar a velocidade de processamento e a quantidade de memória endereçável, gradativamente foram sendo introduzidos novos tamanhos para agrupamentos de *bits*. Os tamanhos de 16, 32 e 64 são os mais comuns, devido ao fato de serem múltiplos de um *byte* garantindo compatibilidade retroativa com os sistemas anteriores.

3.2.2.1 Inteiros

Em um computador, números inteiros são representados nativamente como conjuntos de 1, 2, 4 ou 8 *bytes*, dependendo da faixa de valores que precisam representar. Podem ser inteiros com ou sem sinal, i.e., podem representar números positivos e negativos ou apenas números positivos, respectivamente. Na forma com sinal, utiliza-se o *complemento de 2* (cuja explicação não faz parte do escopo deste trabalho) por permitir que as operações de soma e subtração utilizem os mesmos circuitos (já que uma subtração é apenas a soma com o simétrico, em complemento de 2, de um número).

Considerando-se a representação com sinal, a faixa representável vai de -2^{n-1} e $2^{n-1}-1$, onde n é o número de *bits*, com todos os números inteiros intermediários tendo representação própria. Números maiores ou menores (fora da faixa representável) podem ser representados como vetores de algum formato nativo, mas todos os cálculos e manipulações passam ter de ser realizados em *software*.

Como exemplo, para um número de 8 *bits* com sinal, a faixa representável vai de -128 até 127, representados por $10000000_{(2)}$ e $01111111_{(2)}$ respectivamente. Os números -1, 0 e 1 são representados como $11111111_{(2)}$, $00000000_{(2)}$ e $00000001_{(2)}$ respectivamente.

Em geral, a utilização de aritmética com a representação nativa dos números inteiros é a mais rápida, com as instruções de máquina associadas levando, em média, apenas um ciclo de máquina para serem executadas (ignorando-se *pre-fetch*, *cache*, etc.).

3.2.2.2 Ponto flutuante

Para permitir faixas de operação maiores, bem como o uso de números reais nos cálculos, foram introduzidos os números em ponto flutuante. As formas de representação destes foram padronizadas pela IEEE e detalhes podem ser encontrados em AMD64... (2017).

De modo superficial temos: os números são representados por 1 *bit* de sinal s (-1 ou 1), seguido de alguns *bits* de expoente e e, por fim, os bits significativos m. Os números representáveis então são da forma $s * (1, m) * 2^e$. A faixa numérica representável desta forma torna-se bem maior se comparada com a faixa representável em números inteiros, porém com duas desvantagens:

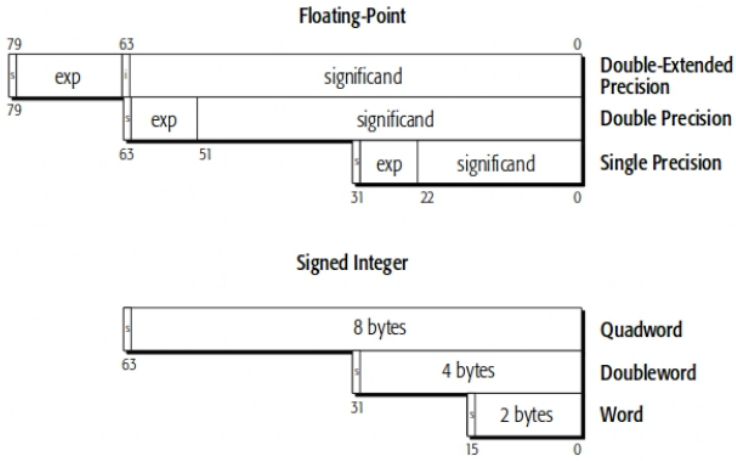
1. a distribuição entre os números não é homogênea, i.e., a distância entre dois números consecutivos próximos do máximo da faixa é bem maior quando comparada com a distância entre dois números próximos do zero;
2. devido à complexidade extra na representação, os cálculos em pontos flutuante gastam mais ciclos de máquina para executar operações semelhantes àquelas com números inteiros.

Desta forma, pode-se afirmar que programas que utilizam apenas operações com números inteiros (nativos) são executados mais rapidamente que aqueles que utilizam ponto flutuante (considerando exatamente as mesmas operações, trocando-se apenas os tipos dos dados). Por outro lado, as operações com números inteiros podem vir a inserir diversos erros de truncagem dos dados que não estariam presentes (ou seriam bem menores) se fossem utilizados números em ponto flutuante, além de as faixas de operação serem bem maiores se for usado ponto flutuante.

Como exemplo de faixas de operação, números em ponto flutuante de 32 *bits* no padrão IEEE 754 estão na faixa $\pm 1,40129846432481707 * 10^{-45}$ até $\pm 3,40282346638528860 * 10^{+38}$, enquanto que um número inteiro de 32 *bits* é capaz de representar apenas a faixa -2147483648 até 2147483647 como pode ser visto em Harold (1998).

Na figura 7 podemos ver diversas formas de se armazenar números em ponto flutuante e como inteiros em um processador AMD de 64 *bits*.

Figura 7 – Representações numéricas. Ponto Flutuante em cima e Inteiros em baixo



Fonte: AMD64. . . (2017)

3.2.2.3 Ponto fixo

A representação em ponto fixo é um meio termo entre as representações em inteiro e em ponto flutuante. Essa representação permite codificar valores com parte inteira e parte fracionária, com muita velocidade nos cálculos mas precisão menor em relação à representação em ponto flutuante.

Os números são armazenados como se fossem números inteiros em complemento de 2, com uma diferença: parte dos dígitos representa potências positivas de 2 e a outra parte, representa potências negativas. Por exemplo, para 8 *bits*, poderia ter-se optado por reservar um conjunto de *bits* para representar potências de 2 com sinal positivo (e.g., 2^4, 2^3, 2^2, 2^1, 2^0, etc.) e outro conjunto para representar aqueles com potência negativa (e.g., 2^{-1}, 2^{-2}, 2^{-3}). Neste exemplo, há uma "vírgula" imaginária implícita entre o 5^o e 6^o *bits*.

Nesta representação, as operações de soma e subtração são realizadas normalmente, como se fossem números inteiros (o que é rápido) e as multiplicações e divisões têm de realizar uma operação extra de deslocamento para compensar o fato de que parte dos dígitos representa valores fracionários.

Caso o local da vírgula decimal (binária) implícita seja escolhido apropriadamente, obtém-se uma faixa de operação razoavelmente boa com tempos de processamento igualmente bons.

3.3 SISTEMAS EMBARCADOS

> Sistemas embarcados estão cada vez mais baratos e acessíveis, demandam menor consumo de energia e, além de mais compactos, possuem maior poder de processamento. Com esse crescente poder de processamento, que é cada vez maior com o passar do tempo, o mundo em que vivemos será cada vez mais micro conectado, onde não só os computadores acessam a internet, mas também os objetos ao nosso redor e, até que em um futuro não tão longilíneo, viveremos em um mundo onde a linha que divide o real do virtual será tênue, se não imperceptível. Esse avanço, assim como os que já se passaram, também será graças aos sistemas embarcados. (SISTEMA. . . , 2013)

Sistema Embarcado é um sistema computacional microprocessado composto de *hardware e software* com propósito dedicado, i.e., construído para um conjunto de tarefas pré-definidas. O "computador" é completamente encapsulado ou dedicado ao sistema que ele controla, diferente do computador com propósito geral (como o computador pessoal). Os sistemas embarcados modernos são geralmente baseados em microcontroladores, i.e., CPUs com memória integrada e/ou interfaces periféricas como PIC ou Arduino e outros mais robustos como a *Raspberry Pi*. Geralmente eles tem como propósito de uso, aplicações computacionais de tempo real. Tendo isso em mente, tais sistemas não costumam ter sua funcionalidade alterada durante o uso, mas isso não impede que seu propósito seja alterado, sendo necessária a reprogramação do mesmo.

Alguns exemplos de uso de Sistemas Embarcados são:

- computadores de bordo automotivos;
- sistemas de controle de acesso biométrico;
- controles inteligentes de temperatura para geladeiras;
- equipamentos médicos (monitores, estetoscópios eletrônicos, etc).

3.3.1 PIC

PIC é uma família de microcontroladores fabricadas pela *Microchip Technology*. Seu nome era inicialmente referido a *Peripheral Interface Controller* ou Controlador de Interface Periférica, posteriormente foi alterado para *Programmable Intelligent Computer* ou Computador Inteligente Programável. Todos os modelos atuais usam memória *flash* para o armazenamento de programas e os mais novos são reprogramáveis. Além disso, a memória de programa e a memória de armazenamento são separadas sendo a de armazenamento de 8-*bits*, 16-*bits* ou 32-*bits*(nos modelos mais avançados). A quantidade de *bits* nas instruções de programas variam de acordo com a família de PIC e podem ser de 12, 14, 16 ou 24 *bits* de comprimento.

3.3.2 Arduino

Arduino é um sistema de placa única comumente utilizado para prototipagem. É também a companhia de mesmo nome que fabrica tal sistema, bem como o projeto e a comunidade de usuários que ajuda a desenvolver o projeto.

Existem vários modelos, cada qual com suas características particulares, porém todos seguindo o mesmo princípio básico: ter uma placa de circuitos única, contendo processador e memória integrados, bem como portas para conexão de periféricos. Conectores na placa oferecem a possibilidade à portas de entrada e saída, facilitando o trabalho de prototipação.

Outra característica comum a todos os modelos é a limitação de *hardware*: os processadores são bem mais simples, lentos e baratos que os topo de linha, para deixá-los mais compatíveis com a realidade dos dispositivos embarcados.

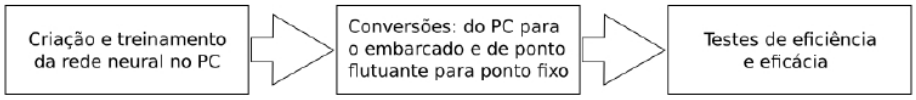
3.3.3 *Rapberry Pi*

De modo similar ao Arduino, *Raspberry Pi* é uma série de computadores em placa única, contendo processador e memória, além de periféricos e portas de entrada e saída. Apesar de também ser usado para prototipação, difere do Arduino por não pretender ser usado apenas em sistemas embarcados, podendo os modelos mais robustos competir com PCs mais simples.

4 METODOLOGIA, PROJETO E DESENVOLVIMENTO

4.1 VISÃO GERAL

Figura 8 – Diagrama esquemático das etapas do processo de desenvolvimento deste trabalho



Fonte: Os autores (2017)

Para alcançar os objetivos deste trabalho foram realizadas as seguintes etapas: criação de uma RN, em ponto flutuante, no PC para o reconhecimento facial usando um banco de dados aberto disponível na internet; criação da ferramenta para a execução de RN qualquer em ponto flutuante; conversão da ferramenta para execução de RN em ponto fixo; treinamento da RN utilizando as ferramentas geradas; verificação da eficiência e eficácia da mesma; conversão da RN para ponto fixo; verificação da eficiência e eficácia da mesma em ponto fixo ainda no PC; conversão das ferramentas de execução de RN para o *Raspberry Pi*; realizada a cópia da RN do ambiente PC, sem modificação, para o ambiente *Raspberry Pi*; realizados os mesmos testes de eficiência e eficácia no ambiente embarcado; adaptação das ferramentas de execução de RN para captura de imagens através de uma câmera USB; verificação da eficiência e eficácia da RN no reconhecimento de faces em tempo real.

O sistema, em resumo, se propõe a verificar se a face de uma pessoa enquadrada em frente a uma câmera USB conectada a um *Raspberry Pi*, condiz com alguma imagem existente em uma base de dados pré cadastrada.

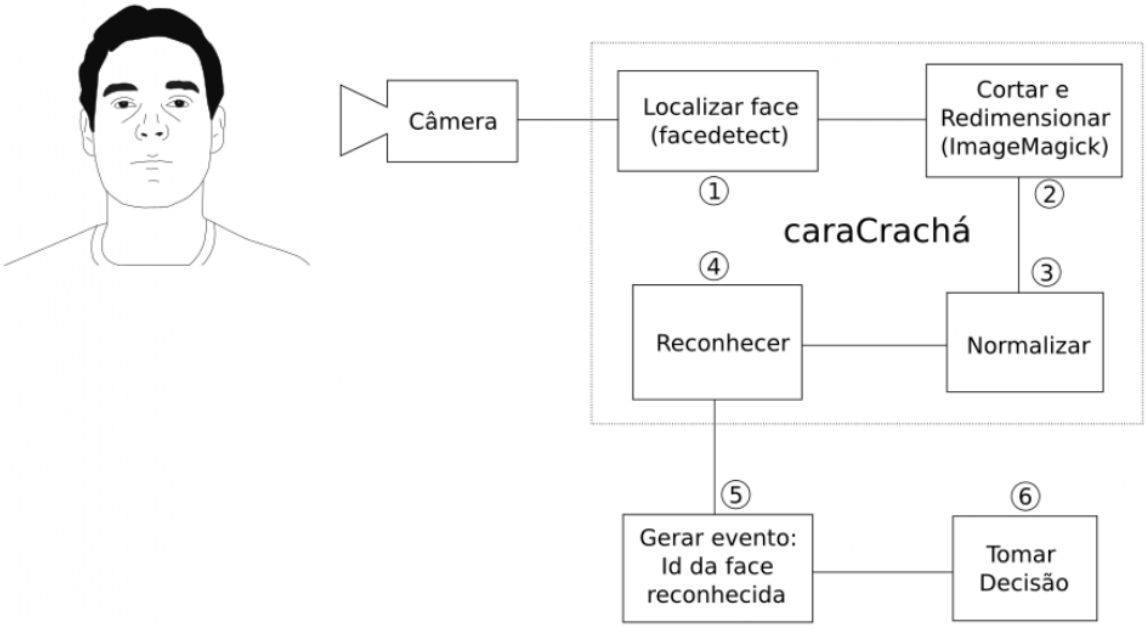
A figura 8 apresenta o esquema simplificado do processo de desenvolvimento do sistema e a figura 9 mostra o sistema em seu estado final.

4.2 METODOLOGIA DO EXPERIMENTO

4.2.1 Etapa de execução no PC

Esta etapa foi executada no PC e compreende o treinamento e avaliação da rede em ponto flutuante, bem como sua conversão para o formato *FANN* e a conversão interna para representação em ponto fixo.

Figura 9 – Diagrama esquemático do sistema - testes



Fonte: Os autores (2017)

4.2.1.1 Considerações gerais

Inicialmente, foi realizada a instalação da biblioteca *Encog* (HEATON, 2017) no PC do experimento. Essa instalação se deu através da ferramenta *Maven* (MAVEN, 2013), que faz o *download* de todos os requisitos para execução dentro do IDE - *NetBeans* (NETBEANS. . . , 2017), mas não deixa a instalação disponível para o restante do sistema. Posteriormente, foi realizada a instalação da biblioteca FANN (NISSEN et al., 2017) na mesma máquina e no mesmo IDE. Esta biblioteca é instalada no sistema. Depois dos *includes* da FANN configurados corretamente no projeto, a compilação dentro do IDE é simples.

Para verificar se o ambiente funcionava corretamente, executou-se o exemplo básico da Encog de reconhecimento de imagens (figuras de moedas). Foram adicionadas algumas moedas extras e versões extras das moedas já presentes no exemplo, para garantir que o exemplo realmente funcionava. O objetivo dessa fase era comprovar que a rede neural (no *Encog*) realmente conseguia reconhecer as imagens.

Da mesma forma, executou-se os exemplos básicos da FANN, especialmente o par de exemplos que tratavam:

1. do treinamento de uma rede neural para se comportar como uma porta lógica XOR;

2. de usar como entrada a rede neural referida previamente, bem como o conjunto de testes na porta lógica XOR e verificar se os resultados conferem com o espe-

rado.

O objetivo dessa fase era comprovar que os exemplos em ponto flutuante da FANN realmente funcionavam.

Posteriormente, tentou-se adaptar um dos exemplos da FANN para trabalhar com ponto fixo.

Os treinamentos de rede neural na FANN são feitos exclusivamente em ponto flutuante para posteriormente, converter as redes geradas para ponto fixo. Desta forma, ao se fazer referência às funções de treinamento no programa compilado em ponto fixo, sempre era exibida uma mensagem de erro dizendo que tais funções não estavam definidas.

De fato, havia na FANN duas coleções de funções muito similares: uma para ponto fixo e outra para ponto flutuante. Entretanto, as funções de treinamento estavam definidas apenas no conjunto para ponto flutuante, mas não no grupo de funções para ponto fixo. Deste ponto em diante, teve-se sempre em mente que os treinamentos jamais poderiam ser realizados em ponto fixo, nesta biblioteca (FANN). A rede neural com ponto fixo só poderia ser obtida através do treinamento completo em ponto flutuante e posterior conversão da rede treinada.

Posteriormente, adaptou-se o exemplo de reconhecimento de imagens da biblioteca *Encog* para reconhecer imagens de faces. Usamos como banco de dados de treinamento o banco disponibilizado gratuitamente para fins de pesquisa por Thomaz (2012). As modificações no exemplo foram as seguintes:

- Na criação do conjunto de treinamento, ao invés de se utilizar a classe *SimpleIntensityDownsample*, optou-se por utilizar uma extensão da mesma criada pelos autores chamada de *SuperSimpleIntensityDownsample*. Mesmo que a classe original trabalhe apenas com tons de cinza, ela ainda guarda três *bytes* para cada *pixel* da imagem (ela é apenas uma variante da classe *RGBDownsample* que não usa todas as informações armazenadas). A classe criada pelos autores armazena apenas 1 *byte* por *pixel*.

- A rede era composta de uma camada de entrada, três camadas escondidas (128, 16 e 8 neurônios) e uma camada de saída. Exceto na camada de entrada, a função de ativação usada foi a tangente hiperbólica.

No anexo C.1 pode ser visto um trecho de código mostrando como recriar uma rede neual com as mesmas especificações.

O banco de dados utilizado possui 200 conjuntos de fotos de voluntários com poses padronizadas. São 14 fotos por pessoa, a saber: foto de frente com expressão

Figura 10 – Imagens de treino “arnaldo” antes de adequá-las como entrada da rede

Fonte: Os autores (2017)

neutra, de frente sorrindo, perfil direito, perfil esquerdo, alguns perfis intermediários, todas com iluminação de estúdio. Também há fotos de frente com expressão neutra e níveis de iluminação variados: ambiente iluminado naturalmente, ambiente escuro.

Para os fins de nossos experimentos, utilizamos apenas as fotos de frente (independente de iluminação) e as fotos intermediárias entre frente e perfil que mais próximas estivessem de fotos frontais. Fotos de 50 indivíduos deste banco foram usadas como treinamento (rotulados "desconhecido"), fotos de 1 indivíduo foram usadas no treinamento e rotuladas "joão"e fotos de outros 31 indivíduos foram usadas como imagens de teste (esperando que a rede identificasse estes como "desconhecido"). Para este exemplo também foram usadas imagens dos autores em posições e condições de iluminação similares às do banco (rotulados "arnaldo"e "diogo").

O objetivo seria treinar uma rede neural para reconhecer os autores, exibindo mensagem com o nome dos mesmos quando lhe fosse apresentada fotos deste, exibindo uma mensagem de "joão"quando lhe fosse apresentada fotos do indivíduo teste retirado do banco e exibindo uma mensagem de “desconhecido” quando lhe fosse apresentada uma das outras fotos presentes no banco.

4.2.1.2 Criação, treinamento e teste de rede em ponto flutuante

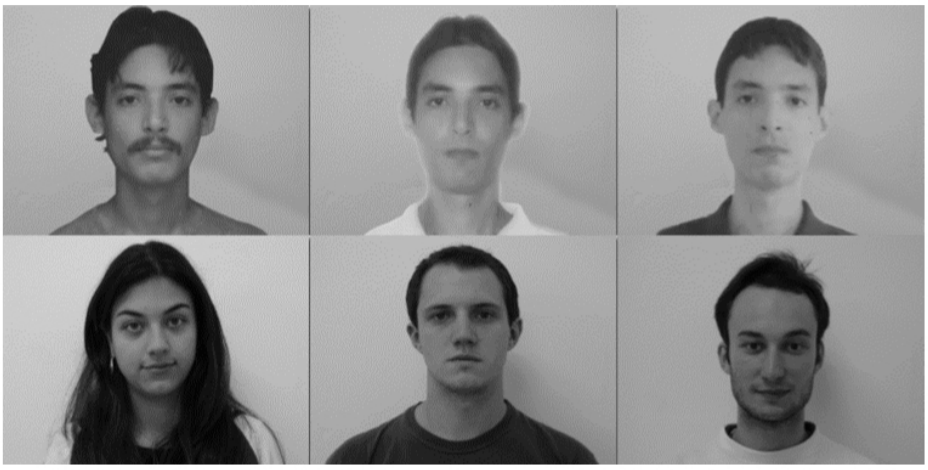
O objetivo desta fase foi criar, treinar e testar um rede neural no PC por meio da biblioteca *Encog* em ponto flutuante (*pf*) para cada conjunto de imagens de testes.

Tentou-se várias configurações de redes (variando-se o número de camadas e o número de neurônios em cada uma delas), até ser obtido um índice de acerto de 90,5% nos testes da rede. Vale ressaltar que o foco deste trabalho não era treinar uma rede, mas sim converter uma rede já treinada e verificar se o comportamento era

Figura 11 – Exemplo de imagens de treino “desconhecido” antes de adequá-las como entrada da rede

Fonte: Thomaz (2012)

Figura 12 – Imagens de teste



Fonte: Os autores (2017) e Thomaz (2012)

compatível com a rede original, portanto foi adotado o referido índice de acerto como aceitável.

Ademais, com as restrições impostas de iluminação, estrutura da rede neural usada para a tarefa e tamanho de entrada da figura, o problema de reconhecimento se torna não trivial. Em geral se usa uma rede muito maior, com entrada previamente tratada e uma rede neural com outra estrutura. Entretanto, como esses fatores são conflitantes com o objetivo de inserir uma rede neural leve (memória e CPU) num sistema embarcado IoT inteligente, os fatores citados foram relaxados.

Figura 13 – Imagens de treino “arnaldo” preparadas. Faces isoladas e redimensionadas para 48x48 pixels

Fonte: Os autores (2017)

4.2.2 Etapa de conversão para o sistema embarcado (*Raspberry PI*)

Nesta etapa foram realizados o desenvolvimento associado à rede em ponto fixo, bem como seu teste.

O desenvolvimento foi realizado primariamente no PC, porém partes do desenvolvimento dos *scripts* usados no sistema foram desenvolvidos diretamente na *Raspberry PI*.

Os testes, por sua vez, foram realizados primariamente na *Raspberry PI*, com testes de validação sendo feitos em ambas as plataformas (PC e *Raspberry Pi*).

4.2.2.1 O sistema CaraCrachá

Criou-se um programa em java que converte uma rede salva no formato da *Encog* para o formato das redes salvas pela FANN. No caso, ambas as redes ainda estão com seus valores em ponto flutuante. Depois, criou-se um programa em C que lê uma rede gravada no formato da FANN em ponto flutuante e a converte para o formato da FANN em ponto fixo.

Ao construir tal programa, verificou-se que a nomenclatura utilizada pelas bibliotecas *Encog* e FANN para denominar as funções de ativação não era a mesma e muitas destas funções não possuem equivalência óbvia. Após analisar as funções disponíveis em ambas as bibliotecas, havia apenas duas em comum: Sigmoide e $\tanh$. Baseado em Facure (2017), optamos por utilizar $\tanh$. Na tabela 1 vê-se uma lista das funções de ativação na *Encog* e sua equivalente na FANN, quando esta foi encontrada.

Tabela 1 – Equivalência entre as funções de ativação da Encog e da FANN

ActivationBiPolar	*
ActivationBipolarSteepenedSigmoid	FANN_SIGMOID_SYMMETRIC_STEPWISE
ActivationClippedLinear	*
ActivationCompetitive	*
ActivationElliott	FANN_ELLIOT
ActivationElliottSymmetric	FANN_ELLIOT_SYMMETRIC
ActivationGaussian	FANN_GAUSSIAN
ActivationLinear	FANN_LINEAR
ActivationLOG	*
ActivationRamp	FANN_THRESHOLD
ActivationSigmoid	FANN_SIGMOID
ActivationSIN	FANN_SIN
ActivationSoftMax	*
ActivationSteepenedSigmoid	FANN_SIGMOID_STEPWISE
ActivationStep	*
ActivationTANH	FANN_SIGMOID_SYMMETRIC
	FANN_COS
	FANN_COS_SYMMETRIC
	FANN_GAUSSIAN_STEPWISE
	FANN_GAUSSIAN_SYMMETRIC
	FANN_LINEAR_PIECE
	FANN_LINEAR_PIECE_SYMMETRIC
	FANN_SIN_SYMMETRIC
	FANN_THRESHOLD_SYMMETRIC

*Usada FANN_LINEAR por falta de equivalência óbvia
Fonte: Os autores (2017)

Durante a etapa de treinamento, foi observado que o normalizador interno da *Encog* gera entradas desnecessárias quando está trabalhando com imagens em tons de cinza, gerando entradas “em branco” (zeros) de modo a completar a mesma quantidade de entradas necessárias a uma imagem colorida. Ou seja, 3 vezes mais informação, das quais apenas $\frac{1}{3}$ é informação útil. Por isso, também foi estendida a classe *SimpleIntensityDownsample* por uma classe que diminui a quantidade de entradas para o valor estritamente necessário.

4.2.2.2 A rede em ponto fixo

O objetivo desta parte é mostrar como foi realizada a conversão da rede original (em ponto flutuante) para uma rede em ponto fixo, através da biblioteca FANN.

Ao se pesquisar como deveria ser o formato dos dados (imagem) que deveriam ser passados como entrada para a rede, descobriu-se que a *Encog* não utiliza uma lógica simples para normalizar os dados. Optou-se por construir a tabela 2 entre as

intensidades dos pixels e os valores normalizados pela *Encog*. Vê-se pela tabela, que a *Encog* prioriza diferenciar muito mais os tons escuros uns dos outros que os tons claros. Os valores dos pixels indo de 0 a 54 (parte escura da imagem) são mapeados para um intervalo de medida 1 (-1.0 a 0.0), enquanto que os valores de 55 a 255 são mapeados para um intervalo de mesma medida (0.0 a 1.0). Por exemplo: os pixels que, em uma imagem em tons de cinza, cujas intensidades de luminosidade/brilho são representadas pelos números 10 e 11 são mapeados por -0,5608 e -0,5373 respectivamente (uma diferença de 0,0235); por outro lado, 244 e 245 são mapeados por 0,9608 e 0,9686 respectivamente (uma diferença de apenas 0,0078, i.e., três vezes maior na faixa escura que na faixa clara, para este exemplo, em particular).

Usando-se a tabela 2 (os valores na tabela estão aproximados até a quarta casa decimal), criou-se um programa normalizador que lê uma imagem no formato PNG e, para cada pixel da mesma, imprime no console o valor tabelado (ou sua versão para ponto fixo). Optou-se por fazer este programa exibir a saída para imagens em tons de cinza (mesmo que a entrada seja colorida - o programa exibe apenas um valor por pixel, ao invés de três) e por treinar a rede com imagens igualmente em tons de cinza. Seres humanos são capazes de identificar uma pessoa mesmo quando lhes é apresentada uma foto da mesma em tons de cinza, o que indica que provavelmente as informações de cor não são realmente necessárias no reconhecimento.

Como em ponto fixo existe um multiplicador fixo para os valores trabalhados, o normalizador lê o arquivo especificador da rede e obtém o multiplicador correto lá contido. A partir daí, ele multiplica os valores da tabela 2 para obter os valores esperados como entrada para a rede em ponto fixo. Em um exemplo bem simples, os valores presentes em uma imagem em tons de cinza vão de 0 a 255, mas a rede neural espera valores de entrada numa faixa como -10000 a 25000, então o normalizador faz a conversão dos valores para a faixa apropriada usando a tabela e um número específico daquela rede neural.

O programa que executa a RN (lida do disco), lê do console as entradas da rede e, após submeter estas à rede, decide se detectou alguém ou não, bem como quem foi detectado, em caso afirmativo.

Optou-se também por reduzir a resolução das imagens de entrada: imagens com resoluções menores acarretam uma menor quantidade de memória para armazenar as mesmas, bem como, uma menor quantidade de memória para armazenar a própria rede neural. E, visto que a rede neural é menor, a quantidade de operações necessárias para uma rede avaliar uma entrada é menor, tornando a sua execução mais rápida.

Como desvantagem, tal rede pode não possuir informações suficientes para tomar a decisão correta. Estes fatores foram equilibrados usando resoluções menores,

Tabela 2 – Valores usados na lógica da *Encog* na normalização antes do treinamento

0	-1,0000	43	-0,1059	86	0,2314	129	0,4824	172	0,6784	215	0,8588
1	-0,8980	44	-0,0980	87	0,2392	130	0,4824	173	0,6863	216	0,8588
2	-0,8275	45	-0,0824	88	0,2471	131	0,4902	174	0,6863	217	0,8667
3	-0,7804	46	-0,0745	89	0,2471	132	0,4902	175	0,6941	218	0,8667
4	-0,7333	47	-0,0667	90	0,2549	133	0,4980	176	0,6941	219	0,8667
5	-0,7020	48	-0,0588	91	0,2627	134	0,5059	177	0,7020	220	0,8745
6	-0,6706	49	-0,0510	92	0,2706	135	0,5059	178	0,7098	221	0,8745
7	-0,6392	50	-0,0431	93	0,2784	136	0,5137	179	0,7098	222	0,8824
8	-0,6078	51	-0,0275	94	0,2784	137	0,5216	180	0,7176	223	0,8824
9	-0,5843	52	-0,0196	95	0,2863	138	0,5216	181	0,7176	224	0,8902
10	-0,5608	53	-0,0118	96	0,2941	139	0,5294	182	0,7255	225	0,8902
11	-0,5373	54	-0,0039	97	0,3020	140	0,5373	183	0,7255	226	0,8980
12	-0,5216	55	0,0039	98	0,3098	141	0,5373	184	0,7333	227	0,8980
13	-0,4980	56	0,0118	99	0,3098	142	0,5451	185	0,7333	228	0,9059
14	-0,4824	57	0,0196	100	0,3176	143	0,5451	186	0,7412	229	0,9059
15	-0,4588	58	0,0275	101	0,3255	144	0,5529	187	0,7412	230	0,9137
16	-0,4431	59	0,0353	102	0,3333	145	0,5608	188	0,7490	231	0,9137
17	-0,4275	60	0,0431	103	0,3333	146	0,5608	189	0,7490	232	0,9216
18	-0,4118	61	0,0510	104	0,3412	147	0,5686	190	0,7569	233	0,9216
19	-0,3961	62	0,0588	105	0,3490	148	0,5686	191	0,7569	234	0,9294
20	-0,3804	63	0,0667	106	0,3569	149	0,5765	192	0,7647	235	0,9294
21	-0,3647	64	0,0745	107	0,3569	150	0,5843	193	0,7725	236	0,9294
22	-0,3490	65	0,0824	108	0,3647	151	0,5843	194	0,7725	237	0,9373
23	-0,3333	66	0,0902	109	0,3725	152	0,5922	195	0,7804	238	0,9373
24	-0,3255	67	0,0980	110	0,3725	153	0,5922	196	0,7804	239	0,9451
25	-0,3098	68	0,1059	111	0,3804	154	0,6000	197	0,7882	240	0,9451
26	-0,2941	69	0,1137	112	0,3882	155	0,6078	198	0,7882	241	0,9529
27	-0,2784	70	0,1216	113	0,3961	156	0,6078	199	0,7961	242	0,9529
28	-0,2706	71	0,1294	114	0,3961	157	0,6157	200	0,7961	243	0,9608
29	-0,2549	72	0,1373	115	0,4039	158	0,6157	201	0,8039	244	0,9608
30	-0,2471	73	0,1451	116	0,4118	159	0,6235	202	0,8039	245	0,9686
31	-0,2314	74	0,1529	117	0,4118	160	0,6314	203	0,8118	246	0,9686
32	-0,2235	75	0,1608	118	0,4196	161	0,6314	204	0,8118	247	0,9686
33	-0,2078	76	0,1608	119	0,4275	162	0,6392	205	0,8196	248	0,9765
34	-0,2000	77	0,1686	120	0,4275	163	0,6392	206	0,8196	249	0,9765
35	-0,1843	78	0,1765	121	0,4353	164	0,6471	207	0,8275	250	0,9843
36	-0,1765	79	0,1843	122	0,4431	165	0,6471	208	0,8275	251	0,9843
37	-0,1686	80	0,1922	123	0,4510	166	0,6549	209	0,8353	252	0,9922
38	-0,1529	81	0,2000	124	0,4510	167	0,6627	210	0,8353	253	0,9922
39	-0,1451	82	0,2078	125	0,4588	168	0,6627	211	0,8431	254	1,0000
40	-0,1373	83	0,2157	126	0,4667	169	0,6706	212	0,8431	255	1,0000
41	-0,1216	84	0,2157	127	0,4667	170	0,6706	213	0,8510		
42	-0,1137	85	0,2235	128	0,4745	171	0,6784	214	0,8510		

Fonte: Os autores (2017)

mas que ainda aparentassem ao olho humano similaridade com as fotos das pessoas sendo testadas.

Todos os teste que poderiam ser feitos em ponto flutuante e em ponto fixo, foram feitos e rodaram em ambos: nos dois *hardwares*, nas duas bibliotecas de rede e ainda nos dois formatos de representação interna da rede. Após isso feito, foi analisado se o aproveitamento da rede para o reconhecimento das imagens se mantém quando muda de pf para pfo.

Nos testes, as redes convertidas no formato do *Encog* para o formato da FANN em pf tiveram 100% de consistência entre resultados obtidos por ambas. Ou seja, quando a RN original apresentava um verdadeiro positivo (VP), um falso positivo (FP), um verdadeiro negativo (VN) ou um falso negativo (FN) para uma dada entrada, a RN convertida apresentava exatamente a mesma saída: um VP, um FP, um VN ou um FN respectivamente. Já as redes convertidas para pfo apresentavam divergência em menos de 3% dos casos testados. Ou seja, a cada 100 entradas fornecidas, 3 poderiam apresentar um FN na rede original, mas um VP na rede convertida, por exemplo.

Algo que ficou claro nesta etapa e na etapa de testes é que a qualidade das imagens de treinamento e o nível de iluminação nas mesmas influencia enormemente nos resultados. Imagens com nível de contraste maior no treinamento propiciam resultados melhores que aquelas com baixo contraste, por exemplo.

4.2.3 Etapa de execução na *Raspberry Pi*

Durante o preparo do ambiente embarcado, foi realizada a instalação, atualização (`apt-get update` e `upgrade`) e configuração do *Raspbian 9* no *Raspberry Pi 3* para execução da RN. Primeiramente foi instalada e configurada a *webcam MS VX6000* e, para isso, foi necessário a instalação e configuração dos seguintes *softwares* e serviços auxiliares: *FS webcam* (HERON, 2017), Codec de Vídeo x264, *FFmpeg* (BELLARD, 2017), *Motion* (MOTION, 2017) (serviço instalado e devidamente configurado para *streaming* de vídeo), *imagemagick* (IMAGEMAGICK, 2017) e o projeto *facedetect* (D'ELIA, 2016).

Ao se utilizar a *webcam Microsoft vx 6.000*, verificou-se a necessidade de vídeo frontal para boa qualidade de imagem, sendo que qualquer fonte de brilho que não seja frontal prejudica muito a qualidade da imagem a ser capturada, devido à geração de sombras que não são tratadas pela câmera.

Outra característica relevante: nas configurações do serviço *motion*, a opção de auto brilho não ajuda na qualidade da captura da imagem; pelo contrário, as imagens ficam exageradamente brancas. Dessa forma, as configurações ideais de ima-

Tabela 3 – Configurações da câmera

Brilho	Contraste	Saturação
157	200	127

Fonte: Os autores (2017)

gem para captura foram as especificadas na tabela 3.

Para capturar uma imagem primeiro foi necessário configurar o *motion* corretamente, pois a câmera sempre inicia com a imagem muito escura, o que inviabiliza o processo de reconhecimento. Primeiro inicia-se o serviço do *motion*, depois o próprio programa do *motion*. Depois de configurado o programa, finaliza-se o mesmo. A partir de agora é possível realizar a captura utilizando o programa *fswebcam*. Para o projeto, o *fswebcam* foi configurado para produzir imagens com resolução de 640x480 pixels.

Uma vez configurado o sistema na *Raspberry PI*, as redes convertidas (pf e pfo) foram copiadas para a mesma. Criou-se então dois *scripts* em BASH que realizam a sequência de operações necessárias (um para a rede em ponto flutuante e outro para a rede convertida para ponto fixo). Um pseudo-código dos passos realizados nos *scripts* pode ser visto em 4.1. O código real dos *scripts* pode ser vistos nos anexos C.2 e C.3.

4.1 – Pseudo-código

```
1 enquanto verdade faça
2     capture uma imagem da câmera (via fswebcam)
3     detectar a maior face presente na mesma (via facedetect)
4     isolar a área de interesse - face - da imagem e redimensioná-
          la para 48x48 pixels (via convert, do imagemagick)
5     normalizar a face (usando nosso normalizador)
6     redirecionar a saída do nosso normalizador para a entrada do
          nosso programa que executa as RN convertidas
7 repetir
```

Fonte: Os autores (2017)

A parte principal do código que converte uma rede no formato Encog para o formato FANN pode ser vista no apêndice C.4 e conversão da rede para ponto fixo é feita pela própria biblioteca FANN como pode ser visto no apêndice C.5.

Como a rede espera que os dados da imagem estejam normalizados segundo as especificações da biblioteca Encog, o normalizador usa os dados da tabela 2 para converter o valor dos pixel para o intervalo esperado. Pode normalizar tanto uma rede em ponto flutuante quanto uma em ponto fixo; neste caso, multiplica os valores normalizados pelo multiplicador para obter valores inteiros na faixa apropriada. A saída

do normalizador é o console padrão, mas este pode ser redirecionado. No apêndice C.6 pode ser visto o código do normalizador.

O código fonte do programa que executa as redes em ponto flutuante e em ponto fixo é compilado duas vezes (com diretivas de compilação apropriadas) para gerar os respectivos códigos objeto. O código fonte do mesmo pode ser visto no apêndice C.7.

4.3 ARQUITETURA DO SISTEMA DE TESTES

Para a realização dos testes necessários foi utilizado o seguinte ambiente:

4.3.1 Hardware

1. *Intel Core i7-4720HQ 2.60 GHz, 16 GB RAM, 1 TB HDD, NVIDIA GeForce GTX 960M 2 GB*;
2. *Raspberry Pi 3 Model B v1.2* rodando *Raspbian 9* (RASPBIAN..., 2017) (*Quad Core 1.2 GHz Broadcom BCM2837 64bit CPU - 1GB RAM* - Cartão de memória *microUSB* de *16GB*);
3. *Webcam USB Microsoft VX-6000*;

4.3.2 Ambiente de sistema de desenvolvimento

1. Sistema Operacional *Linux Mint 18 "Sarah"* (LINUX..., 2006) versão *64 bits* com gerenciador de janelas *cinnamon 3.0.7*;
2. *Netbeans* versão *8.2* usando *OpenJDK 1.8.0_151* (OPENJDK, 2017), uma máquina virtual Java (JAVA, 2017);
3. *Encog* versão *3.3.0*;
4. FANN versão *2.2.0*.

4.3.3 Ambiente embarcado

1. *Raspbian 9, release 2017-11-29*;
2. *FANN* versão *2.2.0*;
3. *Imagemagick* versão *6.9.7-4*;
4. *Fswebcam* versão *20140113*;
5. *Facedetect* versão *0.1*.

4.3.4 Estrutura da RN

1. RN do tipo *FeedForward* de Camadas Múltiplas, contendo 3 camadas escondidas de 128, 16, 8 neurônios respectivamente e saída booleana com 4 neurônios (duas representando os autores, uma representando um indivíduo do banco de dados de exemplo e uma representando desconhecidos em geral). Na implementação na biblioteca *Encog*, foi utilizado o objeto *BasicNetwork* para criação desta rede, com função de ativação *ActivationTANH* em todas as camadas. Essa função foi escolhida para facilitar a conversão de formatos entre *Encog* e FANN;
2. Função de ativação $\tanh$ escolhida devido ao fato da função sigmoide não apresentar resultados satisfatórios. Além disso, segundo Facure (2017):

   > A $\tanh$ se aproxima mais da identidade, sendo assim uma alternativa mais atraente do que a sigmoide para servir de ativação às camadas ocultas das RNAs. o valor da derivada é maior, chegando ao máximo de 1 quando $x = 0$. Por esse motivo, quando uma função sigmoidal precisa ser utilizada, recomenda-se a $\tanh$ no lugar da sigmoide.

3. Entrada da RN com imagens tons de cinza com resolução de 48x48px;
4. Conjunto de treinamento composto de 319 imagens sendo 306 obtidas de um BD aberto (THOMAZ, 2012) e 13 dos autores (6 Arnaldo e 7 Diogo);
5. Conjunto de testes composto de 77 imagens sendo 38 obtidas de um BD aberto e 39 dos autores (26/13 - Arnaldo e Diogo respectivamente).

4.4 MEDIDAS DE DESEMPENHO

Buscando avaliar se o objetivo deste trabalho seria cumprido, foram usadas as duas métricas a seguir:

- **Tempo de execução médio por imagem analisada** - indica se o uso de ponto fixo no lugar de ponto flutuante apresenta algum ganho de desempenho. Em caso positivo, faz sentido pensar em substituir uma forma de representação numérica pela outra, além de tornar mais fácil implementar redes neurais em IoT.

  Para realizar as medições propriamente ditas, usou-se a ferramenta *time*. Mais detalhes sobre o funcionamento da mesma podem ser encontrados no anexo A (principalmente o que vêm a ser tempo “real”, de “usuário” e de “sistema”).

- **Número de erros e acertos** - indica se, ao trocar ponto flutuante por ponto fixo, a confiabilidade de uma rede neural não se altera (e, portanto, trocar uma pela outra não introduz problemas). As medições são feitas contando-se o número de erros e acertos da rede ao realizar o processo de identificação e agrupando os

mesmos como VP, FP, VN e FN, conforme o caso. Calcula-se então um índice de discordância entre as redes

$$100 \times \frac{\text{número de vezes que as redes discordaram}}{\text{número total de testes}}\%$$

5 RESULTADOS

Como já citado, o modelo de rede utilizado é um *Deep FeedForward*. Como o nome indica, têm como estrutura uma rede profunda, com camadas empilhadas e características de fluxo de dados apenas no sentido da camada de saída, sem retroalimentação dos neurônios.

Dadas as condições do experimento, uma das razões para ter-se optado por um número limitado de camadas ocultas no modelo de rede proposto (que é de apenas 3 camadas escondidas), é que os dispositivos embarcados considerados possuem baixo poder de processamento. Assim, muitas camadas poderiam comprometer o tempo de execução.

Ademais, são dispositivos com uma quantidade bastante limitada de memória disponível. Mesmo a rede utilizada no experimento já não cabe na memória de alguns dispositivos. Este também foi um dos motivos de não ser utilizado nenhuma camada de convolução nesta rede, mesmo que a grande maioria da literatura que trata de reconhecimento facial recomende seu uso.

Dispositivos com uma quantidade de memória menos restrita e melhor poder de processamento podem se beneficiar de uma ou mais camadas de convolução, deixando a rede mais eficaz.

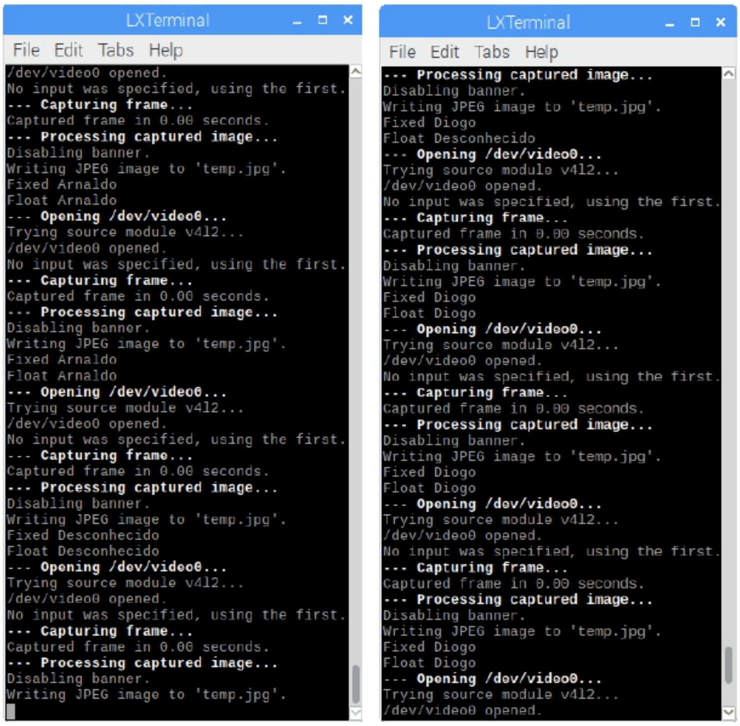
5.1 COMPORTAMENTO DAS REDES TREINADAS E CONVERTIDAS

As figuras 14a e 14b mostram a execução do script descrito no capítulo anterior quando a rede tentava detectar os autores. É possível notar que, exceto por um único caso, em todas as vezes a rede convertida para pfo teve um comportamento idêntico ao da rede convertida para pf.

As tabelas 4 e 5 mostram os resultados da execução da rede em ponto flutuante, quando é submetido a ela o conjunto de testes. O conjunto é compreendido por 68 fotos: 38 “desconhecido”, com nomes de arquivo começando com números entre 51 e 88; 26 “arnaldo”, com nome de arquivo começando com a letra “a”; 4 “diogo”, com nome de arquivo começando com a letra “d”.

Figura 14 – Rede tentando identificar os autores

(a) Arnaldo (b) Diogo



Fonte: Os autores (2017)

Pode-se notar a grande segurança da rede ao afirmar que um indivíduo é um “desconhecido”, pelos valores associados àquela saída serem muito próximos de 1. O mesmo pode-se dizer dos resultados marcados com o rótulo “arnaldo”. Contudo, mesmo a rede igualmente fornecendo valores próximos a 1 para as imagens “d*.png”, vemos que a rede ainda tem problemas em identificar imagens com baixa qualidade[1].

As tabelas 6 e 7 mostram os resultados do mesmo experimento com a mesma rede, convertida para ponto fixo. Pode-se ver que os resultados são razoavelmente consistentes com os resultados anteriores, i.e., a classificação feita por esta rede é a mesma que a feita pela rede anterior, exceto em um caso. Os valores mínimos e máximos associados às saídas, no caso desta rede, variam entre -512 e 512.

[1] Por imagens com baixa qualidade, entende-se aquelas que não apresentam níveis de luminosidade diferentes o suficiente (depois de feita a normalização segundo 4.2.2.2) para pontos adjacentes que representam áreas diferentes de uma face, e.g., lábio e pele ao redor da boca. Maiores informações podem ser vistas em 5.8

Tabela 4 – Resultados de uma rede em ponto flutuante

Arquivo	Valor associado à saída selecionada	Resultado esperado	Resultado obtido
51-11.png	0,991718	desconhecido	desconhecido
52-11.png	0,998819	desconhecido	desconhecido
53-11.png	0,998819	desconhecido	desconhecido
54-11.png	0,998529	desconhecido	desconhecido
55-11.png	0,998754	desconhecido	desconhecido
56-11.png	0,998791	desconhecido	desconhecido
57-11.png	0,998907	desconhecido	desconhecido
58-11.png	0,999137	desconhecido	desconhecido
59-11.png	0,998819	desconhecido	desconhecido
60-11.png	0,998133	desconhecido	desconhecido
61-11.png	0,998819	desconhecido	desconhecido
62-11.png	0,999124	desconhecido	desconhecido
63-11.png	0,998813	desconhecido	desconhecido
64-11.png	0,998819	desconhecido	desconhecido
65-11.png	0,998819	desconhecido	desconhecido
66-11.png	0,998819	desconhecido	desconhecido
67-11.png	0,999138	desconhecido	desconhecido
68-11.png	0,998631	desconhecido	desconhecido
69-11.png	0,995805	desconhecido	desconhecido
70-11.png	0,998059	desconhecido	desconhecido
71-11.png	0,998639	desconhecido	desconhecido
72-11.png	0,998100	desconhecido	desconhecido
73-11.png	0,997269	desconhecido	desconhecido
74-11.png	0,998648	desconhecido	desconhecido
75-11.png	0,998819	desconhecido	desconhecido
76-11.png	0,999082	desconhecido	desconhecido
77-11.png	0,998819	desconhecido	desconhecido
78-11.png	0,999053	desconhecido	desconhecido
79-11.png	0,998788	desconhecido	desconhecido
80-11.png	0,998796	desconhecido	desconhecido
81-11.png	0,998744	desconhecido	desconhecido
82-11.png	0,998819	desconhecido	desconhecido
83-11.png	0,998819	desconhecido	desconhecido
84-11.png	0,996728	desconhecido	desconhecido
85-11.png	0,998924	desconhecido	desconhecido
86-11.png	0,999155	desconhecido	desconhecido
87-11.png	0,999087	desconhecido	desconhecido
88-11.png	0,999086	desconhecido	desconhecido

Fonte: Os autores (2017)

Tabela 5 – Resultados de uma rede em ponto flutuante (cont.)

Arquivo	Valor associado à saída selecionada	Resultado esperado	Resultado obtido
a-a00.png	1,000000	arnaldo	arnaldo
a-a01.png	1,000000	arnaldo	arnaldo
a-a02.png	0,999973	arnaldo	arnaldo
a-a03.png	1,000000	arnaldo	arnaldo
a-a04.png	0,999967	arnaldo	arnaldo
a-a05.png	0,999997	arnaldo	arnaldo
a-a06.png	0,900839	arnaldo	arnaldo
a-a07.png	1,000000	arnaldo	arnaldo
a-a08.png	1,000000	arnaldo	arnaldo
a-a09.png	1,000000	arnaldo	arnaldo
a-a10.png	0,999999	arnaldo	arnaldo
a-a11.png	0,997995	arnaldo	arnaldo
a-a12.png	0,998954	arnaldo	arnaldo
a-a13.png	1,000000	arnaldo	arnaldo
a-a14.png	1,000000	arnaldo	arnaldo
a-a15.png	1,000000	arnaldo	arnaldo
a-a16.png	1,000000	arnaldo	arnaldo
a-a17.png	1,000000	arnaldo	arnaldo
a-a18.png	1,000000	arnaldo	arnaldo
a-a19.png	1,000000	arnaldo	arnaldo
a-a20.png	0,999998	arnaldo	arnaldo
a-a21.png	1,000000	arnaldo	arnaldo
a-a22.png	1,000000	arnaldo	arnaldo
a-a23.png	1,000000	arnaldo	arnaldo
a-a24.png	0,999997	arnaldo	arnaldo
a-a25.png	1,000000	arnaldo	arnaldo
d-a00.png	1,000000	diogo	arnaldo
d-a01.png	0,997985	diogo	diogo
d-a02.png	1,000000	diogo	diogo
d-a03.png	0,960210	diogo	desconhecido

Fonte: Os autores (2017)

Tabela 6 – Resultados de uma rede em ponto fixo

Arquivo	Valor associado à saída selecionada	Resultado esperado	Resultado obtido
51-11.png	512	desconhecido	desconhecido
52-11.png	512	desconhecido	desconhecido
53-11.png	512	desconhecido	desconhecido
54-11.png	512	desconhecido	desconhecido
55-11.png	512	desconhecido	desconhecido
56-11.png	512	desconhecido	desconhecido
57-11.png	512	desconhecido	desconhecido
58-11.png	512	desconhecido	desconhecido
59-11.png	512	desconhecido	desconhecido
60-11.png	512	desconhecido	desconhecido
61-11.png	512	desconhecido	desconhecido
62-11.png	512	desconhecido	desconhecido
63-11.png	512	desconhecido	desconhecido
64-11.png	512	desconhecido	desconhecido
65-11.png	512	desconhecido	desconhecido
66-11.png	512	desconhecido	desconhecido
67-11.png	512	desconhecido	desconhecido
68-11.png	512	desconhecido	desconhecido
69-11.png	512	desconhecido	desconhecido
70-11.png	512	desconhecido	desconhecido
71-11.png	512	desconhecido	desconhecido
72-11.png	512	desconhecido	desconhecido
73-11.png	512	desconhecido	desconhecido
74-11.png	512	desconhecido	desconhecido
75-11.png	512	desconhecido	desconhecido
76-11.png	512	desconhecido	desconhecido
77-11.png	512	desconhecido	desconhecido
78-11.png	512	desconhecido	desconhecido
79-11.png	512	desconhecido	desconhecido
80-11.png	512	desconhecido	desconhecido
81-11.png	512	desconhecido	desconhecido
82-11.png	512	desconhecido	desconhecido
83-11.png	512	desconhecido	desconhecido
84-11.png	512	desconhecido	desconhecido
85-11.png	512	desconhecido	desconhecido
86-11.png	512	desconhecido	desconhecido
87-11.png	512	desconhecido	desconhecido
88-11.png	512	desconhecido	desconhecido

Fonte: Os autores (2017)

Tabela 7 – Resultados de uma rede em ponto fixo (cont.)

Arquivo	Valor associado à saída selecionada	Resultado esperado	Resultado obtido
a-a00.png	512	arnaldo	arnaldo
a-a01.png	512	arnaldo	arnaldo
a-a02.png	512	arnaldo	arnaldo
a-a03.png	512	arnaldo	arnaldo
a-a04.png	512	arnaldo	arnaldo
a-a05.png	512	arnaldo	arnaldo
a-a06.png	512	arnaldo	arnaldo
a-a07.png	512	arnaldo	arnaldo
a-a08.png	512	arnaldo	arnaldo
a-a09.png	512	arnaldo	arnaldo
a-a10.png	512	arnaldo	arnaldo
a-a11.png	512	arnaldo	arnaldo
a-a12.png	512	arnaldo	arnaldo
a-a13.png	512	arnaldo	arnaldo
a-a14.png	512	arnaldo	arnaldo
a-a15.png	512	arnaldo	arnaldo
a-a16.png	512	arnaldo	arnaldo
a-a17.png	512	arnaldo	arnaldo
a-a18.png	512	arnaldo	arnaldo
a-a19.png	512	arnaldo	arnaldo
a-a20.png	512	arnaldo	arnaldo
a-a21.png	512	arnaldo	arnaldo
a-a22.png	512	arnaldo	arnaldo
a-a23.png	512	arnaldo	arnaldo
a-a24.png	512	arnaldo	arnaldo
a-a25.png	512	arnaldo	arnaldo
d-a00.png	512	diogo	arnaldo
d-a01.png	485	diogo	arnaldo
d-a02.png	512	diogo	diogo
d-a03.png	486	diogo	desconhecido

Fonte: Os autores (2017)

Figura 15 – Exemplos de imagens de teste submetidas à rede.

(a) a-a25.png (b) 77-11.png

Fonte: Os autores (2017) e Thomaz (2012)

Note que no exemplo da figura 15 e da tabela 8, a rede teve bastante confiança ao afirmar que a imagem da figura 15b se tratava de um "desconhecido": valor próximo do máximo na coluna desconhecido e próximo do mínimo em todos os outros casos. Por outro lado, apesar dela ter atribuído valor máximo ao rótulo "arnaldo" na imagem da figura 15a, ela não estava bastante segura de que não se tratava de um "desconhecido", pois o valor, apesar de negativo, não estava tão próximo do mínimo nesta coluna.

Tabela 8 – Resultados apresentados pela rede ao se submeter as imagens da figura 15 à RN (valores em ponto flutuante arredondados até a terceira casa decimal)

	Desc.		**Arnaldo**		**Diogo**		**João**		**Sel. pela**
Img.	**pf**	**pfo**	**pf**	**pfo**	**pf**	**pfo**	**pf**	**pfo**	**Rede**
77-11.png	0.999	512	-1.000	-512	-1.000	-512	-0.998	-512	Desc.
a-a25.png	-0.047	-125	1.000	512	-1.000	-512	-0.999	-512	Arnaldo

Fonte: Os autores (2017)

Vale ressaltar ainda que nestes exemplos, a rede em ponto fixo deu como saída os valores compatíveis com os da rede em ponto flutuante.

5.2 CÁLCULO DO CUSTO, EM MEMÓRIA VOLÁTIL, DA REDE

Foi criado neste trabalho um modelo para determinar o consumo de memória, dados parâmetros como o tamanho da entrada e estrutura interna da rede. A forma está descrita a seguir, para o cálculo de consumo de RAM da RN:

$$cmRN = 4 * [ent + \Sigma(cant + 1) * csub)];$$

Onde: $cmRN$ = consumo de memória da RN, ent = Entrada, $cant$ = camada anterior e $csub$ = camada subsequente.

A constante 4 é o número de bytes em memória, por peso, para uma rede em ponto fixo.

No caso específico da rede criada neste trabalho, o cálculo é:

$$4 * [\text{(altura img)} * (\text{(largura img)} + 1) + (128 + 1) * 16 + (16 + 1) * 8 + (8 + 1) * 4].$$

Usando-se a fórmula anterior para imagens de 48x48px, temos:

$$cmRN = 4 * [(48 * 49) + (129 * 16) + (17 * 8) + (9 * 4)]$$
$$cmRN = 18352\text{B ou } \textbf{\textit{17,921875KB}}$$

Ou seja, para a rede rodar são necessários 17,921875 kB de memória para imagens de 48x48px.

Neste caso foi ***desconsiderado*** o espaço de memória ocupado pelo programa para normalizar as imagens e os programas auxiliares de captura, localização e corte da face.

Dessa forma, pode afirmar-se que são necessários ***32KB*** de RAM para rodar essa RN com tranquilidade ao processar imagens de 48x48px.

Considerando que o objetivo inicial deste trabalho seria determinar a viabilidade do uso deste tipo de rede em embarcados e que grande parte desses dispositivos não contam com capacidade suficiente de memória RAM para conter esta rede, podemos afirmar que, apesar disso, ainda há uma boa gama de opções de sistemas embarcados (também conforme imagens posteriores) capazes de utilizar esse tipo de sistema. A exceção se dá na forma de dispositivos mais robustos. Vale salientar também que, mesmo dispositivos com quantidade de memória suficiente podem não apresentar meios viáveis de captura de imagem, inviabilizando desta forma também este processo.

5.3 TEMPOS MEDIDOS DURANTE A EXECUÇÃO DA REDE

Outro fator bastante surpreendente foi o fato de que, nos testes, a rede em ponto fixo executou as classificações em cerca de metade do tempo que a rede em ponto flutuante levava para realizar a mesma tarefa, como pode ser visto na tabela 9.

A tabela 9 sintetiza os tempos medidos durante a execução de testes da rede neural sobre um grupo de testes com 77 indivíduos. Vê-se claramente a influência do poder de processamento do equipamento nos tempos de execução da rede. Além disso, há uma diferença notável entre os tempos de execução das redes em ponto fixo quando comparadas ao ponto flutuante (em média, 0,10s em pfo contra 0,23s

Tabela 9 – Tempo de execução para o conjunto de testes (77 indivíduos) - máquina de treinamento e no embarcado

	PC		***Raspberry***	
	Ponto fixo	**Ponto flutuante**	**Ponto fixo**	**Ponto flutuante**
Real	4,214s	9,343s	71,193s	161,135s
User	**7,412s**	**17,476s**	**133,000s**	**310,600s**
Sys	0,232s	0,308s	2,080s	3,720s

Fonte: Os autores (2017)

Tabela 10 – Comparação dos tempos entre o PC e a *Raspberry*

	PC	***Raspberry***
Real	45,10%	44,18%
User	42,41%	42,82%

Fonte: Os autores (2017)

em pf no PC e 1,73s em pfo contra 4,03s em pf na *Raspberry*), o que demonstra o ganho de desempenho ao utilizar-se o ponto do fixo, favorecendo a sua utilização em equipamentos com baixo poder de processamento. Em termos de comparação, pode-se ver, conforme a tabela 10, que a razão entre os tempos de execução em ponto fixo e ponto flutuante ($\frac{pfo}{pf}$) se mantém muito próximos ao comparar as porcentagens dos tempos relativos do PC e da *Raspberry* (diferença em torno de 1%).

5.4 TEMPOS MEDIDOS NO RECONHECIMENTO EM TEMPO REAL

Visando-se determinar a eficiência do sistema proposto, também foram medidos os tempos de execução do mesmo como um todo, i.e., as imagens sendo capturadas pela câmera e enviadas à rede neural para a identificação das faces. A tabela 11 apresenta os tempos para apenas um quadro (imagem) processado.

A figura 14 mostra a execução do sistema para vários quadros. O tempo médio não foi muito diferente daquele apresentado para um quadro único.

Note que a eficiência observada em 5.3 não foi observada na execução em tempo real. Especulações sobre o porquê podem ser vistas em 5.7.

Tabela 11 – Reconhecimento em tempo real - Tempo de execução

	PC		***Raspberry***	
	Ponto fixo	**Ponto flutuante**	**Ponto fixo**	**Ponto flutuante**
Real	0,594s	2,758s	4,901s	6,048s
User	**0,332s**	**0,484s**	**4,920s**	**7,100s**
Sys	0,620s	0,624s	0,270s	0,360s

Fonte: Os autores (2017)

5.5 EFICÁCIA

Tendo em vista que eficácia é a capacidade de uma tarefa ser realizada de forma correta em uma medida de relação entre efeito e ação, pode se dizer que a mesma é verificável como uma afirmação a qual pode ser atribuindo um valor lógico de verdadeiro ou falso, como uma pergunta cuja resposta é do tipo “sim” ou “não”, por exemplo “a aplicação ou sistema realiza o que se propõe?”

Desta forma, tem-se as afirmações:

- *converter de ponto flutuante para ponto fixo não acarreta em perda significativa de qualidade*, i.e., a conversão é consistente. Verdade, pois, a rede apresentou índice de divergência em torno de apenas 1%;
- *o comportamento apresentado pela rede neural é mantido quando a mesma é convertida do PC para a Raspberry*. Verdade, pois, o comportamento apresentado varia também em aproximadamente 1% conforme pode ser comprovado ao observar a tabela 10.

5.6 EFICIÊNCIA

“É a habilidade (frequentemente mensurável) de evitar o desperdício de materiais, energia, esforços, dinheiro e tempo na execução de alguma coisa ou ao produzir um resultado desejado” (EFFICIENCY, 2018)

Tendo a definição anterior em vista e os tempos medidos em 5.3, percebe-se que a conversão de uma rede neural de ponto flutuante para ponto fixo aumenta a eficiência do sistema, já que os o tempo de processamento gasto em pfo é cerca de 50% do tempo gasto em pf. Além disso, pelo exposto em 5.5, o ganho em eficiência não se deu em detrimento da eficácia.

5.7 GARGALOS

“Em engenharia, gargalo se refere a um fenômeno onde a performance ou a capacidade de um sistema inteiro é limitado por um único ou por um pequeno número de componentes ou recursos.” (BOTTLENECK, 2018)

Mais particularmente, em computação é

“... um componente de *software* que afeta severamente a performance da aplicação.” (BOTTLENECK, 2018)

Apesar de, nos testes com imagens previamente tratadas, as RNs em pfo terem realizado o processamento em cerca de metade do tempo que aquelas em pf, tal

ganho em termos de performance quase não pôde ser sentido nos testes com câmera em tempo real. Dois fatores influenciaram para tanto:

- **Processos auxiliares lentos** entre eles, destaca-se a captura de imagem em si, que na câmera usada chegava a durar alguns segundos em alguns casos. Os processos de localização de faces na imagem e de manipulação da mesma para que as faces tivessem um tamanho padronizado também inseriam tempo de processamento extra.
- **Comunicação entre normalizador e reconhecedor ineficiente** o normalizador escreve os valores da imagem normalizada diretamente no console e o reconhecedor lê seus valores de entrada também do console. É feito então o redirecionamento da saída do primeiro para a entrada do segundo via *pipes* do *Linux*. Este método é bastante ineficiente por envolver a conversão dos dados numéricos de seu formato nativo para código ASCII correspondente e de volta para formato numérico nativo.

5.8 INFLUÊNCIA DA QUALIDADE DE IMAGEM

Foi constatado também que a qualidade das imagens de treinamento afetam bastante o desempenho da rede. Indivíduos cujas fotos tinham alto contraste durante esta fase, tiveram um índice bem maior de identificações positivas do que aqueles cujas fotos não possuíam um nível de contraste adequado.

De modo semelhante, durante a fase de testes, tanto quando os testes foram executados em um conjunto de fotos pré-selecionadas, quanto quando os testes foram realizados em tempo real, por meio de uma *webcam*, foi constatado que condições de baixa iluminação, ou cuja iluminação gere muitas sobras na face, ou ainda efeitos de *lens flare* na câmera, afetam negativamente a capacidade da rede de identificar positivamente os indivíduos.

Ainda assim, na RN usada neste projeto foi obtido um grande número de acertos quando comparado ao número de erros. Vale salientar ainda que, nenhuma imagem de treinamento foi utilizada durante os testes.

5.9 FERRAMENTAS GERADAS

Para execução deste trabalho foram geradas algumas ferramentas necessárias à realização do experimento como um todo, sendo elas:

- **Treinador da rede –** Classe Java que treina uma RN para reconhecer imagens 48x48px em tons de cinza;

- **Conversor *Encog*/FANN –** Classe Java que permite converter RNs gravadas no formato da biblioteca *Encog* para a biblioteca FANN;
- **Normalizador –** Programa de linha de comando (escrito em linguagem C) que lê uma imagem PNG e a normaliza para o intervalo numérico esperado como entrada da rede. Existe nas versões ponto fixo e ponto flutuante;
- **Reconhecedor –** Programa em linha de comando (escrito em linguagem C) que recebe como entrada uma RN convertida e uma imagem normalizada e realiza o reconhecimento. Existe nas versões ponto fixo e ponto flutuante;
- **caraCrachá –** *Script Bash* para capturar imagens, localizar faces, normalizar e reconhecer faces.

5.10 REQUISITOS PARA A EXECUÇÃO DO SISTEMA CARACRACHÁ

Os requisitos mínimos recomendados para a execução do sistema compreendem:

- CPU de 32bits (nativa)
- Memoria RAM 32KB
- Memória de programa 28KB para o nosso programa (desconsiderando o espaço necessário para os programas auxiliares - e.g., captura)
- Uma forma do embarcado capturar imagens (e.g., *webcam*)
- Um programa auxiliar para captura de imagens
- Um programa auxiliar para localizar faces
- Um programa auxiliar para cortar e redimensionar as imagens das faces

6 CONSIDERAÇÕES FINAIS

Podemos verificar e comprovar a viabilidade da criação e treinamento de uma rede neural em ponto flutuante para ponto fixo em um PC e portada para um sistema embarcado, nesse caso uma *Raspberry Pi 3B*.

Além disso foi possível perceber também a viabilidade de uma rede neural de reconhecimento facial de qualidade aceitável sem necessitar de uma rede profunda ou mesmo de uma camada de convolução.

Apesar das considerações das condições de captura de imagem em relação à qualidade da mesma e da luminosidade, a nossa rede após treinada, provou-se capaz de cumprir o objetivo de: dada uma determinada pessoa em frente à câmera, desde que em posição frontal, com poucas sombras e boa iluminação, é possível identificar se essa pessoa é ela mesma ou uma desconhecida, com baixa taxa de erros, seja em ponto flutuante ou ponto fixo, seja rodando em um PC ou em um sistema embarcado.

Considerando também que foi constatada que a conversão da RN de ponto flutuante para ponto fixo, não apresenta perda significativa em relação ao grupo de testes apresentado (1 erro em 74 – conversão eficaz) e que o ganho em tempo de processamento da RN em ponto fixo é de mais de 50% em relação a RN em ponto flutuante (conversão eficiente), conclui-se ser uma técnica muito promissora onde vale a pena a investigação já que no nosso estudo houve apenas a perda de 1% de precisão em relação ao ganho de desempenho neste tipo de sistema. Também é interessante notar que não houve divergência nos resultados individuais e comportamento apresentados pelas redes na transição do PC para a *Raspberry Pi*.

Apesar de comprovada a viabilidade da utilização de uma rede convertida e portada em ambiente embarcado, salienta-se que este trabalho pode ser melhorado de diversas maneiras. Por exemplo: utilizando-se de uma RN melhor treinada e construída; com testes em sistemas embarcados mais robustos; otimizando o processamento/tratamento prévio para captura de imagens; melhorando o sistema de comunicação entre o normalizador e o reconhecedor; avaliando o desempenho da RN com utilização de inteiros de 2 *bytes* em vez de inteiros de 4 *bytes*.

Além disso, um ponto interessante de melhoria desse trabalho é o desenvolvimento de uma estrutura para aplicação da tecnologia de *Yolo Object Detection* para melhorar e otimizar a localização e tratamento da face em frente à câmera, assim obtendo melhores imagens para avaliação de “pessoa desconhecida” ou “conhecida “, além de reduzir o uso de programas auxiliares. *YOLO* é uma abordagem onde a figura é analisada apenas uma única vez. Também seria interessante verificar se o

Yolo Object Detection também funciona satisfatoriamente em ponto fixo. Recomenda-se consultar Redmon e Farhadi (2018) para informações mais detalhadas sobre este assunto.

REFERÊNCIAS

AMD64 architecture programmer's manual volume 1: application programming. AMD64 Technology, 2017. Disponível em: <https://support.amd.com/TechDocs/24592.pdf>. Acesso em: 01 jul. 2018.

BELLARD, Fabrice. **FFmpeg**. 2017. Disponível em: <http://ffmpeg.org/>. Acesso em: 30 dez. 2017.

BOTTLENECK. 2018. Wiki. Disponível em: <https://en.wikipedia.org/wiki/Bottleneck>. Acesso em: 04 jul. 2018.

D'ELIA, Yuri. **facedetect**: a simple face detector for batch processing. 2016. Disponível em: <https://www.thregr.org/~wavexx/software/facedetect/>. Acesso em: 31 dez. 2017.

EFFICIENCY. 2018. Wiki. Disponível em: <https://en.wikipedia.org/wiki/Efficiency>. Acesso em: 10 jul. 2018.

FACURE, Matheus. **Funções de ativação**: entendendo a importância da ativação correta nas redes neurais. 2017. Disponível em: <https://matheusfacure.github.io/2017/07/12/activ-func/>. Acesso em: 29 dez. 2017.

GUPTA, Tushar. **Deep learning**: feedforward neural network. 2015. Livro digital. Disponível em: <https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7>. Acesso em: 30 dez. 2017.

GUPTA, Vikas. **Understanding feedforward neural networks**. 2017. Disponível em: <https://www.learnopencv.com/understanding-feedforward-neural-networks/>. Acesso em: 30 dez. 2017.

HAROLD, Elliotte Rusty. **Java's primitive data types**. 1998. Livro digital. Disponível em: <http://www.cafeaulait.org/course/week2/02.html>. Acesso em: 01 jul. 2018.

HEATON, Jeff. **Encog machine learning framework**. Heaton Research, Inc., 2017. Disponível em: <https://www.heatonresearch.com/encog/>. Acesso em: 30 dez. 2017.

HERON, Philip. **fswebcam**. sanslogic, 2017. Disponível em: <https://github.com/fsphil/fswebcam>. Acesso em: 30 dez. 2017.

IMAGEMAGICK. 2017. Disponível em: <http://imagemagick.org/script/index.php>. Acesso em: 30 dez. 2017.

JAVA. Oracle Corporation, 2017. Disponível em: <https://www.oracle.com/technetwork/pt/java/javase/overview/index.html>. Acesso em: 30 dez. 2017.

LINUX Mint 18 "Sarah". Linux Mark Institute, 2006. Disponível em: <https://linuxmint.com/>. Acesso em: 30 dez. 2017.

MAVEN: version 3.0.5. The Apache Software Foundation, 2013. Wiki. Disponível em: <https://maven.apache.org/>. Acesso em: 07 ago. 2017.

MOTION. Motion-Project, 2017. Disponível em: <https://motion-project.github.io/>. Acesso em: 30 dez. 2017.

NETBEANS IDE. Oracle Corporation, 2017. Disponível em: <https://netbeans.org/>. Acesso em: 30 dez. 2017.

NISSEN, Steffen et al. **FANN**. 2017. Disponível em: <https://github.com/libfann/fann>. Acesso em: 30 dez. 2017.

OPENJDK. Oracle America, Inc., 2017. Disponível em: <https://openjdk.java.net/>. Acesso em: 30 dez. 2017.

RASPBIAN 9. Raspberry Pi Foundation, 2017. Disponível em: <https://www.raspberrypi.org/downloads/raspbian/>. Acesso em: 30 dez. 2017.

REDMON, Joseph; FARHADI, Ali. **YOLOv3**: An incremental improvement. arXiv, 2018. Disponível em: <https://pjreddie.com/darknet/yolo/>. Acesso em: 06 nov. 2018.

SISTEMA Embarcado: O que é? qual sua importância? 2013. Disponível em: <https://www.embarcados.com.br/sistema-embarcado/>. Acesso em: 09 jul. 2018.

THOMAZ, Carlos E. **Fei face database**. 2012. Disponível em: <http://fei.edu.br/~cet/facedatabase.html>. Acesso em: 11 nov. 2017.

APÊNDICE A – LISTA DE SOFTWARES USADOS

Para a confecção deste trabalho de conclusão de curso, foram utilizados diversos softwares de apoio. Neste capítulo são listados os mesmos, bem como uma pequena descrição dos mesmos.

- **Encog** - É um *framework* em Java e C# para aprendizado de máquina que utiliza a licença Apache. Em 30 de dezembro de 2017 podia ser encontrado em https://www.heatonresearch.com/encog/
- **FANN** - É uma biblioteca de código aberto para redes neurais escrita em C. Em 30 de dezembro de 2017 podia ser encontrado em https://github.com/libfann/fann
- ***Raspbian*** - É um sistema operacional de código aberto baseado no *Debian* para a *Raspberry Pi*. Em 30 de dezembro de 2017 podia ser encontrado em https://www.raspberrypi.org/downloads/raspbian/
- ***facedetect*** - É um programa simples, distribuído sob licença GPLv2+, baseado no *OpenCV*, para detecção de faces humanas em imagens. Em 30 de dezembro de 2017 podia ser encontrado em https://www.thregr.org/~wavexx/software/facedetect/
- **Java** - É uma linguagem de programação de propósito geral orientada à objetos. Os programas, nesta linguagem, são compilados para código objeto que será executado em uma máquina virtual Java (JVM na sigla em inglês). Em 26 de julho de 2018 o *kit* de desenvolvimento Java (JDK na sigla em inglês) e o ambiente de execução Java (JRE na sigla em inglês) podiam ser encontrados em http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html
- ***ImageMagick*** - É um *software* livre distribuído sob licença Apache para manipulação de imagens em modo texto. Em 30 de dezembro de 2017 podia ser encontrado em https://www.imagemagick.org/script/index.php
- **Netbeans** - É um Ambiente Integrado de Desenvolvimento (IDE, na sigla em inglês). Em 30 de dezembro de 2017 podia ser encontrado em https://netbeans.org/
- ***fswebcam*** - Programa simples para captura de imagens de *webcams*. Em 30 de dezembro de 2017 podia ser encontrado em https://github.com/fsphil/fswebcam

- ***FFmpeg*** - Conversor e gravador de audio e vídeo distribuído sob licença LGPL. Em 30 de dezembro de 2017 podia ser encontrado em https://www.ffmpeg.org/
- ***Motion*** - Programa para monitorar o sinal de vídeo proveniente de diversos tipos de câmeras. Em 30 de dezembro de 2017 podia ser encontrado em https://motion-project.github.io/
- ***time*** - Ferramenta presente na maioria das distribuições *Linux* para medir o tempo de execução de um programa ou *script*. Retorna três informações de tempo: *real* (real), *user* (usuário) e *system* (sistema). O tempo "real", como o próprio nome implica, representa o tempo gasto pelo programa desde que este começou a ser executado, até o final de sua execução. O tempo de "usuário" é o tempo que a CPU gasta em modo-usuário (fora do *kernel*, ou núcleo do sistema). O tempo de "sistema" é o tempo que a CPU gasta executando chamadas ao *kernel* quando executa o programa. O manual da ferramenta *time* no *Linux* pode ser consultado para maiores informações através do comando `man time` no terminal.

APÊNDICE B – LISTA DE EQUIPAMENTOS USADOS

Neste capítulo são listados os equipamentos utilizados durante o desenvolvimento deste trabalho.

B.1 *RASBERRY PI 3 MODEL B*

É um computador simples em placa única usado para os testes de desempenho.

Figura 16 – *Rasberry PI 3 Model B*

Fonte: Os autores (2017)

B.2 CÂMERA MS-VX6000

Uma *webcam* USB de 5*megapixels* fabricada pela *Microsoft*.

Figura 17 – *Webcam MS-VX6000*

Fonte: Os autores (2017)

B.3 *ARDUINO UNO*

É uma placa de prototipação com *hardware* restrito.

Figura 18 – *Arduino UNO*

Fonte: Os autores (2017)

B.4 COMPUTADOR *LAPTOP*

PC onde foi desenvolvida a espinha dorsal do trabalho.

APÊNDICE C – LISTAGENS

C.1 – Trecho de código exemplificando como criar uma rede neural com as mesmas especificações da rede usada neste trabalho

```
1 this.network = new BasicNetwork();
2 this.network.addLayer(new BasicLayer(null, true, this.training.
      getInputSize()));
3 this.network.addLayer(new BasicLayer(new ActivationTANH(), true,
      128));
4 this.network.addLayer(new BasicLayer(new ActivationTANH(), true,
      16));
5 this.network.addLayer(new BasicLayer(new ActivationTANH(), true,
      8));
6 this.network.addLayer(new BasicLayer(new ActivationTANH(), false,
      this.training.getIdealSize()));
7 this.network.getStructure().finalizeStructure();
8 this.network.reset();
```

Fonte: Os autores (2017)

C.2 – caraCrachaFloat.sh

```
1 #!/bin/bash
2
3 while [true ]; do
4     fswebcam -r 640x480 --no-banner temp.jpg
5     facedetect --biggest temp.jpg | while read x y w h; do
6         convert temp.jpg -crop ${w}x${h}+${x}+${y} \\
7         -resize 48x48 temp.jpg
8     done
9     echo Float $(./normalizer facenn.net temp.png | \\
10        ./runNetwork facenn.net)
11    rm temp.png temp.jpg
12 done
```

Fonte: Os autores (2017)

C.3 – caraCrachaFixed.sh

```
1 #!/bin/bash
2
3 while [true ]; do
4     fswebcam -r 640x480 --no-banner temp.jpg
```

```
5     facedetect --biggest temp.jpg | while read x y w h; do
6         convert temp.jpg -crop ${w}x${h}+${x}+${y} \\
7         -resize 48x48 temp.jpg
8     done
9     echo Fixed $(./normalizer_fixed facenn_fixed.net \\
10        temp.png | ./runFixedNetwork facenn_fixed.net)
11    rm temp.png temp.jpg
12 done
```

Fonte: Os autores (2017)

C.4 – Conversor de rede no formato Encog para o formato FANN

```
1  public static String convert(BasicNetwork network)
2  {
3    StringBuilder result = new StringBuilder();
4    result.append("FANN_FLO_2.1\n");
5    result.append("num_layers=").append(network.getLayerCount()).
       append("\n");
6    result.append("learning_rate=0.700000\n");
7    result.append("connection_rate=1.000000\n");
8    result.append("network_type=0\n");
9    result.append("learning_momentum=0.000000\n");
10   result.append("training_algorithm=2\n");
11   result.append("train_error_function=1\n");
12   result.append("train_stop_function=1\n");
13   result.append("cascade_output_change_fraction=0.010000\n");
14   result.append("quickprop_decay=-0.000100\n");
15   result.append("quickprop_mu=1.750000\n");
16   result.append("rprop_increase_factor=1.200000\n");
17   result.append("rprop_decrease_factor=0.500000\n");
18   result.append("rprop_delta_min=0.000000\n");
19   result.append("rprop_delta_max=50.000000\n");
20   result.append("rprop_delta_zero=0.100000\n");
21   result.append("cascade_output_stagnation_epochs=12\n");
22   result.append("cascade_candidate_change_fraction=0.010000\n");
23   result.append("cascade_candidate_stagnation_epochs=12\n");
24   result.append("cascade_max_out_epochs=150\n");
25   result.append("cascade_min_out_epochs=50\n");
26   result.append("cascade_max_cand_epochs=150\n");
27   result.append("cascade_min_cand_epochs=50\n");
28   result.append("cascade_num_candidate_groups=2\n");
29   result.append("bit_fail_limit=9.99999977648258209229e-03\n");
30   result.append("cascade_candidate_limit=1.0000000000000000000e
       +03\n");
31   result.append("cascade_weight_multiplier=4.00000059604644775390e
       -01\n");
```

```
32  result.append("cascade_activation_functions_count=10\n");
33  result.append("cascade_activation_functions=3 5 7 8 10 11 14 15
       16 17 \n");
34  result.append("cascade_activation_steepnesses_count=4\n");
35  result.append("cascade_activation_steepnesses
       =2.50000000000000000000e-01 5.00000000000000000000e-01
       7.50000000000000000000e-01 1.00000000000000000000e+00 \n");
36  result.append("layer_sizes=");
37  for (int i = 0; i < network.getLayerCount(); ++i)
38  {
39    result.append(network.getLayerNeuronCount(i) + 1).append(" ");
40  }
41  result.append("\n");
42  result.append("scale_included=0\n");
43  result.append("neurons (num_inputs, activation_function,
       activation_steepness)=");
44  for (int i = 0; i < network.getInputCount() + 1; ++i)
45  {
46    result.append("(0, 0, 0.00000000000000000000e+00) ");
47  }
48  for (int layer = 1; layer < network.getLayerCount(); ++layer)
49  {
50    for (int neuron = 0; neuron < network.getLayerNeuronCount(
         layer); ++neuron)
51    {
52      result.append("(");
53      result.append(network.getLayerNeuronCount(layer - 1) + 1);
54      result.append(", ");
55      result.append(encogActivationTypeToFANN(network.
           getActivation(layer)));
56      result.append(", 1.00000000000000000000e+00) ");
57    }
58    result.append("(0, 5, 1.00000000000000000000e+00) ");
59  }
60  result.append("\n");
61  int layerShitf = 0;
62  result.append("connections (connected_to_neuron, weight)=");
63  for (int layer = 1; layer < network.getLayerCount(); ++layer)
64  {
65    for (int toIdx = 0; toIdx < network.getLayerNeuronCount(layer)
         ; ++toIdx)
66    {
67      for (int fromIdx = 0; fromIdx < network.getLayerNeuronCount(
           layer - 1) + 1; ++fromIdx)
68      {
69        result.append("(");
70        result.append(fromIdx + layerShitf);
```

```
71          result.append(", ");
72          result.append(toScientificNotation(network.getWeight(layer
                - 1, fromIdx, toIdx)));
73          result.append(") ");
74        }
75      }
76      layerShitf += network.getLayerNeuronCount(layer - 1) + 1;
77    }
78    result.append("\n");
79    return result.toString();
80 }
81
82 private static int encogActivationTypeToFANN(ActivationFunction
       activationType)
83 {
84    if(activationType instanceof ActivationElliott)
85      return FannActivationTypes.FANN_ELLIOT.ordinal();
86    if(activationType instanceof ActivationElliottSymmetric)
87      return FannActivationTypes.FANN_ELLIOT_SYMMETRIC.ordinal();
88    if(activationType instanceof ActivationLinear)
89      return FannActivationTypes.FANN_LINEAR.ordinal();
90    if(activationType instanceof ActivationSigmoid)
91      return FannActivationTypes.FANN_SIGMOID.ordinal();
92    if(activationType instanceof ActivationGaussian)
93      return FannActivationTypes.FANN_GAUSSIAN.ordinal();
94    if(activationType instanceof ActivationRamp)
95      return FannActivationTypes.FANN_THRESHOLD.ordinal();
96    if(activationType instanceof ActivationSIN)
97      return FannActivationTypes.FANN_SIN.ordinal();
98    if(activationType instanceof ActivationSteepenedSigmoid)
99      return FannActivationTypes.FANN_SIGMOID_STEPWISE.ordinal();
100   if(activationType instanceof ActivationBipolarSteepenedSigmoid)
101     return FannActivationTypes.FANN_SIGMOID_SYMMETRIC_STEPWISE.
          ordinal();
102   if(activationType instanceof ActivationTANH)
103     return FannActivationTypes.FANN_SIGMOID_SYMMETRIC.ordinal();
104
105   return FannActivationTypes.FANN_LINEAR.ordinal();
106 }
```

Fonte: Os autores (2017)

C.5 – Conversor de rede em ponto flutuante para rede em ponto fixo

```
1 struct fann *ann;
2 ann = fann_create_from_file(nome_do_arquivo);
3
```

```
4 if(!ann)
5 {
6   printUsage();
7   return (EXIT_FAILURE);
8 }
9
10 unsigned int multiplicador = fann_save_to_fixed(ann, argv[2]);
```

Fonte: Os autores (2017)

C.6 – Código fonte para *normalizer* e *normalizer_fixed*

```
1 void process_file(unsigned int multi) {
2   for (int y = 0; y < height; y++) {
3     png_byte* row = row_pointers[y];
4     for (int x = 0; x < width; x++) {
5       unsigned char value;
6       png_byte* ptr;
7       switch(png_get_color_type(png_ptr, info_ptr)) {
8         case PNG_COLOR_TYPE_GRAY:
9           ptr = &(row[x]);
10          value = ptr[0];
11          break;
12        case PNG_COLOR_TYPE_GRAY_ALPHA:
13          ptr = &(row[x*2]);
14          value = getGrayValue(ptr[0], ptr[1]);
15          break;
16        case PNG_COLOR_TYPE_RGB:
17          ptr = &(row[x*3]);
18          value = getGrayValueFromRGB(ptr[0], ptr[1], ptr[2]);
19          break;
20        case PNG_COLOR_TYPE_RGBA:
21          ptr = &(row[x*4]);
22          value = getGrayValueFromRGBA(ptr[0], ptr[1], ptr[2], ptr
                [3]);
23      }
24 #ifdef FIXEDFANN
25     printf("%d ", (int) (normalizer_data[value] * multi));
26 #else
27      printf("%lf ", normalizer_data[value]);
28 #endif
29     }
30   }
31 }
```

Fonte: Os autores (2017)

C.7 – Código fonte para *runNetwork* e *runFixedNetwork*

```
struct fann *ann;
ann = fann_create_from_file(argv[1]);
if(!ann)
{
  abort_("Error creating ann --- ABORTING.");
}
fann_type inputs[ann->num_input];

for(int i = 0; i < ann->num_input; ++i)
{
  if(
#ifdef FIXEDFANN
  scanf("%d", &inputs[i])
#else
  scanf("%f", &inputs[i])
#endif
  == 0)
  {
    abort_("Failed to read data");
  }
}
fann_type *outputs;
outputs = fann_run(ann, inputs);
fann_type classe = outputs[0];
int indice_classe = 0;
for(int i = 1; i < ann->num_output; ++i)
{
  if(outputs[i] > classe)
  {
    classe = outputs[i];
    indice_classe = i;
  }
}
```

Fonte: Os autores (2017)