Juan Alejandro Terenzi Cuttle

Utilização de redes convolucionais profundas para estimativa de ângulos de pose de fácies obtidas através do *Kinect*®

Florianópolis

2018

Juan Alejandro Terenzi Cuttle

Utilização de redes convolucionais profundas para estimativa de ângulos de pose de fácies obtidas através do *Kinect*®

Monografia submetida ao Programa de Graduação em Ciência da Computação para a obtenção do Grau de Bacharel.

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Ciência da Computação

Orientador: Prof. Dr. Mauro Roisenberg

Florianópolis

2018

Cuttle, Juan Alejandro Terenzi

Utilização de redes convolucionais profundas para estimativa de ângulos de pose de fácies obtidas através do *Kinect*® / Juan Alejandro Terenzi Cuttle; Orientador: Prof. Dr. Mauro Roisenberg– 2018.

106 p. : il. (algumas color.) ; 30 cm.

Trabalho de Conclusão de Curso de Graduação – Universidade Federal de Santa Catarina, Departamento de Informática e Estatística, Ciência da Computação, Florianópolis, 2018.

Inclui referências

1. Ciência da Computação. 2. Inteligência Artificial. 3. Deep Learning. I. Roisenberg, Mauro. II. Universidade Federal de Santa Catarina

Juan Alejandro Terenzi Cuttle

Utilização de redes convolucionais profundas para estimativa de ângulos de pose de fácies obtidas através do *Kinect*®

Esta Monografia foi julgada aprovada para a obtenção do Título de "Bacharel em Ciência da Computação", e aprovada em sua forma final pelo Programa de Graduação em Ciência da Computação.

Florianópolis, 22 de Novembro de 2018:

Prof. Dr. Rafael Luiz Cancian
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Mauro Roisenberg
Orientador

Prof. Dr. Elder Rizzon Santos
Universidade Federal de Santa Catarina

Prof. Dr. Ricardo Azambuja Silveira
Universidade Federal de Santa Catarina

Florianópolis
2018

"Ideias, e somente ideias, podem iluminar a escuridão." - Mises, Ludwig V.

Resumo

A capacidade de reconhecer padrões e construir modelos analíticos a partir de dados, é uma das principais características das técnicas de aprendizado de máquina.

O reconhecimento da posição da cabeça a partir de imagens obtidas de câmeras dotadas de sensores de profundidade permite uma série de interações entre seres humanos e computadores em uma variedade de aplicações que vão de jogos até o controle de cadeiras de rodas por cadeirantes. O uso deste tipo de equipamento permite que se obtenha dados de profundidade, tornando possível obter o ângulo de inclinação da cabeça nos três eixos, o que seria muito difícil apenas a partir de imagens convencionais.

Uma câmera relativamente barata, e capaz de capturar estas imagens de profundidade é o sensor *Kinect*®, desenvolvido pela empresa Microsoft. Este captura imagens de 640x480 pixels, uma resolução boa para distinguir a posição da cabeça numa imagem de profundidade.

As redes neurais convolucionais são uma arquitetura de redes neurais profundas (deep learning) que têm sido utilizadas com sucesso em uma série de problemas de visão computacional relacionados com a identificação e reconhecimento de objetos em imagens.

Este trabalho procurou então utilizar imagens capturadas por um sensor *Kinect*®, para treinar uma Rede Neural Convolucional de Camada Profunda e reconhecer nas imagens os ângulos de pose de cabeça de seres humanos. A Rede treinada foi então utilizada numa implementação simples onde o reconhecimento correto de imagens de profundidade demonstra aplicações práticas desta interface humano-computador, assim como seu potencial.

Palavras-chaves: Computação, *Kinect*®, Inteligência Artificial, Redes Neurais Artificiais, Visão Computacional, Reconhecimento de Ângulos de Pose de Cabeça.

Abstract

The capacity to recognise patterns and create analytical models from data, is one of the main characteristics of Machine Learning techniques.

The recognition of head positions from 3D (depth) images allows for a multitude of interactions between human beings and computers, in a variety of applications ranging from games to wheelchair control. The usage of devices containing depth sensors allows for the capturing of depth data, making the obtainment of head inclination angles on the three axes possible, which would be very difficult only through conventional images.

A relatively cheap camera, which can capture these 3D images is the *Kinect*® sensor, developed by Microsoft. This device captures images of 640x480 pixels, a good enough resolution to distinguish the head's position in a depth image.

Convolutional Neural Networks are an architecture of Deep Neural Networks (Deep Learning) which have been used with success in a plethora of computer vision problems, related to the identification and recognition of objects in images.

This work sought to utilize images captured by a *Kinect*®, to train a Deep Convolutional Neural Network and to recognise in images the angles in which human heads are posing. The trained Network was then used in a simple example software where the correct recognition of depth images demonstrates practical applications of this human-machine interface, as well as its potential.

Palavras-chaves: Computer Science, *Kinect*®, Artificial Intelligence, Convolutional Neural Networks, Computer Vision, Head Pose.

Sumário

1 Introdução

Desde a década de 1970, os computadores pessoais mudaram significativamente em questão de capacidade de processamento, armazenamento de dados, e dispositivos de entrada e saída. Porém, a comunicação do usuário com o programa aplicativo ou o computador em si manteve-se quase estática, com o mouse e o teclado servindo como as formas de controle disponíveis para manipular o computador em tempo real.

Alternativas a este paradigma são estudadas em uma área da Ciência da Computação denominada HCI (Human-Computer Interaction, ou Interação Humano-Computador). Entre estas alternativas, podemos salientar a voz e imagens (imagens do usuário e o ambiente atual deste).

A importância de se estudar e aplicar formas inusuais de interação humano-computador deve-se levar em conta alguns aspectos. Entre eles destaco a praticidade [como controlar casas inteligentes com a voz (Portet et al. (2013))] e a flexibilidade, como permitir que pessoas com movimento reduzido ainda se divirtam com jogos digitais (SPECIALEFFECT).

O foco deste trabalho se dá nas posições de cabeça, as quais são acessíveis para a grande maioria das pessoas, e podem ser facilmente identificadas, estando o usuário sentado ou de pé. Além disso, as diferentes posições são capazes de uma variedade de comandos de software, tornando esta opção muito generalizável para aplicações futuras.

Posições de cabeça são geralmente definidas como uma comunicação não-verbal, uma categoria de expressão a qual além de cabeça, engloba padrões de movimentos das mãos, braços, torso e olhos. No caso particular da cabeça, temos 3 (três) eixos onde ela pode se mover, estes são: vertical, transversal e longitudinal (face voltada para um lado e para outro, face voltada para cima e para baixo, e cabeça inclinada para um lado e para outro, respectivamente).

Trabalhos recentes de capturas de movimentos faciais foram geralmente feitos com imagens RGB extraídas de vídeo (Murphy-Chutorian e Trivedi (2009)). Porém, o uso destas imagens 2D têm apresentado dificuldades de precisão, um dos motivos sendo a ausência de textura em algumas regiões faciais. A principal alternativa é a de imagens de profundidade, a qual têm apresentado bons resultados, confirmando sua validade (Weise, Leibe e Gool (2007)).

Sensores de profundidade baratos ainda são poucos, porém, existem algumas opções, como o $AsusXTion^{®}$ e $MicrosoftKinect^{®}$. Para realizar este trabalho foi escolhido o sensor $Kinect^{®}$, devido à abundância de exemplos de uso dele em aplicações

computacionais diversas e à sua compatibilidade com o ambiente MATLAB, também utilizado neste trabalho.

Para detectar a posição da cabeça, o *Kinect*® captura uma imagem de profundidade, a qual é então submetida a um pré-processamento, e finalmente classificada. Os resultados poderão ser posteriormente utilizados em quaisquer aplicações, com o limite da variedade delas sendo a criatividade humana. Neste caso, foi utilizado um jogo como prova de conceito, porém, devido à natureza genérica da posição da cabeça, os softwares que podem utilizar esta interação podem ser os mais diversos.

As técnicas mais utilizadas para reconhecimento de pose de cabeça atualmente (visão computacional, como definidas por Al-Rahayfeh e Faezipour (2013)) são : Local Binary Pattern (uma detecção de mudança da posição facial, usando uma análise de textura Siriteerakul, Sato e Boonjing (2011)), baseado no Lucas-Kanade (detecta o rosto, detecta as narinas, e utiliza Lucas-Kanade para seguir a posição das narinas Zhao, Wang e Fu (2012)), utilizando redes neurais (backpropagation, Zhao e Yan (2011)) e Random forests (coleções de árvores de decisão, Fanelli et al. (2013)) .

O ambiente MATLAB oferece muitas vantagens. Em especial oferece interfaces simples para configuração, treinamento, avaliação e utilização de redes Deep Learning, através do pacote TOOLBOX. Além disto, ele possui suporte de hardware para o dispositivo *Kinect*®, o que permite uma integração completa desde a captura das imagens, processamento na linguagem MATLAB e utilização de redes neurais de camada profunda para classificá-las.

Este trabalho busca analisar posições dinâmicas (sequência de movimentos), utilizando uma biblioteca de redes neurais em MATLAB para fazer a classificação destas posições. Isto foi feito através da detecção em frames individuais, onde cada frame é submetido à rede, a saída é gerada, e esta é enviada como entrada ao programa aplicativo, o qual por sua vez, responde de acordo.

1.1 Objetivos

A proposta deste relatório é estudar análises prévias de redes neurais artificiais, aplicadas para reconhecimento de pose facial. Com uma introdução de seus fundamentos e uma breve apresentação aos principais conceitos existentes nesse ramo.

Após esta introdução, deseja-se treinar uma rede neural para reconhecer e classificar poses faciais capturadas através de um sensor *Kinect*®, usando redes neurais convolucionais.

Os objetivos são, portanto, divididos em:

1.1.1 Objetivo Geral

Implementar uma rede neural e treiná-la para reconhecer posições de cabeça, capturadas através de um sensor *Microsoft Kinect*®, usando redes neurais convolucionais.

1.1.2 Objetivos Específicos

- Levantamento da literatura buscando técnicas que já foram utilizadas para esta tarefa e similares.
- Obtenção de dados já disponíveis na comunidade acadêmica para este problema.
- Desenvolvimento de metodologia para aplicação dos dados obtidos para treinamento de redes convolucionais.
- Utilizar redes neurais de camada profunda para classificação de ângulos de fácies em imagens de profundidade.
- Realização de experimentos, utilizando capturas em tempo real para controlar um software simples, para avaliar a rede neural implementada.

1.2 Metodologia

A primeira etapa deste trabalho consistirá em encontrar bases de dados com imagens de profundidade (de rostos) já existentes. Feito isto, os dados serão analisados para encontrar maneiras de pré-processamento que facilitem o treinamento e precisão final da rede neural a ser construída.

Após esta fase inicial relacionada com os dados, será feita uma análise dos conceitos e das técnicas mais utilizadas na área de reconhecimento de posição facial e similares (focando em redes neurais), identificando aquelas que são mais comuns e/ou possuem maiores taxas de acerto. Para tal, serão buscados principalmente artigos de trabalhos no mesmo campo de estudo.

Finalmente, será implementado um software `MATLAB` que utilizará as técnicas determinadas como mais precisas para localizar o rosto, e através de uma rede neural configurada e treinada para tal, identificar o ângulo no qual a cabeça está virada.

A partir desta identificação, um software poderá ser desenvolvido, o qual utilizará as imagens capturadas pelo *Kinect*®, e o resultado da entrada destas imagens tratadas na rede, para gerar os comandos deste programa aplicativo.

Será então avaliada a efetividade da rede neural desenvolvida através do software exemplo. Neste caso, será avaliado a corretude dos comandos enviados ao jogo, e a facilidade de controlar o programa através das posições de cabeça.

1.3 Organização do Texto

No capítulo 1 são expostos alguns conceitos, motivações e metodologias escolhidas para a realização deste trabalho.

No capítulo 2 são explicados mais profundamente os diversos conceitos utilizados neste trabalho, relacionando-os com o problema em questão. Além disto será mostrado o estado da arte na área de reconhecimento de padrões.

No capítulo 3 é apresentada a proposta deste relatório. Aqui estão descritos os processos de decisão feitos neste trabalho. Também estão descritos o ambiente e algoritmos; a origem e tratamento dos dados; como foi escolhida e configurada a rede neural; e como a rede é utilizada em um programa de demonstração qualquer.

No capítulo 4 são detalhados os experimentos com o tratamento dos dados, configuração da rede, e análise dos resultados. E é descrito o processo de revisão e ajuste contínuo necessário para otimizar a qualidade dos resultados finais, de modo a demonstrar a possibilidade prática de utilizar a rede em outras situações.

Por fim no capítulo 5 são apresentadas as conclusões sobre este trabalho, seus resultados e seu impacto. São sugeridos trabalhos futuros e feitas as considerações finais.

2 Revisão Bibliográfica

2.1 Trabalhos correlatos

Os trabalhos em reconhecimento de posições de cabeça são muitos (Murphy-Chutorian e Trivedi (2009)). Como visto em Fanelli, Gall e Gool (2011), as técnicas envolvidas nesse reconhecimento podem ser divididas naquelas que utilizam imagens 2D e naquelas que utilizam imagens de profundidade. Dentro da categoria 2D podemos dividir os métodos em mais duas categorias: baseada em aparência, e baseada em características.

Estes métodos baseados em aparência utilizam o rosto como um todo para avaliar a posição da cabeça. São geralmente feitos discretizando as diferentes poses e aprendendo a detectar cada uma individualmente (Viola, Jones e Viola (2003)), ou usando modelos estatísticos da forma do rosto e da aparência para detectar a posição em tempo real (Cootes, Edwards e Taylor (2001)).

Já os métodos baseados em características utilizam certas partes da cabeça para detectá-la e determinar sua posição (exs.: nariz, olhos). Neste caso, dependem que estas partes apareçam em todas as imagens (ex.: Vatahska, Bennewitz e Behnke (2007)) ou que certas partes apareçam em certas poses (ex.: Yao e Cham (2004)).

Em geral, porém, métodos com imagens 2D apresentam problemas grandes com iluminação, falta de características e/ou oclusões. Precisam então ser encontradas alternativas.

Entre as alternativas mais promissoras, estão as abordagens que utilizam imagens de profundidade. Estas abordagens se popularizaram recentemente devido ao surgimento de equipamentos sensores de baixo custo, como o *Kinect*®. Este problema então se concentra em um reconhecimento de padrões corporais em imagens 3D (de profundidade) e diversos pesquisadores têm utlizado as imagens de profundidade (sozinhas ou aliadas às imagens 2D) para realizar o reconhecimento das posições de cabeça.

Zhao e Yan (2011) utilizaram redes neurais como uma técnica relativamente simples de implementação e que se mostrou capaz de obter excelentes resultados.

Fanelli et al. (2013) propuseram e desenvolveram *Random Forests* para alcançar o mesmo objetivo. Os autores descrevem como estas árvores são capazes de simplificar o problema maior em problemas menores, pois cada nodo da árvore é um preditor de algo simples na imagem. Combinados na forma descrita, eles são capazes de reconhecer padrões complexos, como fácies.

O termo *Forest* se refere a forma como estas árvores estão dispostas como uma

"floresta", onde cada árvore é treinada com um subconjunto aleatório dos dados de treinamento, dificultando o processo de *overfitting*. Este processo ocorre quando uma técnica de aprendizagem de máquina ajusta seus pesos demais, apenas para o conjunto de treinamento, tornando-se incapaz de reconhecer o padrão em dados novos.

2.2 Redes Neurais Artificais

Redes Neurais Artificiais são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência. Uma grande rede neural artificial pode ter centenas ou milhares de unidades de processamento; já o cérebro de um mamífero pode ter muitos bilhões de neurônios (Carvalho ()).

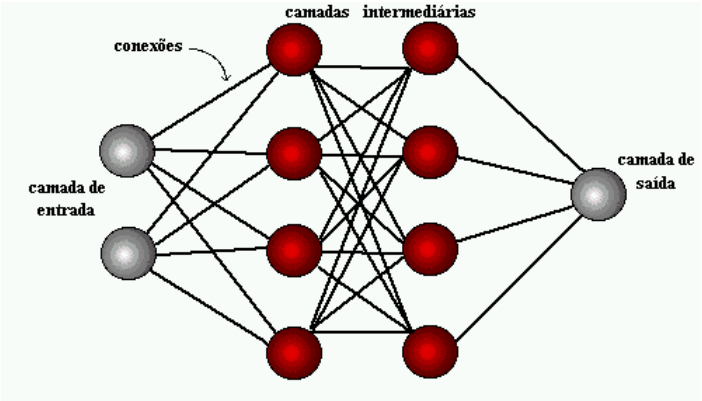


Figura 1 – Visão esquemática de uma rede neural genérica.
Fonte: Carvalho ()

Uma rede neural artificial é composta de 3 partes: Camada de entrada, onde o dado entra (exemplos: pixel, letra, som). Camadas intermediárias ou escondidas, onde os dados passam por funções diversas (como funções de ativação). Camada de saída, onde o resultado é disposto da forma desejada (um valor, uma categoria, 2 valores etc).

2.3 *Perceptrons*

O nodo (neurônio) de uma rede neural artificial é denominado *Perceptron*. Nas décadas de 1950 e 1960, Rosenblatt (1958) desenvolveu sua teoria com base nos trabalhos de Mcculloch e Pitts (1943) (apud Nielsen (2015)).

Na figura 2, a saída está descrita como *output*. O cálculo do *perceptron* é feito da seguinte maneira: $threshold = \sum_j w_j x_j$, onde w_j é o j-ésimo peso, aplicado na j-ésima entrada x_j e threshold é o valor com base no qual o perceptron gera a saída 0 ou 1. Ainda, a saída pode ser calculada com base nesse resultado sem sê-lo diretamente. Por exemplo (figura 2):

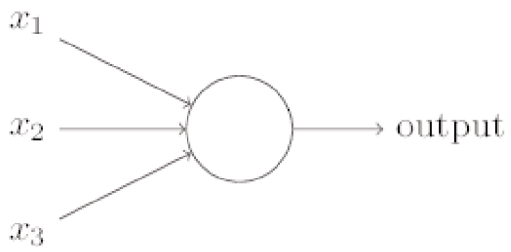


Figura 2 – Visão esquemática de um Perceptron clássico.
Fonte: Nielsen (2015)

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{2.1}$$

Esta função resulta no seguinte gráfico, chamado *Step* (figura 3):

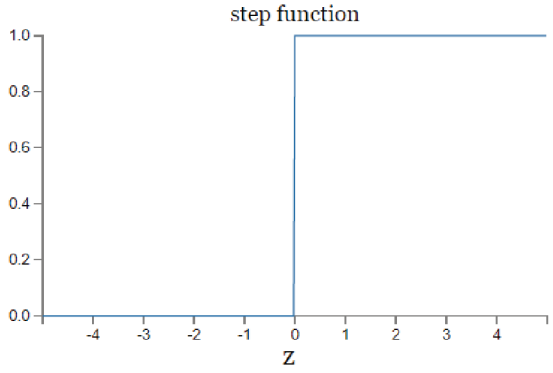


Figura 3 – Gráfico da função de um Perceptron clássico.
Fonte: Nielsen (2015)

2.3.1 *Sigmoid Neuron*

Suponha que você deseje classificar dígitos manuscritos. A diferença entre eles pode ser bem sutil, dada a variedade de formas de escrevê-los e às características da caligrafia de cada indivíduo. Ao aprender, uma rede de perceptrons precisará fazer ajustes pequenos em seus pesos, e ao fazer isto, ajustar apenas um pouco o resultado final da rede. Porém, isto não ocorre em uma rede de *perceptrons* convencional (Nielsen (2015)). Felizmente, há uma maneira de garantir que a rede não se comportará de forma errática em seu treinamento.

A solução jaz em um tipo de neurônio artificial, o *Sigmoid Neuron*, ou "neurônio sigmoide". Sua estrutura se assemelha à de um *perceptron*, mas a sua maneira de calcular a saída é dada por (Nielsen (2015)):

$$\sigma = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}. \tag{2.2}$$

O gráfico desta função é denominado *Sigmoid*, e é da forma (figura 4):

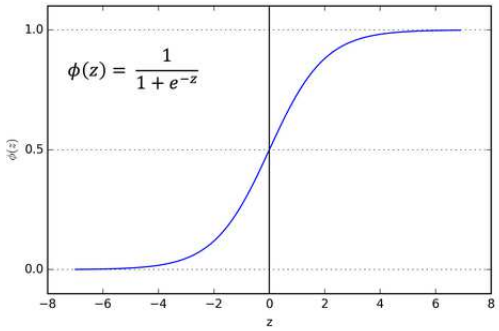


Figura 4 – Gráfico da função Sigmoide.
Fonte: Sharma (2017)

Perceba que este gráfico nada mais é do que uma versão "suave"da função *Step*, do *perceptron* simples. E é aí que está a "mágica"do neurônio sigmoide. Ao variar a saída pouco em relação à entrada, podemos ter um comportamento muito mais "suave"no treinamento da rede neural. Por isso que esta forma de neurônio aritifical é mais comum que o *perceptron* em trabalhos da área (Nielsen (2015)).

2.4 *Funções de ativação*

Cada saída de um nodo (z) é calculada a partir de x e y (saídas dos nodos anteriores), e após este cálculo, passa por uma função não-linear.

O cálculo é dado de forma genérica $z^{[n]} = W^{[n]} \cdot a^{[n-1]} + b^{[n]}$, cuja saída é um vetor com os valores de saída dos neurônios da n-ésima camada. Então, a função f, tipicamente não-linear, aplicada nesse vetor é chamada função de ativação ou de transferência. De maneira genérica, chamamos de "ativação"a em uma camada n a aplicação da função de derivação (de ativação) $a^{[n]} = f(z^{[n]})$. Na primeira camada há a diferença de que não há cálculo, pois não houveram nodos antes. Logo, a saída é igual à entrada: $a^{[0]}$.

Abaixo seguem as funções mais comuns em redes neurais:

- Função Logística

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

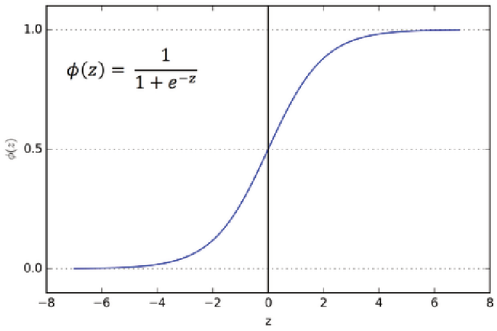


Figura 5 – Gráfico da função Logística.
Fonte: Sharma (2017)

- Função Tangente Hiperbólica

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.4}$$

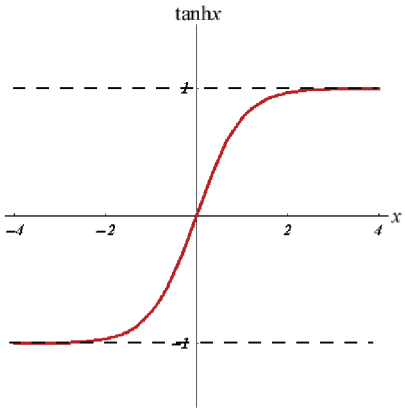


Figura 6 – Gráfico da função Tangente Hiperbólica.
Fonte: EFUNDA

- Rectified Linear Unit - ReLU

$$ReLU(z) = max(0, z) \tag{2.5}$$

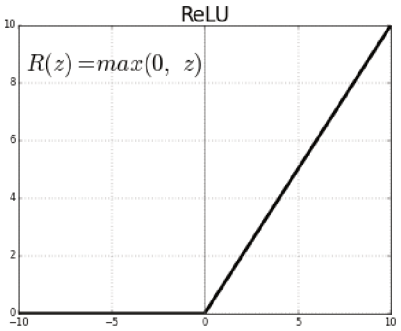


Figura 7 – Gráfico da função ReLU.
Fonte: Sharma (2017)

2.5 *Deep Learning*

Outra técnica de aprendizagem de máquina muito utilizada hoje para identificar e modelar dados com vários níveis de abstração é a chamada *Deep Learning*. Ela utiliza o algoritmo *Error Back-Propagation* ("Retropropagação do Erro") para ajustar seus parâmetros de modo a aumentar o nível de abstração com cada camada de sua arquitetura. Até o século XXI, o uso desta técnica estava estagnado por ser muito custoso em termos de processamento e memória. Como visto em LeCun, Bengio e Hinton (2015), isto agora não é mais um problema, e grandes avanços no processamento de imagens, vídeo, áudio e texto foram possíveis em um pequeno espaço de tempo.

Ela é popular devido a sua versatilidade, sendo capaz de receber diversos formatos de dados de entrada, e automaticamente ajustar seus parâmetros de forma a identificar desde diferenças simples (byte a byte, ou bordas simples) até características abstratas (como ângulo de visualização de um objeto).

Devido a sua generalidade, é usada em campos muito variados, como: previsão de interações entre substâncias químicas (Ma et al. (2015) apud LeCun, Bengio e Hinton (2015)), análise de dados de aceleradores de partículas (Ciodaro et al. (2012) e KAGGLE apud LeCun, Bengio e Hinton (2015)), e predição fenótipa de genes (Leung et al. (2014) e Xiong e al. (2015) apud LeCun, Bengio e Hinton (2015)). Também tem sido usado na área de linguagens naturais, em classificação de tópicos, respostas à perguntas (Bordes, Chopra e Weston (2014) apud LeCun, Bengio e Hinton (2015)), entendimento (Collobert e al. (2011) apud LeCun, Bengio e Hinton (2015)) e tradução (Jean et al. (2015) e Sutskever, Vinyals e V. (2014) apud LeCun, Bengio e Hinton (2015)).

2.5.1 *Supervised Learning*

A forma mais comum de aprendizagem de máquina é *Supervised Learning* (aprendizagem supervisionada). Quando se deseja quem uma rede neural seja capaz de identificar um elemento de uma imagem ou classificá-la em um subconjunto de possíveis interpre-

tações, primeiro é necessário descrever à rede como ela deve classificar imagens. Isto é feito dando à rede um conjunto de exemplos como entrada, junto com seus resultados esperados, para que desta forma a máquina seja capaz de tentar simular o raciocínio.

Para que ela seja capaz de ajustar seus parâmetros, ela deve ter uma forma de "saber"se esse ajuste está tornando ela mais precisa, ou menos. Isto tipicamente é feito por um algoritmo chamado *Gradient Descent*, o qual, para cada peso, calcula a tangente da função do erro, dados os pesos. Assim, ele ajusta os pesos na direção correta para reduzi-lo a cada iteração. O erro em função dos pesos pode ser visto como um terreno montanhoso, com altos e baixos, onde a "altura"corresponde ao tamanho da diferença entre o resultado esperado e o obtido naquela iteração. O gráfico desta função do erro em relação aos pesos possui a forma genérica da figura 8.

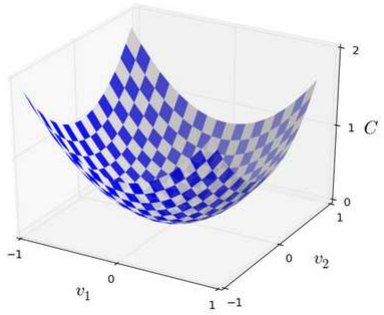


Figura 8 – Gráfico da função Gradiente.
Fonte: Nielsen (2015)

Inicialmente, um valor é dado a cada peso (o ponto inicial é definido). Depois, para determinar em qual "direção"modificar os pesos, usamos a derivada da função de erro para (a tangente da função no ponto atual). Uma vez escolhida a direção e sentido, basta escolher o tamanho do "salto". Este é dado da seguinte forma:

- Variação do erro em relação à n entradas

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + ... + \frac{\partial C}{\partial v_n}\Delta v_n. \tag{2.6}$$

- Vetor Gradiente

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)^T. \tag{2.7}$$

- Variação dos pesos com base em η e o Vetor Gradiente

$$\Delta v = -\eta \nabla C, \tag{2.8}$$

Na equação acima, η regula o tamanho do "salto". Este salto não pode ser pequeno demais, ou o aprendizado será muito lento, ou grande demais, ou ele pode impedir a rede de convergir em um erro aceitável. Assim, temos que escolher um η ou através de experimentos, ou através de ajustes automáticos ao observar a divergência (repetindo o passo com um "salto menor").

O algoritmo de gradiente mais comum é o *Stochastic Gradient Descent (SGD)*. Ele escolhe um subconjunto dos dados de treino para atingir um erro aproximado. Então ele escolhe outro subconjunto (sem repetir dados de treino entre subconjuntos) e reduz o erro mais ainda a partir dele. Dessa maneira, em cada iteração, ele ajusta os pesos até começar a ter dificuldade para obter maior precisão. Ele então repete o processo com subconjuntos diferentes até atingir um erro satisfatório. Após o treino, ele é apresentado a um conjunto de novos exemplos para verificar se está genérico o bastante.

2.5.2 *Back-Propagation*

Criada na década de 1970, e popularizada por Rumelhart, Hinton e Williams (1986). Para calcular o gradiente de forma rápida em uma rede com múltiplas camadas, viu-se necessário uma técnica nova. Nesse artigo foram mostradas várias redes neurais onde o algoritmo *Back-Propagation* foi muito mais veloz que técnicas antigas (apud Nielsen (2015))

Para calcular o gradiente de uma rede com múltiplas camadas, usa-se a técnica *Back-Propagation*. Ela usa derivadas [da função de erro] em cadeia, para calcular os pesos de uma camada com base na saída dela (entrada da próxima camada). Desta forma, pode-se atualizar a rede inteira, começando na saída (resultado) e descendo até a camada de entrada.

Com uma rede neural multinível é possível separar classes de dados de entrada. Na Figura 9, isto está mostrado nas cores azul e vermelho.

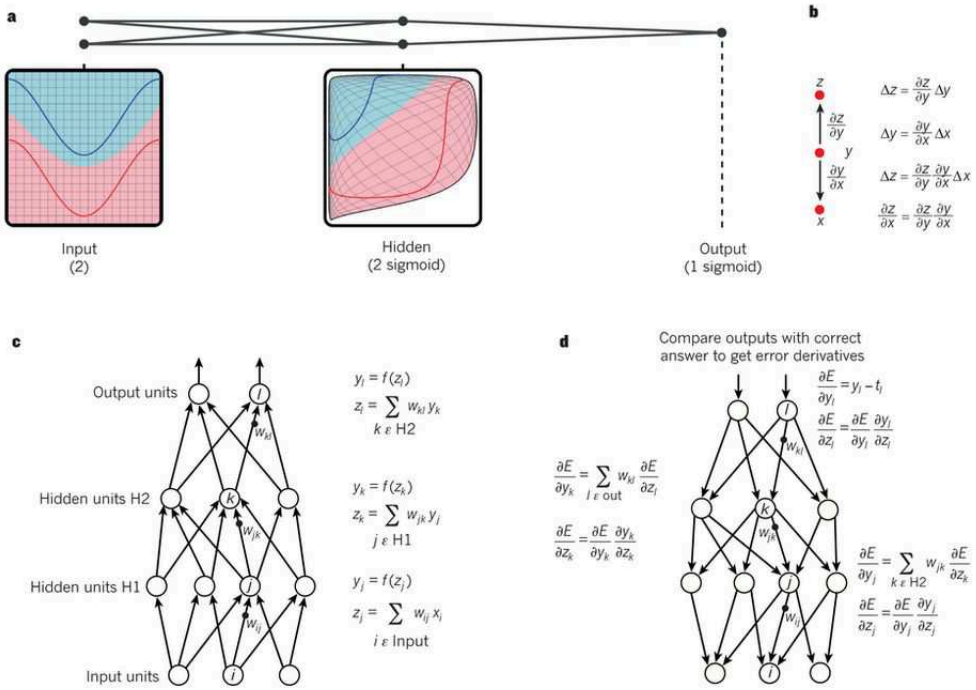


Figura 9 – Visão esquemática de uma rede neural e do algoritmo de *Back-Propagation*.
Fonte: LeCun, Bengio e Hinton (2015)

2.5.3 *Convolutional Neural Networks*

Como descrito em Vargas, Paes e Vasconcelos (2016), CNNs (*Convolutional Neural Networks*, ou Redes Neurais Convolucionais) são uma variante das redes de Perceptrons de Múltiplas Camadas (MLPs), compostas de múltiplas camadas com funções diferentes. As principais destas funções são a convolução e o pooling, explicadas em maiores detalhes abaixo.

Tendo sido desenvolvidas com o objetivo de processar múltiplas entradas de dados independentes, como as intensidades de cada uma de 3 cores em uma imagem só. Outros exemplos são sinais e sequências (áudio), espectogramas, vídeo e imagens volumétricas (LeCun, Bengio e Hinton (2015)).

Em imagens em vídeos são muito populares para fazer a detecção, classificação e reconhecimento, dada sua capacidade de manter e considerar as relações de vizinhança entre os pixels.

A convolução é um cálculo, feito em cima das diversas partes da imagem. Cada camada de convolução possui vários neurônios, cada um dos quais aplica um filtro na imagem. Este filtro "reconhece" padrões dentro da imagem, onde a combinação de múltiplos filtros torna a rede capaz de reconhecer padrões complexos, que se repetem de forma similar em situações nas quais os pixels em si possuem muita diferença. Exemplos destes usos são: detecção de rostos, texto, e humanos completos (LeCun, Bengio e Hinton (2015)).

Em imagens, a convolução usa um conjunto de filtros (onde o tamanho do filtro

representa o tamanho da região verificada) para calcular o quanto uma determinada região da imagem se assemelha ao filtro (ex.: se o filtro busca linhas verticais, ele vai percorrer a imagem, aplicando convoluções, para encontrar linhas verticais). Este processo ocorre na etapa de convolução da figura 10.

Após executar uma convolução, é comum utilizar funções de ativação para modificar os dados recebidos de uma maneira não-linear. Esta não-linearidade permite às camadas seguintes tornar as categorias de saída lineramente separáveis (Vargas, Paes e Vasconcelos (2016)).

A outra categoria de camada é a chamada camada de *Pooling*, ou agrupamento. A funçao desta camada é reduzir a dimensionalidade dos dados internos da rede. Esta redução dimensional é importante para acelerar o treinamento, porém, também serve a função de criar invariância espacial.

Esta camada funciona da seguinte forma: a entrada é dividida em múltiplas saídas, por exemplo (Vargas, Paes e Vasconcelos (2016)), janelas 4x4. A função da camada de *Pooling* é de selecionar um destes valores apenas, para representar o conjunto. Diversas funções podem exercer este papel, porém, a mais comum é a de máximo. Desta maneira, a camada de *Pooling* reduz a altura e largura dos mapas (agrupamento das saídas dos neurônios da camada anterior, resultado de um filtro comum sobre a entrada).

Ao aplicar o algoritmo de CNN em um conjunto grande de dados de treino, os filtros ideais para aquele problema são encontrados e combinados corretamente. Assim, caso deseja-se encontrar rostos, a rede pode utilizar filtros com olhos, bocas, narizes e orelhas (ou dois círculos para os olhos, uma reta horizontal para a boca etc).

Comparadas com redes neurais convencionais, vemos que esta se mostra mais adequada ao reconhecimento de padrões complexos, como formas humanas (ou de outros animais). Tudo isto mantendo a generalidade de redes neurais como um todo, levando a uma grande popularidade recente delas até para padrões bem específicos (Levi e Hassner (2015)), especialmente graças aos avanços velozes na capacidade de processamento de GPUs.

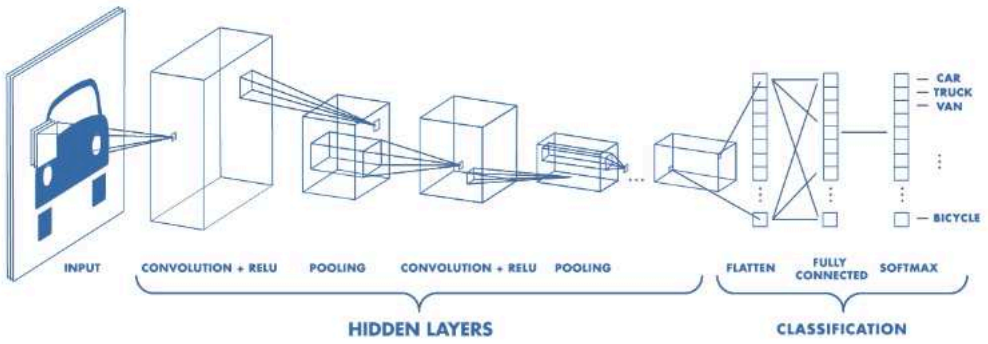


Figura 10 – Visão esquemática de uma Rede Neural Convolucional Classificadora. Fonte: CORNELISSE

As arquiteturas desta categoria de rede neural são diversas, Vargas, Paes e Vasconcelos (2016) citam LeNet-5 com duas camadas de convolução seguidas de uma de *pooling* e outra convolução. Outros exemplos citados no mesmo artigo são da GoogLeNet com cinco camadas de convolução, todas seguidas por uma de *pooling*. Para cada problema, costumam ser testadas arquiteturas de maneira a encontrar a ideal para aquele específico, não havendo uma arquitetura que resolva todos os problemas.

No caso de um problema de classificação, ela deve conter ao final das convoluções e *poolings* uma camada totalmente conectada, a qual tem a função de traçar um caminho de decisão a partir das camadas anteriores, para cada classe de resposta.

Após esta camada totalmente conectada, a rede deve conter uma etapa de classificação. Esta etapa é composta por uma função SoftMax, a qual gera um resultado de classificação a partir das classes de resposta da camada anterior (esta função é a mais comum, porém, existem outras que cumprem o mesmo papel).

3 Apresentação da Proposta

3.1 Proposta

A proposta deste trabalho é estudar os conceitos de redes neurais, determinar qual técnica utilizar e implementá-la em uma aplicação capaz de identificar ângulos de cabeça em imagens de profundidade.

Devido a sua biblioteca capaz de interagir com o *Kinect*®, foi escolhido o ambiente de programação MATLAB, versão 2018a e a rede neural utilizada será de um pacote da ferramenta, a qual conta com pacotes de redes neurais convolucionais, assim como redes neurais mais genéricas.

Após análise do problema e pesquisas teóricas, a técnica mais promissora parece ser a de redes neurais convolucionais. Esta automaticamente se adaptaria com filtros para reconhecer o ângulo correto de forma simples.

3.1.1 Arquitetura proposta

Primeiro, as imagens são capturadas utilizando as funções `MATLAB` que interagem com o *Kinect*®. Então, as imagens em si (640x480 pixels RGB e 640x480 pixels de profundidade) são colocadas em variáveis do ambiente. Após estas operações iniciais, as matrizes ainda são muito grandes e possuem muito ruído para servirem de entrada para uma rede neural.

Figura 11 – Imagem RGB como capturada por um KINECT®.
Fonte: ETHZ

Para extrair apenas a região da cabeça, é necessário localizá-la de maneira auto-

mática. Portanto, foi utilizado um algoritmo chamado Viola-Jones Viola e Jones (2001) disponibilizado pelo ambiente MATLAB para localizar o rosto na captura RGB e devolver o *bounding box* (caixa de contenção, coordenadas iniciais X e Y do rosto, e tamanho nos dois eixos). Esta área tem a função de recortar as imagens de profundidade, que são as utilizadas neste trabalho como entrada da rede neural.

Figura 12 – Imagem em escala de cinza, demonstrando o uso do algoritmo Viola-Jones ao detectar quase todas as fácies. Quadrados em torno dos rostos são as *bounding boxes*, desenhadas na imagem.

Fonte: Viola e Jones (2001)

O algoritmo Viola-Jones foi proposto por Paul Viola e Michael Jones em 2001. Em busca de um algoritmo rápido e eficiente para detectar fácies, 3 contribuições foram alcançadas pela dupla: Imagem Integral (uma imagem calculada a partir da imagem original, a qual acelera o processamento do algoritmo em si), um algoritmo de aprendizagem de máquina o qual seleciona as partes mais importantes da imagem para assim reduzir o processamento em áreas inúteis para a detecção, e um método de encadeamento de classificadores, de maneira a eliminar regiões e imagens inteiras que não possuem fácies, para assim acelerar dramaticamente o processamento total.

Por estas razões, o algoritmo proposto por Viola e Jones atinge sozinho 15 frames por segundo (nos testes feitos pelos proponentes em 2001), e é portanto utilizado em inúmeras aplicações, inclusive de tempo real, como é o caso deste trabalho.

Para demonstrar o recorte possibilitado pelo algoritmo Viola-Jones, seguem exemplos de capturas (RGB) recortadas, na figura 17.

Após o recorte das imagens de profundidade (com o mesmo *bounding box* das RGBs), elas ficam com dimensões diferentes, dado que as cabeças estão a distâncias diferentes da câmera, podem estar em posições estranhas ou até mesmo ter forma e tamanho diferentes de indivíduo para indivíduo. Nestes casos, as *bounding boxes* se ajustam em torno da fácie, o que detecta ela corretamente. Porém, isto é um problema devido à rede

neural aceitar apenas entradas de mesma dimensão para a qual foi treinada. Note as dimensões diferentes nos recortes (a) e (b) em 13.

(a) (b)

Figura 13 – Recortes de imagem RGB, demonstrando resultados após a aplicação do algoritmo Viola-Jones. Figura (a) ficou com dimensões 84x84, e figura (b) com dimensões 95x95

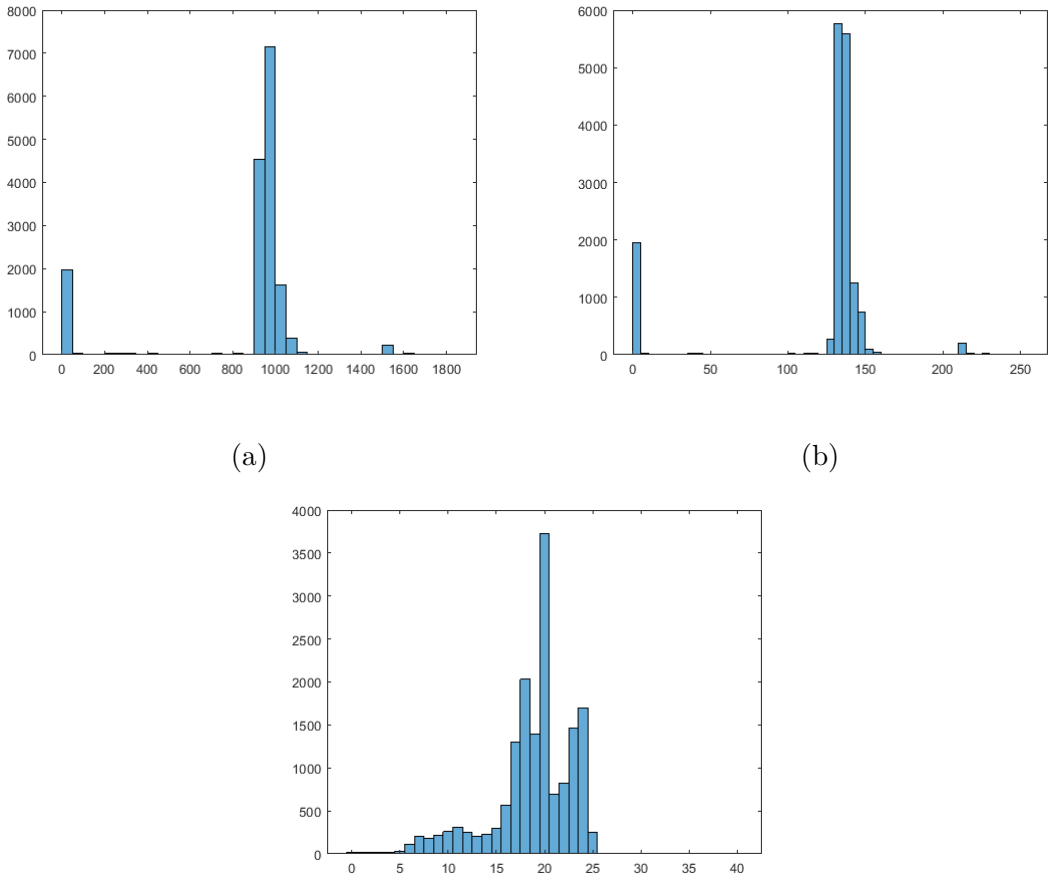
Fonte: Autor

Para corrigir isto, as imagens de profundidade também foram ajustadas para um tamanho padrão de 128x128 pixels, para que a mesma rede sirva para todas as imagens, assim como as imagens sejam de um tamanho efetivo para processamento pela rede.

Figura 14 – Recortes de imagem RGB, demonstrando resultados após a aplicação do algoritmo Viola-Jones e redimensionamento para 128x128 pixels.

Fonte: Autor

Uma vez de tamanho padrão, as imagens são submetidas a um processamento para torná-las mais compactas (no formato de matrizes `MATLAB`), no caso, reduzir o intervalo de valores para cada pixel para [0, 255]. Em seguida, são removidos o fundo e outras partes indesejadas que podem confundir a rede neural, como cabelo e áreas próximas demais à câmera, as quais podem aparecer sem valor indevidamente (valor zero). Exemplos passo-a-passo deste processamento, visto através dos histogramas e das imagens (a), (b) e (c) das figuras 15 e 16, respectivamente.



(a) (b)

(c)

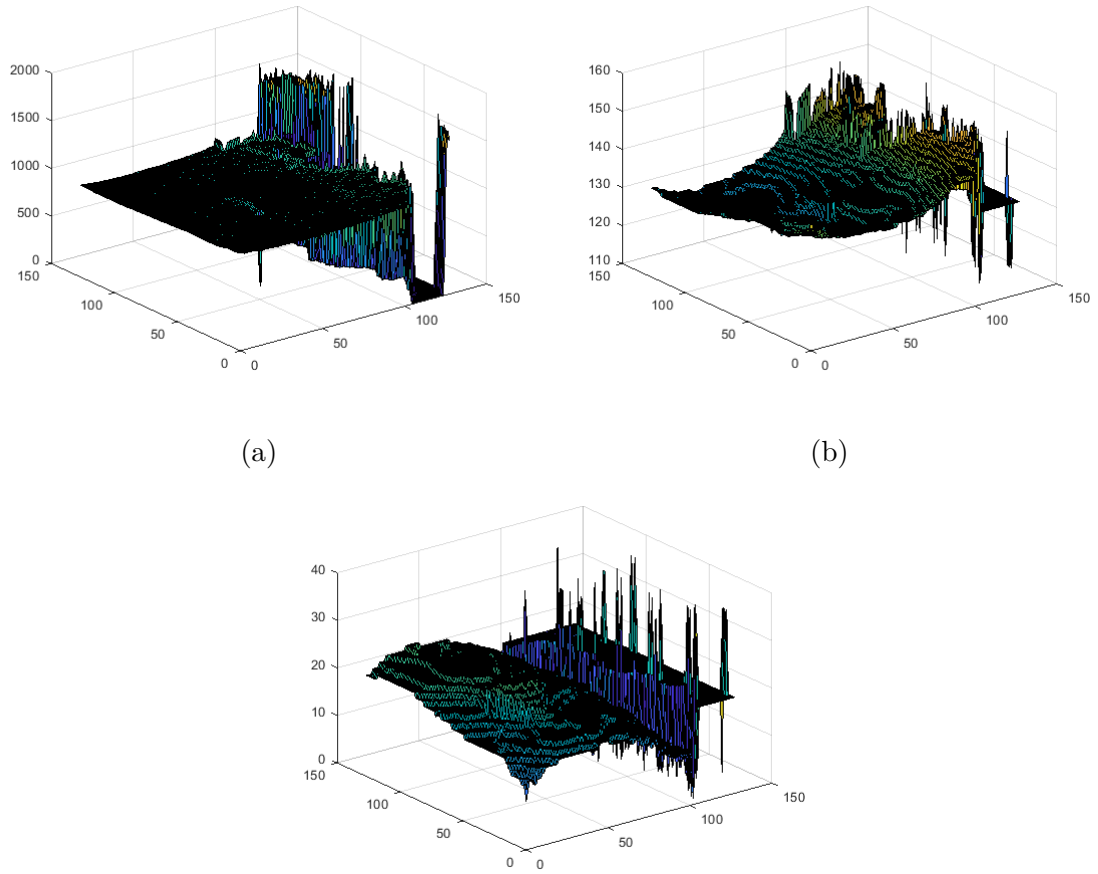
Figura 15 – Histogramas parciais do pré-processamento das imagens de profundidade. Em (a) temos a imagem como capturada pelo sensor. Em (b) temos a imagem com valores modificados para serem entre 0 e 255. Em (c) temos a imagem com uma máscara aplicada, onde os dados mencionados que poderiam confundir a rede foram removidos, e portanto, pronta para ser submetida à rede neural.

Fonte: Autor

Tirados os ruídos e simplificadas as imagens, elas estão prontas para serem submetidas à rede neural. Cada imagem de profundidade entra na rede neural como matriz (128x128 pixels) e a rede calcula a categoria à qual a imagem pertence (Muito à esquerda, pouco à esquerda, centrada, pouco à direita e muito à direita). Ou no caso de uma rede de regressão, tenta prever o ângulo da cabeça em um determinado eixo (caso este dado esteja na base de treinamento).

Caso esteja sendo treinada, neste momento a rede neural então ajusta seus pesos e filtros, de forma a identificar informações relevantes, como talvez a posição do nariz, dos olhos ou do queixo. Dados suficientes exemplos e tempo para processamento, teremos a rede treinada e pronta para identificar ângulos em novas imagens.

Se a entrada for para uso na classificação de imagens, ela retorna a categoria com



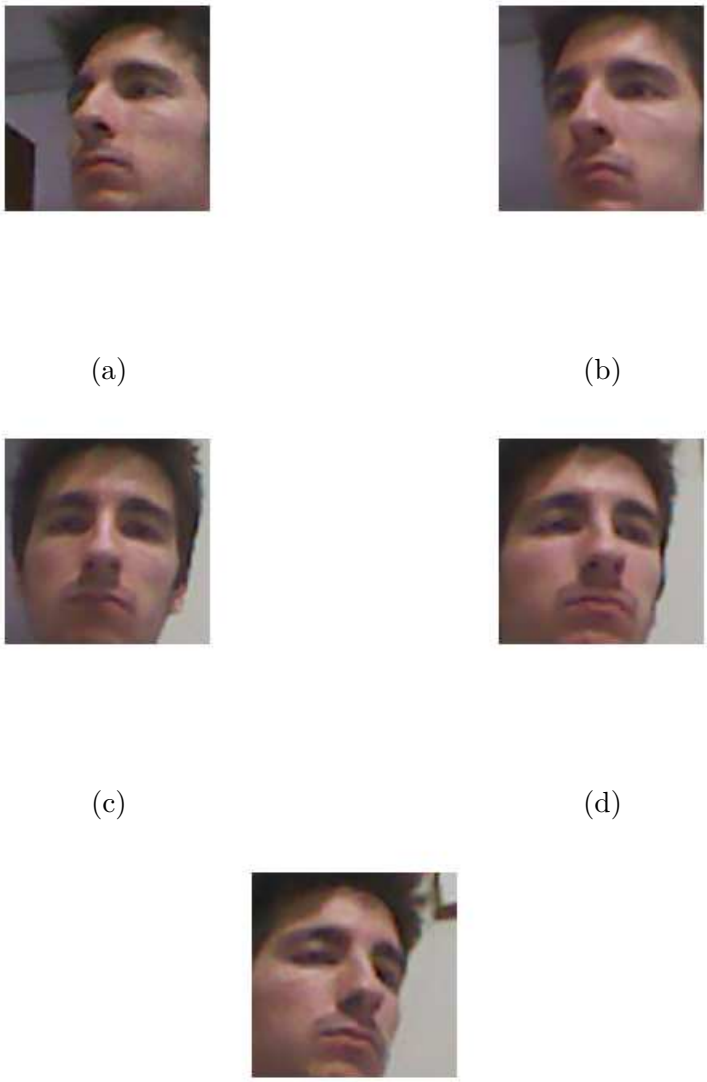
(a) (b)

(c)

Figura 16 – Imagens parciais do pré-processamento das imagens de profundidade. Em (a) temos a imagem como capturada pelo sensor. Em (b) temos a imagem com valores modificados para serem entre 0 e 255. Em (c) temos a imagem com uma máscara aplicada, onde os dados mencionados que poderiam confundir a rede foram removidos, e portanto, pronta para ser submetida à rede neural.

Fonte: Autor

acurácia de 90%, tendo como erro mais comum alguma categoria próxima a ela, o que permite que mesmo em caso de erro, a rede ainda possa ser utilizada de maneira confiável para aplicações que capturam imagens continuamente. Matrizes de confusão dos dados de treinamento e validação da rede neural, assim como a matriz de confusão das imagens da figura 17 estão apresentadas na figura 20.



(a) (b)

(c) (d)

(e)

Figura 17 – 5 Imagens de exemplo para testar a rede neural. (a) Muito à esquerda, (b) pouco à esquerda, (c) centralizada, (d) pouco à direita e (e) muito à direita.

Fonte: Autor

3.2 Bases de dados

Na primeira tentativa foi obtida uma base de dados do Laboratório de Visão Computacional do Instituto Federal de Tecnologia de Zurique (ETHZ, em alemão *Eidgenössische Technische Hochschule Zürich*). Esta base conta com 15677 imagens em formato RGB e profundidade obtidas por um KINECT®, assim como máscaras para isolar os rosto, os ângulos em matrizes de Euler e a posição da cabeça em 3 (três) dimensões. As matrizes de Euler são uma forma de representar os 3 (três) ângulos de rotação de objetos no espaço,

denominados (em inglês) *yaw*, *pitch* e *roll*. Estes se referem à rotação do corpo nos eixos vertical, transversal e longitudinal, respectivamente. Esta base de dados foi gerada por Fanelli et al. (2013) e utilizada no artigo referenciado, assim como em outros trabalhos referenciados na página ETHZ.

Alguns exemplos destas imagens abaixo, na figura 18:

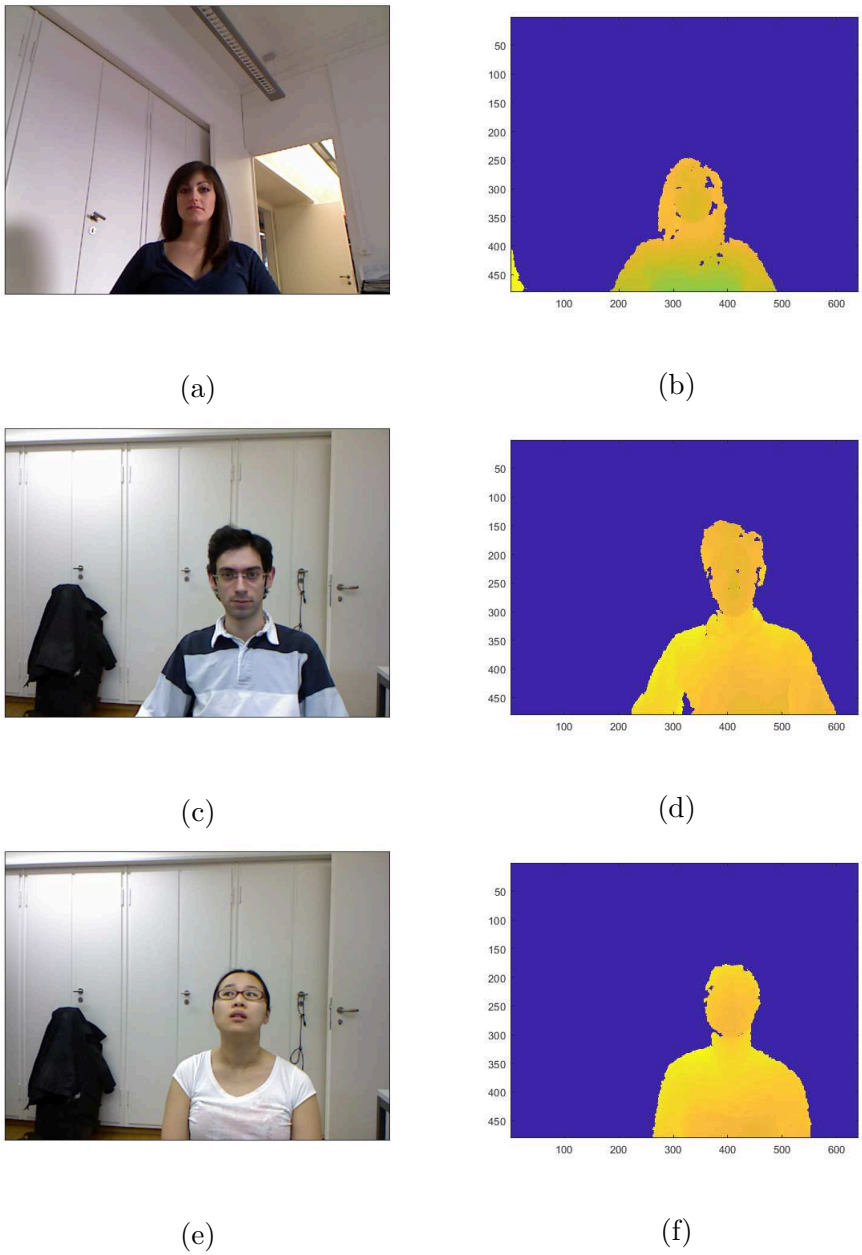


Figura 18 – Exemplos de imagens da base de dados *Head-Pose DataBase* da ETHZ, mostrando RGB e sua respectiva imagem de profundidade.

Fonte: Autor

Estas 15677 imagens foram cortadas conforme as máscaras, tratadas e submetidas a uma rede neural convolucional para treino. Esta rede utilizou em torno de 11000 imagens para treinar e as outras 4500 para validação. Após o treino, era necessário fazê-la

reconhecer imagens novas, capturadas pelo autor ou por usuários dos softwares que virão a utilizar os resultados destas capturas para seus próprios fins.

A rede neural convolucional escolhida foi de regressão, tendo como entrada uma imagem de profundidade tratada (128x128) e como saída o ângulo da cabeça no eixo vertical. Porém, devido a problemas com o formato muito diferente das imagens da biblioteca, com as imagens do autor, os resultados da rede foram consistentemente insatisfaórios, levando uma segunda tentativa.

Nesta segunda tentativa de treinar uma rede para classificar imagens de rostos, foi gerada uma base de dados própria, contendo 200 imagens de cada categoria (Muito à esquerda, pouco à esquerda, centrada, pouco à direita e muito à direita), totalizando 1000 imagens de profundidade. As imagens foram capturadas com fundos diferentes e distâncias diferentes (ambos para todas as categorias) para "incentivar"a rede convolucinal a identificar as características corretas a se buscar nas imagens.

Estas imagens próprias foram capturadas buscando maximizar a variância de todas as características, exceto aquilo que deseja-se reconhecer (posição da cabeça em torno do eixo vertical de rotação). Então além de fundos e distâncias diferentes, foram capturadas imagens com ângulos diferentes em torno dos eixos horizontal e longitudinal, piscando, torcendo o rosto e contraindo músculos da fácie. Especificamente para a variação de distância, esta foi feita dentro da possibilidade do espaço físico disponível ao autor no momento de construção da base. Estas distâncias foram consideradas aceitáveis devido a serem normais para um usuário de computador, onde o usuário se distanciaria menos da câmera do que em um outro dispositivo como o XBox, dispositivo desenvolvido para utilização com o *Kinect*® em jogos.

Então, estas imagens foram tratadas para remover o fundo e outras partes que apareciam na imagem mas não faziam parte daquilo que se desejava levar em consideração, gerando valores muito grandes ou pequenos. Esta decisão foi tomada com o objetivo de garantir que a rede não seja demasiado influenciada por fatores externos ao rosto em si, o que geraria tanto uma rede ineficaz como resultados insatisfatórios (ver histogramas na figura 15 e imagens 3D na figura 16).

Feito isso, foi gerado um vetor de 1000 posições, o qual continha os resultados esperados, em ordem, das 1000 imagens da base de dados do experimento. Após este procedimento, bastava fazer a seleção para treinar a rede. O procedimento escolhido para determinar os dados para treino e validação gera uma permutação sem repetições dos números que viriam a ser as posições de imagens na base de dados (128x128x1000, a permutação age sobre este último valor). Esta permutação seleciona randomicamente tanto os dados de treino quanto de validação, assim como suas respectivas saídas esperadas, sem permitir intersecção entre estes dois conjuntos, para efetividade máxima do treino da rede.

Uma vez obtidos os dados de treino e validação, estes foram submetidos desta vez a uma arquitetura de rede neural convolucional classificadora, com o restante da arquitetura similar, finalizando em uma classificação em 5 categorias, sendo elas: Muito à esquerda, pouco à esquerda, centrada, pouco à direita e muito à direita. A rede foi treinada e o resultado obtido inicialmente foi em torno de 89% de acurácia nos dados de validação e próximo a 100% nos dados de treino, sendo evidente o resultado muito satisfatório (para acurácia, ver figura 20).

A arquitetura escolhida para as redes neurais deste trabalho está descrita abaixo, e no apêndice:

```
layers = [
    % Camada de entrada
    imageInputLayer([128 128 1])

    % Primeira camada escondida
    convolution2dLayer(11,30,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    % Segunda camada escondida
    convolution2dLayer(5,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    % Terceira camada escondida
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    % Quarta camada escondida
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    % Camada final
    fullyConnectedLayer(5)
```

```
softmaxLayer
classificationLayer];
```

Nas camadas de convolução, o primeiro parâmetro indica o tamanho do filtro utilizado (neste caso foram filtros de mesma largura e altura). O segundo parâmetro indica a quantidade de filtros que serão aplicados nesta camada. Nas camadas de *Pooling*, o primeiro parâmetro indica o tamanho da região de *Pooling*. Quando a rede neural foi configurada para regressão, a camada final era de: dropoutLayer(0.2), fullyConnectedLayer(1) e regressionLayer, nesta ordem.

4 Experimentos e Resultados

4.1 Experimentos

4.1.1 Experimento 1

Utilizando os dados tratados da base de dados de ETHZ (), foram obtidos os ângulos de Euler dos arquivos de *ground truth* da base assim como as imagens de profundidade foram tratadas de maneira a torná-las mais adequadas para a rede (compactas e sem ruídos de fora da cabeça).

Foi então treinada uma rede neural convolucional de regressão, com o objetivo de prever o ângulo (em graus) da cabeça no eixo vertical.

Esta rede foi treinada com um subconjunto de 10000 imagens da base de dados mencionada. As saídas iriam de valores negativos à esquerda, a positivos à direita, com valores próximos de zero quando a cabeça estivesse voltada ao centro da câmera.

4.1.2 Resultados do experimento 1

Para a base de dados do ETHZ (), os resultados foram muito bons, com os ângulos previstos pela rede coincidindo fortemente com os ângulos esperados.

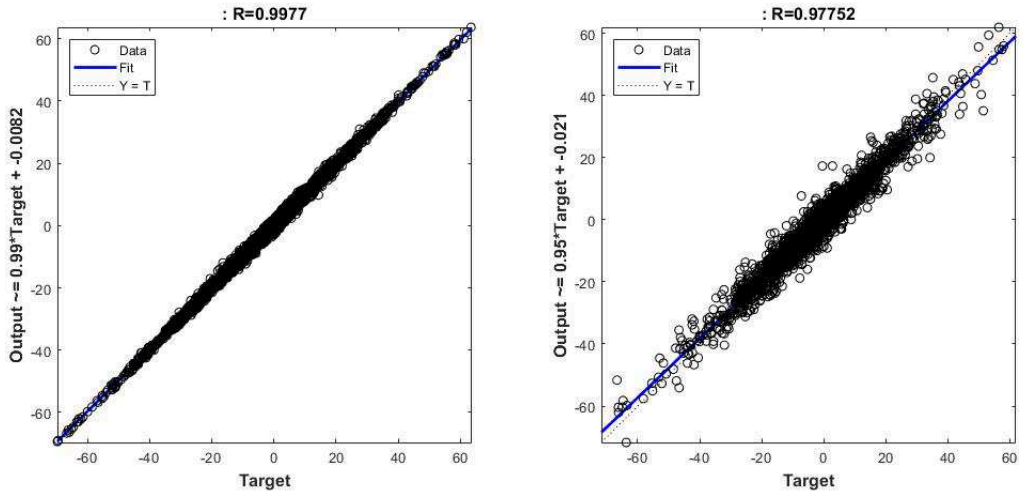
Porém, ao aplicar estes resultados aos exemplos do autor, a qualidade das previsões caía significativamente. Foram capturadas e tratadas 5 imagens (figura 17), de um extremo à outro de ângulos capturáveis, estas imagens foram tratadas da mesma forma que as da base de treinamento, e submetidas à rede.

Os resultados para as imagens da figura 17 foram: [-0.9713 -1.5730 -0.7911 2.1332 -11.1040]. Portanto, ângulos muito díspares dos resultados esperados, os quais deveriam no mínimo, formar uma sequência crescente de valores. Foram tentadas algumas variações, porém, os resultados foram similarmente insatisfatórios.

4.1.3 Experimento 2

Para este experimento, foi construída uma nova base de dados, descrita na seção 3.2 deste documento. Esta base de dados consiste em 1000 imagens de tamanho 128x128, com o mesmo tratamento dado a elas, porém, todas do autor e com suas saídas dadas em categorias, ao invés de ângulos (as cinco categorias descritas na seção 3.2).

Foi treinada uma nova rede neural, desta vez para classificação a partir de dados capturados pelo autor. O restante da configuração da rede foi idêntico, apenas substituindo



(a) (b)

Figura 19 – Estas imagens mostram o quanto as saídas da rede condizem com o resultado esperado para estes conjuntos de dados (no eixo vertical estão os valores de saídas, e no horizontal os valores esperados). Na imagem (a), os dados são os de treinamento da rede. Na imagem (b), os dados são os de validação da mesma rede.

Fonte: Autor

a última camada (*regression layer*) para 3 camadas: *fullyConectedLayer(5)* (para as 5 categorias), *softMaxLayer* e *classificationLayer*.

Os resultados das matrizes de confusão estão expostos abaixo:

4.1.4 Resultados do experimento 2

Como é possível ver, a precisão foi alta (90%) e, mesmo em caso de erro, este é sutil para o funcionamento de um programa que interprete as saídas como um intervalo de valores, pois o comando errado não seria tão diferente do comando desejado. Ainda assim, isto é um caso relativamente raro, mesmo em uma rede treinada com apenas 1000 exemplos.

Para treinar a rede, foi utilizado um computador disponibilizado pelo laboratório do orientador deste trabalho, o qual possui como características: $Intel^{®}Core^{TM}$ i7-8700K CPU, 64GB *RAM* e placa de vídeo *GeForce GTX 1080 Ti*. Com esta configuração, o tempo de treinamento foi de 3 minutos e 18 segundos, para um total de 200 épocas e 5200 iterações. A taxa de aprendizado foi de $1e-07$.

4.1.5 Experimento 3

Uma vez com a rede neural treinada com os dados do autor, foi possível executar um experimento para verificar como seria seu desempenho em uma situação mais realista.

(a) (b)

(c)

Figura 20 – Matrizes de confusão para 3 conjuntos de dados. Em (a) temos a matriz de confusão dos dados de treinamento. Em (b) temos a matriz de confusão dos dados de validação da rede neural. Em (c) temos a matriz de confusão das 5 imagens da próxima figura.

Fonte: Autor

Foi escolhido um teste de tempo real, onde imagens seriam capturadas, submetidas à rede e as saídas verificadas para conferir não apenas a acurácia, mas também a velocidade resultante do tratamento e processamento dos dados em um cenário real.

Foram obtidas 100 imagens dessa forma, e verificadas manualmente para comparar com os movimentos recém feitos pelo autor. Conforme demonstrado pelas matrizes de confusão (figura 20), a taxa de acerto foi alta, com os erros sendo de categorias similares (posições de cabeça similares).

Também foram constatados alguns erros provenientes do algoritmo de Viola-Jones, pois, como este depende de localizar o rosto, quando o usuário vira seu rosto muito para

os lados, para cima ou para baixo, o algoritmo falha e a rede não possui entrada válida naquele frame.

Outro problema também advindo do algoritmo Viola-Jones foi quando este localiza múltiplos rostos, onde há 2 (duas) situações possíveis: Existem de fato múltiplos rostos, onde poderíamos colocar todos na rede, causando interferência; Não há múltiplos rostos (falso positivo), neste caso o correto seria ignorar os errados.

Devido ao objetivo deste trabalho não ser determinar se parte de uma imagem é um rosto ou não, e sim determinar a posição da cabeça, optou-se por utilizar apenas o primeiro rosto detectado. Caso nenhum tenha sido localizado, a "saída"da rede ao invés de ser computada, foi colocada como *default* "rosto centrado"(para os testes iniciais).

4.1.6 Resultados do experimento 3

Quanto à precisão, o experimento 3 obteve o mesmo valor dado que a rede neural utilizada foi a mesma, e os resultados foram apenas aplicados em uma situação diferente.

Estes resultados demonstraram que a rede era capaz de funcionar em uma situação de tempo real, com suficiente taxa de acerto para ser utilizada em uma aplicação real.

4.1.7 Experimento 4

Utilizando a mesma rede neural do experimento anterior, treinada com a base de dados do autor, foi desenvolvido um algoritmo que captura as imagens do usuário, em tempo real, e as submete à rede.

Foi escolhido para a rede um cenário de utilização para jogos. Para tal, foi baixado o programa multijogador FuzzyTruck, disponibilizado pelo orientador deste trabalho. Como este jogo recebe comandos via entrada de texto na interface de rede, ele pôde ser facilmente adaptado para receber comandos pelo ambiente MATLAB, onde a rede foi desenvolvida e é executada.

De maneira similar ao experimento anterior, este apenas inclui a utilização da saída da rede para enviar um comando por TCP/IP para o jogo FuzzyTruck, "girando" o "volante" do caminhão em tempo real, de acordo com a movimentação da cabeça do jogador, comprovando a eficácia da rede neural desenvolvida.

O objetivo deste jogo é estacionar um caminhão, o qual é mostrado em uma interface gráfica, com o estacionamento sendo o espaço entre as duas linhas na parte de baixo da tela. Cada vez que um ângulo é enviado ao servidor, este calcula a nova posição e desenha na tela o caminhão nesta nova posição (sem apagar a posição anterior, permitindo a visualização do caminho completo percorrido).

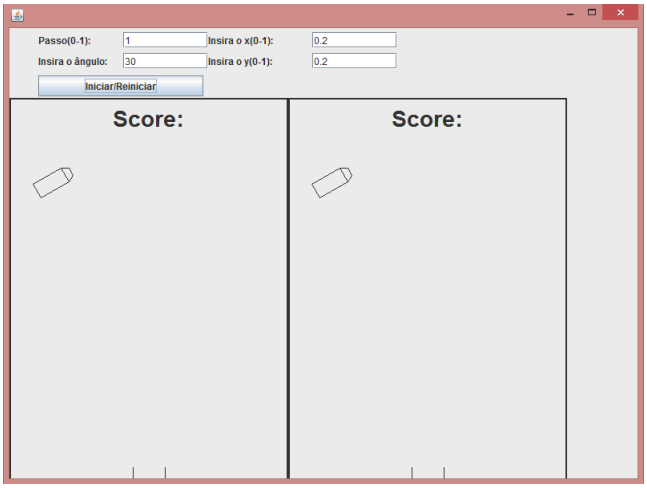


Figura 21 – Tela inicial do jogo FuzzyTruck.
Fonte: Autor

4.2 Resultados do experimento 4

Ajustando a equação para determinar o "movimento do volante", consegui-se uma conversão bastante adequada da saída da rede para o comando. O resultado foi próximo a um estacionamento completo, assim como foram também feitos testes para ver se o caminhão andava em círculos ou em linha reta, caso mantido o rosto virado em uma posição por longo período de tempo (figuras 22 e 23).

Os resultados dos experimentos confirmaram o esperado pelas matrizes de confusão, tendo grande precisão (80-90%), sendo que mesmo em casos de erro, ainda era possível corrigir a rota (no caso do caminhão).

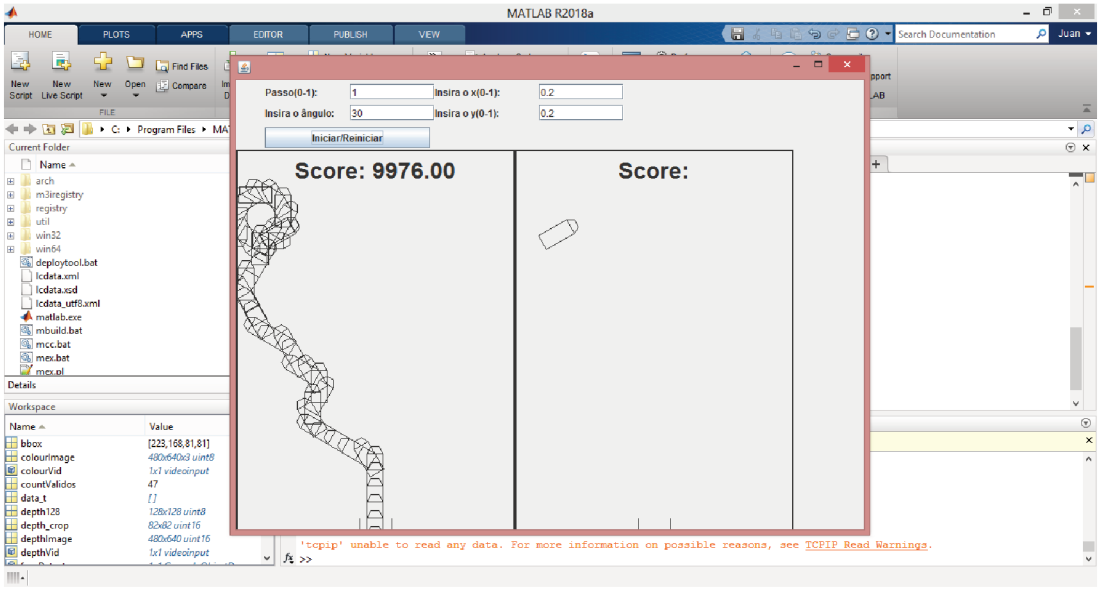


Figura 22 – Caminho percorrido pelo caminhão, através dos movimentos de cabeça classificados pela rede neural.

Fonte: Autor

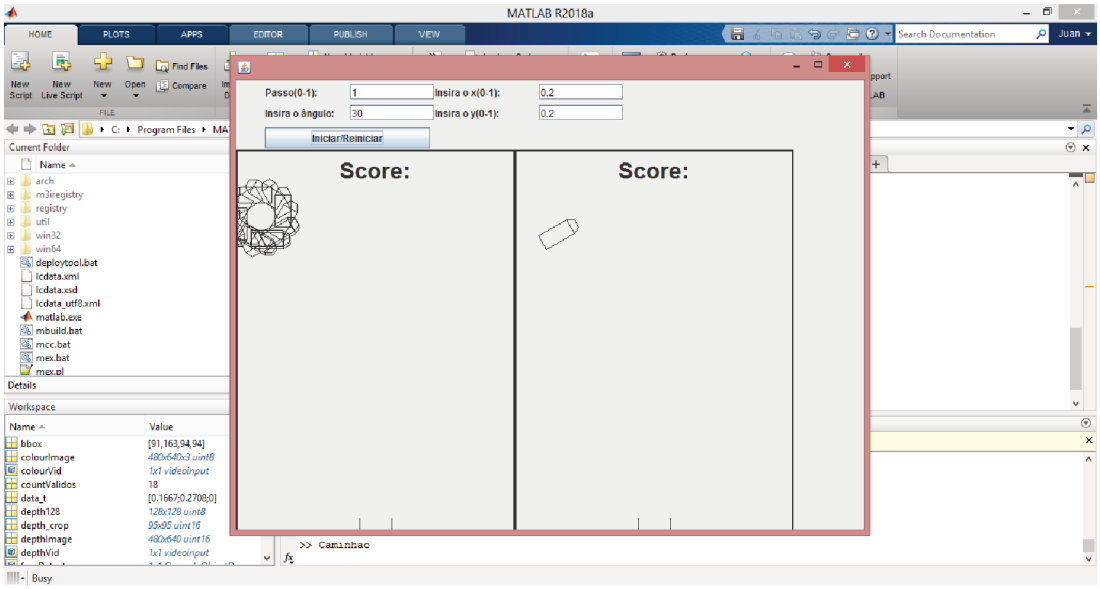


Figura 23 – Caminho percorrido pelo caminhão, com a cabeça parada e voltada para à esquerda.

Fonte: Autor

Com a precisão observada, foi possível fazer qualquer caminho que o usuário desejasse (limitado apenas aos comandos originais do software em questão). Portanto a precisão da rede foi bastante satisfatória, enquanto a velocidade nos experimentos de tempo real foi de 0.3 frames/segundo.

5 Conclusões

A precisão alcançada pela rede com apenas 1000 imagens demonstrou que o pré-processamento foi bom, assim como a captura pelo *Kinect*® teve uma qualidade suficiente de detalhes para o problema. Isto demonstrou que é possível aplicar esta rede para identificar posições de cabeça em ambientes diversos, inclusive de baixo custo.

Pelo mesmo motivo, o conceito de desenvolver aplicações controladas por movimentos de cabeça foi comprovado, utilizando Redes Neurais Convolucionais (de classificação, neste caso).

Foram identificadas 2 (duas) principais dificuldades. A primeira foi a dificuldade com o algoritmo Viola-Jones, o qual pode retornar falsos positivos. Isto deve ser contornado de uma maneira efetiva. No caso deste trabalho, foi apenas necessário selecionar a primeira fácie, porém, isto pode apresentar um problema maior em trabalhos que utilizem múltiplas fácies simultaneamente. Felizmente, o problema é relativamente raro (menos de 1% de falsos positivos).

A segunda dificuldade foi com relação ao sensor *Kinect*® em si. Quando o usuário se situa muito próximo ao sensor (aproximadamente 50 centímetros ou menos), o resultado da captura de profundidade nas regiões da imagem demasiado próximas ao sensor será zero, muito diferente do valor de 800 ou mais, de pouco mais distante da câmera do que isto.

Fora isto, trabalhar com o *Kinect*® foi uma boa experiência, considerando seu baixo custo e flexibilidade de uso. A captura simultânea de vídeo de cor (RGB) e de profundidade, com uma qualidade surpreendente é muito útil para aplicações deste tipo, especialmente quando a aplicação que venha a utilizar uma rede neural como a desenvolvida neste trabalho for pensada para ser distribuída para um grande número de pessoas.

Contruir esta rede neural e utilizá-la em uma aplicação de tempo real demonstrou que a viabilidade de trabalhos semelhantes existe e está ao alcance de muitos. Ao longo deste desenvolvimento, múltiplos novos trabalhos foram pensados que poderiam partir de princípios semelhantes.

Um caminho seria retornar a tentativa de construir uma rede neural de regressão, para talvez fazer comandos em um intervalo contínuo de posições, ao invés de discreto, como aquele desenvolvido neste trabalho.

Outra possibilidade é a de expandir a base de dados gerada pelo autor, para abranger mais distâncias, assim como utilizar pessoas diferentes para treinar a rede. Além da base de dados maior, também podem ser feitas mais categorias além das 5 posições

deste trabalho, dando maior flexibilidade aos comandos possíveis.

Com relação à rede, podem-se testar diferentes arquiteturas, com mais ou menos camadas, filtros maiores ou menores, mais convoluções, mais poolings, menos convoluções e outras combinações, para desenvolver a melhor configuração de rede possível para este problema específico.

Finalmente, podem ser classificadas imagens com maior especificação de posições. Isto pode ser feito tendo a rede de regressão com 3 (três) saídas (uma para cada ângulo de Euler), 3 redes neurais distintas (uma para cada ângulo), ou ainda uma rede classificadora treinada para distinguir categorias mais variadas (exemplos: cima-direita, baixo-centro etc).

Referências

AL-RAHAYFEH, A.; FAEZIPOUR, M. Eye tracking and head movement detection: A state-of-art survey. *IEEE J Transl Eng Health Med*, 2013.

BORDES, A.; CHOPRA, S.; WESTON, J. Question answering with subgraph embeddings. *Conference on Empirical Methods in Natural Language Processing*, 2014. Disponível em: <http://arxiv.org/abs/1406.3676v3>.

CARVALHO, A. P. de Leon F. de. *Redes Neurais Artificiais*. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/>. Acesso em: 30.10.2018.

CIODARO, T. et al. Online particle detection with neural networks based on topological calorimetry information. *Journal of Physics: Conference Series 368*, 2012.

COLLOBERT, R.; AL. et. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, p. 2493–2537, 2011.

COOTES, T. F.; EDWARDS, G. J.; TAYLOR, C. J. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 23, n. 6, p. 681–685, June 2001. ISSN 0162-8828.

CORNELISSE, D. *An intuitive guide to Convolutional Neural Networks*. Disponível em: <https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050>. Acesso em: 26.10.2018.

EFUNDA. Disponível em: <http://www.efunda.com/math/hyperbolic/display.cfm?name=tanh>. Acesso em: 21.06.2018.

ETHZ, H. P. D. *Random Forests for Real Time Head Pose Estimation*. Disponível em: <https://data.vision.ee.ethz.ch/cvl/gfanelli/head_pose/head_forest.html#>. Acesso em: 16.09.2018.

FANELLI, G. et al. Random forests for real time 3d face analysis. *Int. J. Comput. Vision*, v. 101, n. 3, p. 437–458, February 2013.

FANELLI, G.; GALL, J.; GOOL, L. V. Real time head pose estimation with random regression forests. In: *Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2011. p. 617–624.

JEAN, S. et al. On using very large target vocabulary for neural machine translation. *Proceedings of the Association for Computational Linguistics Anthology*, 2015. Disponível em: <http://arxiv.org/abs/1412.2007>.

KAGGLE. *Higgs boson machine learning challenge*. 2014. Disponível em: <https://www.kaggle.com/c/higgs-boson>.

KINECT. *Kinect for Xbox 360*. Disponível em: <http://www.xbox.com/en-US/xbox-360/accessories/kinect/>. Acesso em: 15.11.2016.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, 2015. Disponível em: <https://www.nature.com/articles/nature14539/>. Acesso em: 10.5.2018.

LEUNG, M. K. et al. Deep learning of the tissue-regulated splicing code. *Oxford Academic Bioinformatics 30*, p. 121–129, 2014.

LEVI, G.; HASSNER, T. Age and gender classification using convolutional neural networks. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. [S.l.: s.n.], 2015.

MA, J. et al. Deep neural nets as a method for quantitative structure-activity relationships. *Journal of Chemical Information and Modeling*, p. 263–274, 2015.

MATLAB. Disponível em: <https://www.mathworks.com/products/matlab.html>.

MCCULLOCH, W. S.; PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 115–133, 1943.

MURPHY-CHUTORIAN, E.; TRIVEDI, M. M. Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 31, n. 4, p. 607–626, April 2009. ISSN 0162-8828.

NIELSEN, M. A. *Neural Networks and Deep Learning*. [s.n.], 2015. Disponível em: <http://neuralnetworksanddeeplearning.com/>.

PORTET, F. et al. Design and evaluation of a smart home voice interface for the elderly – Acceptability and objection aspects. *Personal and Ubiquitous Computing*, Springer Verlag, v. 17, n. 1, p. 127–144, 2013. Impact-F=1.13 estim. in 2012. Disponível em: <https://hal.archives-ouvertes.fr/hal-00953242>.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back propagating errors. v. 323, p. 533–536, 10 1986.

SHARMA, S. *Activation Functions: Neural Networks*. 2017. Disponível em: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>. Acesso em: 21.06.2018.

SIRITEERAKUL, T.; SATO, Y.; BOONJING, V. Estimating change in head pose from low resolution video using lbp-based tracking. In: *2011 International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS)*. [S.l.: s.n.], 2011. p. 1–6.

SPECIALEFFECT. *Beating physical disability to enjoy video games*. Disponível em: <http://www.specialeffect.org.uk/>. Acesso em: 15.11.2016.

SUTSKEVER, I.; VINYALS, O.; V., L. Q. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, p. 3104–3112, 2014.

TOOLBOX, D. L. *Deep Learning Toolbox - MATLAB*. Disponível em: <https://www.mathworks.com/products/deep-learning.html>. Acesso em: 03.07.2018.

VARGAS, A. C.; PAES, A.; VASCONCELOS, C. Um estudo sobre redes neurais convolucionais e sua aplicaçao em detecção de pedestres. *CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 29. (SIBGRAPI)*, 2016. Disponível em: <<http://urlib.net/rep/8JMKD3MGPAW/3ME3L2P>>.

VATAHSKA, T.; BENNEWITZ, M.; BEHNKE, S. Feature-based head pose estimation from images. In: *2007 7th IEEE-RAS International Conference on Humanoid Robots*. [S.l.: s.n.], 2007. p. 330–335. ISSN 2164-0572.

VIOLA, M.; JONES, M. J.; VIOLA, P. Fast multi-view face detection. In: *Proc. of Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2003.

VIOLA, P.; JONES, M. *Rapid object detection using a boosted cascade of simple features*. 2001.

WEISE, T.; LEIBE, B.; GOOL, L. V. Fast 3d scanning with automatic motion compensation. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2007. p. 1–8. ISSN 1063-6919.

XIONG, H. Y.; AL. et. The human splicing code reveals new insights into the genetic determinants of disease. *Science*, 2015.

YAO, J.; CHAM, W.-K. Efficient model-based linear head motion recovery from movies. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* [S.l.: s.n.], 2004. v. 2, p. II–II. ISSN 1063-6919.

ZHAO, Y.; YAN, H. Head orientation estimation using neural network. In: *Proceedings of 2011 International Conference on Computer Science and Network Technology*. [S.l.: s.n.], 2011. v. 3, p. 2075–2078.

ZHAO, Z.; WANG, Y.; FU, S. Head movement recognition based on lucas-kanade algorithm. In: *2012 International Conference on Computer Science and Service System*. [S.l.: s.n.], 2012. p. 2303–2306.

6 Apêndice

6.1 APÊNDICE A - Artigo sobre o TCC

Utilização de redes convolucionais profundas para estimativa de ângulos de pose de fácies obtidas através do Kinect®

Juan A. T. Cuttle[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

juan.cuttle@grad.ufsc.br

Abstract. *In the presente article a CNN (Convolutional Neural Network) was trained, and utilized to estimate the pose of human faces in 3D images, captured by a depth sensor of a Kinect®.*

Resumo. *No presente trabalho foi treinada uma CNN (Convolutional Neural Network), e utilizada para estimar a posição de fácies, em imagens 3D capturadas por um sensor de profundidade de um dispositivo Kinect®.*

1. Introdução

Como descrito em [Vargas, Paes e Vasconcelos, 2016], CNNs (*Convolutional Neural Networks*, ou Redes Neurais Convolucionais) são uma variante das redes de Perceptrons de Múltiplas Camadas (MLPs), compostas de múltiplas camadas com funções diferentes. As principais destas funções são a convolução e o pooling.

O foco deste trabalho será em classificar posições de cabeça, as quais são acessíveis para a grande maioria das pessoas, e podem ser facilmente identificadas, estando o usuário sentado ou de pé. Além disso, as diferentes posições são capazes de uma variedade de comandos de software, tornando esta opção muito generalizável para aplicações futuras.

A categoria de aplicações que foi testada foi a de tempo real, a qual requer análise de posições dinâmicas (sequência de movimentos), para a qual foi utilizada uma biblioteca de redes neurais em MATLAB para fazer a classificação destas posições. Isto foi feito através da detecção em frames individuais, onde cada frame é submetido à rede, a saída é gerada, e esta é enviada como entrada ao programa aplicativo, o qual por sua vez, responde de acordo.

Ao final, os resultados foram analisados para determinar sua aplicabilidade e áreas de melhoria, assim como foram feitas sugestões de trabalhos futuros baseados em CNN's e reconhecimento de posições de cabeça.

2. Procedimento Experimental

Para a realização dos experimentos foi utilizado o ambiente MATLAB, principalmente sua *Deep Learning Toolbox*. Como base de dados, foram capturadas imagens RGB e de profundidade através do sensor Kinect®, sempre aos pares.

O algoritmo Viola-Jones [Viola e Jones, 2001] localiza todos os rostos em uma imagem RGB. Como desejamos utilizar neste trabalho apenas imagens com um único rosto, somente utilizamos um e rejeitamos os falsos positivos.

Após este processo, podemos selecionar apenas a região da imagem de profundidade onde sabemos estar o rosto, e portanto, a cabeça da pessoa. Para corrigir por cabeças mais distantes ou próximas, assim como posições mais inusuais, as quais geram regiões maiores ou menores de presença da cabeça detectadas pelo algoritmo, estas imagens são então corrigidas para um formato padrão de 128x128 pixels.

Feito este ajuste apenas da região desejada da imagem, a próxima etapa é a de modificar os valores dos pixels para acelerar o treinamento, assim como tornar a rede ao final mais precisa. Para tal, o intervalo dos valores dos pixels foi modificado para [0, 255] e as imagens foram "mascaradas", para remover valores muito discrepantes do rosto, como o fundo e regiões muito próximas da câmera, aos quais são atribuídos valor 0 (zero) pelo sensor.

Uma vez pré-processadas, as imagens de profundidade estão prontas para serem submetidas para a rede neural. As imagens foram distribuídas em 5 (cinco) categorias: Muito à esquerda, esquerda, centrada, direita e muito à direita.

A rede neural convolucional foi treinada utilizando 1000 imagens de profundidade (200 de cada categoria), separando 700 destas para treinamento e 300 para validação.

A rede foi então testada em um software *FuzzyTruck*, onde o objetivo é guiar o caminhão até uma parte da tela, delimitada por dois traços. Para tal, devem ser enviados comandos para o jogo, de maneira que estes comandos em sequência levem a um caminho de sucesso até a área de destino.

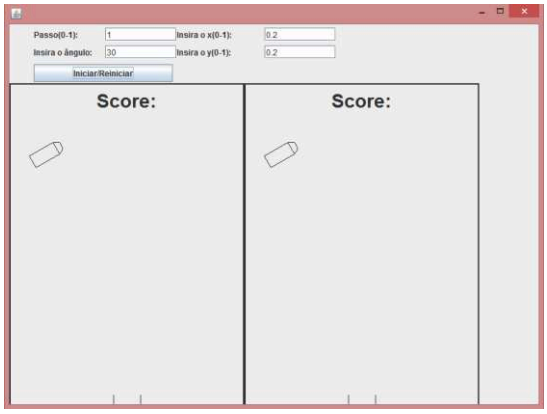


Figura 1: Tela inicial do jogo FuzzyTruck

Com a saída da rede classificada em 5 (cinco) categorias, estas foram utilizadas como movimentos para o "volante" (muito à esquerda, esquerda, centrado, direita e muito à direita). A conversão para este jogo foi direta e as saídas foram transformadas em ângulos para cada movimento.

3. Resultados e Discussões

Foram feitos alguns testes iniciais para verificar a precisão geral da rede, resultando na medida de ~89% de acurácia, como mostram as matrizes de confusão abaixo:

Figura 2: Matriz de confusão do teste inicial com uma imagem de cada categoria

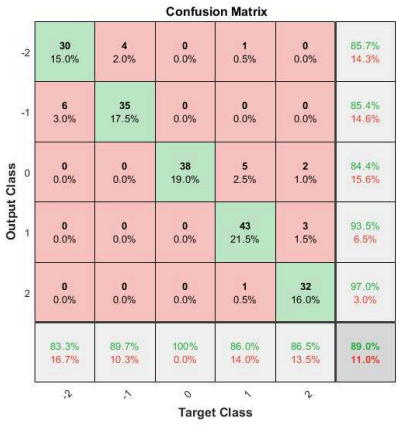


Figura 3: Matriz de confusão para o conjunto de validação da rede, como efeito de comparação

Após constatação de um nível bom de acurácia, foi feito o teste com o jogo *FuzzyTruck*. As saídas da rede foram calculadas em ângulos para "virar o volante" do caminhão, e os caminhos alcançados estão demonstrados nas capturas de tela ao final das partidas:

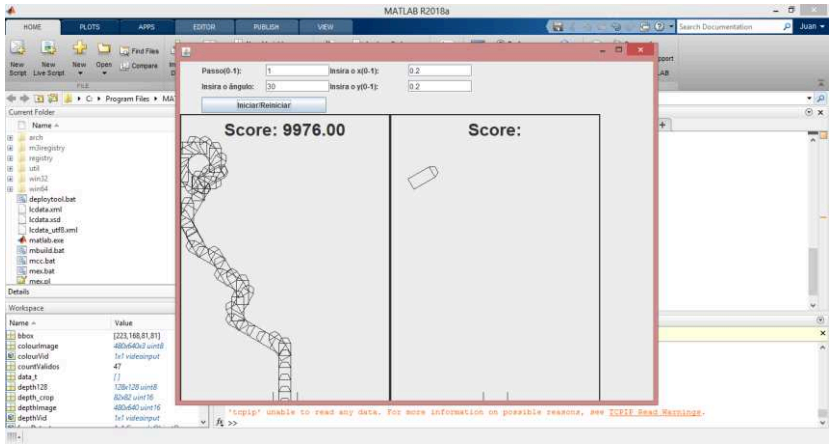


Figura 4: Um caminho de estacionamento de sucesso, conforme os objetivos do jogo.

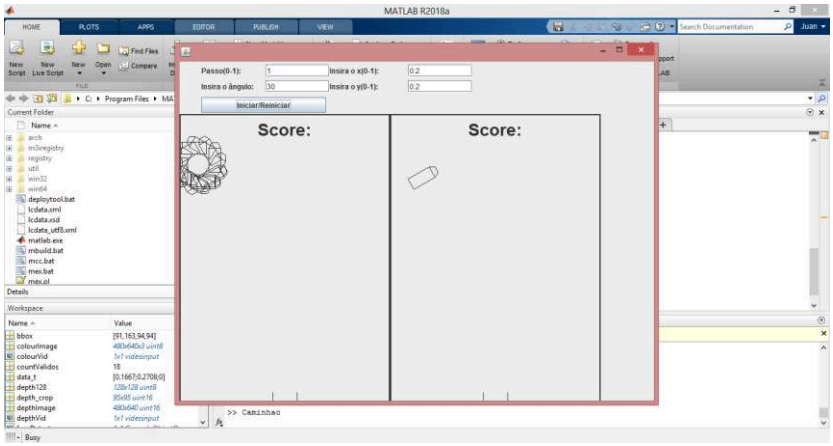


Figura 5: Um caminho circular que evidencia o resultado do usuário manter-se com a cabeça virada para o lado, o que resulta em um "volante" mantido para o lado no jogo.

A partir destes resultados, podemos ver que a precisão constatada foi de fato satisfatória pois, mesmo em caso de erro, a saída predita era próxima em ângulo à saída esperada, não havendo súbitas "viradas" em momento algum dos experimentos.

4. Conclusões

O que se pode concluir a partir dos experimentos feitos com a rede neural e, em particular com o uso do software *FuzzyTruck*, foi que a utilização de redes neurais convolucionais de camada profunda para detecção de posição de fácies em imagens de profundidade não somente é possível, como possui alta precisão, suficiente para utilizar em aplicações reais.

A velocidade de 0.3 frames/s deixou a desejar e, como trabalho futuro, a melhoria deste tempo de processamento é desejável. A rede também não foi testada com pessoas diferentes, podendo ter problemas devido ao treinamento exclusivo com imagens de uma só pessoa. Utilizar os mesmos princípios com uma base de dados maior e mais variada é importante caso a intenção seja de construir uma rede para uso comercial, ou apenas de valor mais prático.

Outras configurações e arquiteturas de rede também podem ser exploradas, de maneira a buscar maior precisão ou até mesmo maior velocidade de classificação. Para um controle mais preciso da aplicação, podem ser feitas mais categorias, classificando a partir de 2 (dois) ou até dos 3 (três) eixos de rotação. Uma alternativa a isto é desenvolver uma rede de regressão, a qual teria como requisito os ângulos das imagens de treino e validação, e também poderia ser feita com outra arquitetura e em mais de um eixo de rotação.

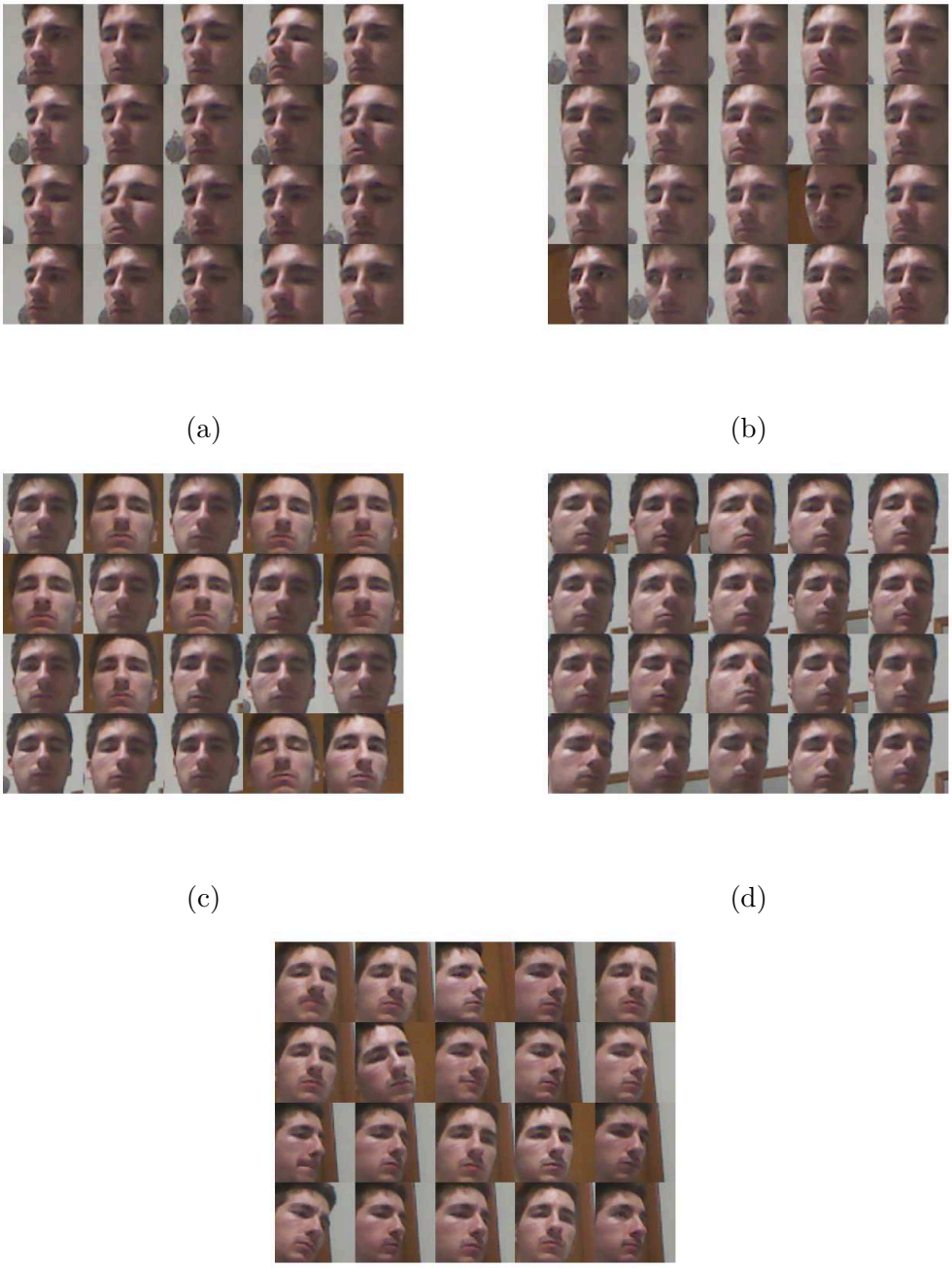
Referências

C Vargas, A Paes, and C Vasconcelos. Um estudo sobre redes neurais convolucionais e sua aplicaçao em detecção de pedestres. CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 29. (SIBGRAPI), 2016.

Deep Learning Toolbox. Deep learning toolbox - matlab.

Paul Viola and Michael Jones. “Rapid object detection using a boosted cascade of simple features”, 2001.

6.2 APÊNDICE B - Exemplos da base de dados utilizada



(a) (b)

(c) (d)

(e)

Figura 24 – 5 montagens com 20 imagens exemplo de cada categoria. (a) Muito à esquerda, (b) pouco à esquerda, (c) centralizada, (d) pouco à direita e (e) muito à direita.

Fonte: Autor

6.3 APÊNDICE C - Código-fonte

6.3.1 APÊNDICE C1 - Leitura das imagens do ETHZ em formato .bin, para um formato diretamente exportável para o MATLAB

```
#include <iostream>

using namespace std;
//#include "io_sample.cpp"
#include <string>
#include <fstream>
#include <cstdlib>

#include <direct.h>
#include <io.h>
#include "Windows.h"
#include <vector>

float* read_gt(const char* fname){

        //try to read in the ground truth from a binary file
        FILE* pFile = fopen(fname, "rb");
        if(!pFile){
                //std::cerr << "could not open file " <<
                // fname << std::endl;
                return NULL;
        }

        float* data = new float[6];

        bool success = true;
        success &= ( fread( &data[0], sizeof(float), 6, pFile) == 6 );
        fclose(pFile);

        if(success)
                return data;
        else{
                delete data;
```

```
                return NULL;
        }

}

int16_t* loadDepthImageCompressed( const char* fname ){

        //now read the depth image
        FILE* pFile = fopen(fname, "rb");
        if(!pFile){
                std::cerr << "could not open file " << fname << std::endl
                return NULL;
        }

        int im_width = 0;
        int im_height = 0;
        bool success = true;

    // read width of depthmap
        success &= ( fread(&im_width,sizeof(int),1,pFile) == 1 );
        // read height of depthmap
        success &= ( fread(&im_height,sizeof(int),1,pFile) == 1 );

        int16_t* depth_img = new int16_t[im_width*im_height];

        int numempty;
        int numfull;
        int p = 0;

        while(p < im_width*im_height ){

                success &= ( fread( &numempty,sizeof(int),1,pFile) == 1 )

                for(int i = 0; i < numempty; i++)
                        depth_img[ p + i ] = 0;

                success &= ( fread( &numfull,sizeof(int), 1, pFile) == 1
                success &= ( fread( &depth_img[ p + numempty ],
                sizeof(int16_t), numfull, pFile) == (unsigned int) numfu
```

```
                p += numempty+numfull;

        }

        fclose(pFile);

        //cout << "Largura: "<< im_width << ". Altura: "
        // << im_height << "\n";

        if(success)
                return depth_img;
        else{
                delete depth_img;
                return NULL;
        }
}

void listDir(char* path)
{
        cout << _chdir((char*)path) << "\n";

        system("dir /B | findstr \".bin\" > temp.txt ");

        ifstream input( "temp.txt" );

        for( std::string line; getline( input, line ); )
        {
                //cout << line << "\n";
        std::string nomeFile = line;
        std::string nomeFileNew = nomeFile.substr(0,
        nomeFile.length()-4);
        cout << "Reading file: " << nomeFileNew << "\n";
        nomeFileNew += ".txt";
        //cout << "Writing to file: " << nomeFileNew << "\n";

        //cin >> nomeFile;
        ofstream myfile;
        //myfile.open ("example.txt");
        myfile.open(const_cast<char*>(nomeFileNew.c_str()));
```

```
        int16_t* imagem = loadDepthImageCompressed(
            (char*)nomeFile.data());

        int iterador = 0;
        while (iterador < 307200){
            myfile << imagem[iterador] << "\t";
            iterador++;

            if (iterador % 640 == 0){
                myfile << "\n";
            }
        }
        myfile.close();
        }
}

// Invocar a funcao main, substituindo o diretorio enviado para
// listDir() pelos locais das pastas do hpdb. Exs.:
// C:/Users/Juan/Documentos/UFSC/TCC/hpdb/01 para a primeira pasta,
// C:/Users/Juan/Documentos/UFSC/TCC/hpdb/02 para a segunda, etc.
int main(int argc, char** argv)
{
    // Repetir com todos os 24 diretorios do Head-Pose DataBase
    // (hpdb)
    listDir("C:/Users/Juan/Documents/UFSC/TCC/hpdb/24");
    return 0;
}
```

6.3.2 APÊNDICE C2 - Isolamento dos rostos a partir das máscaras fornecidas pelo ETHZ (Bounding boxes para as imagens de profundidade)

```
cd C:\Users\Juan\Documents\UFSC\TCC\head_pose_masks\01\;

dir **/*.png;

folder = dir('**/*.png');

max_size = length(folder);
```

```
b_boxes = zeros(max_size, 4);

%masks = zeros(480, 640, max);

%frame1 = masks(:, :, 1);

for i = 1:max_size

filename = folder(i).name;

%masks(:, :, i) = imread(filename);
mask = imread(filename);

[row, col] = find(mask);
min_c = min(col);
max_c = max(col);
min_l = min(row);
max_l = max(row);

b_box = [min_c, min_l, max_c-min_c, max_l-min_l];

b_boxes(i, :) = b_box;

end

clearvars frame1 folder filename max_size row col min_c max_c min_l max_l mas
```

6.3.3 APÊNDICE C3 - Obtenção das imagens RGB em resolução 128x128 e tons de cinza

```
cd C:\Users\Juan\Documents\UFSC\TCC\hpdb\24\;

dir **/*.png;

folder = dir('**/*.png');

max = length(folder);

%rgbs = zeros(480, 640, 3, max);
grayscale = zeros(128, 128, max);
```

```
%frame1 = rgbs(:, :, :, 1);

for i = 1:max

filename = folder(i).name;

rgb = imread(filename);
rgb = im2double(rgb);

rgb = rgb2gray(rgb);
example_cropped = imcrop(rgb, b_boxes(i, :));
fixed_size = imresize(example_cropped,[128 128]);

grayscale(:, :, i) = fixed_size;

end

 for i = 1:max
     rgbs(:, :, i+rgbs_size) = grayscale(:, :, i);
 end

 rgbs_size = rgbs_size + max;

clearvars frame1 folder filename max example_cropped fixed_size;
```

6.3.4 APÊNDICE C4 - Importação das imagens de profundidade obtidas pelo software C++ anterior, para o MATLAB

```
%% Import data from text file.

% Script for importing data from the following text file:

%

%     C:\Users\Mauro\Downloads\example.txt

%
```

```
% To extend the code to different selected data or a different text file,

% generate a function instead of a script.



% Auto-generated by MATLAB on 2018/08/27 16:13:03



%% Initialize variables.
cd C:\Users\Juan\Documents\UFSC\TCC\hpdb\01;

dir **/*depth.txt;

folder = dir('**/*depth.txt');

max = length(folder);

%frames = zeros(480, 640, max);
frames = zeros(128, 128, max);

%frame1 = frames(:, :, 1);

for i = 1:max

filename = folder(i).name;
%if (~contains(filename, "pose"))
delimiter = '\t';



%% Format for each line of text:

%   column1: double (%f)

%       column2: double (%f)

%   column3: double (%f)
```

```
%        column4: double (%f)
%    column5: double (%f)
%        column6: double (%f)
%    column7: double (%f)
%        column8: double (%f)
%    column9: double (%f)
%        column10: double (%f)
%    column11: double (%f)
%        column12: double (%f)
%    column13: double (%f)
%        column14: double (%f)
%    column15: double (%f)
%        column16: double (%f)
%    column17: double (%f)
%        column18: double (%f)
%    column19: double (%f)
%        column20: double (%f)
%    column21: double (%f)
%        column22: double (%f)
```

```
%    column23: double (%f)
%        column24: double (%f)
%    column25: double (%f)
%        column26: double (%f)
%    column27: double (%f)
%        column28: double (%f)
%    column29: double (%f)
%        column30: double (%f)
%    column31: double (%f)
%        column32: double (%f)
%    column33: double (%f)
%        column34: double (%f)
%    column35: double (%f)
%        column36: double (%f)
%    column37: double (%f)
%        column38: double (%f)
%    column39: double (%f)
%        column40: double (%f)
%    column41: double (%f)
%        column42: double (%f)
```

```
%    column43: double (%f)
%        column44: double (%f)
%    column45: double (%f)
%        column46: double (%f)
%    column47: double (%f)
%        column48: double (%f)
%    column49: double (%f)
%        column50: double (%f)
%    column51: double (%f)
%        column52: double (%f)
%    column53: double (%f)
%        column54: double (%f)
%    column55: double (%f)
%        column56: double (%f)
%    column57: double (%f)
%        column58: double (%f)
%    column59: double (%f)
%        column60: double (%f)
%    column61: double (%f)
```

```
%        column62: double (%f)
%    column63: double (%f)
%        column64: double (%f)
%    column65: double (%f)
%        column66: double (%f)
%    column67: double (%f)
%        column68: double (%f)
%    column69: double (%f)
%        column70: double (%f)
%    column71: double (%f)
%        column72: double (%f)
%    column73: double (%f)
%        column74: double (%f)
%    column75: double (%f)
%        column76: double (%f)
%    column77: double (%f)
%        column78: double (%f)
%    column79: double (%f)
%        column80: double (%f)
%    column81: double (%f)
```

```
%        column82: double (%f)
%    column83: double (%f)
%        column84: double (%f)
%    column85: double (%f)
%        column86: double (%f)
%    column87: double (%f)
%        column88: double (%f)
%    column89: double (%f)
%        column90: double (%f)
%    column91: double (%f)
%        column92: double (%f)
%    column93: double (%f)
%        column94: double (%f)
%    column95: double (%f)
%        column96: double (%f)
%    column97: double (%f)
%        column98: double (%f)
%    column99: double (%f)
%        column100: double (%f)
```

```
%    column101: double (%f)
%        column102: double (%f)
%    column103: double (%f)
%        column104: double (%f)
%    column105: double (%f)
%        column106: double (%f)
%    column107: double (%f)
%        column108: double (%f)
%    column109: double (%f)
%        column110: double (%f)
%    column111: double (%f)
%        column112: double (%f)
%    column113: double (%f)
%        column114: double (%f)
%    column115: double (%f)
%        column116: double (%f)
%    column117: double (%f)
%        column118: double (%f)
%    column119: double (%f)
%        column120: double (%f)
```

```
%    column121: double (%f)
%        column122: double (%f)
%    column123: double (%f)
%        column124: double (%f)
%    column125: double (%f)
%        column126: double (%f)
%    column127: double (%f)
%        column128: double (%f)
%    column129: double (%f)
%        column130: double (%f)
%    column131: double (%f)
%        column132: double (%f)
%    column133: double (%f)
%        column134: double (%f)
%    column135: double (%f)
%        column136: double (%f)
%    column137: double (%f)
%        column138: double (%f)
%    column139: double (%f)
```

```
%       column140: double (%f)
%   column141: double (%f)
%       column142: double (%f)
%   column143: double (%f)
%       column144: double (%f)
%   column145: double (%f)
%       column146: double (%f)
%   column147: double (%f)
%       column148: double (%f)
%   column149: double (%f)
%       column150: double (%f)
%   column151: double (%f)
%       column152: double (%f)
%   column153: double (%f)
%       column154: double (%f)
%   column155: double (%f)
%       column156: double (%f)
%   column157: double (%f)
%       column158: double (%f)
%   column159: double (%f)
```

```
%         column160: double (%f)
%    column161: double (%f)
%         column162: double (%f)
%    column163: double (%f)
%         column164: double (%f)
%    column165: double (%f)
%         column166: double (%f)
%    column167: double (%f)
%         column168: double (%f)
%    column169: double (%f)
%         column170: double (%f)
%    column171: double (%f)
%         column172: double (%f)
%    column173: double (%f)
%         column174: double (%f)
%    column175: double (%f)
%         column176: double (%f)
%    column177: double (%f)
%         column178: double (%f)
```

```
%    column179: double (%f)
%        column180: double (%f)
%    column181: double (%f)
%        column182: double (%f)
%    column183: double (%f)
%        column184: double (%f)
%    column185: double (%f)
%        column186: double (%f)
%    column187: double (%f)
%        column188: double (%f)
%    column189: double (%f)
%        column190: double (%f)
%    column191: double (%f)
%        column192: double (%f)
%    column193: double (%f)
%        column194: double (%f)
%    column195: double (%f)
%        column196: double (%f)
%    column197: double (%f)
%        column198: double (%f)
```

```
%    column199: double (%f)
%        column200: double (%f)
%    column201: double (%f)
%        column202: double (%f)
%    column203: double (%f)
%        column204: double (%f)
%    column205: double (%f)
%        column206: double (%f)
%    column207: double (%f)
%        column208: double (%f)
%    column209: double (%f)
%        column210: double (%f)
%    column211: double (%f)
%        column212: double (%f)
%    column213: double (%f)
%        column214: double (%f)
%    column215: double (%f)
%        column216: double (%f)
%    column217: double (%f)
```

```
%        column218: double (%f)
%    column219: double (%f)
%        column220: double (%f)
%    column221: double (%f)
%        column222: double (%f)
%    column223: double (%f)
%        column224: double (%f)
%    column225: double (%f)
%        column226: double (%f)
%    column227: double (%f)
%        column228: double (%f)
%    column229: double (%f)
%        column230: double (%f)
%    column231: double (%f)
%        column232: double (%f)
%    column233: double (%f)
%        column234: double (%f)
%    column235: double (%f)
%        column236: double (%f)
%    column237: double (%f)
```

```
%         column238: double (%f)
%    column239: double (%f)
%         column240: double (%f)
%    column241: double (%f)
%         column242: double (%f)
%    column243: double (%f)
%         column244: double (%f)
%    column245: double (%f)
%         column246: double (%f)
%    column247: double (%f)
%         column248: double (%f)
%    column249: double (%f)
%         column250: double (%f)
%    column251: double (%f)
%         column252: double (%f)
%    column253: double (%f)
%         column254: double (%f)
%    column255: double (%f)
%         column256: double (%f)
```

```
%    column257: double (%f)
%        column258: double (%f)
%    column259: double (%f)
%        column260: double (%f)
%    column261: double (%f)
%        column262: double (%f)
%    column263: double (%f)
%        column264: double (%f)
%    column265: double (%f)
%        column266: double (%f)
%    column267: double (%f)
%        column268: double (%f)
%    column269: double (%f)
%        column270: double (%f)
%    column271: double (%f)
%        column272: double (%f)
%    column273: double (%f)
%        column274: double (%f)
%    column275: double (%f)
%        column276: double (%f)
```

```
%    column277: double (%f)
%        column278: double (%f)
%    column279: double (%f)
%        column280: double (%f)
%    column281: double (%f)
%        column282: double (%f)
%    column283: double (%f)
%        column284: double (%f)
%    column285: double (%f)
%        column286: double (%f)
%    column287: double (%f)
%        column288: double (%f)
%    column289: double (%f)
%        column290: double (%f)
%    column291: double (%f)
%        column292: double (%f)
%    column293: double (%f)
%        column294: double (%f)
%    column295: double (%f)
```

```
%        column296: double (%f)
%    column297: double (%f)
%        column298: double (%f)
%    column299: double (%f)
%        column300: double (%f)
%    column301: double (%f)
%        column302: double (%f)
%    column303: double (%f)
%        column304: double (%f)
%    column305: double (%f)
%        column306: double (%f)
%    column307: double (%f)
%        column308: double (%f)
%    column309: double (%f)
%        column310: double (%f)
%    column311: double (%f)
%        column312: double (%f)
%    column313: double (%f)
%        column314: double (%f)
%    column315: double (%f)
```

```
%         column316: double (%f)
%    column317: double (%f)
%         column318: double (%f)
%    column319: double (%f)
%         column320: double (%f)
%    column321: double (%f)
%         column322: double (%f)
%    column323: double (%f)
%         column324: double (%f)
%    column325: double (%f)
%         column326: double (%f)
%    column327: double (%f)
%         column328: double (%f)
%    column329: double (%f)
%         column330: double (%f)
%    column331: double (%f)
%         column332: double (%f)
%    column333: double (%f)
%         column334: double (%f)
```

```
%    column335: double (%f)
%        column336: double (%f)
%    column337: double (%f)
%        column338: double (%f)
%    column339: double (%f)
%        column340: double (%f)
%    column341: double (%f)
%        column342: double (%f)
%    column343: double (%f)
%        column344: double (%f)
%    column345: double (%f)
%        column346: double (%f)
%    column347: double (%f)
%        column348: double (%f)
%    column349: double (%f)
%        column350: double (%f)
%    column351: double (%f)
%        column352: double (%f)
%    column353: double (%f)
%        column354: double (%f)
```

```
%    column355: double (%f)
%        column356: double (%f)
%    column357: double (%f)
%        column358: double (%f)
%    column359: double (%f)
%        column360: double (%f)
%    column361: double (%f)
%        column362: double (%f)
%    column363: double (%f)
%        column364: double (%f)
%    column365: double (%f)
%        column366: double (%f)
%    column367: double (%f)
%        column368: double (%f)
%    column369: double (%f)
%        column370: double (%f)
%    column371: double (%f)
%        column372: double (%f)
%    column373: double (%f)
```

```
%       column374: double (%f)
%   column375: double (%f)
%       column376: double (%f)
%   column377: double (%f)
%       column378: double (%f)
%   column379: double (%f)
%       column380: double (%f)
%   column381: double (%f)
%       column382: double (%f)
%   column383: double (%f)
%       column384: double (%f)
%   column385: double (%f)
%       column386: double (%f)
%   column387: double (%f)
%       column388: double (%f)
%   column389: double (%f)
%       column390: double (%f)
%   column391: double (%f)
%       column392: double (%f)
%   column393: double (%f)
```

```
%         column394: double (%f)
%    column395: double (%f)
%         column396: double (%f)
%    column397: double (%f)
%         column398: double (%f)
%    column399: double (%f)
%         column400: double (%f)
%    column401: double (%f)
%         column402: double (%f)
%    column403: double (%f)
%         column404: double (%f)
%    column405: double (%f)
%         column406: double (%f)
%    column407: double (%f)
%         column408: double (%f)
%    column409: double (%f)
%         column410: double (%f)
%    column411: double (%f)
%         column412: double (%f)
```

```
%    column413: double (%f)
%        column414: double (%f)
%    column415: double (%f)
%        column416: double (%f)
%    column417: double (%f)
%        column418: double (%f)
%    column419: double (%f)
%        column420: double (%f)
%    column421: double (%f)
%        column422: double (%f)
%    column423: double (%f)
%        column424: double (%f)
%    column425: double (%f)
%        column426: double (%f)
%    column427: double (%f)
%        column428: double (%f)
%    column429: double (%f)
%        column430: double (%f)
%    column431: double (%f)
%        column432: double (%f)
```

```
%    column433: double (%f)
%        column434: double (%f)
%    column435: double (%f)
%        column436: double (%f)
%    column437: double (%f)
%        column438: double (%f)
%    column439: double (%f)
%        column440: double (%f)
%    column441: double (%f)
%        column442: double (%f)
%    column443: double (%f)
%        column444: double (%f)
%    column445: double (%f)
%        column446: double (%f)
%    column447: double (%f)
%        column448: double (%f)
%    column449: double (%f)
%        column450: double (%f)
%    column451: double (%f)
```

```
%        column452: double (%f)
%    column453: double (%f)
%        column454: double (%f)
%    column455: double (%f)
%        column456: double (%f)
%    column457: double (%f)
%        column458: double (%f)
%    column459: double (%f)
%        column460: double (%f)
%    column461: double (%f)
%        column462: double (%f)
%    column463: double (%f)
%        column464: double (%f)
%    column465: double (%f)
%        column466: double (%f)
%    column467: double (%f)
%        column468: double (%f)
%    column469: double (%f)
%        column470: double (%f)
%    column471: double (%f)
```

```
%        column472: double (%f)
%    column473: double (%f)
%        column474: double (%f)
%    column475: double (%f)
%        column476: double (%f)
%    column477: double (%f)
%        column478: double (%f)
%    column479: double (%f)
%        column480: double (%f)
%    column481: double (%f)
%        column482: double (%f)
%    column483: double (%f)
%        column484: double (%f)
%    column485: double (%f)
%        column486: double (%f)
%    column487: double (%f)
%        column488: double (%f)
%    column489: double (%f)
%        column490: double (%f)
```

```
%    column491: double (%f)
%        column492: double (%f)
%    column493: double (%f)
%        column494: double (%f)
%    column495: double (%f)
%        column496: double (%f)
%    column497: double (%f)
%        column498: double (%f)
%    column499: double (%f)
%        column500: double (%f)
%    column501: double (%f)
%        column502: double (%f)
%    column503: double (%f)
%        column504: double (%f)
%    column505: double (%f)
%        column506: double (%f)
%    column507: double (%f)
%        column508: double (%f)
%    column509: double (%f)
%        column510: double (%f)
```

```
%    column511: double (%f)
%        column512: double (%f)
%    column513: double (%f)
%        column514: double (%f)
%    column515: double (%f)
%        column516: double (%f)
%    column517: double (%f)
%        column518: double (%f)
%    column519: double (%f)
%        column520: double (%f)
%    column521: double (%f)
%        column522: double (%f)
%    column523: double (%f)
%        column524: double (%f)
%    column525: double (%f)
%        column526: double (%f)
%    column527: double (%f)
%        column528: double (%f)
%    column529: double (%f)
```

```
%         column530: double (%f)
%    column531: double (%f)
%         column532: double (%f)
%    column533: double (%f)
%         column534: double (%f)
%    column535: double (%f)
%         column536: double (%f)
%    column537: double (%f)
%         column538: double (%f)
%    column539: double (%f)
%         column540: double (%f)
%    column541: double (%f)
%         column542: double (%f)
%    column543: double (%f)
%         column544: double (%f)
%    column545: double (%f)
%         column546: double (%f)
%    column547: double (%f)
%         column548: double (%f)
%    column549: double (%f)
```

```
%        column550: double (%f)
%    column551: double (%f)
%        column552: double (%f)
%    column553: double (%f)
%        column554: double (%f)
%    column555: double (%f)
%        column556: double (%f)
%    column557: double (%f)
%        column558: double (%f)
%    column559: double (%f)
%        column560: double (%f)
%    column561: double (%f)
%        column562: double (%f)
%    column563: double (%f)
%        column564: double (%f)
%    column565: double (%f)
%        column566: double (%f)
%    column567: double (%f)
%        column568: double (%f)
```

```
%    column569: double (%f)
%        column570: double (%f)
%    column571: double (%f)
%        column572: double (%f)
%    column573: double (%f)
%        column574: double (%f)
%    column575: double (%f)
%        column576: double (%f)
%    column577: double (%f)
%        column578: double (%f)
%    column579: double (%f)
%        column580: double (%f)
%    column581: double (%f)
%        column582: double (%f)
%    column583: double (%f)
%        column584: double (%f)
%    column585: double (%f)
%        column586: double (%f)
%    column587: double (%f)
%        column588: double (%f)
```

```
%    column589: double (%f)
%        column590: double (%f)
%    column591: double (%f)
%        column592: double (%f)
%    column593: double (%f)
%        column594: double (%f)
%    column595: double (%f)
%        column596: double (%f)
%    column597: double (%f)
%        column598: double (%f)
%    column599: double (%f)
%        column600: double (%f)
%    column601: double (%f)
%        column602: double (%f)
%    column603: double (%f)
%        column604: double (%f)
%    column605: double (%f)
%        column606: double (%f)
%    column607: double (%f)
```

```
%        column608: double (%f)
%    column609: double (%f)
%        column610: double (%f)
%    column611: double (%f)
%        column612: double (%f)
%    column613: double (%f)
%        column614: double (%f)
%    column615: double (%f)
%        column616: double (%f)
%    column617: double (%f)
%        column618: double (%f)
%    column619: double (%f)
%        column620: double (%f)
%    column621: double (%f)
%        column622: double (%f)
%    column623: double (%f)
%        column624: double (%f)
%    column625: double (%f)
%        column626: double (%f)
%    column627: double (%f)
```

```
%        column628: double (%f)
%    column629: double (%f)
%        column630: double (%f)
%    column631: double (%f)
%        column632: double (%f)
%    column633: double (%f)
%        column634: double (%f)
%    column635: double (%f)
%        column636: double (%f)
%    column637: double (%f)
%        column638: double (%f)
%    column639: double (%f)
%        column640: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%

%% Open the text file.
fileID = fopen(filename,'r');
```

```
%% Read columns of data according to the format.

% This call is based on the structure of the file used to generate this

% code. If an error occurs for a different file, try regenerating the code

% from the Import Tool.

dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType',
'ReturnOnError', false);


%% Close the text file.

fclose(fileID);


%% Post processing for unimportable data.

% No unimportable data rules were applied during the import, so no post

% processing code is included. To generate code which works for

% unimportable data, select unimportable cells in a file and regenerate the

% script.


%% Create output variable

example1 = [dataArray{1:end-1}];

example_cropped = imcrop(example1, b_boxes(i, :));
fixed_size = imresize(example_cropped,[128 128]);

frames(:, :, i) = fixed_size;
```

```
end

 for i = 1:max
     faces(:, :, i+faces_size) = frames(:, :, i);
 end

 faces_size = faces_size + max;

%% Clear temporary variables

clearvars filename delimiter formatSpec fileID dataArray ans i max dir da
```

6.3.5 APÊNDICE C5 - Importação dos ground-truths, ou seja, as saídas esperadas para as imagens, considerando uma rede neural de regressão

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\Juan\Documents\UFSC\TCC\hpdb\01\frame_00004_pose.txt
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2018/09/01 15:47:04


%%%%%%%%%% Para separar este ground_truth em angulos de Euler (X, Y
%%%%%%%%%% e Z), utilizar a funcao do Matlab:
%%%%%%%%%% [Z Y X] = rotm2eul(gt(1:3, :))


%% Initialize variables.

cd C:\Users\Juan\Documents\UFSC\TCC\hpdb\24;

dir **/*pose.txt;

folder = dir('**/*pose.txt');
```

```
max = length(folder);

gts = zeros(4, 3, max);

for i = 1:max

filename = folder(i).name;
delimiter = ' ';

%% Format for each line of text:
%   column1: double (%f)
%       column2: double (%f)
%   column3: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'MultipleDeli
'ReturnOnError', false);

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Create output variable
%frame00004pose = table(dataArray{1:end-1}, 'VariableNames', {'VarName1','Var
example = [dataArray{1:end-1}];
```

```
gts(:, :, i) = example;

end

for i = 1:max
    ground_truths(:, :, i+gt_size) = gts(:, :, i);
end

gt_size = gt_size + max;

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans example fold
```

6.3.6 APÊNDICE C6 - Configuração de parâmetros da CNN

```
% entrada: ETH: 128x128x15677 rostos pre-processados
% Base de dados propria: 128x128x1000 rostos pre-processados
% saida: ETH: 15677x3 angulos ZYX do rosto.
% Base de dados propria: vetor de categorias Matlab, com as
% categorias esperadas (-2, -1, 0, 1, 2).
% angulo: no ETH: primeiro, segundo ou terceiro valor do ground-truth
% (1, 2 ou 3). Na base de dados propria, apenas 1
% pctTreino: percentual das imagens para treino. Ex.: 0.7-> 70
% porcento
% pctValid: percentual de imagens para validacao. 1-pctTreino
function [XTrain, YTrain, XValidation, YValidation] = ConfigParamNN(entra

% Entrada de treinamento, 128x128 1 canal, 100 imagens
numTreino = floor(pctTreino*size(entrada, 3));
XTrain = zeros(128, 128, 1, numTreino);
YTrain = zeros(numTreino, 1);

% Permutacao entre as possiveis imagens de rosto, para obter 100 aleatori
perm = randperm(size(entrada,3));

treino=perm(1:numTreino);
valid=perm(numTreino+1:end);

% Saida esperada dos dados. ETH: angulo->eixo: 1 -> Z, 2->Y, 3->X
```

```
% Base de dados propria angulo sempre 1, devido a saida baseada em
% apenas um angulo

for i=1:numTreino
    XTrain(:, :, :, i) = entrada(:, :, perm(i));
    YTrain(i, :) = saida(perm(i), angulo);
end



% Entradas de validacao, mesmo formato (128x128, 1 canal, numValid exemplos)
XValidation = zeros(128, 128, 1, size(entrada,3)-numTreino);
YValidation = zeros(size(entrada,3)-numTreino, 1);

% Nova permutacao, para obter numValid exemplos aleatorios diferentes das im
% de treinamento

% Saidas esperadas dos exemplos de validacao
for i = 1:size(entrada,3)-numTreino
    XValidation(:, :, :, i) = entrada(:, :, perm(numTreino+i));
    YValidation(i, :) = saida(perm(numTreino+i), angulo);
end

YTrain = categorical(YTrain);
YValidation = categorical(YValidation);

%clearvars i;
```

6.3.7 APÊNDICE C7 - Definição das camadas da CNN e de suas configurações

```
layers = [
    % Camada de entrada
    imageInputLayer([128 128 1])

    % Primeira camada escondida
    convolution2dLayer(11,30,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)
```

```
    % Segunda camada escondida
    convolution2dLayer(5,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    % Terceira camada escondida
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    % Quarta camada escondida
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    % Camada final
    fullyConnectedLayer(5)
    softmaxLayer
    classificationLayer];


miniBatchSize  = 30;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('sgdm',...
    'MiniBatchSize',miniBatchSize,...
    'MaxEpochs',200,...
    'InitialLearnRate',1e-4,...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.1,...
    'LearnRateDropPeriod',50,...
    'Shuffle','every-epoch',...
    'ValidationData',{XValidation,YValidation},...
    'ValidationFrequency',validationFrequency,...
    'ValidationPatience',Inf,...
    'Plots','training-progress',...
    'Verbose',true);
```

```
net = trainNetwork(XTrain,YTrain,layers,options);
```

6.3.8 APÊNDICE C8 - Teste em tempo real para verificar as saídas da rede

```
colourVid = videoinput('kinect', 1, 'RGB_640x480'); % video normal
depthVid = videoinput('kinect', 2, 'DEPTH_640x480'); % video de profundidade

countValidos = 0;
faceDetector = vision.CascadeObjectDetector();
saida = zeros(10, 1);

for i=1:10
colourImage = getsnapshot(colourVid);
depthImage = getsnapshot(depthVid);

bbox = step(faceDetector, colourImage);
if size(bbox, 1) > 0
    if size(bbox, 1) < 2

        countValidos = countValidos + 1

        depth_crop = imcrop(depthImage, bbox(1, :));
        depth128 = imresize(depth_crop,[128 128]);

        im = depth128;
        im1 = uint8(255 * mat2gray(im));

        im1(find(im1 > 180)) = 0;

        m=median(median(im1));
        min=m-20; max=m+20;
        mask = im1 < min | im1 > max;
        im1(mask)=mean([min,max]);
        depth128=max-im1;

        saida(countValidos) = classify(net, depth128);

    end
```

```
end
end

stop(colourVid);
stop(depthVid);

clearvars bbox depth128 depth_crop depthImage depthVid faceDetector i im
```

6.3.9 APÊNDICE C9 - Captura da imagem e submissão da classificação ao FuzzyTruck em tempo real

```
colourVid = videoinput('kinect', 1, 'RGB_640x480'); % video normal
depthVid = videoinput('kinect', 2, 'DEPTH_640x480'); % video de profundid
% Viola-Jones do Matlab
faceDetector = vision.CascadeObjectDetector();
saidas = zeros(100, 1);

countValidos = 0;
t = tcpip('localhost', 4321); % conexao com o FuzzyTruck
fopen(t);
fprintf(t, 'r');
data_t = fscanf(t, '%f␣%f␣%f');
while(~isempty(data_t))
    countValidos = countValidos + 1;

    colourImage = getsnapshot(colourVid);
    depthImage = getsnapshot(depthVid);

    % Corte da imagem por Viola-Jones
    bbox = step(faceDetector, colourImage);
    if size(bbox, 1) > 0
        if size(bbox, 1) < 2



            depth_crop = imcrop(depthImage, bbox(1, :));
            depth128 = imresize(depth_crop,[128 128]);

            im = depth128;
            im1 = uint8(255 * mat2gray(im));
```

```
            im1(find(im1 > 180)) = 0;

            m=median(median(im1));
            min=m-20; max=m+20;
            mask =  im1 < min | im1 > max;
            im1(mask)=mean([min,max]);
            depth128=max-im1;
            virar = classify(net, depth128); % classificacao
            fprintf(t,'%f\n', (double(virar)-3)*15); % conversao para angulo
            fprintf(t, 'r');
            data_t = fscanf(t, '%f %f %f');
        end
    end
    saidas(countValidos, 1) = virar;
end
```