TRABALHO DE GRADUAÇÃO

Reconhecimento de Orientação Facial Utilizando Redes Neurais Convolucionais

Rafaela Sinhoroto Lima

Brasília, dezembro de 2019

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Reconhecimento de Orientação Facial Utilizando Redes Neurais Convolucionais

Rafaela Sinhoroto Lima

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Prof. Dr. Marcus V. Chaffim Costa, FGA/UnB _____
Orientador

Prof Dr. Flávio de Barros Vidal, CIC/UnB _____
Co-Orientador

Prof. Dr. Alexandre Romariz, ENE/UnB _____
Examinador externo

Prof. Dr. Cristiano Miosso, FGA/UnB _____
Examinador externo

Brasília, dezembro de 2019

FICHA CATALOGRÁFICA

LIMA, RAFAELA S.
Reconhecimento de Orientação Facial Utilizando Redes Neurais Convolucionais

[Distrito Federal] 2019.

vii, 73p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2019). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Aprendizagem de Máquina
2.Redes Neurais Artificiais
3. Redes Neurais Convolucionais
4.Estimativa de Orientação Facial

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

LIMA, R.S. (2019). Reconhecimento de Orientação Facial Utilizando Redes Neurais Convolucionais. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n^{\circ}ZZ$, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 73p.

CESSÃO DE DIREITOS

AUTOR: Rafaela Sinhoroto Lima

TÍTULO DO TRABALHO DE GRADUAÇÃO: Reconhecimento de Orientação Facial Utilizando Redes Neurais Convolucionais.

GRAU: Engenheiro ANO: 2019

Rafaela Sinhoroto Lima

SQN 314, BL E, APT 514.

70767-050 Brasília – DF – Brasil.

Dedicatória

Dedico este presente documento, primeiramente, à minha mãe, meu maior suporte fora da Universidade.

Dedico, em seguida, aos professores que me inspiraram e ensinaram ao longo de todo o curso, em especial ao Prof. Alexandre Zaghetto, cuja capacidade de educar se equipara apenas à sua habilidade videogame-ística.

Por fim, dedico a todos os amigos de curso que seguiram o caminho mecatrônico comigo, com carinho especial para aqueles com quem virei noites terminando trabalhos.

Rafaela Sinhoroto Lima

Agradecimentos

À minha mãe Sandra Lucia Sinhoroto*, pelo seu apoio incondicional durante toda a minha vida.*
Aos professores Marcus Chaffim *e* Flávio Vidal*, pela orientação e auxílio durante este presente trabalho.*
Aos meus amigos Carlos Rocha*,* Francisco Matheus *e* Pablo Santos*, pelos muitos momentos que me ajudaram ao longo do curso.*

Rafaela Sinhoroto Lima

RESUMO

Neste trabalho é apresentada uma proposta a partir do uso de Redes Neurais Convolucionais (CNNs), com o objetivo de extrair a informação de orientação facial de um usuário, a partir de uma imagem digital. O objetivo principal deste projeto é extrair informações que possam ser aplicadas como entrada em sistemas de controle (ex. sistemas de direção de cadeira de rodas motorizada, braços robóticos auxiliares de pessoas com mobilidade reduzida, entre outros). Para a solução apresentada são utilizados modelos de CNNs multicanais, as quais realizam a estimativa dos ângulos que descrevem a orientação da face do indivíduo no espaço tridimensional, as quais atingiram erros médios absolutos de 6,64°, 5,70° e 3,91° para os ângulos de guinada, inclinação e rolagem, respectivamente.

Palavras Chave: Aprendizagem de Máquina, Redes Neurais Artificiais, Redes Neurais Convolucionais, Estimativa de Orientação de Face.

ABSTRACT

In this work, a solution based on the use of Convolutional Neural Netwroks is presented, with the goal to extract the head pode information of an user, from a digital image input. The main objective of this project is to extract information that can be used as control system input (for example, a motorized wheelchairs driving system, auxiliar robotic arms for people with reduced mobility, among others). Multi-channel CNN models are used in the presented solution, which estimate the angles that describe the head pose orientation of the subject in tridimensional space, and that reached mean absolute errors of 6.64°, 5.70° and 3.91° for the yaw, pitch and roll angles, respectively.

Keywords: Machine Learning, Artificial Neural Networks, Convolutional Neural Networks, Head Pose Estimation.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

AFLW	*Annotated Facial Landmarks in the Wild*
AFW	*Annotated Faces in the Wild*
API	*Application Programming Interface*
CLR	*Cyclic Learning Rate*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
DCNN	*Deep Convolutional Neural Network*
DDR4	*Double Data Rate [4th Generation]*
DOF	*Degrees of Freedom*
FDDB	*Face Detection Data Set and Benchmark*
FPGA	*Field-Programmable Gate Array*
GPU	*Graphics Processing Unit*
IoT	*Internet of Things*
KNN	*K-Nearest Neighbours*
MAE	*Mean Absolute Error*
ML	*Machine Learning*
MLP	*Multilayer Percetron*
MSE	*Mean Squared Error*
OOM	*Out Of Memory*
PCD	Pessoa Com Deficiência
RAM	*Random Access Memory*
ReLU	*Rectified Linear Unit*
RGB	*Red Green Blue*
RMSE	*Root Mean Squared Error*
RNA	Rede Neural Artificial
SGD	*Stochastic Gradient Descent*
SVM	*Support-Vector Machine*

Capítulo 1

Introdução

O presente trabalho apresenta a UnBFace, uma rede neural convolucional projetada para estimar a orientação facial de uma usuário a partir de imagens digitais com três canais de cor RGB sem informação de profundidade. Sua estrutura é baseada na arquitetura InceptionV3, adaptada para solução de problemas de regressão ao invés de classificação, treinada com bases de dados públicas, com implementação em código livre e processamento em ambientes gratuitos de fácil acesso.

O conceito de Rede Neural, e mais amplamente de Aprendizagem de Máquina, é um conceito antigo[7]. A ideia de redes neurais como elementos de computação data de 1943, com o modelo matemático de um neurônio artificial apresentado por Warren McCulloch, um neurocientista, e Walter Pitts, um estudioso da área da lógica. Quinze anos depois, em 1968, o psicólogo estadunidense Frank Rosenblatt propôs o primeiro modelo de aprendizagem de máquina supervisionada, o *perceptron*.

Após um atraso na evolução das Redes Neurais de cerca de 15 anos[1], vários estudos realizados na década de 80 renovaram o interesse da comunidade científica na área, que persiste até hoje. Além disso, o constante avanço do poder de processamento e da capacidade de armazenamento dos computadores modernos têm permitido o uso de arquiteturas de redes neurais cada vez mais complexas com um volume de dados cada vez maior. Tais fatores têm culminado numa grande popularidade das redes neurais, em diversas áreas da computação e da engenharia, desde a manutenção de plataformas de *web search* como o Google® a aplicações no mercado financeiro.

As redes neurais têm seu grande poder computacional devido a sua estrutura maciça paralelamente distribuída somada a sua habilidade de aprender a generalizar[8]. Uma das atuais arquiteturas de redes neurais de maior sucesso são as *Convolutional Neural Netwroks* (CNN), ou Redes Neurais Convolucionais, em português. CNN's são simplesmente redes neurais que utilizam operações de convolução em pelo menos uma de suas camadas.

1.1 Descrição do Problema

A determinação da orientação no espaço tridimensional da face de uma pessoa a partir de uma imagem bidimensional é um problema não-trivial; Depende de inúmeros parâmetros e necessita demonstrar invariância a diversos fatores, de maneira a se tornar inviável projetar e desenvolver os filtros e os algoritmos capazes de realizar essa tarefa de maneira predefinida que não permita adaptações sem modificar o programa como um todo (*hard-coded*). Com isso, este trabalho busca aplicar uma rede neural convolucional profunda à solução, buscando resultados estado-da-arte a partir da escolha correta do modelo e do ajuste satisfatório dos hiper-parâmetros a partir de uma coleção suficientemente robusta e variada de imagens para treinamento e validação.

1.2 Motivação

Rotineiramente as pessoas usam a orientação da face para transmitir importante informação interpessoal, desde apontar uma direção ou um alvo com a direção da cabeça até usar gestos adicionais como acenos ou balanços para indicar confusão, consideração ou concordância[9]. O posicionamento da cabeça é um fator importante na comunicação não-verbal. Por isso, tal como reconhecimento de fala, se torna importante e relevante considerar processos capazes de estimar a orientação de uma cabeça como ferramenta útil para preencher a lacuna entre pessoas e computadores.

Uma vez que a aprendizagem de máquina e as redes neurais convolucionais se apresentam como boas ferramentas para a solução de problemas de análise de *big data* nos campos de visão computacional e reconhecimento de padrões[10], se torna atraente o seu uso para aplicações de sistemas de controle dos mais diversos tipos, como o desenvolvimento de um controle de braço robótico para Pessoas com Deficiência (PCDs) com pouca ou nenhuma mobilidade dos membros, mas com controle fino dos movimentos da cabeça, ou para o uso em interfaces usuário-máquina em *smartphones* e outros equipamentos, ou em aplicações de robótica social.

Redes neurais convolucionais, em especial as apresentadas inicialmente por LeCun *et al.*[11], possuem um desempenho de destaque na área de visão computacional. Com sua capacidade de aprender a reconhecer padrões espaciais em imagens ajustando os pesos das conexões entre os neurônios vizinhos por meio de operações de convolução, as CNNs têm se tornado cada vez mais populares em soluções de detecção, classificação e reconhecimento de objetos em imagens.

1.3 Objetivos

Dado o problema descrito na Seção 1.2 e a motivação do uso de estratégias de aprendizagem de máquina, torna-se o objetivo principal deste trabalho apresentar um projeto de rede neural convolucional capaz de extrair a informação de orientação no espaço tridimensional da face do usuário a partir de uma imagem colorida estática, para aplicação em sistemas de controle baseados em posição do rosto ou cabeça.

É possível também estabelecer objetivos específicos de cada etapa do trabalho:

- Adquirir uma base de imagens de rostos suficientemente bem-construída e variada para que seja possível ter o maior número de orientações representadas durante o treinamento, e aplicar estratégias de aumento de dados, caso necessário;
- Projetar e desenvolver uma arquitetura de rede neural convolucional profunda que seja capaz de ser treinada para estimar três ângulos de orientação (rolagem, arfagem e guinada) a partir de uma imagem estática de um rosto, sem informação adicional de profundidade;
- Relatar os resultados obtidos, expor as principais dificuldades encontradas e trazer sugestões de modificações e trabalhos futuros.

1.4 Descrição dos capítulos

- Capítulo 2: Apresentação da fundamentação teórica contendo os conteúdos necessários para entendimento da solução proposta incluindo reconhecimento de padrões, aprendizagem de máquina, redes neurais artificiais, redes neurais convolucionais e estimação de orientação facial, e a descrição de trabalhos correlatos ao projeto e sua contribuição para a área de estimação de orientação facial;
- Capítulo 3: Apresentação da solução proposta para o problema descrito, contendo as decisões de projeto realizadas ao longo do trabalho e as justificativas de tais escolhas, para a arquitetura da rede neural convolucional, a configuração de seus parâmetros, as bases de dados e as ferramentas de implementação e processamento;
- Capítulo 4: Apresentação dos resultados experimentais obtidos ao longo das diversas etapas de execução, comparando os efeitos das mudanças realizadas no desempenho da rede e justificando as escolhas transportadas entre etapas;
- Capítulo 5: Recapitulação dos processos realizados ao longo do trabalho, apresentando as principais contribuições geradas pela solução proposta e possíveis sugestões de trabalhos futuros para aprimoramento do projeto.

Capítulo 2

Revisão Bibliográfica

2.1 Fundamentação Teórica

2.1.1 Reconhecimento de Padrões

Reconhecimento de padrões é definido por Duda, Hart e Stork (2001) como "o ato de assimilar dados brutos e tomar uma ação a partir da 'categoria' do padrão que reside dentro dos dados fornecidos[12]." Sabendo que os processos de tomada de decisão de um ser humano estão relacionados ao reconhecimento de padrões, por exemplo, sabendo que o próximo movimento em um jogo de xadrez é decidido baseado no padrão atual do tabuleiro, Fukunaga (2013) afirma que o objetivo da área de estudo do reconhecimento de padrões consiste em esclarecer os complicados mecanismos dos processos de tomada de decisão e automatizar tais funções utilizando computadores[13].

2.1.2 Aprendizagem de Máquina

Segundo Shalev-Shwartz e Ben-David (2014), Aprendizagem de Máquina (do inglês *Machine Learning*, geralmente abreviado para ML), também conhecida como aprendizagem automatizada, é o desejo de programar computadores de maneira que eles possam aprender a partir das informações que lhes são fornecidas. Uma vez que o processo de aprendizagem é definido, em termos simplificados, como sendo a transformação de experiência em conhecimento, é possível descrever um algoritmo de aprendizagem como um programa que transforma a entrada dos dados de treinamento, representando a experiência, em conhecimento, que normalmente se dá na forma de outro programa de computador capaz de realizar uma tarefa[14].

Reconhecimento de padrões é um dos grandes exemplo do uso da aprendizagem de máquina. Duda *et al.* (2001) afirmam que é natural o desejo de projetar e construir máquinas que conseguem aprender a reconhecer padrões[12]. Sobretudo porque o processo de resolver os inúmeros problemas que surgem ao projetar tais sistemas nos permitem entender melhor e apreciar mais os sistemas de reconhecimento de padrões que existem na natureza, como o próprio cérebro humano.

Não são todos os problemas que necessitam de algoritmos de aprendizagem para serem resol-

vidos. Há diversas situações onde um simples comando de cálculo ou de movimentação executado repetidamente é o suficiente para chegar à solução. Por exemplo, chegar a uma aproximação satisfatória de uma integral complexa em um tempo mínimo não necessita que o computador aprenda cálculo e integração. Basta que este realize sucessivas aproximações com uma conta rápida, composta de somas e multiplicações, até alcançar uma certa precisão. Computadores modernos são extremamente rápidos e eficientes na realização de tarefas simples, mas repetitivas.

Quais seriam, então, os casos onde é necessário aplicar aprendizagem de máquina? Shalev-Shwartz e Ben-David estabelecem dois critérios em sua obra *Understanding Machine Learning: From theory to algorithms*[14] (Entendendo Aprendizagem de Máquina: Da teoria a aos algoritmos, em uma tradução livre do inglês):

- **A tarefa é muito complexa para programar:** existem inúmeras tarefas que são realizadas rotineiramente por pessoas ou animais cujo processo de execução não é compreendido de maneira profunda, e frequentemente se resume a "aprender com a experiência". Tarefas como dirigir, reconhecer falas e compreender imagens são alguns exemplos de tarefas onde algoritmos de aprendizagem alcançam resultados satisfatórios, depois de expostos a uma quantidade suficiente de exemplos de treinamento;

  Também se encontram neste caso as tarefas que estão além da capacidade humana. Tarefas relacionadas à análise de conjuntos de dados vastos e complexos se beneficiam do uso de de estratégias de ML pois algoritmos de aprendizagem são capazes de aprender a encontrar informações relevantes enterradas em arquivos de dados que são demasiadamente grandes e/ou complexos para o entendimento humano. Aprender a detectar e reconhecer padrões relevantes em conjuntos de dados extensos é um domínio promissor em diversas áreas, da astrofísica ao comércio eletrônico, em parte devido à grande capacidade de armazenamento e ao sempre crescente poder de processamento dos computadores modernos.

- **A tarefa requer adaptabilidade:** A maioria das ferramentas de programação tem sua rigidez como uma grande limitação. Uma vez que um programa foi escrito e instalado, ele permanece inalterado. Porém, existem tarefas que mudam ao longo do tempo ou de usuário para usuário. Ferramentas de ML, por serem programas cujo comportamento de adapta aos dados de entrada, oferecem uma solução para esse problema. Tarefas que se encaixam nessa descrição são, por exemplo, a decodificação de texto manuscrito, classificação de e-mails como spam/não-spam e reconhecimento de fala.

Em um cenário típico de aprendizagem de máquina se encontra uma medida de saída, que pode ser tanto quantitativa quanto categórica, a ser prevista a partir de um conjunto de características ou *features*. Encontra-se também um conjunto de dados de treinamento, que é observado em função de seu resultado e/ou da medida de suas características para um certo conjunto de objetos. A partir desses dados, se constrói um modelo de predição, ou aprendiz, que dever ser capaz de prever o resultado para novos objetos. Um bom aprendiz é aquele que prevê corretamente tal resultado[15].

É possível dividir o campo de aprendizagem de máquina em diversos subcampos de acordo

com diversos parâmetros que determinam o comportamento do algoritmo de aprendizagem. São parâmetros de classificação de tipos de aprendizagem:

- **Supervisão:** Talvez o principal eixo de distinção entre os diversos algoritmos de ML, é possível separar o campo da aprendizagem de máquina em aprendizagem supervisionada e não-supervisionada.
  - **Aprendizagem supervisionada:** Dado que o processo de aprender envolve uma interação entre o aprendiz e o ambiente onde ele se encontra, define-se aprendizagem supervisionada como o processo no qual o ambiente cumpre o papel de um professor que "supervisiona" o aprendiz ao fornecer exemplos de treino que possuem informação significativa sobre os dados (conhecida como *labels* ou rótulos). A tarefa do aprendiz passa a ser, então, aplicar o conhecimento adquirido para encontrar a informação relevante faltando nos exemplos de teste a partir dos rótulos dos exemplos de treino. Ou seja, a experiência adquirida com a aprendizagem é a de prever as *labels* que estão faltando nos dados de teste. Os rótulos que acompanham os dados de treino são chamados geralmente de alvo. Um exemplo de tarefa que requer este tipo de aprendizagem é a classificação de e-mails como spam/não-spam. O programa é alimentado uma grande variedade de e-mails rotulados como spam ou não e deve extrair desses e-mails elementos que permitem classificar um novo exemplo (um novo e-mail recebido) como spam ou não;
  - **Aprendizagem não-supervisionada:** Diferentemente da aprendizagem supervisionada, a aprendizagem não-supervisionada não possui distinção entre os casos usados para treino e os casos usados para teste. Nesta situação, o ambiente passa a ser apenas um fornecedor de dados, permitindo que o algoritmo chegue às suas próprias conclusões, sem supervisão. A experiência adquirida é, então, a capacidade de apresentar um resumo ou versão condensada dos dados para tirar alguma informação ou realizar alguma ação com os casos de teste. Um exemplo de tarefa que requer este tipo de aprendizagem é a de agrupar um conjunto de dados em subconjuntos de itens similares entre si, como, por exemplo, agrupar as cores presentes em uma imagem colorida.

  Existe ainda uma configuração intermediária de aprendizagem, chamada de *reinforcement learning* ou aprendizagem de reforço. Sutton e Barto (2018) definem aprendizagem de reforço como aprender o que fazer, como mapear situações em ações, de maneira a maximizar uma recompensa. O aprendiz não é informado de quais ações deve tomar, mas deve descobrir através de tentativa e erro quais delas retornam a maior recompensa[16]. A interação entre o aprendiz e o ambiente é, neste caso, ditada pelas necessidades do aprendiz. O aprendiz, também chamado de agente, explora o ambiente, que por sua vez fornece informações importantes para o aprendiz.

- **Influência no ambiente:** É possível classificar o esforço de aprendizagem pelo papel do aprendiz como ativo ou passivo.
  - **Aprendiz ativo:** É aquele que interage ativamente com o ambiente durante o treina-

mento, por exemplo, levantando questionamentos ou realizando experimentos buscando a informação necessária para que possa aprender;

  - **Aprendiz passivo:** É aquele que observa apenas a informação passada pelo ambiente ou professor, sem influenciá-lo ou direcionar a maneira como recebe as informações.

- **Acúmulo de dados:** Também é um parâmetro de classificação de algoritmos de aprendizagem a forma como é feita a alimentação dos exemplos de treinamento para o aprendiz. Neste caso, separam-se os algoritmos em *online* e por lote.

  - **Aprendizagem *online*:** Ocorre quando o algoritmo deve responder concomitantemente ao processo de aprendizagem, dando menor importância no início para a taxa de acerto do que para a capacidade de se manter atualizado sobre o problema a ser resolvido. Por exemplo, um algoritmo corretor de ações deve fazer decisões diárias, inicialmente com grande porcentagem de falha, mas que melhoram ao longo do tempo e da experiência adquirida;
  - **Aprendizagem por lote:** Ocorre quando o aprendiz deve primeiro processar grandes quantidades de dados para depois aplicar sua experiência adquirida na realização de uma tarefa. Este é o caso da maioria dos algoritmos de mineração de dados.

Dada a grande quantidade de parâmetros que definem modelos de ML, não é de estranhar que existam diversos algoritmos das mais variadas implementações. De treinamento supervisionado existem as representações por instância dos elementos de entrada, como é o caso dos métodos *K-Nearest Neighbours* (KNN) e *Support Vector Machine* (SVM); representações em hiper-planos, como o classificador ingênuo de Bayes e a regressão logística; representações em conjuntos de regras; redes neurais; árvores de decisão; e representações em modelos gráficos, como redes Bayesianas e campos condicionais aleatórios. Para treinamento não supervisionado, duas técnicas muito utilizadas são as análises de *cluster* ou agrupamento, e a análise dos componentes principais.

As tarefas de aprendizagem, de maneira geral, descrevem o tipo de processamento de observações de acordo com o sistema de aprendizagem aplicado. Uma observação representa um conjunto de características de um determinado objeto de estudo, enquanto cada entrada x_i representa uma característica específica deste objeto. Existem muitos tipos de processamento no campo de ML, cada um associado a um tipo de tarefa: classificação, regressão, transcrição, tradução, etc[17].

Uma vez que existem inúmeras tarefas que podem ser realizadas por meio de aprendizagem de máquina, é necessário analisar o tipo de processamento mais adequado para cada. Por exemplo, em tarefas de classificação, o sistema deve designar, para cada observação de entrada, uma das K classes disponíveis, produzindo assim a representação em função matemática $R^n \rightarrow \{1,\ ...,\ k\}$. Em tarefas de regressão, o sistema deve ser capaz de de prever um valor numérico para cada observação de entrada, produzindo a função $R^n \rightarrow R$. Ou seja, regressão e classificação são tarefas similares, diferindo apenas no formato de saída.

Para avaliar a qualidade do ajuste do modelo após seu treinamento é necessário utilizar uma medida quantitativa do desempenho, em outras palavras, é necessário quantificar o quão bem as previsões do aprendiz correspondem aos dados observados. Isso é feito medindo-se o quanto a

Tabela 2.1: Comparação do efeito da variância do erro nas métricas MAE e RMSE (Fonte: [18]).

Variável	Caso 1	Caso 2	Caso 3	Caso 4	Caso 5
e_1	2	1	1	0	0
e_2	2	1	1	0	0
e_3	2	3	1	1	0
e_4	2	3	5	7	8
$\sum \lvert e_i \rvert$	8	8	8	8	8
MAE	2	2	2	2	2
$\sum \lvert e_i \rvert^2$	16	20	28	50	64
RMSE	2,0	2,2	2,6	3,5	4,0

resposta prevista pelo modelo está distante do valor real do objeto de entrada, em um "sistema de coordenada" adequado à natureza da tarefa. Por exemplo, em tarefas de classificação se utiliza a acurácia, a capacidade do modelo de classificar corretamente as observações, ou a taxa de erros, que é a medida complementar da acurácia ou taxa de acertos.

Em tarefas de regressão, utilizam-se medidas que medem direta ou indiretamente a distância do valor previsto pelo regressor do valor real fornecido, utilizando *Mean Squared Error* (MSE), *Mean Absolute Error* (MAE), *Root Mean Squared Error* (RMSE), entre outras métricas. O cálculo do erro médio absoluto ou MAE é relativamente direto e consiste na média aritmética simples da magnitude dos erros de cada caso de teste. Já as métricas MSE e RMSE calculam a média aritmética simples do quadrado da magnitude do erro (e o RMSE finaliza calculando a raiz quadrada do erro quadrático médio). Como resultado disso, erros grosseiros, de grande magnitude, têm uma influência relativamente maior no erro total do que erros pequenos. Uma propriedade decorrente disto que é frequentemente esquecida é o fato de que tanto o MSE quanto o RMSE aumentam conforme a variância associada à distribuição de frequência das magnitudes de erro aumenta.[18]

A Tabela 2.1 ilustra essa propriedade mencionada no parágrafo anterior apresentando uma comparação entre conjuntos hipotéticos de um erros. Conforme a variação de magnitude do erro aumenta gradualmente (de zero no Caso 1 até a variação máxima no Caso 5), o valor do RMSE também aumenta.

Em um estudo conduzido em 2005 sobre as vantagens da métrica MAE sobre o RMSE na avaliação de desempenho médio de modelos, Willmott e Matsuura concluíram que medidas de erro média baseadas na soma do quadrado dos erros são função de ambos o erro médio e a distribuição da magnitude dos erros e, por isso, não descrevem apenas o erro médio do desempenho de um modelo. O valor RMSE tende a se tornar cada vez maior do que o valor do MAE conforme a distribuição de magnitudes de erro se torna mais diversa, e não necessariamente monotonamente. Isso significa que, para uma dada distribuição de valores de erro, o valor da medida RMSE pode ser exponencialmente maior do que o valor do MAE para o mesmo conjunto. Uma vez que não há nenhuma interpretação clara aparente da métrica RMSE ou de suas medidas relacionadas, Willmott e Matsuura (2005) recomendam que esta não seja mais usada na literatura, já que o erro médio absoluto (MAE) é a medida mais natural e não-ambígua da magnitude média do erro de

um modelo[18].

O problema central a ser resolvido com aprendizagem de máquina se resume a estimar os parâmetros de um modelo para que esse seja capaz de ter um bom desempenho nas métricas aplicadas em elementos estranhos aos dados fornecidos durante o treinamento. Essa capacidade de se ajustar bem a novas observações recebe o nome de capacidade de generalização, que pode ser interna ou externa. À capacidade de inferir dados que se encaixam dentro do conjunto de observações de entrada, mas que não foram vistas durante o treinamento dá-se o nome de interpolação. Já a habilidade de prever algo completamente inédito e externo às observações de entrada dá-se o nome de extrapolação. De maneira geral, é mais difícil de se obter um bom desempenho em extrapolação do que em interpolação.

Dois fatores que podem interferir na capacidade de generalização são o *overfitting*, ou sobre-ajuste, e o *underfitting*, ou sub-ajuste. O sub-ajuste ocorre quando o modelo não consegue ajustar seu parâmetros aos dados de entrada, seja pela pouca quantidade de informação fornecida ou pela inadequação do modelo ao espaço de hipótese requisitado. Por outro lado, sobre-ajuste, como o nome indica, é o problema oposto, onde os parâmetros são demasiadamente ajustados aos dados de entrada, restringindo ou até mesmo perdendo a capacidade de ajuste às novas informações. Por exemplo, tentar aproximar uma função polinomial de quarto grau por uma função linear causará sub-ajuste pela inadequação do modelo, enquanto o uso de uma função polinomial de grau 15 causará sobre-ajuste devido ao excesso de parâmetros a serem ajustados.

Por ser o foco deste trabalho, a próxima seção irá apresentar as redes neurais com a profundidade necessária para o entendimento da nomenclatura e convenções presente na literatura sobre o assunto.

2.1.3 Redes Neurais

Haykin (2004) define uma Rede Neural Artificial (RNA) como sendo um "processador paralelamente distribuído e constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso". Uma RNA se assemelha ao cérebro pelo fato de que o conhecimento é adquirido pela rede a partir de seu ambiente por meio de um processo de aprendizagem, onde forças de conexões entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.

O estudo das Redes Neurais Artificiais (RNA) é um tópico especifico da área de Inteligência Artificial, da subárea que estuda os modelos conexionistas dos quais as redes neurais artificias fazem parte. A motivação para este estudo vem da área do conhecimento chamada Neurociência que é o estudo do sistema nervoso e especialmente do cérebro. A inspiração oriunda do estudo da Neurociência e das redes neurais biológicas possibilitou numerosos avanços no desenvolvimento de sistemas inteligentes e, em especial, as RNAs vêm sendo utilizadas para resolver uma variedade de problemas tais como reconhecimento de padrões, predição, otimização, controle e outros.[1]

O uso de redes neurais oferece a um sistema computacional inúmeras propriedades úteis como não-linearidade, que permite prever resultados mais complexos do que simples funções lineares;

adaptabilidade; tolerância a falhas, entre outras.

Histórico das RNAs Em 1943, McClulloch e Pitts sugeriram um modelo matemático para o funcionamento do neurônio biológico. O neurônio lógico, nome pelo qual ficou conhecido, nada mais representa que uma célula nervosa com a propriedade de poder estar excitada ou inibida. O estado de excitação ou inibição é determinado pela chamada função de ativação. A inspiração para o desenvolvimento das chamadas Redes Neurais Artificiais, ou *Artificial Neural Networks* (ANNs) nasceu a partir do estudo das células nervosas que compõem o cérebro biológico.

Após o trabalho pioneiro de McClulloch e Pitts, uma segunda contribuição foi dada por Rosenblatt que apresentou na década de 60 o teorema da convergência dos perceptrons e, posteriormente, o trabalho de Minsky e Papert sobre as limitações da utilização de perceptrons simples. Os resultados obtidos por Minsky e Papert levaram a um desencorajamento à continuação da pesquisa e evolução das RNAs por cerca de 15 anos. Foi somente na década de 80 que vários estudos renovaram o interesse da comunidade científica na área principalmente devidos aos trabalhos de Hopfield, com a abordagem da energia, e Werbos, com o algoritmo de aprendizagem de perceptrons multicamadas (multilayer feed-forward networks), posteriormente popularizado por Rumelhart em 1986[1].

O cérebro humano Principal inspiração das redes neurais, o sistema nervoso humano pode ser representado por um sistema de três estágios, demonstrado no diagrama de blocos da Figura 2.1

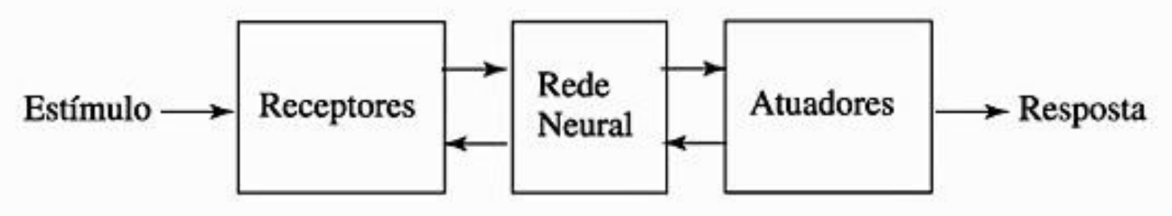


Figura 2.1: Representação em diagrama de blocos do Sistema Nervoso humano (Fonte: [19]).

O cérebro é a peça central do sistema e é representado pela rede neural. A rede neural é continuamente alimentada com informações de estímulos, que são interpretadas e usadas para produzir uma resposta adequada. Os receptores recebem os estímulos externos e os convertem em sinais elétricos capazes de serem interpretados pelo cérebro, que por sua vez envia as respostas também na forma de impulsos elétricos para os atuadores, que convertem tais impulsos em alguma saída do sistema.

A célula fundamental do sistema nervoso central é o neurônio biológico, cuja função é conduzir impulsos elétricos. O neurônio pode ser dividido em três partes: dendritos, que são finos prolongamentos ramificados cuja função é receber os estímulos dos vários neurônios vizinhos; corpo celular, que abriga as principais organelas celulares do neurônio e é responsável pelo processamento de toda a informação que chega pelos dendritos e a partir dele indicar se o neurônio deve ou não instigar um impulso elétrico junto ao seu axônio; e axônio, que é um prolongamento do corpo

celular cuja função é guiar os impulsos elétricos para outros neurônios de conexão ou de junção neuro-muscular. A terminação do axônio é composta de ramos chamados de terminais sinápticos.

A Figura 2.2 apresenta um dos tipos mais comuns de neurônios corticais, a célula piramidal, porém os neurônios aparecem em uma grande variedade de tamanhos e formas variando de acordo com sua localização no cérebro.

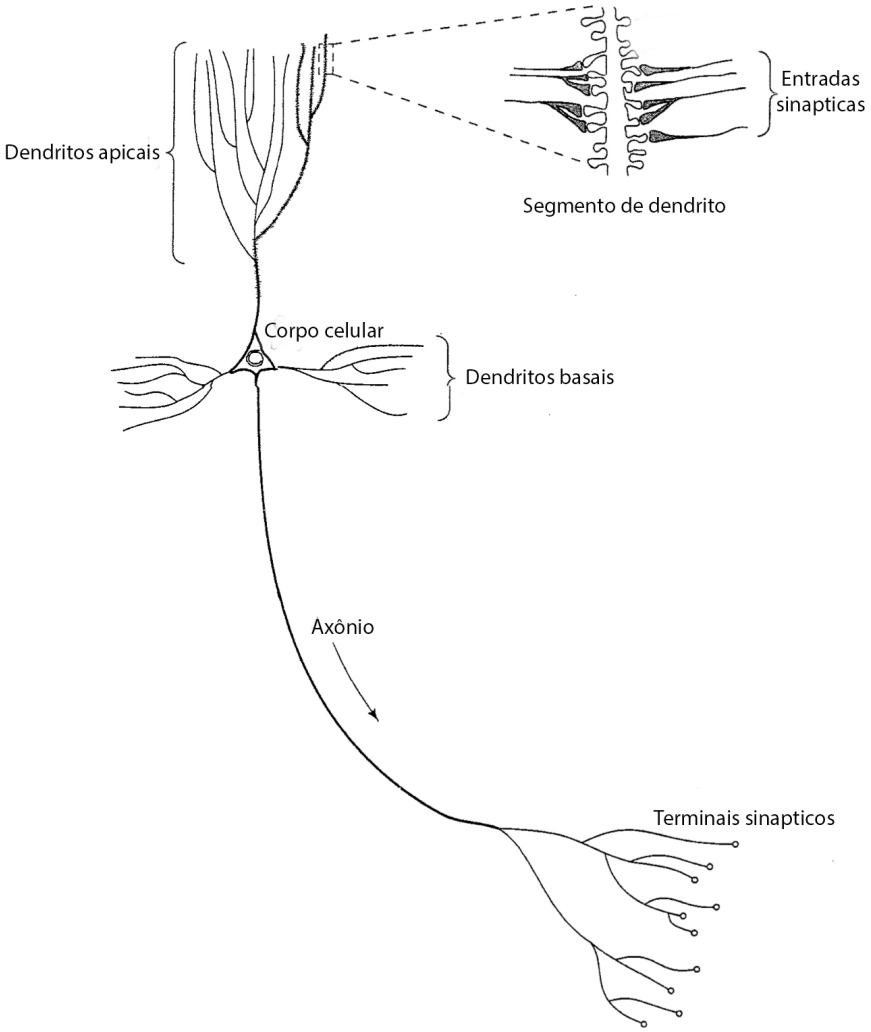


Figura 2.2: Esquema de um dos tipos de nurônio existentes no cérebro humano: a célula piramidal (Fonte: [19]).

A região de conexão entre ou neurônios é chamada de sinapse. A sinapse é uma região que permite a transmissão de um impulso elétrico de uma neurônio para outro por meio de reações químicas, e não conexões elétricas diretas como é o caso dos transistores num processador.

Diferentemente das portas lógicas que compõem os transistores dos computadores modernos, os neurônios orgânicos são lentos. Enquanto o tempo de chaveamento chaveamento de um transistor é algo em torno de um nanossegundo, o neurônio biológico chega a ser um milhão de vezes mais lento, levando em média um milissegundo. Para compensar a deficiência em velocidade, a quantidade de

Tabela 2.2: Comparação entre cérebro e computador (Fonte: [20]).

	Cérebro	**Computador**
Elementos Processadores	$\sim 10^{11}$ neurônios	$\sim 10^{9}$ transistores
Forma de processamento	Massivamente paralelo	Geralmente serial
Tipo de Memória	Associativa	Endereçada
Tempo de chaveamento	$\sim 1ms$	$\sim 1ns$

neurônios é espantosa, com muito mais conexões internas, de maneira que o processamento seja o mais efetivo possível.

A Tabela 2.2 apresenta uma breve comparação entre arquiteturas do cérebro humano e de um computador moderno. Assumindo que uma pessoa adulta qualquer reconhece um rosto em aproximadamente 0.1 segundos, a velocidade de chaveamento de um neurônio biológico de um milissegundo implica que o reconhecimento facial é feito em aproximadamente 100 passos. Em 100 passos sequenciais, um computador convencional de arquitetura von Neumann dificilmente finaliza uma tarefa complexa. Isso mostra que o processamento massivo paralelo do cérebro humano ainda é muito mais eficiente do que o processamento serial dos computadores modernos, apesar do elemento processador dos computadores ser melhor, individualmente[20].

As redes neurais artificiais buscam processar informação de maneira semelhante ao cérebro humano, com o objetivo de tornar mais eficiente a realização de tarefas de aprendizado, que ainda são custosas em tempo e recurso.

Modelo computacional do neurônio O elemento fundamental de construção das redes neurais artificias é o neurônio simples de entrada única, apresentado na Figura 2.3.

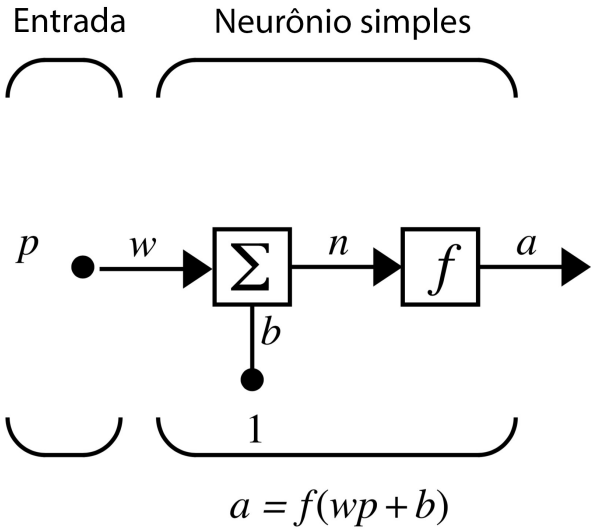


Figura 2.3: Modelo do neurônio artificial simples de entrada única (Fonte: [19]).

Há três operações distintas no modelo de neurônio artificial de entrada única. Uma entrada

escalar p é multiplicada por um peso escalar w para formar o produto wp, também escalar. Em seguida, a entrada wp é adicionada de um viés escalar b para formar a entrada n. Por fim, a entrada n passa pela função f onde é transformada em um valor de saída a, também escalar. Os nomes das três operações descritas são: função de peso, função de entrada e função de transferência.

É necessário ressaltar que w e b são ambos parâmetros ajustáveis do neurônio. A ideia de rede neural é ajustar esses parâmetros iterativamente até que a rede exiba o comportamento desejado. Assim, treina-se a rede para fazer um trabalho específico a partir do ajuste dos parâmetros de peso e/ou viés de cada neurônio.

O mesmo neurônio simples de entrada única da Figura 2.3 pode ser estendido para lidar com com entradas que são vetores. O escalar p passa a ser um vetor de entrada com R elementos (como demonstra a Figura 2.4)

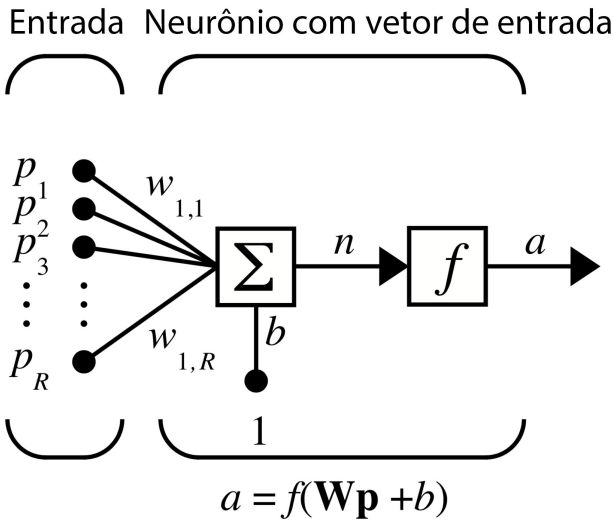


Figura 2.4: Neurônio artificial simples estendido para acomodar um vetor de entrada de tamanho R (Fonte: [19]).

Os elementos de entrada individuais $p_1, p_2, ..., p_R$ são multiplicados pelos pesos $w_{1,1}; w_{1,2}; ...; w_{1,R}$, e os valores ponderados são repassados para o bloco de função soma $\sum$, cuja saída é Wp, o produto escalar da matriz-coluna W e do vetor p. Essa saída é então adicionada do viés b para formar a entrada n da função de transferência f.

Representação matemática É possível representar o neurônio artificial por meio de um modelo matemático não linear, apresentado na Figura 2.5.

Nesta representação, o conjunto de estímulos $\{p_1, p_2, p_3, ..., p_R\}$ tem cada um de seus elementos amplificados ou atenuados pelo pesos sinápticos $w_{1,1}, w_{1,2}, w_{1,3}, w_{1,1}, ..., w_{1,R}$, produzindo uma soma ponderada

$$s = \sum_{i=1}^{R} w_{1,i} p_i \ , \tag{2.1}$$

onde o índice i indica o i-ésimo estímulo aplicado ao neurônio.

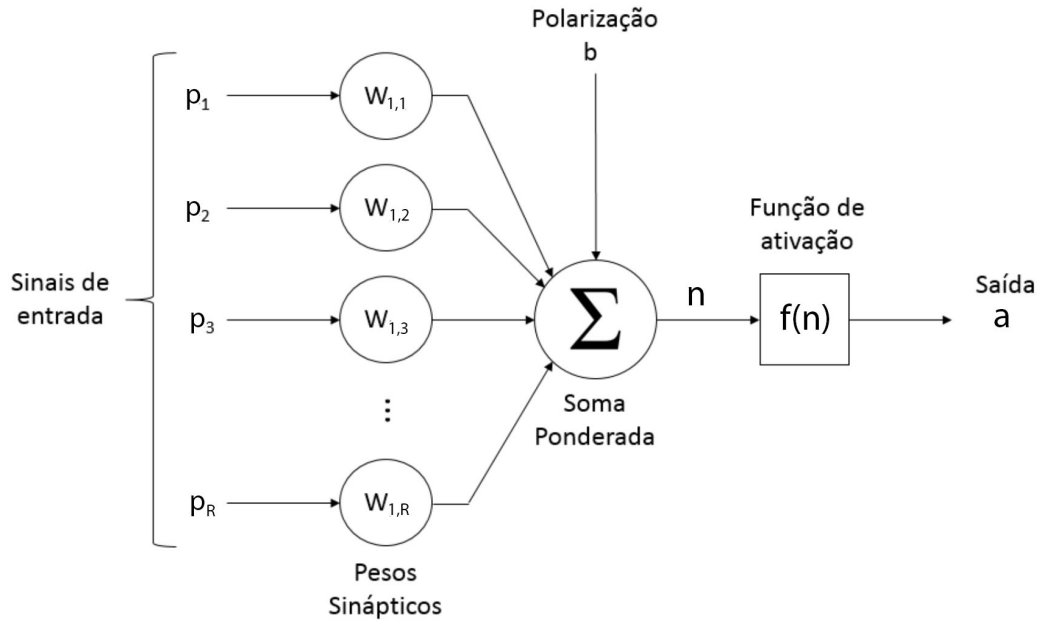


Figura 2.5: Modelo matemático de um neurônio com R sinais de entrada (Fonte: [19]).

À soma ponderada resultante da Equação 2.1 pode ser adicionado o valor b, denominado de viés, com o intuito de realizar uma polarização no neurônio. Essa polarização busca aumentar ou diminuir o valor de s com a finalidade de facilitar ou dificultar a ativação no neurônio, conforme o valor do viés.

Com isso, define-se a entrada n para a função de transferência $f(n)$ do neurônio pela Equação 2.2, abaixo:

$$n = W_p + b \ . \tag{2.2}$$

A função de transferência $f(n)$ define a saída a do neurônio. Quando o valos do viés b é nulo, $a = f(s)$, caso contrário, $a = f(n)$. Ou seja, a saída a é definida por:

$$a = \begin{cases} f(n), & se\ b\ \neq\ 0; \\ f(W_p), & se\ b\ =0. \end{cases} \tag{2.3}$$

A função de transferência $f(n)$, também chamada de "função de ativação", define a saída do neurônio em função do conjunto de entradas, da distribuição dos pesos sinápticos e do viés introduzido à entrada. Existem diversos tipos de função de transferência, e a escolha do tipo aplicado depende de fatores como a posição do neurônio na rede e o tipo de situação a ser tratada.

Por exemplo, os neurônios das camadas escondidas, ligados a outros neurônios, geralmente possuem alguma das funções de transferência clássicas descritas na Figura 2.6.

As funções clássicas da Figura 2.6 tem como característica um intervalo de saída de [0,1],

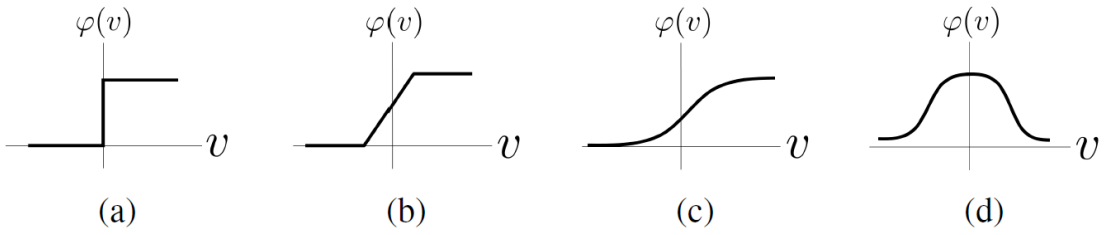


Figura 2.6: Quatro tipos clássicos de função de ativação: (a) limiar, (b) linear por partes, (c) sigmoide e (d) gaussiana (Fonte: [1]).

permitindo apenas valores positivos limitados de saída. Existem funções com intervalos variados, como [-1,1], [0,∞] ou até mesmo [$-\infty$,∞]. Exemplos de funções de ativação com esses intervalos são a tangente hiperbólica (Figura 2.7), a unidade linear retificada (*Rectified Linear Unit*, ReLU), Figura 2.8) e a linear $y = x$.

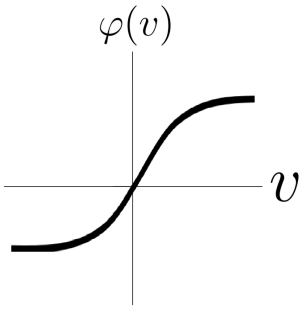


Figura 2.7: Função de ativação tangente hiperbólica (Adaptado de: [1]).

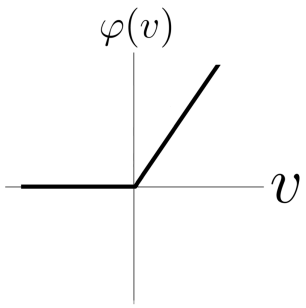


Figura 2.8: Função de ativação ReLU (Adaptado de: [1]).

Qualquer que seja a função de ativação escolhida, é desejado que esta seja diferenciável[1] em todos os pontos da entrada para realizar o processo de aprendizagem da rede, que será explicado no tópico 'Treinamento de uma Rede Neural' da Subseção 2.1.3.

Arquitetura de uma RNA Toda notação matemática apresentada até o momento diz respeito a um único neurônio. No entanto, as redes neurais artificiais projetadas para solução de problemas reais demandam que vários neurônios sejam conectados para que a rede seja capaz de aprender e armazenar conhecimento.

A arquitetura de uma RNA se refere, então, à organização da estrutura da rede: à quantidade e tipos de camadas, de conexões, de parâmetros e de neurônios ou unidades de aprendizado. As redes são organizadas em camadas, em formato de cadeia, de maneira que o valor de saída de uma camada sirva de entrada para a camada seguinte.

Por exemplo, se na primeira camada, ligada diretamente à entrada, tem-se a seguinte expressão para a saída h:

[1]Uma função é diferenciável se as derivadas laterais são definidas e iguais.

$$h^{(1)} = g^{(1)}(W^{(1)T}x_0 + b^{(1)}) \ . \tag{2.4}$$

Para a i-ésima entrada, tem-se que

$$h^{(i)} = g^{(i)}(W^{(i)T}x_{i-1} + b^{(i)}) \ , \tag{2.5}$$

em que x_{i-1} é a saída da camada anterior $h^{(i-1)}$.

Os principais tipos de arquitetura de rede neural são as redes recorrentes e as redes *feed-forward* ou alimentadas adiante. A Figura 2.9 ilustra alguns tipos comuns de arquitetura de redes neurais.

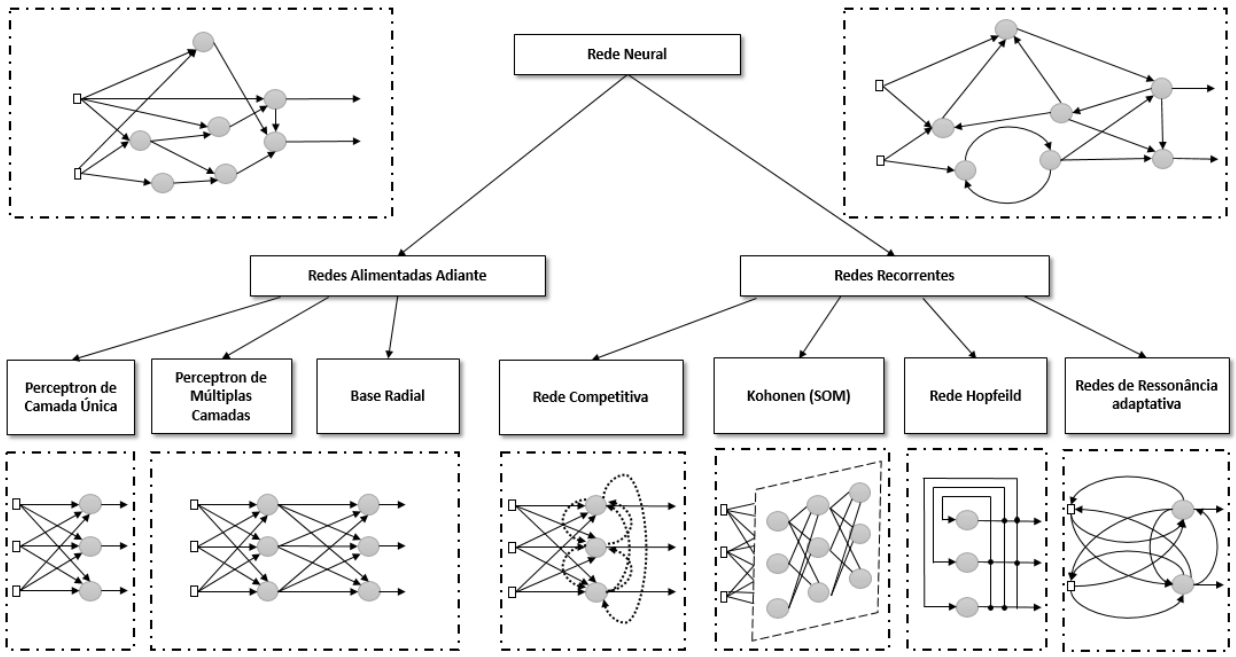


Figura 2.9: Alguns exemplos de arquiteturas alimentadas adiante e recorrentes de RNAs (Fonte: [1]).

O nome 'recorrente' refere-se às redes que possuem pelo menos uma estrutura de retroalimentação, onde uma camada anterior é alimentada pela saída de uma camada posterior a ela. Essa estrutura de rede é capaz de processar como entrada uma sequência de elementos e manter em suas camadas escondidas uma vetor que contem, implicitamente, informações sobre a história dos elementos da sequência de entrada. Esse fato torna as redes recorrentes eficientes em trabalhos informação temporal ou de posição em uma sequência, como reconhecimento de fala e texto ou detecção de movimento em vídeo. Redes recorrentes são poderosas e dinâmicas, mas em contrapartida são complicadas de serem treinadas.

O termo *feed-forward* se refere às redes que "repassam" o conhecimento adiante, sem ciclos e sem retroalimentações. A informação de entrada é apresentada por um vetor x com n elementos, modificada por cálculos intermediários e extraída ao final de acordo com o tipo de tarefa sendo realizada. O tipo de rede *feed-forward* mais comum é o *Multilayer Perceptron* (MLP), ou *perceptron*

multicamadas, em português. '*Perceptron*' é o nome dado ao algoritmo de classificação binária proposto por Frank Rosenblatt em 1958 e definido por

$$f(x) = \begin{cases} 1, & W^T x \ + \ b > 0; \\ 0, & \textit{caso contrário.} \end{cases} \tag{2.6}$$

Os MLPs consistem em n repetições da estrutura do *perceptron* ou *perceptron* de camada única, onde n indica o número de camadas escondidas.

A principal limitação do *perceptron* é a sua incapacidade de classificar padrões que não são linearmente separáveis, como é o caso do famoso detector de diferenças XOR, ou-exclusivo. A partir dessa incapacidade surge a necessidade d adição de mais camadas de neurônios. Quanto mais camadas escondidas uma rede possui, mais complexos podem ser seus problemas de classificação. Ao mesmo tempo, mais complexo se torna seu processo de treinamento. As redes alimentadas adiante com muitas camadas escondidas são denominadas redes de aprendizagem profunda.

Treinamento de uma Rede Neural Ao processo de ajustar os pesos sinápticos de uma rede neural iterativamente por meio da interação da mesma com o ambiente com os dados de entrada dá-se o nome de treinamento da rede.

A principal tarefa de uma RNA consiste na aprendizagem de um modelo no ambiente em que está inserida, com os dados que lhe são apresentados. Representa-se a aprendizagem como um conjunto de métodos que permita um aprendiz ser alimentado com dados para então estimar os parâmetros do modelo capaz de detectar as representações necessárias para a solução de um problema. Se o aprendizado for supervisionado, por exemplo, seu desempenho é estimado comparando-se a saída da rede com o verdadeiro valor.

O treinamento da rede neural consiste no processo da estimar tal modelo que minimiza o erro entre a saída do sistema e a resposta desejada, no caso do aprendizado supervisionado. Para estimar esse modelo, é preciso ajustar os pesos de forma a minimizar o erro do modelo, descrito pela função de custo, e esse ajuste é feito utilizando otimizadores iterativos baseados em gradientes que conduzem a função de custo a um valor mínimo que atenda às especificações de desempenho de um determinado projeto de aprendiz.

Bishop (2006) define a função de custo (*cost function*), também conhecida como função de perda (*loss function*) como "uma medida única global de perdas incorridas ao se tomar qualquer uma das decisões ou ações disponíveis[21]."

Existem diversas funções de custo, que variam principalmente de acordo com a tarefa a ser realizada. Para tarefas de classificação binária, é possível se utilizar funções de custo simples, como a perda $\mathrm{L}^{0\text{-}1}$, definida por

$$L^{0-1}(\mathbf{y}, f(\mathbf{x})) = \sum \begin{cases} 0, & f(x_i) \ = \ y_i; \\ 1, & f(x_i) \ \neq \ y_i. \end{cases} \tag{2.7}$$

Apesar da simplicidade da perda $\mathrm{L}^{0\text{-}1}$ (Equação 2.7) a função mais comum de custo para tarefas

de classificação binária é a função de custo logarítmica, também conhecida como *cross entropy loss*, descrita pela equação

$$L^{log}(\mathbf{y}, f(\mathbf{x})) = -\sum y_i log(f(x_i)) \ , \tag{2.8}$$

na qual $f(x_i)$ é uma estimativa de probabilidade (ou seja, um valor entre 0 e 1) gerada por uma função de ativação de limiar, sigmoide ou similar (Figura 2.6). Para classificação em N classes, calcula-se a perda logarítmica para cada classe independentemente e soma-se o resultado.

Para redes que solucionam problemas de regressão, como é o caso deste presente trabalho, uma função de perda comum é a perda erro quadrático, denominada de L^{SE} e dada por

$$L^{SE}(\mathbf{y}, f(\mathbf{x})) = \sum (y_i \ - \ f(x_i))^2 \ , \tag{2.9}$$

em que $f(x_i)$ é o valor previsto pela rede para o i-ésimo elemento de entrada.

Muitas vezes a perda erro quadrático é estendida para a função perda erro quadrático médio ($\mathrm{L}^{\mathrm{MSE}}$), onde o valor da perda é dividido pelo número de elementos de entrada.

As funções de custo descritas acima calculam o erro relativo apenas à última camada (ou única camada, caso seja o caso) da rede neural. Para treinar redes com múltiplas camadas utiliza-se o algoritmo de treinamento *back-propagation* proposto por Rumelhart e McClelland em 1986, ou um de seus derivados modernos.

A ideia básica do algoritmo de *back-propagation* possui duas etapas:

1. Propagação para frente: Uma entrada $\mathbf{x}$ é apresentada à rede e essa os efeitos dessa informação inicial são propagados pelas camadas da rede até atingir a camada de saída, produzindo um valor ou conjunto de valores de saída $\hat{y}$ e um valor de custo $L(\hat{y})$. Durante essa etapa os valores das sinapses e dos vieses são mantidos constantes;

2. Retro-propagação do erro ou propagação para trás: etapa que dá nome ao algoritmo, a informação de custo é propagada por toda a rede no sentido contrário, realizando o ajuste dos valores das sinapses e vieses de acordo com o gradiente, buscando melhorar a resposta da rede e diminuir o custo ou erro.

Matematicamente, observa-se a Figura 2.10 para acompanhamento da definição das equações.

A propagação tem início com o cálculo do chamado campo local induzido, $u_j(m)$, para cada neurônio da camada escondida, de acordo com a equação

$$u_j(m) = \sum_{i=0}^{N1} w_{j,i}(m) x_i(m) \ , \tag{2.10}$$

na qual j é o j-ésimo neurônio da camada oculta, $x_i(m)$ é o i-ésimo elemento do vetor de entrada durante a iteração m, $w_{j,i}(m)$ é o valor do peso da conexão do elemento i da entrada com o neurônio j da camada oculta e N1 é o número total de elementos do vetor de dados de entrada.

A adição do viés na Equação 2.10 se dá quando $i = 0$, que define $x_0 = 1$ e $w_{j,0}(m) = b_j(m)$, este último a definição do viés do neurônio j. A partir disso, a saída do neurônio j é definida pela

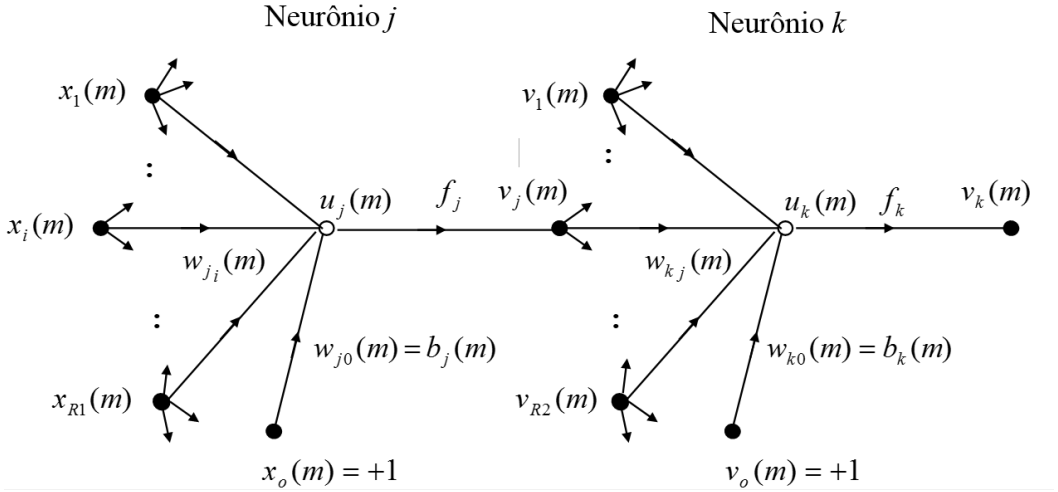


Figura 2.10: MLP com um vetor de entrada, uma camada escondida e uma camada de saída para detalhamento matemático do algoritmo de *back-propagation* (Fonte: [1]).

equação

$$v_j(m) = f_j(u_j(m)) \ , \tag{2.11}$$

na qual $f_j()$ é a função de ativação do neurônio j.

De maneira similar à apresentada pela Equação 2.10, o campo local induzido para o neurônio k da camada de saída é dado por

$$u_k(m) = \sum_{j=0}^{N2} w_{k,j}(m) v_j(m) \ , \tag{2.12}$$

em que $v_j(m)$ é a saída do neurônio j da camada escondida na iteração m, $w_{k,j}(m)$ é o valor do peso da conexão entre o neurônio j da camada oculta e o neurônio k da camada de saída e N2 é o número de neurônios da camada escondida.

Novamente a adição do viés ocorre quando $j = 0$, $v_0 = 1$ e $w_{k,0}(m) = b_k(m)$, este último o viés do neurônio k da camada de saída. A saída do neurônio k é dada por

$$v_k(m) = f_k(u_k(m)) \ , \tag{2.13}$$

em que $f_k()$ é a função de ativação do neurônio k.

Para este exemplo, o custo será dado pelo erro simples, definido na Equação 2.14:

$$e_k(m) = y_k(m) - v_k(m) \ , \tag{2.14}$$

na qual $y_k(m)$ é o resultado esperado da saída e $v_k(m)$ a saída obtida pelo processo de propagação para frente, que se encerra após este cálculo.

Uma vez obtido o erro, é necessário retro-propagá-lo, ajustando as sinapses e vieses da rede. O primeiro passo para realizar a retro propagação dos erros é calcular os gradientes locais de cada neurônio.

Para cada neurônio da camada de saída os gradientes são dados por

$$\delta_k(m) = f'_k(v_k(m))e_k(m) \ , \tag{2.15}$$

e para cada neurônio da camada escondida os gradientes são dados pela equação

$$\delta_j(m) = f'_j(v_j(m)) \sum_{k=1}^{N3} \delta_k(m) w_{k,j}(m) \ , \tag{2.16}$$

na qual N3 é a quantidade de neurônios da camada de saída.

Uma vez calculados os gradientes locais, estes são utilizados para calcular a atualização das sinapses e bias. A atualização dos valores das sinapses que ligam os neurônios da camada escondida aos neurônios da camada de saída é realizada segundo a equação

$$w_{k,j}(m+1) = w_{k,j}(m) + \eta v_j(m) \delta_k(m) \ , \tag{2.17}$$

e, similarmente, a atualização das sinapses que ligam os sinais da camada de entrada aos neurônios da camada escondida é dada por

$$w_{j,i}(m+1) = w_{j,i}(m) + \eta x_i(m) \delta_j(m) \ . \tag{2.18}$$

Finalmente, os vieses da camada escondida e da camada oculta são atualizados, respectivamente, pela Equação 2.19 e pela Equação 2.20:

$$b_k(m+1) = b_k(m) + \eta \delta_k(m) \ ; \tag{2.19}$$

$$b_j(m+1) = b_j(m) + \eta \delta_j(m) \ . \tag{2.20}$$

O termo η é a taxa de aprendizagem e $f'()$ é a diferenciação em relação ao argumento. Para que a rede neural melhore sua performance é necessário que vários conjuntos de entradas padrão sejam inseridos na rede e que a propagação e a retro-propagação ocorram diversas vezes. Por fim, um critério de parada deve ser criado, visto que o resultado ótimo pode não ser encontrado. Um exemplo de critério de parada é, por exemplo, quando a taxa absoluta da variação do erro médio quadrático para um conjunto de padrões de entrada ter alcançado um valor mínimo desejado.

2.1.4 Redes Neurais Convolucionais

Uma Rede Neural Convolucional, ou CNN, do original em inglês, é definida por Goodfellow, Bengio e Courville (2016) como uma "rede especializada no processamento de dados com topologia semelhante a grades[8]." Por exemplo, dados de entrada que são imagens apresentam sua informação por meio de grades 2D de pixeis.

O conceito de Rede Neural Convolucional pode ser datado do final da década de 1980, quando LeCun *et al.* (1989) publicaram um artigo no qual estava proposto que boas generalizações de tarefas complexas podem ser obtidas a partir do projeto de uma arquitetura de rede que contém uma

certa quantidade de conhecimento prévio sobre tais tarefas[22]. Para exemplificar essa afirmação, os autores aplicaram o algoritmo de *back-propagation* no treinamento de uma rede multicamadas para a solução de um problema real de reconhecimento de dígitos manuscritos contidos nos códigos postais dos envelopes que circulavam pelos correios dos Estados Unidos. Embora não fosse o primeiro estudo sobre o problema, foi um dos primeiros a apresentar o diferencial da alimentação da rede ser feita diretamente com imagens, e não com vetores de características.

O nome convolucional se refere a operação matemática realizada neste tipo de rede. Ou seja, uma rede neural convolucional é simplesmente uma rede neural que utiliza a operação de convolução

$$s(t) = (x * w)(t) = \int x(a)w(t-a)d(a) \tag{2.21}$$

no lugar da multiplicação de matrizes em pelo menos uma de suas camadas. A convolução é definida matematicamente como um operador linear aplicado a duas funções cujo objetivo é medir uma terceira área originada da sobreposição dessas em função do deslocamento entre elas.

Por se tratar do ambiente discreto do processamento das máquinas, é comum se utilizar o operador convolucional discreto da Equação 2.22:

$$s[T] = \sum_{a=-\infty}^{\infty} x[a]w[T-a] \ . \tag{2.22}$$

Nas redes neurais, o primeiro argumento se refere às entradas da rede, enquanto o segundo argumento se refere ao *kernel* ou filtro de convolução. A saída das camadas convolucionais é chamada de mapa de características[8].

Além disso, nas aplicações de aprendizagem de máquina, tanto a entrada quanto o *kernel* são normalmente matrizes multidimensionais. Por esse motivo, as operações de convolução em CNNs são geralmente aplicadas em mais de um eixo por vez. A Figura 2.11 demonstra a ideia da convolução em mais de um eixo simultaneamente.

Em uma imagem bidimensional de entrada I, é aconselhável aplicar um filtro K que também seja bidimensional. A operação de convolução passa a ser, então, descrita pela Equação 2.23.

$$S[i,j] = (I * K)[i,j] = \sum_{m}\sum_{n} I[m,n]K[i-m, j-n] \tag{2.23}$$

O termo 'convolucional' tornou-se popular em 1998, com a publicação do artigo *Gradient-Based Learning Applied to Document Recognition*[11] (Apredizagem Baseada em Gradiente Aplicada ao Recohecimento de Documentos, em uma tradução livre do título original), escrito por Yann LeCun, Léon Bottou, Yoshua Bengio e Patrick Haffner. Nele, LeCun *et al.* explicam que a operação aplicada à extração das *features* da imagem para armazenamento nos mapas de características é equivalente à operação de convolução, por isso o nome Rede Convolucional. Nesse caso, o filtro da operação de convolução é o conjunto de pesos sinápticos utilizados pelas unidades dos mapas de características.

Segundo LeCun *et al.* (1998), Redes Neurais Convolucionais combinam três ideias de arquite-

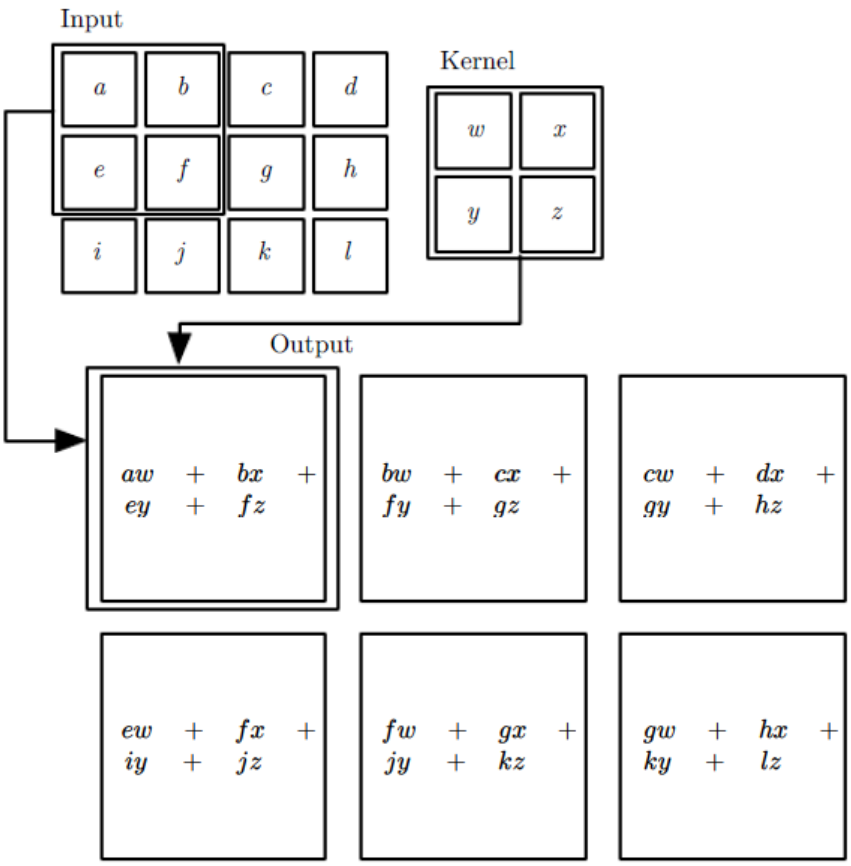


Figura 2.11: Um exemplo de uma operação de convolução 2D com a saída restrita a posições do filtro inteiramente contido na imagem, chamada de "convolução válida" (Fonte: [8]).

tura para garantir algum grau de invariância a variações de escala, deslocamento e distorção[11]: campos receptivos locais, replicação de pesos e sub-amostragem temporal ou espacial, de acordo com o dado de entrada.

Os autores descrevem o funcionamento de uma rede convolucional típica, denominada LeNet-5, para explicar tais ideias de arquitetura. A LeNet-5 recebe como entrada imagens de caracteres aproximadamente normalizados no tamanho e centralizados. Cada unidade em uma camada recebe o informações de um conjunto de unidades localizadas em uma pequena seção da camada anterior, chamado de campo receptivo local. Com estes campos, os neurônios conseguem extrair características visuais elementares como bordas anguladas, terminações de linhas, cantos, entre outras. Essas características elementares são então combinadas nas camadas subsequentes para detectar características de mais alto nível[11].

Unidades em uma camada são organizadas em planos dentro dos quais todas as unidades compartilham o mesmo conjunto de pesos sinápticos, e o conjunto das saídas das unidades desses planos é chamda de mapa de características. Uma camada convolucional completa é composta de vários mapas de características, cada um com um vetor de pesos específico e distinto dos outros, de maneira que múltiplas características possam ser extraídas em cada local.

Ainda no caso da LeNet-5, cada unidade em um mapa de característica tem 25 entradas conectadas a uma região de tamanho 5 por 5 da entrada ou camada anterior. Essa área é denominada campo receptor daquela unidade. Os campos receptores de unidades vizinhas possuem seus centros em unidades vizinhas correspondentes na camada anterior, o que causa sobreposição entre esses campos. Adicionalmente, uma vez que todas as unidades em um mapa de características compartilham o mesmo conjunto de 25 pesos, elas detectam a mesma característica em todos os pontos possíveis da entradas. Outros mapas de características da mesma camada, com outros conjuntos de pesos, extraem outras características locais, e por isso uma camada convolucional completa é composta por mais de um campo receptivo.

LeCun *et al.* afirmam ainda que, uma vez que a característica foi detectada, sua localização exata perde importância, podendo até prejudicar o processo de identificação do padrão em outras instâncias de entrada. Uma maneira simples de reduzir a precisão com a qual a posição de características distintas estão codificadas em um mapa de características é reduzir a resolução espacial de tal mapa, por meio do uso de camadas de sub-amostragem. As camadas de sub-amostragem realizam uma operação de média local e uma passagem de filtro com passo maior que um, diminuindo a resolução do mapa de características e, consequentemente, a sensitividade da saída a deslocamentos e distorções.

Para Goodfellow, Bengio e Courville (2016), as propriedades que tornam mais eficiente o processamento dos dados de entrada nas camadas convolucionais são: as interações esparsas, o compartilhamento de parâmetros e a representação equivariante.

Interações esparsas: Tradicionalmente, as camadas das redes neurais utilizam multiplicação matricial para descrever a interação doa entrada com os pesos sinápticos, de maneira que cada unidade da entrada de uma camada interaja com cada unidade da saída da camada anterior e vice-versa. Esse método não escala bem para imagens completas de entrada. Por exemplo, para uma rede treinada utilizando a base de dados CIFAR-10, com formato de entrada 32x32x3 (uma imagem colorida com 32 pixeis de altura por 32 pixeis de largura com três canais de cor RGB), um único neurônio completamente conectado na primeira camada escondida de uma RNA teria, sozinho, 3072 parâmetros ajustáveis. Esse valor rapidamente se torna intratável conforme as dimensões da imagem de entrada crescem (uma imagem real pequena, de 200x200x3, geraria 120000 pesos para o mesmo neurônio descrito anteriormente).

Redes convolucionais, porém, possuem tipicamente interações esparsas, também chamadas de conectividade esparsa ou pesos esparsos. Isso é alcançado utilizando um *kernel* com dimensões menores do que as dos dados de entrada. Por exemplo, a imagem de entrada pode ter algo em torno dos milhares ou até milhões de pixeis, mas é possível detectar informações pequenas importantes, como bordas, utilizando filtros que ocupam apenas dezenas ou centenas de pixeis. Isso significa que menos parâmetros necessitam ser armazenados, o que tanto reduz os requerimentos de memória do modelo, quanto melhora sua eficiência, já que o processo de gerar a saída passa a ser concluído com menos operações.

Compartilhamento de parâmetros: Em uma RNA tradicional, cada elemento de uma matriz de pesos é usado exatamente uma vez ate a camada de saída. Esse peso é multiplicado uma

vez por um elemento de entrada e nunca é revisitado. A ideia do compartilhamento de parâmetros, também chamada de "pesos atados", consiste em amarrar o valor do peso aplicado a uma entrada ao valor do mesmo peso aplicado em outro lugar, de modo a utilizar o mesmo parâmetro para mais de uma função no modelo. Nas redes convolucionais, cada elemento do *kernel* é usado em todas as posições de entrada, e o compartilhamento dos pesos usado pela operação de convolução implica que o modelo aprende apenas um conjunto de parâmetros global, ao invés de aprender um conjunto separado para cada local. Isso reduz os requisitos de armazenamento do modelo e torna as redes convolucionais mais eficientes no requerimento de memória, além de reduzir a possibilidade de ocorrer o sobre-ajuste.

Representação equivariante: Como consequência do compartilhamento de parâmetros, CNNs são equivariantes. Uma função é equivariante se, dada uma modificação na entrada, a saída da função obedece à mesma modificação. Matematicamente, $f(x)$ é equivariante a uma função $g(x)$ se $g(f(x)) = f(g(x))$. Em uma rede convolucional, se $g(x)$ é uma função qualquer que aplica uma transformação de translação à entrada, então a função de convolução é equivariante a g, o que fornece à rede invariância a aspectos irrelevantes da imagem, como translação da posição de elementos, condições de cor e iluminação, entre outros.

Outra característica que difere as redes convolucionais das redes neurais artificiais clássicas é a presença de camadas de *pooling* ou agrupamento. As camadas de agrupamento tem como tarefa realizar a sub-amostragem, de forma a reduzir progressivamente o tamanho espacial da representação para diminuir a quantidade de parâmetros computados e fornecer um certo grau de invariância a deslocamento.

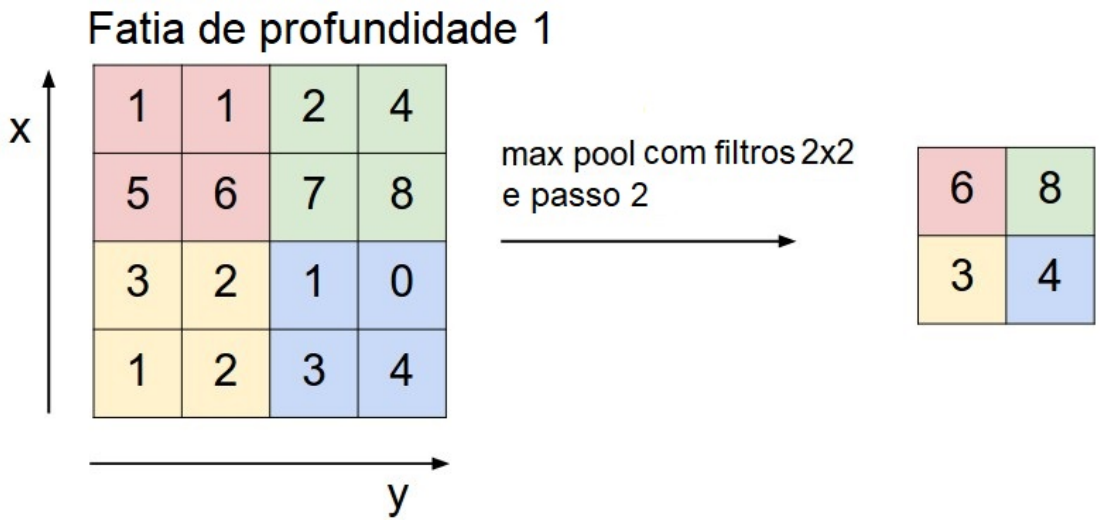


Figura 2.12: Operação de *pooling* ou agrupamento por valor máximo.

As funções de agrupamento ou *pooling* substituem as saídas da rede em um certo local por um valor que seja representativo desse subconjunto. Os tipos de operação de agrupamento que existem são as de valor máximo e de valor médio. Uma camada de *max pooling* (Figura 2.12) resume uma vizinhança retangular ao valor máximo presente na mesma, enquanto uma camada de *average pooling* substitui uma área retangular por um único valor equivalente à média aritmética simples

de todos os elementos.

As reduções realizadas pelas camadas de agrupamento melhoram a eficiência computacional da rede e diminuem a memória necessária para armazenamento dos parâmetros, já que cada cama seguinte a uma camada de *pooling* possui um número reduzido de entradas para processar.

Uma vez que todos pesos de uma rede neural convolucional, das conexões densas, das camadas de convolução e das camadas de subamostragem, são aprendidos com o método de *back-propagation*, é possível considerar que as CNNs sintetizam seu próprio extrator de características (LeCun 1998), evitando que esse trabalho seja feito manualmente pelo programador.

2.1.5 Estimação de Orientação de Face

Desde cedo, é possível observar nos seres humanos a habilidade de interpretar a orientação e o movimento da cabeça de uma pessoa rapidamente e sem esforço, permitindo que se infira as intenções dos outros que estão próximos e que se compreenda uma importante forma de comunicação não verbal. A facilidade com que alguém realiza essa tarefa disfarça e esconde a real dificuldade de um problema que vem desafiando sistemas computacionais há décadas. Segundo Murphy-Chutorian e Manubhai (2008), no contexto do campo de visão computacional, *head pose estimation* (estimativa de orientação de face, em uma tradução livre do inglês) consiste no processo de inferir a orientação espacial de uma cabeça humana a partir de imagens digitais[9].

Assim como outras tarefas de processamento de visão facial, um estimador ideal de orientação de face deve demonstrar invariância a diversos fatores modificadores de imagem, incluindo tanto fenômenos físicos, como distorção da câmera e geometria projetiva, como a aparência biológica, expressão facial e a presença de acessórios como óculos e chapéus.

Graus de Liberdade Apesar de parecer um problema simples de se visualizar, há diversas maneiras de se interpretar a frase "estimativa de orientação de face". Em seu nível mais grosseiro, o processo de *head pose estimation* se atenta a algoritmos que identificam a cabeça em uma categoria dentre poucas orientações discretizadas, por exemplo, um rosto de frente para a câmera em oposição a um rosto de perfil, tanto esquerdo quanto direito. Do outro lado do espectro, uma estimativa fina de orientação de face pode se tornar uma medida contínua de angulação atravessando múltiplos graus de liberdade (DOF, do inglês *Degrees of Freedom*). Um sistema que estima apenas um grau de liberdade, abrangendo o movimento horizontal da esquerda para a direita, ainda assim é considerado um estimador de orientação de face tanto quanto a abordagem mais complexa que estima uma orientação e posição complexa em 3D enquanto incorpora DOF adicionais como movimento dos músculos faciais e da mandíbula.

Enquanto no contexto de visão computacional a estimativa de orientação de face é mais comumente interpretada como a habilidade de inferir o posicionamento da cabeça de uma pessoa relativo à vista de uma câmera, de maneira rigorosa, *head pose estimation* se refere à habilidade de inferir a orientação de uma cabeça relativa a um sistema de coordenadas global.

A amplitude de movimento de cabeça de um homem adulto abrange flexão e extensão sagital

(que pode ser vista de perfil, neste caso, movimento do queixo para cima e para baixo) de -60,4° a 69,6°, uma inclinação frontal lateral (dobra do pescoço e angulação da cabeça para a esquerda ou para a direita) de -40,9° a 36,6° e uma rotação axial horizontal (rotação da cabeça da direita para a esquerda) de -79,8° a 75,3°, de acordo com um estudo conduzido por Ferrario *et al.*, em 2002[23].

Há uma ambiguidade na combinação entre rotação muscular e orientação relativa que é frequentemente omitida em tarefas de estimativa de orientação de cabeça. Murphy-Chutorian e Trivedi (2008) destacam que a vista de perfil de uma cabeça não parece a mesma quando a câmera captura a imagem pela lateral comparado com quando a câmera captura a imagem de frente e a cabeça está virada para o lado[9].

Figura 2.13: Visão de perfil pela posição da câmera (Fonte: [2]).

Figura 2.14: Visão de perfil pela movimentação da cabeça (Fonte: [3]).

A ambiguidade surge do fato de não haver uma certeza absoluta da posição da câmera no sistema de posicionamento global. A Figura 2.13 e a Figura 2.14 demonstram esse efeito. Em ambas as imagens, a posição da cabeça em relação à câmera é praticamente idêntica, mas a diferença no pescoço e nos ombros indica uma diferença fundamental no sistema de coordenada global, à qual o corpo adere.

Apesar desse problema, geralmente presume-se que a cabeça humana pode ser modelada como um objeto rígido separado do corpo. Sob essa suposição, a cabeça humana fica limitada a três graus de liberdade, que podem ser caracterizados por três ângulos rolagem (*roll*), inclinação(*pitch*) e guinada (*yaw*), retratados na Figura 2.15.

Métodos de estimativa de orientação de face Murphy-Chutorian e Manubhai (2008) dividem os métodos existentes de *head pose estimation* em oito categorias organizadas em função da abordagem fundamental que constitui a base de sua implementação[9]. São elas:

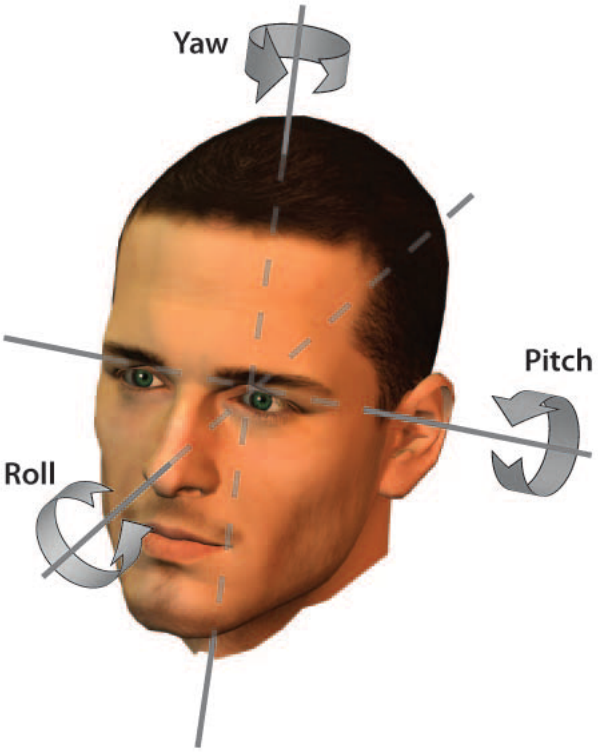


Figura 2.15: Os três graus de liberdade de uma cabeça humana podem ser descritos pelos três ângulos de rotação rolagem (do inglês *roll*), inclinação ou arfagem (do inglês *pitch*) e guinada (do inglês *yaw*) (Fonte: [9]).

- Métodos de *template* de aparência:

  São os métodos que comparam uma nova imagem de uma cabeça com um conjunto de exemplos devidamente rotulados com um valor discreto de orientação, em busca da vista mais similar;

- Métodos de vetor de detecção:

  São os métodos que treinam uma série de detectores de cabeça sintonizados em uma orientação específica e designam uma orientação discreta ao detector com maior suporte;

- Métodos de regressão não-linear:

  São os métodos que utilizam ferramentas de regressão não-linear para desenvolver um mapeamento funcional do espaço da imagem de entrada para o espaço das medidas de orientação de cabeça (Figura 2.16);

- Métodos de incorporação de espaço:

  São os métodos que procuram espaços de baixa dimensão que modelam as variações contínuas na orientação da cabeça. Novas imagens podem ser embutidas nesses espaços e então usadas para regressão incorporadas ou *template matching*;

- Métodos flexíveis:

São os métodos que realizam o encaixe de um modelo não-rígido à estrutura facial de cada individuo no espaço das imagens. A orientação da cabeça é então estimada a partir de comparações a nível de características ou *features*, ou a partir da instanciação de parâmetros do modelo;

- Métodos geométricos:

  São os métodos que utilizam a localização de pontos de referência como os olhos, a boca e a ponta do nariz para determinar a orientação da cabeça a partir de sua configuração relativa;

- Métodos de rastreamento:

  São os métodos que recuperam a mudança de posição global da cabeça a partir da observação do movimento desta entre quadros (*frames*) de um vídeo;

- Métodos híbridos:

  São métodos que combinam uma ou mais das estratégias mencionadas anteriormente para superar as limitações inerentes a qualquer abordagem simples e individual.

As CNNs pertencem à categoria dos métodos de regressão não-linear, que aprendem a estimar uma posição facial por meio do aprendizado de um mapeamento funcional não linear do espaço da imagem para o espaço de um ou mais ângulos de orientação, de acordo com a quantidade de DOFs.

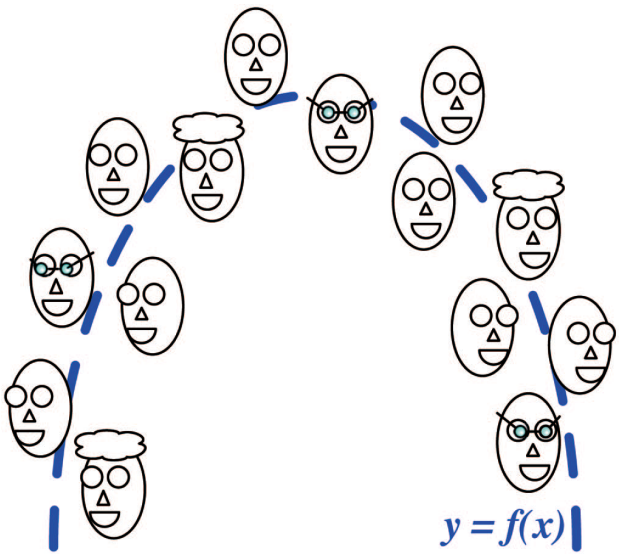


Figura 2.16: **Regressão não-linear** é um método que proporciona um mapeamento funcional do espaço da imagem ou dos dados para o espaço de medidas de orientação de cabeça (Fonte: [9]).

O principal atrativo desses métodos é o fato de que, dado um conjunto de dados de treinamento devidamente rotulados, se torna possível construir uma modelo que irá proporcionar uma

estimativa de orientação discreta ou contínua para qualquer novo exemplar de entrada. A limitação dessa abordagem é que não é óbvio ou mesmo fácil de se perceber o quão bem uma ferramenta específica de regressão será capaz de aprender o mapeamento correto.

Das ferramentas de regressão não-linear usadas para estimar posição de cabeça, as redes neurais têm sido as mais utilizadas e divulgadas na literatura. Um exemplo é o MLP, que pode ser treinado por algoritmos de *backpropagation* e é capaz de estimar a orientação de uma cabeça a partir de recortes de imagens de uma cabeça, que não é necessariamente a mesma da que se busca a posição estimada.

O MLP também pode assumir diferentes configurações. Por exemplo, os nós de saída do MLP podem corresponder a posições discretizadas, e neste caso um *kernel* gaussiano pode ser utilizado para suavizar as orientações de treinamento para atender às semelhanças de posições próximas. Apesar do sucesso dessa estratégia, ela é demasiadamente similar aos vetores de detecção e aos métodos de *template* de aparência, no sentido de providenciar apenas uma estimativa grosseira de orientação em intervalos e localizações discretas.

O mesmo MLP pode ser treinado para realizar estimativas finas de orientação de cabeça sobre uma extensão contínua de orientações, com algumas modificações na configuração. Neste caso, a rede possui um *output* para cada DOF e a ativação da saída é proporcional à orientação correspondente. Alternativamente, um conjunto de redes MLP com um único neurônio de saída pode ser trainado individualmente para cada DOF. Esta última abordagem tem sido usada principalmente para casos em que as imagens das cabeças são adquiridas por múltiplas câmeras distantes em ambientes internos, e somada a filtros de cor ou subtração de fundo para detecção da região do rosto e a filtros de Bayes para unir e suavizar as estimativas de cada câmera individual.

As vantagens das abordagens que utilizam redes neurais são inúmeras. Esses sistemas trabalham rápido, necessitam apenas de recortes etiquetados de rostos para treino, têm bom desempenho para imagens vindas de câmeras distantes ou próximas e entregam estimativas de orientação de cabeça das mais precisas na prática. A principal desvantagem desses métodos é que eles estão propensos a erros causados por má localização da cabeça. Uma sugestão para esse problema é o uso de camadas de convolução (ou seja, de CNNs) para estender o MLP através da modelagem explícita de invariância de deslocamento, escala e distorção.

2.2 Trabalhos correlatos

Nesta seção, serão introduzidos trabalhos correlatos, principalmente, na área de *head pose estimation* utilizando CNNs, porém alguns trabalhos utilizando outros métodos ou sobre *face alignment* ou alinhamento de rosto também serão discutidos.

O problema de estimar a orientação no espaço tridimensional de um rosto a partir de uma imagem bidimentsional existe há décadas. Em 1998, Chen *et al.*[24] propuseram um método que calcula propriedades geométricas como a área e o centro das regiões do rosto, da pele e do cabelo, e através do eixo de menor inércia, é capaz de estimar uma posição 3D com três graus de liberdade

(sobre três eixos independentes de rotação).

Métodos que utilizam redes neurais são mais recentes. Em 2004, Seeman *et al.*[25] apresentaram uma abordagem de *head pose estimation* focada em aplicações de interação homem-máquina, levando em consideração o fato de que a pessoa pode mover-se livremente pelo ambiente e estar mais próxima ou mais afastada da câmera. O método utiliza uma câmera estéreo como ferramenta de aquisição das imagens de rosto, que passam por um pré-processamento para extração de informação de profundidade aplicando-se modelagem 3D e rastreamento de rosto. Essa informação de profundidade é extraída e alimentada a uma rede neural *feedforward* simples de três camadas escondidas, que é capaz de estimar ângulos de panorâmica e inclinação de -90° a 90°. Além disso, os autores afirmam que o sistema é capaz de realizar processamento em tempo real.

Em 2005, Voit *et al.*[26] propuseram um sistema de estimar a orientação horizontal (*yaw*) de um rosto que utiliza múltiplas câmeras não para obter informação de profundidade mas para garantir pelo menos uma vista facial do usuário em qualquer ponto no ambiente. As imagens são alimentadas em redes neurais que foram treinadas com exemplos de cabeças rotacionadas para classificar a vista de cada câmera. Quando há mais de uma estimativa por usuário, elas são combinadas em uma hipótese conjunta, que, segundo os autores, é capaz de reduzir o erro médio de estimativa para rostos conhecidos em até 50%, demonstrado experimentalmente.

Um dos principais desafios atrelados ao mundo da aprendizagem de máquina e das redes neurais é a quantidade pura disponível de dados para treinamento, sem uso de *data augmentation*. Os bancos de imagens de rostos disponíveis são geralmente limitados a vistas frontais, oferecem apenas um pequeno número de anotações e informações sobre os rostos e suas orientações, e muitas vezes são adquiridos em ambientes controlados, que não necessariamente refletem as condições de uso. Buscando superar essas limitações, em 2011, Köstinger *et al.*[27] apresentaram uma base de dados inédita, denominada *Assorted Facial Landmarks in the Wild* (AFLW). A AFLW fornece uma coleção em larga escala de imagens coletadas do ambiente de compartilhamento de fotos Flickr, exibindo uma grande variedade de aparências de rostos (variando em posição, expressão, etnia, idade, gênero, etc.) e de meios de captura e condições ambientais (fotos com fundo simples, fundo detalhado, de múltiplos rostos, de longe, de perto, etc.). São quase 26000 rostos em aproximadamente 22000 imagens, com até 21 anotações por imagem. Além disso, Köstinger *et al.* também forneceram um conjunto de ferramentas para facilitar a integração com outras bases de dados e suas anotações associadas. Segundo os autores, a AFLW é "bem adequada para treinar e testar algoritmos de detecção de rostos em múltiplas vistas, de localização de marcos faciais e de estimativa de orientação facial".

Em 2012, Zhu e Ramanan [28] introduziram uma nova abordagem, apresentando um modelo unificado capaz de detectar rostos, estimar a orientação e localizar marcos destes em imagens confusas de ambientes reais. O sistema é baseado em misturas de árvores com um conjunto compartilhado de peças. Cada marca facial é modelada como uma peça e usada em misturas globais para capturar mudanças topológicas devido a variações no ponto de vista da câmera de aquisição. Além de apresentarem resultados extensos em bancos de dados já consolidados, os autores também trouxeram um novo conjunto de dados "in-the-wild", denominado *Assorted*

Faces in the Wild (AFW), para demonstrar que o modelo conjunto do sistema proposto avança o estado da arte nas três tarefas propostas. Os autores ainda afirmam que o modelo se compara favoravelmente a sistemas comerciais treinados com bilhões de exemplos, tal como Google Picasa e o website 'face.com', mesmo "treinado modestamente com (apenas) centenas de imagens".

Seguindo a mesma linha de modelos multi-tarefa, Zhang e Zhang [29] propuseram em 2014 um sistema de aprendizagem profunda multi-tarefa para melhorar a performance de algoritmos de detecção facial em múltiplas vistas, visto que este é um problema desafiador devido às mudanças drásticas na aparência dos rostos sob diferentes orientações, expressões faciais e condições de iluminação. O sistema descrito é uma *Deep Convolutional Neural Network* (DCNN) que é capaz de aprender simultaneamente a fronteira de decisão entre rosto/não-rosto, a solução do problema de estimar a orientação facial (embora discretizada) e a solução do problema de localizar os marcos faciais. O sistema multi-tarefas é usado principalmente para aprimorar ainda mais a acurácia de um classificador. Utilizando a base de dados desafiadora *Face Detection Dataset and Benchmark* (FDDB), uma base de dados projetada para estudar o problema de detecção facial livre, contendo anotações de mais de 5000 rostos em um conjunto de 2845 imagens retiradas da base de dados *Faces in the Wild*[30], os autores afirmam uma melhora de mais de 3% na taxa de detecção facial com uma mesma taxa de falsos-positivos, quando comparado com outros métodos estado-da-arte.

Ranjan, Patel e Chellappa [31] apresentaram em 2017 um algoritmo para realização simultânea de detecção de rostos, localização de marcos faciais, estimativa de orientação e reconhecimento de gênero utilizando redes neurais convolucionais profundas. Denominado *HyperFace*, o método proposto combina as camadas intermediárias de uma DCNN utilizando uma outra CNN separada seguida de um algoritmo aprendiz multi-tarefa que opera nas características combinadas. Além disso, os autores também propõem variações na *HyperFace* em busca de melhoria na performance. Uma das variações consiste na chamda *HyperFace-ResNet*, que é construída a partir do modelo *ResNet-101* e, se aproveitando dos pesos pré-treinados e arquitetura consolidada, consegue melhorar significativamente o desempenho em todas as tarefas. A melhora no desempenho, porém, vem a custo da velocidade, uma vez que a rede agora possui mais operações de convolução. A segunda modificação atua, então, para melhorar a velocidade do algoritmo, utilizando um detector de rosto rápido de alta recordação para gerar "propostas de regiões" e diminuir o custo da operação de busca das áreas de interesse. Segundo os autores, os modelos propostos demonstraram, após experimentação extensiva, serem capaz de capturar informação tanto local quanto global em rostos e terem um desempenho melhor em todas as tarefas listadas do que muitos algoritmos competitivos de cada área.

Estimar a orientação da cabeça de uma pessoa é um problema crucial para um grande número de aplicações, como sistemas que estimam a direção do olhar de uma pessoa a algoritmos que executam alinhamento de modelos de rosto 3D. Tradicionalmente, a estimativa da orientação da cabeça é feita a partir da estimativa da localização de alguns marcos faciais (olhos, ponta nariz, cantos da boca) no rosto que se quer estimar, seguida de um mapeamento da correspondência 2D/3D com uma cabeça humana média. Ruiz, Chong e Rehg (2018) [32] argumentam que este tipo de solução é um método frágil que depende inteiramente do desempenho da localização de marcos faciais, do modelo externo da cabeça humana e do ajuste *ad-hoc* dos parâmetros do sistema.

Em contrapartida, Riuz *et al.* apresentaram a um método elegante e robusto para determinar a orientação de uma cabeça a partir do treinamento de uma CNN com função múltipla de custo para prever os ângulos intrínsecos (*roll*, *pitch* e *yaw*) diretamente das intensidades de imagem por meio de classificação e regressão conjuntas. A rede, denominada Hopenet, foi treinada usando o conjunto de dados 300W-LP, um vasto conjunto de imagens de rosto expandido sinteticamente, e os autores apresentaram resultados estado-da-arte em testes empíricos com bases de dados de imagens de rostos *in-the-wild*. Adicionalmente, Ruiz, Chong e Rehg testaram a solução também em uma base de dados para sistemas que utilizam informação de profundidade e afirmam estar preenchendo a lacuna de desempenho entre os métodos de *head pose estimation* que utilizam apenas imagens e os que utilizam informação de profundidade. Os modelos e os códigos de treinamento e teste são todos *open-source*, disponibilizados em um repositório GitHub pelos autores[2].

Por se tratar de um objeto rígido em três dimensões, é natural e intuitivo descrever a orientação da cabeça de uma pessoa utilizando ângulos de Euler, comumente denominados de roll, pitch e yaw, conforme mostrado na figura Figura 2.15. Essa representação, porém, esta sujeita ao chamado *gimbal lock*, onde o alinhamento de dois dos três eixos em um plano causa a perda[3] de um grau de liberdade, em termos computacionais. Buscando evitar esse problema, Hsu *et al.*[4] propuseram, em 2018, a QuatNet, uma CNN com função de custo múltipla, combinando uma função de perda de regressão L2 com uma função de perda ordinal, dedicada a estimar a orientação de uma cabeça a partir de uma imagem RGB, sem informação adicional de profundidade. A função de perda de regressão ordinal é usada para considerar a propriedade não estacionária observada conforme a aparência do rosto muda de acordo com diferentes orientações de cabeça e para garantir a aprendizagem de características robustas, enquanto a função de perda de regressão L2 utiliza as características robustas para fornecer predições precisas de ângulos a partir de imagens de entrada. Utilizando representação de rotação no espaço 3D com quatérnions ao invés de ângulos de Euler, a QuatNet obtém desempenho estado-da-arte nas bases de dados AFLW[27] e AFW[28], além de diminuir a lacuna de desempenho com métodos que utilizam informação de profundidade.

[2] `https://github.com/natanielruiz/deep-head-pose`

[3] Trata-se de um problema de ambiguidade. Os três eixos ainda podem mover-se livremente, porém a descrição desse movimento, até que os eixos desalinhem-se, está limitada a duas dimensões, uma vez que não há como garantir sobre qual dos dois eixos alinhados o objeto está rotacionando.

Capítulo 3

Metodologia Proposta

Neste capítulo, será apresentada a solução proposta e seus parâmetros definidos, bem como os motivos que levaram a tais escolhas ou, quando for o caso, será apresentada uma suposição que se deseja comprovar ou comparar nos resultados obtidos.

3.1 Arquitetura da Rede

Mesmo após limitar dentro do universo das Redes Neurais Convolucionais, as possibilidades de *design*, arquitetura e implementação são quase infinitas. Desde o número de neurônios nas camadas escondidas, passando pelo número de camadas escondidas e pelo tipo de camadas escondidas, passando também pelo tamanho dos filtros de convolução e pelos métodos de *pooling*, até por fim chegar no formato de entrada dos dados e nas funções de ativação das saídas, a escolha do formato da rede é uma das mais difíceis durante todo o processo.

Por questão de facilidade, foi escolhido utilizar uma arquitetura convolucional já consolidada e previamente treinada para extrair *features* de imagens para tarefas de classificação de objetos, em computadores muito mais potentes com muito mais dados do que seria possível ter acesso como estudante.

Hsu *et al.*[4] projetaram e apresentaram uma rede baseada na GoogLeNet, também conhecida como Inception V1[5]. A GoogLeNet foi o ponto de partida para a solução proposta, a partir da qual os autores modificaram as camadas densas ao fim da rede para que estas realizassem tarefas de regressão em vez de classificação. A estrutura Inception, porém, possui versões mais recentes, mais profundas e com melhor desempenho, e optou-se por utilizar a Inception V3[6]. A principal diferença entre as versões das DCNNs Inception listadas consiste no bloco de processamento que dá o nome à rede. Um 'módulo *inception*' original é um bloco de três operações de convolução, com filtros 1x1, 3x3 e 5x5, e uma operação de *max pooling* seguidos de uma concatenação de filtros, podendo conter ou não operações que reduzem a dimensionalidade. A Figura 3.1 apresenta um esquema de um módulo *inception* com redução de dimensões. Já a Inception V3 apresenta um módulo *inception* com as operações de convolução com filtros de maior dimensão (3x3 e 5x5) fatoradas em duas operações de convolução assimétricas (1xn seguido de nx1), que possui o mesmo

efeito de uma única operação com filtro nxn porém é mais barata computacionalmente, para grades de tamanho médio (Szegedy *et al.*[6] afirmam que com n = 7 é possível alcançar ótimos resultados). A Figura 3.2 apresenta um diagrama do módulo *inception* com camadas fatorizadas.

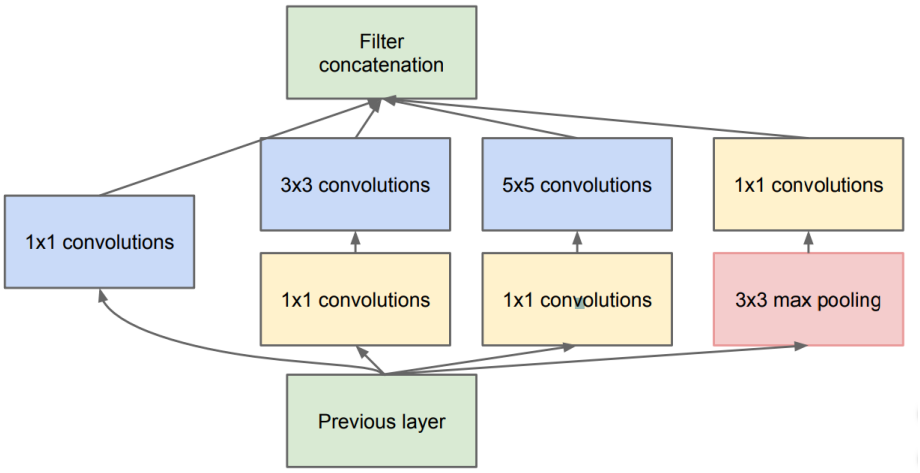


Figura 3.1: Módulo *inception* da Inception V1 (Fonte: [5]).

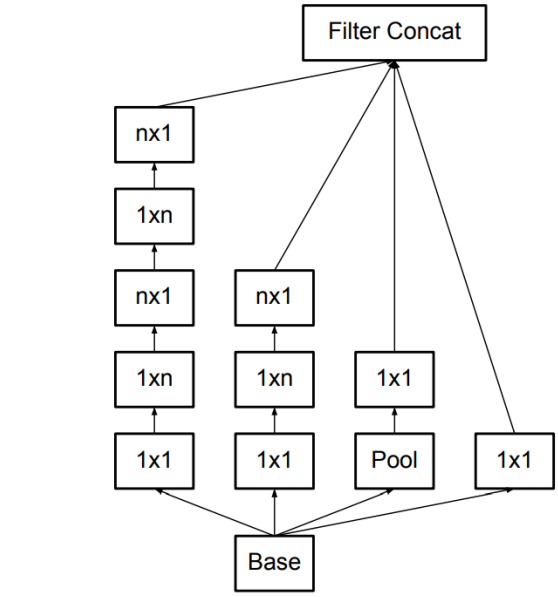


Figura 3.2: Módulo *inception* da Inception V3 (Fonte: [6]).

A Figura 3.3 apresenta um diagrama do sistema proposto neste trabalho, e um fluxograma das camadas da GoogLeNet(Inception V1) e da Inception V3 se encontram no Anexo I e no Anexo II, respectivamente.

Além disso, serão feitas duas versões, com diferentes tipos de dados de entrada, para futura comparação no Capítulo 4.

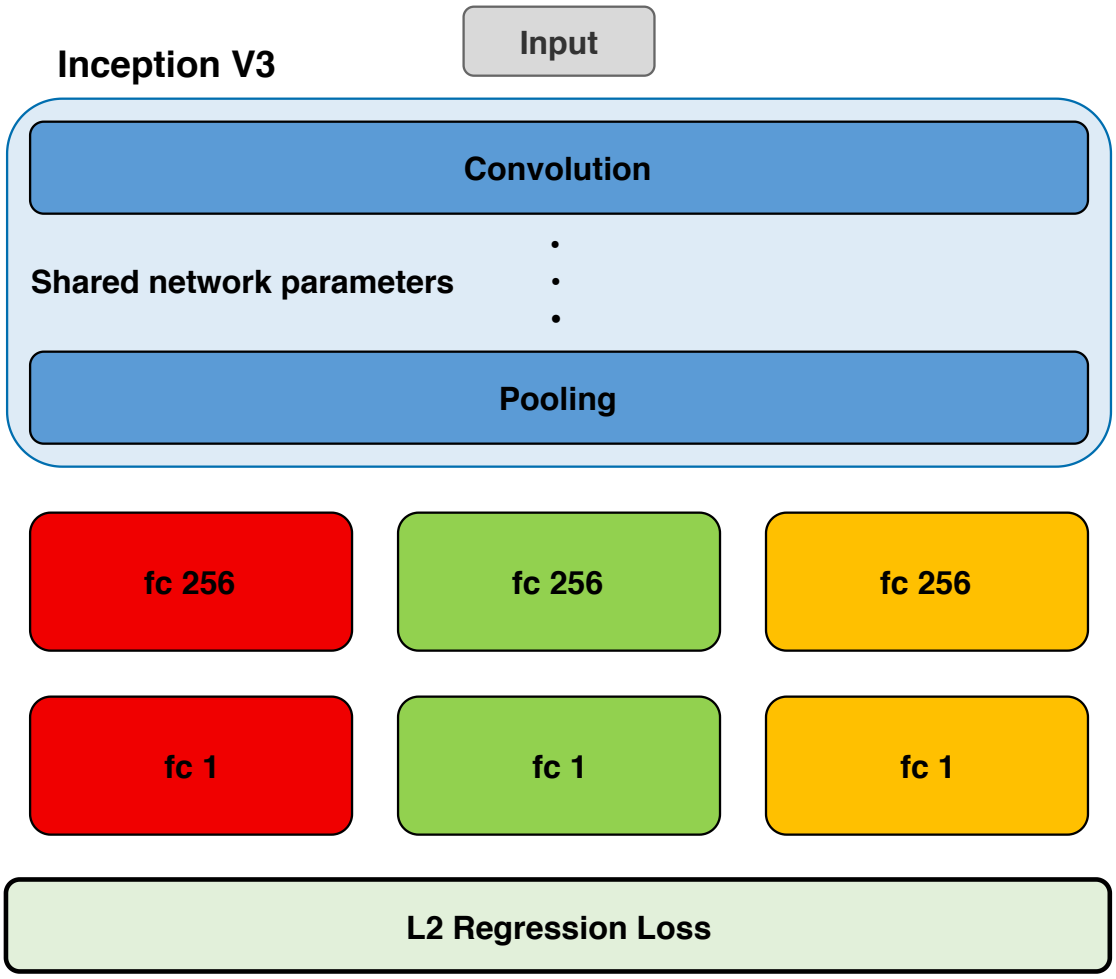


Figura 3.3: Diagrama das camadas da solução proposta (Adaptado de [4]).

3.1.1 Parâmetros da Rede

Uma vez que a base da solução proposta é uma CNN previamente treinada de arquitetura consolidada, há algumas maneiras de se prosseguir com o treinamento após as modificações: manter os pesos sinápticos das camadas de convolução já existentes e treinar apenas as novas camadas de regressão adicionadas para a nova tarefa; realizar um processo de *fine-tuning*, onde os valores já existentes são usados como os valores iniciais dos pesos e a rede completa é treinada, com pesos da base InceptionV3 ajustados a uma taxa de aprendizagem reduzida em relação às novas camadas adicionadas de acordo com a tarefa; e treinar a solução completa do zero, usufruindo apenas da arquitetura consolidada. Todas as estratégias possuem vantagens e desvantagens. Por exemplo, se manter as camadas de convolução da rede base intacta implica em uma velocidade rápida de iteração para as poucas camadas densas de saída, ao mesmo tempo essa estratégia restringe a capacidade que a rede tem de aprender e limita a minimização da função de custo a uma região pré-definida. Por outro lado, treinar a rede completa novamente dá maior flexibilidade ao treinamento, mas demanda muito mais tempo para concluir o aprendizado.

Ainda sobre a inicialização dos pesos sinápticos e seus efeitos no desempenho da rede, Pan e

Yang (2009) alertam sobre os perigos da transferência negativa, um fenômeno que ocorre quando "os dados e tarefa do domínio de origem do conhecimento contribuem para uma performance reduzida da aprendizagem no domínio de destino".[33] Embora saber evitar transferência negativa seja um assunto de extrema importância, Pan e Yang relatam em seu estudo sobre *Transfer Learning* realizado em 2009 que poucas pesquisas sobre esse tópico haviam sido conduzidas e publicadas até então. Apesar disso, Rosenstein *et al.*[34] demonstraram empiricamente que uma das causas de transferência negativa é uma alta dessemelhança entre as tarefas de origem e de destino.

Considerando a possibilidade de ocorrência de da transferência negativa, optou-se por sempre treinar a rede por completo, sem aprendizado por transferência ou ajuste fino de pesos previamente treinados. Quanto à inicialização dos pesos sinápticos, utilizou-se os pesos já existentes na InceptionV3 para os casos de treinamento cujo objetivo era comparar o efeito de mudanças na rede, visando assegurar as mesmas condições iniciais em cada caso, porém as diversas execuções da solução final escolhida deram-se a partir de inicializações aleatórias.

Outra escolha a ser feita é a do otimizador da rede, cuja função é minimizar a função de custo com cada iteração. O projeto apresentado por Hsu *et al.* utiliza o clássico método da descida estocástica do gradiente (do inglês, *Stochastic Gradient Descent*, SGD), mas há anos novos otimizadores vêm sendo desenvolvidos, dentre eles o Adam, um algoritmo para otimização baseado em gradiente de primeira ordem de funções estocásticas objetivas, a partir de estimativas adaptativas de momentos de ordem mais baixa, apresentado por Kingma e Ba em 2014[35]. Embora, de maneira geral, o método de otimização não seja uma escolha crucial para a capacidade do sistema solucionar o problema proposto, a escolha do otimizador para o modelo de aprendizagem profunda pode significar a diferença na obtenção de bons resultados em minutos, horas ou dias.

Vale ressaltar que a escolha de um otimizador possui um parâmetro estritamente relacionado ao método utilizado: a variação da taxa de aprendizagem. Enquanto o SGD clássico mantém um único valor de taxa de aprendizagem (denominado alfa) para todas as iterações de atualização peso sinápticos, outros otimizadores possuem taxa de aprendizagem variável de concepção, como o previamente citado Adam[35], que computa uma taxa de aprendizagem adaptativa individualmente para cada parâmetro (peso sináptico) do modelo, a partir de estimativas dos primeiros gradientes. Além disso, é possível estabelecer rotinas de decaimento da taxa de aprendizagem, uma estratégia baseada na observação de que passos grandes de ajuste dos pesos são desejáveis no início, mas conforme o processo se aproxima da solução, passos cada vez menores ajudam na convergência da rede para um mínimo.

Uma estratégia de variação da taxa de aprendizagem é a chamada taxa de aprendizagem cíclica, ou, em inglês, *Cyclic Learning Rate* (CLR), proposta em 2017 por Leslie Smith[36]. A ideia da taxa de aprendizagem cíclica é eliminar a necessidade de encontrar experimentalmente os melhores valores de taxa de aprendizagem, variando esta de maneira cíclica entre valores razoáveis de borda máxima e mínima, ao invés de só aplicar rotinas de decaimento monotônico.

Hsu *et al.*[4] utilizaram no projeto original da QuatNet o otimizador SGD com taxa de aprendizagem inicial $5x10^{-3}$ e um decaimento programado pelo qual a taxa é reduzida pela metade a

cada 10000 iterações, mas neste trabalho foram realizados alguns testes experimentais com o objetivo de comparar a eficiência da taxa de aprendizagem cíclica aplicada à solução. Os resultados dos casos de treino estão descritos no Capítulo 4.

3.2 Método de Aquisição

Muitos trabalhos recentes que fornecem predições precisas de orientações de cabeça utilizam, além de uma imagem RGB como entrada, informação adicional de profundidade, geralmente obtida através de equipamentos especializados como câmeras de profundidade ou o *Kinect*® da Microsoft. Entretanto, câmeras de profundidade e outras ferramentas similares requerem sensoreamento ativo (ou seja, devem interagir com o ambiente e o usuário, de alguma forma), que pode ser comprometido em ambientes ao ar livre ou com fontes externas de ruído (por exemplo, ambiente com muito barulho interferem em equipamentos de ultrassom). Além disso, para aplicações que requerem reações imediatas e quantidades maciças de processamento de dados em um curto espaço de tempo, o uso de câmeras de profundidade apresentam um custo alto de computação adicional. Hsu *et al.* (2018) sugerem que um sistema de *head pose estimation* que trabalha apenas com a informação contida em imagens RGB funciona como um meio-termo entre agilidade e acurácia, além de exibir o potencial para mais diversas aplicações[4].

Baseado nisso, para este trabalho foi decidido utilizar como entrada da rede fotos RGB adquiridas de câmeras fotográficas comuns comerciais, sem especificação de lente ou modelo, em formato de arquivo de imagem comum (.jpg ou .png), de modo que seja possível montar uma integração rápida da rede com o *stream* de entrada de dados (uma câmera ligada a um computador).

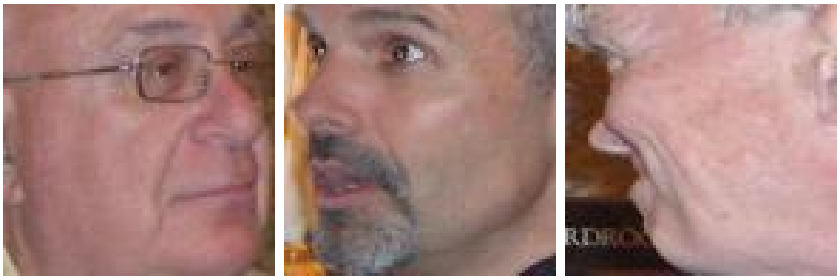
3.3 Dados de Treino e Teste

Decidido método de aquisição, torna-se necessário encontrar bases de dados que possuam uma quantidade suficiente de informação para um treinamento satisfatório.

Hsu *et al.*[4] listam em seu artigo quatro bases de dados que podem ser utilizadas para treinar uma CNN para estimar a orientação de uma cabeça de uma pessoa: AFLW[27], AFW[28], 300W-LP[37] e Biwi Kinect Head Pose Database[38], este último uma base de dados de posições de cabeça com informação de profundidade (nuvens de pontos 3D geradas pelo Kinect, da Microsoft), mas que também disponibiliza a imagem RGB correspondente. Foram escolhidas as bases AFW e AFLW por serem as mais voltadas para o objetivo deste trabalho. A Figura 3.4 e a Figura 3.6 apresentam exemplos das imagens contidas nas bases AFW e AFLW, respectivamente, enquanto a Figura 3.5 e a Figura 3.7 apresentam os recortes dos rostos dos respectivos exemplos.

Após adquiridas as bases, notou-se uma discrepância das mesmas com as informações previamente obtidas. Segundo Koestinger *et al.*[27], a base AFLW é composta de 21997 imagens contendo 25993 exemplos de rostos. Porém a contagem final de imagens foi 21123, com 24384 rostos. A base foi adquirida por meio de um link fornecido pelos próprios autores, que continha,

Figura 3.4: Exemplo de imagem contida na base de dados AFW (Fonte: [28]).



(a) (b) (c)

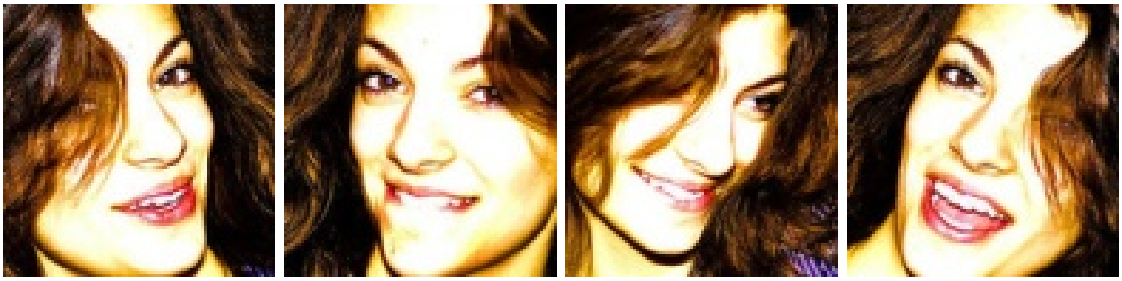
Figura 3.5: Recortes dos rostos da Figura 3.4.

além de pastas comprimidas das imagens em si, arquivos .md5 contendo o *checksum* das pastas comprimidas para verificar a integridade dos arquivos baixados e garantir que o arquivo está completo e não corrompido. A verificação foi feita para todos os arquivos baixados e em todos os casos houve correspondências dos *checksum*, portanto, se desconhece o motivo da discrepância no número de imagens.

Uma vez adquiridas as bases de dados, é necessário realizar uma divisão destas em conjuntos de treino e de teste, ou em conjuntos de treino e de validação.

Enquanto não há nenhuma resposta definitiva para como deve ser feita essa divisão e qual deve ser a razão entre o tamanho do conjunto de treino para o tamanho do conjunto de validação, estudos realizados sobre esse processo sugerem uma dependência do tamanho do conjunto de dados a ser dividido. Guyon (1997)[39] sugere, para grandes bases de dados em problemas de classificação, uma fórmula baseada na quantidade de classes e na complexidade do modelo, geralmente relacionada à quantidade de parâmetros ajustáveis ou outras medidas. Por exemplo, para uma rede cujo objetivo é classificar uma entrada em uma entre dez classes, para uma arquitetura simples, com cerca de cem parâmetros ajustáveis (baixa complexidade), é recomendado separar um quarto dos

Figura 3.6: Exemplo de imagem contida na base de dados AFLW (Fonte: [27]).



(a) Sup. Esq. (b) Sup. Dir. (c) Inf. Esq. (d) Inf. Dir.

Figura 3.7: Recortes dos rostos da Figura 3.6.

dados para validação. Já com uma arquitetura de alta complexidade, com uma quantidade de parâmetros uma ordem de grandeza acima, 10% dos dados para validação é o suficiente, segundo Guyon.

Em uma tentativa de extrapolar essa noção para o problema de regressão, a divisão de casos de treino e validação será feita de acordo com o tamanho da base de dados utilizada. No caso da AFW, que contém ao todo menos de 500 imagens, a divisão será 75% dos dados para treino e 25% para validação. Já para o AFLW, que contém quase 25000 exemplos de rostos, a divisão será 90% das imagens para treino e 10% para validação. A ideia é que uma base de dados extensa consegue fornecer mais casos de teste com uma porcentagem menor da base total do que uma base pequena. Enquanto um quarto da base AFW se resume a apenas 117 imagens, 10% da base AFLW corresponde a 2439 exemplos de rostos, um valor 20x maior. Um conjunto maior de

imagens consegue fornecer dados de validação suficientes com uma divisão mais estreita do que uma base de dados limitada.

Algo a se decidir também é o tamanho das imagens que serão alimentadas para a rede. Esse é um aspecto que afeta diretamente tanto na velocidade de treinamento da rede (quanto maior a imagem de entrada, mais operações por camada) quanto nos requisitos computacionais (quanto maiores as dimensões das imagens de entrada, maior a quantidade de memória necessária para armazenar todos os pesos e maior o poder de processamento necessário para realizar e armazenar o resultado das operações de convolução). Existem diversos tamanhos descritos na literatura, que variam com a tarefa que a rede deve cumprir e com o equipamento à disposição dos pesquisadores de cada sistema. Às vezes a decisão do formato da entrada é feito pela própria base de dados, como é o caso das bases CIFAR10[40] e STL10[41], cujas imagens possuem dimensão de 32x32 pixeis e 96x96 pixeis, respectivamente.

Nem o AFW nem o AFLW restringem o formato de entrada dos dados, logo, se torna parte do projeto da rede escolher um formato padrão para os dados de entrada. Neste trabalho, essa escolha se deu de maneira empírica, comparando alguns formatos mais comuns e considerando eventuais limitações da rede. A Seção 4.2 do Capítulo 4 descreve essa análise.

É importante também levar em consideração o formato dos dados do vetor alvo, que contém as informações dos ângulos de orientação de cabeça de cada imagem que será usada para treinar e testar a rede. A natureza do problema que este projeto busca resolver permite a representação da informação de ângulo tanto em graus quanto em radianos. Escolheu-se utilizar ângulos em radiano pois seu valores mais próximos ou pertencentes ao intervalo [-1,1] evita que o valor da função de custo "exploda" durante o treinamento, ou seja, que ultrapasse o valor máximo que uma variável pode armazenar, devido a algum ruído ou perturbação ou má inicialização dos pesos. Com ângulos em grau, previsões erradas podem gerar erros quadráticos na casa de centenas ou até milhares, por item de entrada.

3.4 Implementação

A melhor escolha de ferramenta para implementação da solução são as ferramentas computacionais de aprendizagem de máquina desenvolvidas em código livre, por permitirem livre-divulgação e serem de fácil acesso e uso, em quase todos os ambientes.

Duas grandes ferramentas populares atuais do campo de ML são o Keras[42] e o TensorFlow[43]: TensorFlow é uma plataforma de ponta a ponta para aprendizagem de máquina escrita em código aberto que possui um ecossistema compreensivo e flexível de ferramentas, bibliotecas e recursos comunitários para facilmente desenvolver e publicar aplicações ML com desempenho estado-da-arte; Já Keras é uma Interface de Programação de Aplicações (API, do inglês *Application Programming Interface*) de redes neurais de alto nível, escrita em linguagem de programação Python e capaz de rodar em cima de plataformas como o TensorFlow e desenvolvida com um foco em permitir experimentação rápida, sendo capaz de ir do projeto aos resultados com o menos atraso possível.

Uma vez que a API Keras já está escrita em linguagem Python, esta se torna a primeira opção de linguagem de implementação da solução. Python é uma linguagem de programação de alto nível e para propósitos gerais de fácil acesso mantida por uma organização sem fins lucrativos cuja missão é avançar a tecnologia *open source* relacionada à linguagem Python e divulgar o uso desta.

Python é uma linguagem popular com muitas bibliotecas oficiais e *community-made* dos mais diversos domínios de aplicações: de desenvolvimento web a computação científica-numérica. Para este trabalho, era necessário obter, além das ferramentas de aprendizagem de máquina e redes neurais, módulos de processamento de imagens, de programação científica e de manipulação de gráficos. Para isso, foram escolhidas as bibliotecas Scikit-image[44], Numpy[45] e Matplotlib[46], respectivamente. Dentre as implementações feitas pela comunidade está a implementação para Keras da taxa de aprendizagem cíclica feita por Kenstler em 2018[47].

No Apendice II se encontra disponível o código em Python da versão final da solução apresentada, criada após as variadas análises de projeto serem feitas. As análises de projeto estão descritas no Capítulo 4.

3.5 Ambiente de Processamento

Existem diversas opções para a escolha do ambiente de processamento, desde computadores locais com hardware (geralmente *Graphic Processing Units* (GPUs)) dedicado para execução de tarefas de aprendizagem de máquina até serviços de processamento em nuvem que oferecem esse tipo de processamento dedicado para usuários que não possuem esse hardware em suas máquinas locais.

Um desses serviços de computação em nuvem é o *Google Colaboratory*, uma ferramenta gratuita de pesquisa para projetos educativos e acadêmicos de ML.

Um estudo conduzido por Carneiro *et al.*[48] em 2018 mostrou que o *Google Colaboratory*, também frequentemente chamado de *Google Colab*, possui hardware que alcança um desempenho similar aos computadores com hardware dedicado. Os autores do estudo observaram que é vantajoso utilizar o Colab para experimentos de ML quando não se tem um equipamento de processamento (GPU) mais robusta do que uma Tesla K80, da NVIDIA. É, ainda, possível usar o Colab para outras aplicações que requerem uma GPU que não de aprendizagem profunda, sem necessidade de configuração extra. Adicionalmente, por se tratar de um serviço de computação em nuvem, o compartilhamento dos *notebooks*, como são chamadas as instâncias, com códigos e resultados, é simples e direto.

Apesar disso, o Google Colab não é um ambiente sem desvantagens. Carneiro *et al.* listaram como sendo as principais incoveniências o limite de tempo de uso de GPU, a necessidade de transferência de dados por meio do Google Drive ou de repositórios Git, o limite de transferência de dados entre o Drive e o Colab, e a falta de núcleos CPU (*Central Processing Unit*). Além disso, o hardware do Colab não é possível de ser escalado e é inviável para a solução de problemas maiores.

Outra ferramenta de processamento em nuvem para tarefas de aprendizagem de máquina é o Intel AI DevCloud[1]. Segundo Apeland (2017), o DevCloud é um aglomerado de processadores escaláveis Intel® Xeon® com software pré-compilado otimizado para a arquitetura Intel®, incluindo distribuições próprias da linguagem Python v2.7 e v3.6, otimização de software para Keras e para Tensorflow. Seu acesso é disponibilizado para desenvolvedores, cientistas de dados, professores, estudantes e *start-ups*, e os usuários contam com uso de processadores Intel® Xeon® com 24 núcleos e 96GB de *Random Access Memory* (RAM) DDR4, e 200GB disponíveis para armazenamento de dados[49].

A Intel oferece três ambientes de acordo com a carga de trabalho do usuário: Centro de dados, voltado para a aplicação de algoritmos de aprendizado de máquina e aprendizagem profunda para treinamento e inferência mais rápidos; *Edge* ou Borda, um ambiente de desenvolvimento de borda heterogêneo para *Internet of Things* (IoT); e FPGA (*Field-Programmable Gate Array*), voltado para o desenvolvimento de soluções programáveis validadas em hardware FPGA de ponta. Para este trabalho, o ambiente mais adequado é o 'Centro de Dados'.

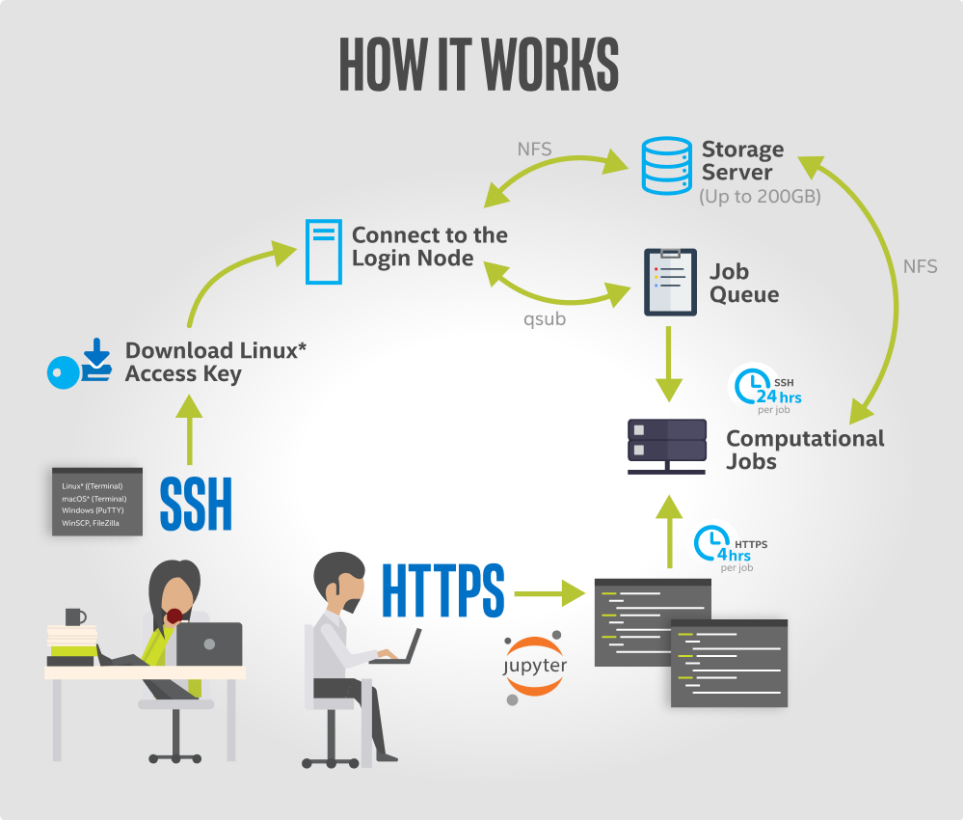


Figura 3.8: Diagrama dos meios de conexão com a plataforma Intel DevCloud. O acesso principal é feito com uma chave de acesso SSH para ambientes Linux e permite acesso ao servidor de armazenamento e ao serviço de agendamento de trabalhos, com limite de 24 horas de processamento contínuo. O outro método de acesso consiste em uma conexão HTTPS via Jupyter Notebook e permite trabalhos interativos de até quatro horas de processamento (Fonte: [50]).

[1] `https://software.intel.com/devcloud`

Diferentemente do *Google Colab*, cujo processamento é em tempo real com conexão gerenciada pelo *browser* de internet, o DevCloud permite agendamento de trabalho, permitindo que a rede seja treinada à distância sem necessidade de acompanhamento ou de manter o computador local ligado, permitindo 24h de uso do processador. O DevCloud também permite o mesmo modelo de conexão do *Google Colab*, chamado de 'trabalho interativo' (não agendado), porém este tem limite de conexão com o servidor de 4h. A Figura 3.8 ilustra o processo de conexão aos processadores.

Outra possível opção para o processamento seria o ML Engine[2], também da Google, que é um serviço pago de processamento em nuvem para aplicações de aprendizagem de máquina. A análise desse ambiente fica como trabalho futuro.

[2] `https://cloud.google.com/ml-engine/`

Capítulo 4

Resultados

Este capítulo apresenta os resultados obtidos experimentalmente após o treinamento do modelo sob variadas condições, buscando comparar as escolhas feitas na etapa de projeto, de maneira a obter a melhor solução possível para o problema apresentado.

4.1 Resultados de Referência

Hsu *et al.*[4] apresentaram em 2018, juntamente com a QuatNet, um projeto chamado EulerNet. A rede EulerNet tem seu design e implementação semelhantes aos da QuatNet, porém ela busca estimar os ângulos de orientação de cabeça em ângulos de Euler, similar ao que foi planejado e executado neste trabalho.

Buscando obter valores base para a validação das escolhas de projeto realizadas para este trabalho, primeiramente tentou-se reproduzir os resultados obtidos por uma das instâncias da EulerNet descritas em [4], denominada de Euler_b por se tratar de um estágio intermediário antes do projeto final da EulerNet.

A Euler_b possui as camadas convolucionais e de *pooling* retiradas da GoogLeNet/ InceptionV1, inicializada com os pesos já previamente treinados no *dataset* ImageNet ILSVRC, seguidas de três conjuntos de camadas completamente conectadas independentes, uma para cada ângulo a ser estimado (*yaw*, *pitch* e *roll*). O formato das imagens de entrada da rede é 227x227 pixeis e três canais de cor RGB. O tamanho do *mini-batch*, ou seja, a quantidade de imagens de treino em uma passagem completa do algoritmo de *back-propagation* foi configurado como 96, e a Euler_b foi treinada por 30.000 iterações, com taxa de aprendizagem inicial definida em 0,005, sendo dividida pela metade a cada 10.000 iterações. O projeto foi desenvolvido utilizando o *framework* Caffe. Hsu *et al.* treinaram a Euler_b com as quatro bases de dados mencionadas na Seção 3.3.

A rede desenvolvida para obtenção dos valores de erro de referência neste trabalho utiliza como base a InceptionV3 no lugar da V1, foi implementada utilizando o *framework* Tensorflow e utilizou o número de épocas em vez do de iterações para o controle do treinamento e da variação da taxa de aprendizagem. Sabendo que o número de iterações para completar uma época pode ser

Tabela 4.1: Comparação do desempenho da Euler_b com uma versão alfa do presente projeto, denominada UnBFace_a.

Rede	Tempo de Treino	Yaw MAE	Pitch MAE	Roll MAE	Rede MAE
Euler_b	$\sim 4,5$ horas	5,440°	6,432°	6,495°	6,122°
UnBFace_a	$\sim 3,5$ horas	6,961°	6,073°	4,618°	5,884°

obtido pela divisão da quantidade total de exemplos de treinamento pelo tamanho do *mini-batch*, utilizando a base de dados AFLW, com 24384 exemplos de rostos, estimou-se 24384/96 = 254 iterações por época, o que resulta em 30.000/254 = 118.11 épocas no total. Utilizaram-se 120 épocas, número suficientemente próximo. O decaimento da taxa de aprendizagem foi programado utilizando as ferramentas de *callback* do Keras para alterar o valor a cada 40 épocas.

Sobre o hardware dedicado, Hsu *et al.* utilizaram uma GPU NVIDIA GTX 1080, enquanto as instâncias de conexão com o *Google Colab* forneceram acesso a uma Tesla P100-PCIE-16GB.

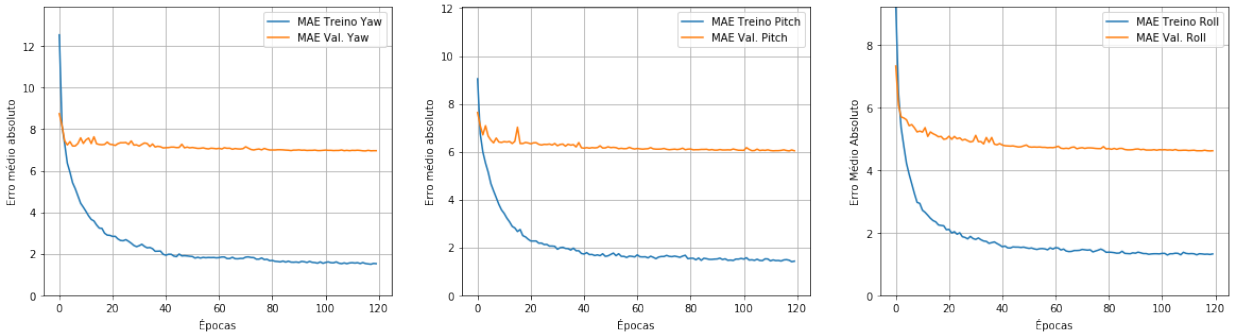


Figura 4.1: Histórico dos valores de MAE por época para a rede com configuração baseada na Euler_b.

A Figura 4.1 apresenta o resultado do treinamento da rede desenvolvida para obtenção dos resultados de comparação, e a Tabela 4.1 compara os resultados obtidos em [4] com os obtidos neste presente trabalho, que será denominado UnBFace_a.

Nota-se que as diferenças entre os projetos, por mais que não numerosas, foram o suficiente para evitar uma reprodução fiel dos resultados descritos no artigo. Apesar disso, estes valores de erro estão aquém do estado da arte. Hsu *et al.* conseguiram, na versão oficial da EulerNet, um erro médio de 5,02°. Visando alcançar essa marca, modificações foram feitas no projeto da solução apresentada neste trabalho, a serem descritas no restante deste capítulo.

4.2 Análise do formato dos dados de entrada

É conhecido o fato de que o formato dos dados de entrada influencia no tempo de treinamento da rede e no uso de memória, como descrito na Seção 3.3. É possível supor, então, que também tenha influência nos resultados. Como a base de dados AFLW possui recortes de cabeças de

dimensões mínimas (Figura 4.2c), optou-se por fazer uma análise do desempenho da solução proposta de acordo com o formato dos dados de entrada.

(a) 227x227

(b) 96x96

(c) 25x25

Figura 4.2: Exemplo de imagem sobre-dimensionada. Na Figura 4.2a, a informação visual dos padrões do nariz é prejudicada.

O processo de treinamento para esta etapa foi feito utilizando a base de dados AFW e as informações dos ângulos em graus, e processado numa GPU Tesla K80. As Figuras 4.3 a 4.6 apresentam os o histórico de treinamento de cada dimensionamento.

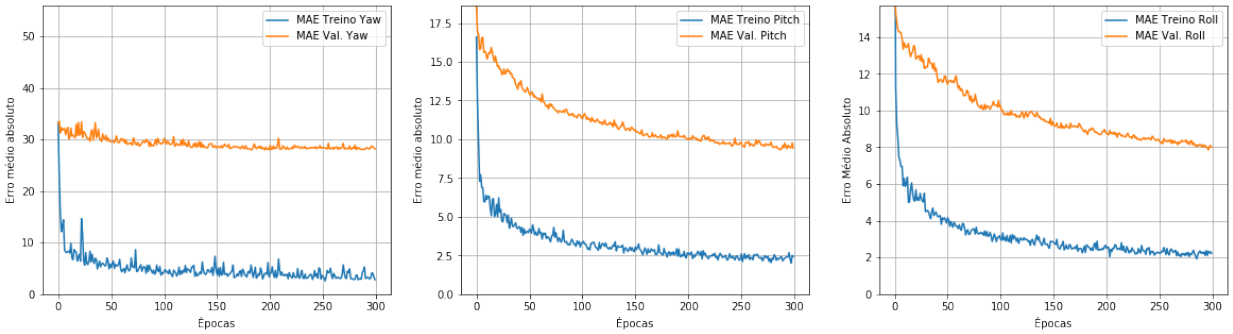


Figura 4.3: Histórico dos valores de MAE por época para imagem de entrada de dimensão 227x227 pixeis.

As configurações da rede, com exceção do formato dos dados, foram as mesmas: O treinamento se deu utilizando a solução inicializada com os pesos pré-treinados da ImageNet na base Inception V3; o otimizador escolhido foi o SGD clássico com decaimento programado de 3,33x10^{-5} a cada iteração; A taxa de aprendizagem inicial foi configurada em 0.05, e o número de épocas máximo definido em 300; O *batch size* foi definido em 96 e uma estratégia de parada antecipada foi im-

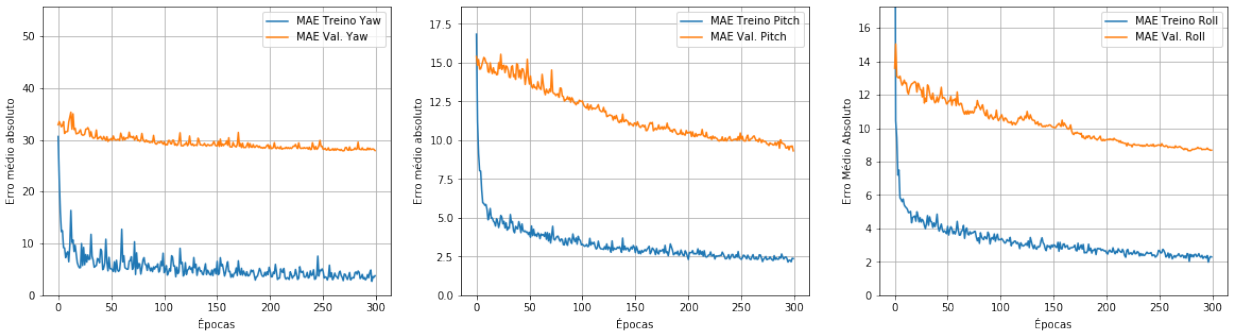


Figura 4.4: Histórico dos valores de MAE por época para imagem de entrada de dimensão 256x256 pixeis.

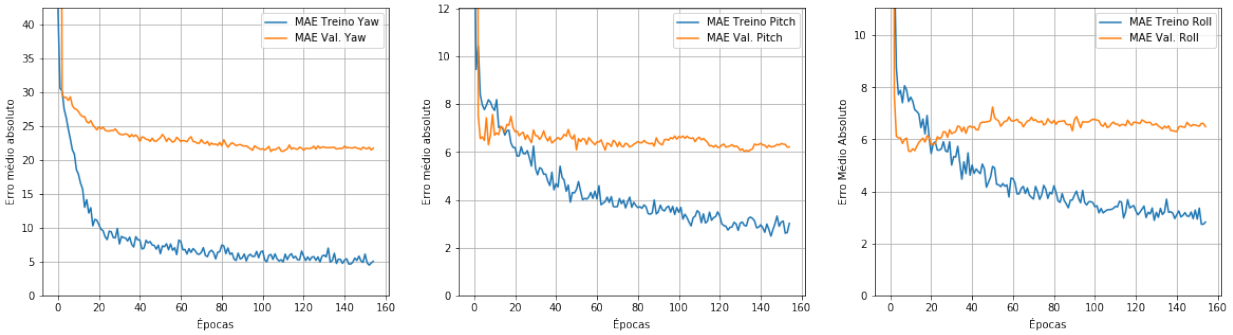


Figura 4.5: Histórico dos valores de MAE por época para imagem de entrada de dimensão 96x96 pixeis.

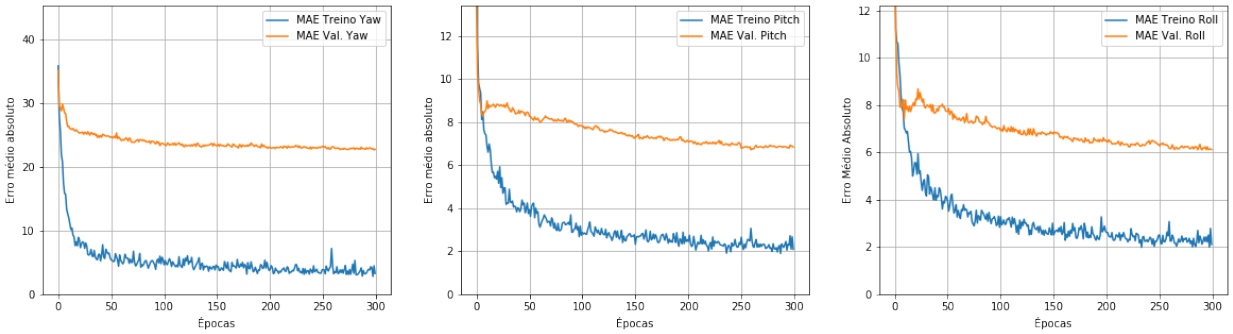


Figura 4.6: Histórico dos valores de MAE por época para imagem de entrada de dimensão 128x128 pixeis.

plementada (caso não houvesse diminuição no valor da função de perda gerada pelo conjunto de validação após 50 épocas, o treinamento seria encerrado); A divisão treino/validação foi 75% das imagens para treino e 25% para validação durante o treino. No Apêndice I encontra-se um link para o repositório no GitHub que contém o código em Python utilizado para esse processamento, gerado automaticamente pelo ambiente Google Colab. A Tabela 4.2 apresenta um resumo dos resultados para cada dimensão de imagem de entrada escolhida.

Tabela 4.2: Resultados dos treinamentos realizados para análise da dimensão da imagem de entrada da rede. *A implementação do Keras da rede Inception V3 não permite valores dimensionais de entrada menores que 75 pixeis. **Durante o início do treinamento, o processo lançou uma exceção de falta de memória (OOM, *Out of Memory*) ao alocar o tensor de uma das camadas de convolução, interrompendo o processamento.

Input shape	Tempo de execução	*Yaw* MAE	*Pitch* MAE	*Roll* MAE
32x32	Nenhum*	-	-	-
96x96	6min	21,24°	6,02°	6,30°
128x128	12min	22,67°	6,70°	6,10°
227x227	36min	28,01°	9,32°	7,86°
256x256	49min	27,87°	9,31°	8,64°
299x299	4min**	-	-	-

A partir das informações da Tabela 4.2, o formato de entrada escolhido foi 96x96, pois não só foi o mais rápido quanto também foi o que teve os menores erros absolutos médios em dois dos três ângulos de saída, em especial no *yaw*. Além disso, foi o único caso em que a parada antecipada foi aplicada, indicando que essa dimensão de input fez com que a rede alcançasse a solução do problema em um menor número de operações.

4.3 Treinamento da Rede de Canal Único

Decidido o tamanho da imagem, foi realizado um primeiro teste com as mesmas especificações da Seção 4.1 com apenas o formato dos dados de entrada da rede modificado, de 227x227x3 para 96x96x3, procurando-se observar uma leve melhora nas previsões baseando-se no que foi observado da Seção 4.2. A Figura 4.7 apresenta o histórico de treinamento da rede, e a Tabela 4.3 compara os valores de tempo de treinamento, MAE dos ângulos de Euler e erro médio da rede.

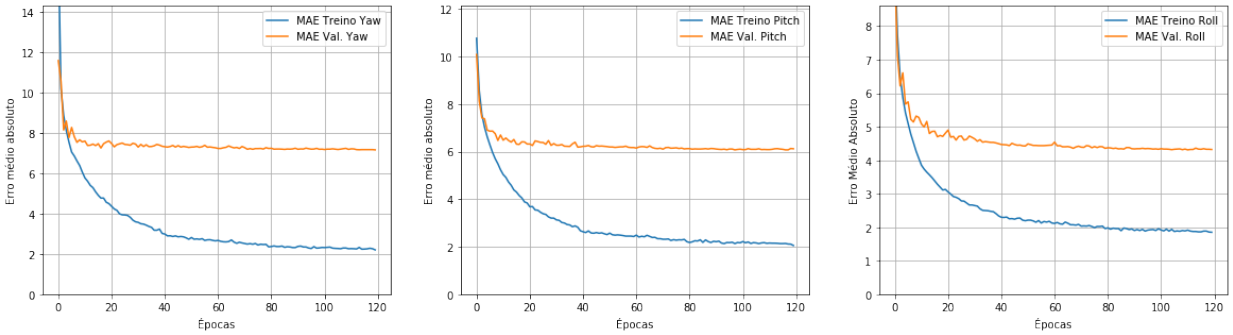


Figura 4.7: Histórico dos valores de MAE por época para a rede com configuração UnBFace_b.

Ao contrário do que era esperado baseado nos resultados da análise dimensional da Seção 4.2, obteve-se uma piora no resultado do ângulo *yaw*. Porém observa-se uma melhora no erro médio da rede e uma notável diminuição no tempo de treino.

Tabela 4.3: Comparação do desempenho da UnBFace_a com uma outra versão intermediária do presente projeto, denominada UnBFace_b. A única diferença entre as duas consiste no formato dos dados de entrada.

Rede	Tempo de Treino	Yaw MAE	Pitch MAE	Roll MAE	Rede MAE
UnBFace_a	$\sim 3,5$ horas	6,961°	6,073°	4,618°	5,884°
UnBFace_b	$\sim 1,1$ horas	7,168°	6,073°	4,343°	5,861°

Considerando os efeitos da mudança de formato dos dados de entrada descritos na Tabela 4.3 optou-se por continuar com as imagens de dimensão reduzida.

4.4 Aplicação da Taxa de Aprendizagem Cíclica

Em uma tentativa de forçar o algoritmo de minimização da função de custo a explorar melhor o espaço da imagem da função de custo, e a fim de evitar o ajuste manual do valor ótimo da taxa de aprendizagem, escolheu-se utilizar a estratégia de Taxa de Aprendizagem Cíclica, apresentada na Seção 3.1.1, e avaliar os efeitos de seu uso no projeto da presente solução. A Figura 4.8 apresenta o histórico de treinamento da rede com a função de ciclagem da taxa de aprendizagem adicionada, e a Tabela 4.4 compara o desempenho da solução com e sem taxa cíclica, ambas para imagens de entrada de dimensão 96x96 pixeis.

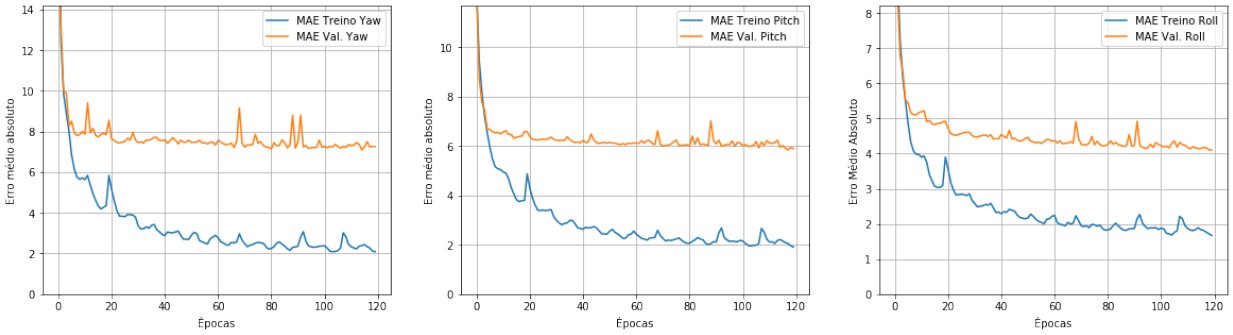


Figura 4.8: Histórico dos valores de MAE por época para a rede com estratégia de ciclagem do valor da taxa de aprendizagem, denominada UnBFace_c.

Tabela 4.4: Comparação do desempenho da UnBFace_b com uma outra versão intermediária do presente projeto, denominada UnBFace_c. A única diferença entre as duas consiste no método de variação da taxa de aprendizagem.

Rede	Tempo de Treino	Yaw MAE	Pitch MAE	Roll MAE	Rede MAE
UnBFace_b	$\sim 1,1$ horas	7,168°	6,073°	4,343°	5,861°
UnBFace_c	$\sim 1,2$ horas	7,173°	5,919°	4,320°	5,804°

De imediato nota-se um efeito nos gráficos apresentados na Figura 4.8 que não apareceu nas

seções anteriores (não é possível observar esse fenômeno na Figura 4.1 nem na Figura 4.7). Tanto a linha dos valores de erro do conjunto de treino quanto do conjunto de validação refletem a ciclagem da taxa de aprendizagem, que é regida por uma onda triangular[47].

Além disso, embora tenha havido um aumento de aproximadamente 5 minutos no tempo total de treino, houve uma melhora no erro médio da maioria dos ângulos estimados, resultando numa diminuição no erro médio da rede.

4.5 Adição da base de dados AFW

Buscando melhorar o desempenho da rede, decidiu-se agregar aos dados existentes uma nova base de dados, na expectativa de melhorar a capacidade de predição da solução com uma gama maior de exemplos fornecidos. A Figura 4.9 apresenta o histórico de treinamento da rede, e a Tabela 4.5 compara o desempenho da solução antes e depois de adicionados os exemplos contidos na base de dados AFW, embaralhando os vetores de imagens na tentativa de equilibrar a presença dos novos exemplos em ambos os conjuntos de imagens (de treinamento e de validação).

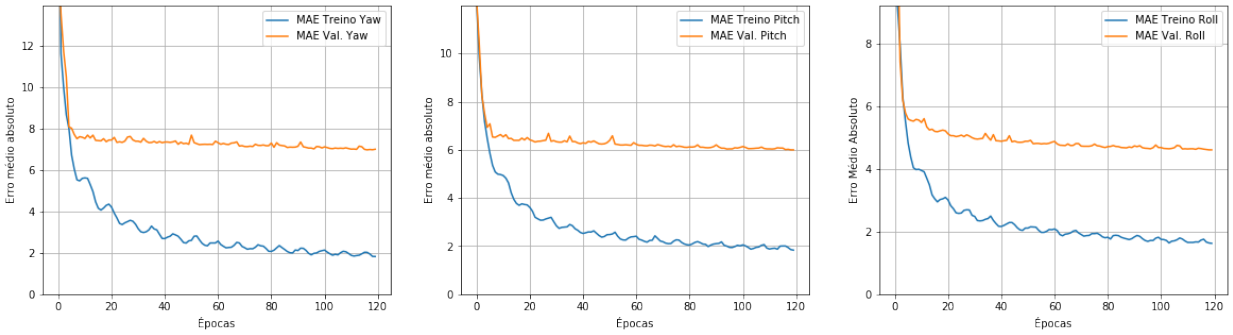


Figura 4.9: Histórico dos valores de MAE por época para a rede com base de dados combinada, denominada UnBFace_d.

Tabela 4.5: Comparação do desempenho da UnBFace_c com uma outra versão intermediária do presente projeto, denominada UnBFace_d. A única diferença entre as duas consiste na adição de mais exemplos de treino e validação.

Rede	Tempo de Treino	Yaw MAE	Pitch MAE	Roll MAE	Rede MAE
UnBFace_c	$\sim 1,2$ horas	7,173°	5,919°	4,320°	5,804°
UnBFace_d	$\sim 1,2$ horas	6,973°	5,999°	4,612°	5,861°

Ressalta-se que o principal eixo de avaliação da orientação de cabeça oferecido pelo conjunto de dados AFW é o *yaw*, ou a direção que a pessoa está com a cabeça virada em torno do pescoço (para a direita ou para a esquerda), o que limita a quantidade de informação fornecida quando comparado com o conjunto de imagens mais completo fornecido por [27].

A adição da base de dados AFW, por ser muito menor que a base de dados AFLW, não apresentou modificação no tempo de treinamento, e, embora tenha apresentado piora no desempenho

do ângulo *roll*, houve uma diminuição de 0,2° no erro médio da estimativa do ângulo *yaw*, como era desejado. Por esse motivo, e também pelo fato de que mais exemplos de entrada significa uma gama maior de configurações observada pela solução, optou-se por manter os dados de entrada em conjunto, buscando resolver eventuais problemas de desempenho modificando-se a inicialização.

4.6 Treinamento da Rede Multi-canal

A última modificação realizada consistiu na implementação de uma rede multi-canais que fornece à solução uma informação mais detalhada. O segundo canal oferece à solução como imagem de entrada um recorte da região dos olhos de cada rosto que é alimentado à rede por completo. A posição dos olhos conforme capturada pela câmera é grande indicação dos ângulos *yaw*, geralmente dado pela visibilidade dos olhos, e *roll*, indicado pela inclinação da reta que conecta os dois olhos, que pode ser percebida pelas redes convolucionais com filtros que realizam operação de derivada.

A Figura 4.10 apresenta o histórico de treinamento da rede multi-canais, e a Tabela 4.6 compara o desempenho desta com a rede de canal de entrada único.

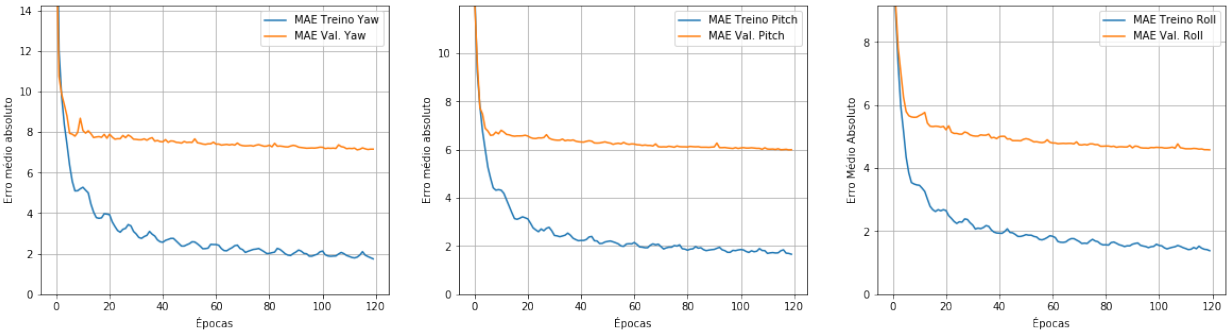


Figura 4.10: Histórico dos valores de MAE por época para a rede multi-canais, denominada UnBFace_e.

Tabela 4.6: Comparação do desempenho da UnBFace_d com uma outra versão intermediária do presente projeto, denominada UnBFace_e. A diferença entre as duas consiste na adição de um segundo canal de entrada com informações cruciais de orientação de dois dos três ângulos destacadas.

Rede	Tempo de Treino	Yaw MAE	Pitch MAE	Roll MAE	Rede MAE
UnBFace_d	$\sim 1,2$ horas	6,973°	5,999°	4,612°	5,861°
UnBFace_e	$\sim 2,3$ horas	7,156°	5,999°	4,584°	5,913°

De acordo com a Tabela 4.6, a adição do segundo canal de camadas acarretou num aumento demasiado do tempo de treinamento, sem apresentar melhora significativa que justificasse sua implementação. A principal justificativa para tal consiste na hipótese de que um recorte simples da região dos olhos não acrescenta o suficiente à gama de informação disponível para o treinamento

da rede, mas não é possível excluir outras hipóteses inerentes ao treinamento de uma rede neural (por exemplo, uma inicialização dos pesos das conexões em uma região que não permitiu melhor otimização) sem futuras repetições do treinamento com outras inicializações que não os pesos pré-treinados com a base de imagens ImageNet.

4.6.1 Melhorias Experimentais Na Solução: tamanho do *mini-batch*

Melhorias experimentais são mudanças que não estavam no planejamento do trabalho, mas que foram feitas visando diminuir o erro de maneira significativa. Durante todas as execuções descritas nas Seções 4.1, 4.3, 4.4, 4.5 e 4.6, o erro médio da solução obtido esteve sempre próximo dos 5,8°, a maior diferença sendo apresentada pela UnBFace_e, com 5,9°. Buscando alcançar uma taxa de erro que conseguisse romper a barreira dos 5,8°, decidiu-se alterar um dos parâmetros que permaneceram contante ao longo de todas as etapas planejadas: O tamanho do *mini-batch*, ou seja, a quantidade de imagens de treino alimentadas por iteração do algoritmo de *back-propagation*.

A Figura 4.11 apresenta o histórico de treinamento da solução multi-canais com tamanho de *mini-batch* de 128 imagens, a Figura 4.12, com tamanho de *mini-batch* de 256, e a Figura 4.13, tamanho de *mini-batch* de 64. Em seguida, a Figura 4.14 apresenta o histórico de treinamento da solução de canal único (UnBFace_d) com tamanho de *mini-batch* de 64 e a Tabela 4.7 compara o desempenho de cada instância citada.

Tabela 4.7: Comparação do resultados obtidos variando-se o tamanho do *mini-batch* nas configurações do treinamento da rede..

Arquitetura	*Batch size*	Tempo de execução	*Yaw* MAE	*Pitch* MAE	*Roll* MAE
UnBFace_e	128	2h55	7,18°	6,03°	4,57°
UnBFace_e	256	3h45	7,80°	6,50°	5,09°
UnBFace_e	64	2h05	6,82°	5,76°	4,22°
UnBFace_d	64	1h08	6,64°	5,70°	3,91°

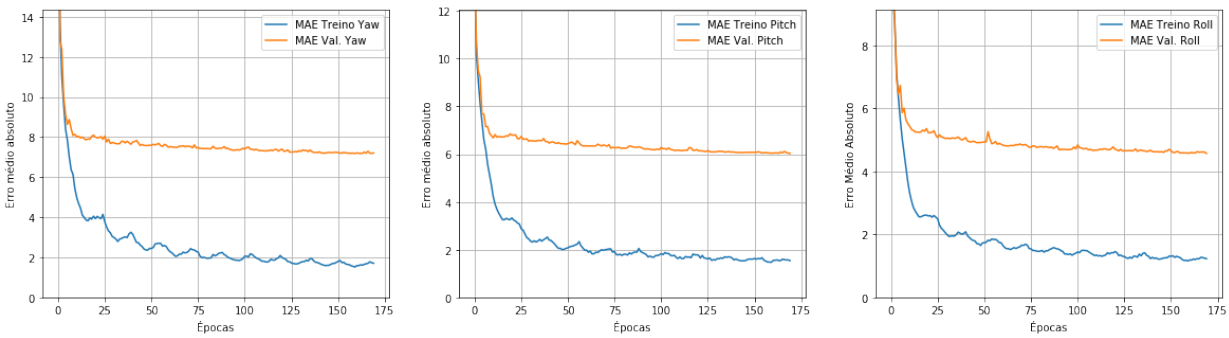


Figura 4.11: Histórico dos valores de MAE por época para a rede multi-canais, denominada UnBFace_e, treinada com mini-batch de tamanho 128 (originalmente 96).

Por se tratar de melhorias experimentais, optou-se por realizar diversos treinamentos com

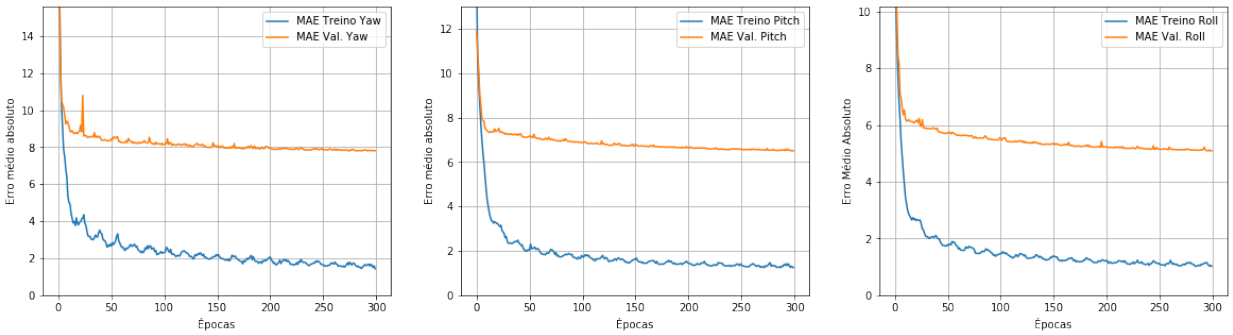


Figura 4.12: Histórico dos valores de MAE por época para a rede multi-canais, denominada UnBFace_e, treinada com mini-batch de tamanho 256 (originalmente 96).

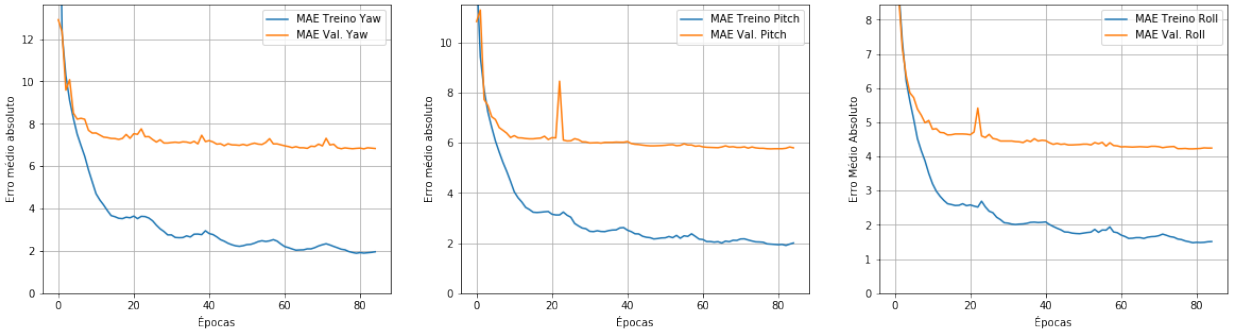


Figura 4.13: Histórico dos valores de MAE por época para a rede multi-canais, denominada UnBFace_e, treinada com mini-batch de tamanho 64 (originalmente 96).

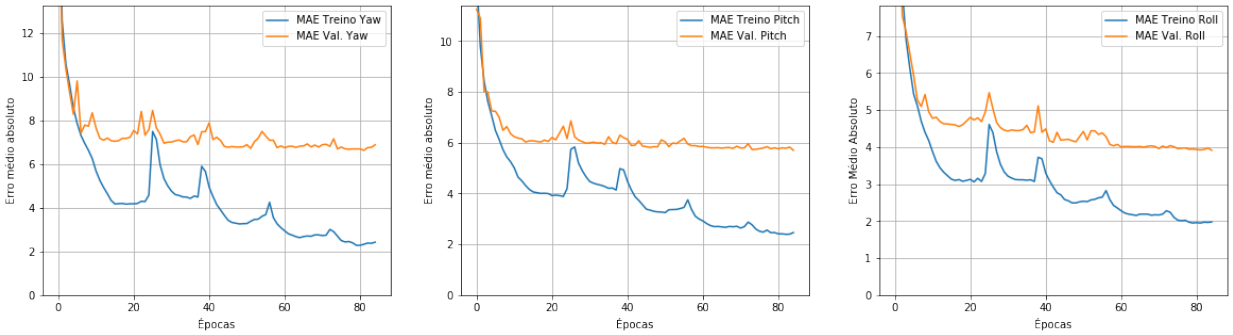


Figura 4.14: Histórico dos valores de MAE por época para a rede com base de dados combinada, denominada UnBFace_d, treinada com mini-batch de tamanho 64 (originalmente 96).

variadas inicializações para cada tamanho de *mini-batch*, e os resultados apresentados a nesta seção tratam-se dos melhores obtidos.

Capítulo 5

Conclusões

Redes neurais convolucionais são ferramentas poderosas que podem ser aplicadas às mais diversas tarefas do âmbito da visão computacional, entre estas a estimação da orientação da face de uma pessoa a partir de uma imagem comum RGB do rosto do usuário, sem informação adicional de profundidade.

Por outro lado, o projeto e execução de uma rede que seja capaz de resolver um problema de tamanha complexidade com desempenho satisfatório não é tarefa trivial. É possível encontrar na literatura estudos e projetos de soluções para o problema de estimar a orientação da face de um usuário a partir de uma imagem digital que datam do século passado. Encontra-se desde a utilização de métodos que estimam a orientação facial a partir do cálculo de propriedades geométricas do rosto[24] a exemplos de soluções que eliminam a necessidade de suposições prévias sobre a localização dos chamados marcos faciais (olhos, nariz, boca)[32]. Dentre os projetos mais aplicados se encontram os que fazem uso das Redes Neurais Convolucionais, utilizadas por Zhang e Zhang[29], Ranjan *et al.*[31], Ruiz *et al.*[32], Hsu*et al.*[4], entre outros, bem como pelo presente trabalho.

A presente solução, denominada UnBFace, apresenta uma rede neural convolucional baseada na arquitetura InceptionV3 adaptada para solução de problemas de regressão ao invés de classificação, implementada em código livre[42, 43] e processada em um ambiente gratuito de fácil acesso[48]. Treinada principalmente na base de dados AFLW[27], a UnBFace alcançou resultados comparáveis com valores presentes na literatura obtidos sob condições similares. A Tabela 5.1 apresenta um resumo comparativo dos valores expostos ao longo do Capítulo 4.

A melhor solução, tanto em termos de valor do erro médio dos três canais de estimação quanto do tempo de treinamento reduzido foi a arquitetura UnBFace_d_64: uma rede de canal único, com formato de entrada de 96x96, taxa de aprendizagem cíclica e tamanho do *mini-batch* de 64 imagens. Seu erro médio de 5,42° ultrapassa o resultado obtido por Hsu *et al.* em um projeto similar (EulerNet_b), e não fica muito aquém do obtido na versão final da EulerNet, 5,02°, uma vez que o conjunto de dados alimentados à EulerNet foi consideravelmente maior do que utilizado pela UnBFace, indicando uma possibilidade de melhora no desempenho com a adição de mais exemplos.

Tabela 5.1: Comparação do desempenho das diversas instâncias da UnBFace por meio dos erros médios das estimativas obtidos variando uma característica por vez, do formato dos dados de entrada (UnBFace_a para UnBFace_b) a adição de um segundo canal de entrada (UnBFace_d para UnBFace_e).

Rede	Tempo de Treino	Yaw MAE	Pitch MAE	Roll MAE	Rede MAE
UnBFace_a	$\sim 3,5$ horas	6,96°	6,07°	4,62°	5,88°
UnBFace_b	$\sim 1,1$ horas	7,17°	6,07°	4,34°	5,86°
UnBFace_c	$\sim 1,2$ horas	7,17°	5,92°	4,32°	5,80°
UnBFace_d	$\sim 1,2$ horas	6,97°	5,99°	4,61°	5,86°
UnBFace_e	$\sim 2,3$ horas	7,16°	5,99°	4,58°	5,91°
UnBFace_e_128	$\sim 2,9$ horas	7,18°	6,03°	4,57°	5,93°
UnBFace_e_256	$\sim 3,8$ horas	7,80°	6,50°	5,09°	6,46°
UnBFace_e_64	$\sim 2,1$ horas	6,82°	5,76°	4,22°	5,60°
UnBFace_d_64	$\sim 1,1$ horas	6,64°	5,70°	3,91°	5,42°

Com isso, a Seção 5.1, Trabalhos Futuros, busca oferecer soluções e sugestões aos problemas encontrados durante o desenvolvimento do presente trabalho.

5.1 Trabalhos futuros

Ângulos de Euler estão sujeitos a um problema conhecido como *Gimbal Lock*, que pode ter sérias consequências se ocorrer em sistemas de controle críticos. Para resolver essa questão utiliza-se a representação de rotação no espaço tridimensional utilizando quatérnions. A mudança para quatérnions, porém, implica na necessidade de modificar o projeto tanto na estrutura quanto no tratamento do dados (como os quatérnions devem ser unitários, é necessário aplicar alguma estratégia de regularização da norma), e portanto torna-se necessário aprofundar o estudo dessa representação de rotação antes de se aplicar ao projeto.

Independentemente da maneira de descrever a informação de orientação no espaço tridimensional, os resultados experimentais obtidos demonstram a necessidade de uma base de dados mais abrangente. Além da AFLW e da AFW utilizadas para treinar a UnBFace, é interessante agrupá-las a outros conjuntos de imagens para estimação de orientação facial, como o 300W-LP[1] e o BIWI[2].

Um próximo passo natural a este trabalho é a aplicação da UnBFace e suas estimativas em um sistema de controle real, ainda que simplificado, para validação da confiabilidade dos valores estimados e de sua usabilidade em sistemas de controle, como ponto de partida da motivação deste presente projeto.

[1] `http://www.cbsr.ia.ac.cn/users/xiangyuzhu/projects/3DDFA/main.htm`
[2] `https://data.vision.ee.ethz.ch/cvl/gfanelli/head_pose/head_forest.html`

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ZAGHETTO, C. Detecção do mal-posicionamento rotacional de dedos em dispositivos de captura de impressões digitais multivista sem toque utilizando redes neurais artificiais. 2016.

[2] CHUCK, W. *Man Wears Orange Knit Cap and Shirt*. 2019. `https://www.pexels.com/photo/man-wears-orange-knit-cap-and-shirt-2802601/`.

[3] RALD, J. *Standing Man Wearing Gray and Black Outer S*. 2012. `https://www.pexels.com/photo/standing-man-wearing-gray-and-black-outer-s-2918513/`.

[4] HSU, H.-W. et al. QuatNet: Quaternion-based head pose estimation with multiregression loss. *IEEE Transactions on Multimedia*, IEEE, v. 21, n. 4, p. 1035–1046, 2018.

[5] SZEGEDY, C. et al. Going deeper with convolutions. In: *Computer Vision and Pattern Recognition (CVPR)*. [s.n.], 2015. Disponível em: <http://arxiv.org/abs/1409.4842>.

[6] SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 2818–2826.

[7] HAYKIN, S. S. et al. *Neural networks and learning machines*. [S.l.]: New York: Prentice Hall, 2009.

[8] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] MURPHY-CHUTORIAN, E.; TRIVEDI, M. M. Head pose estimation in computer vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 31, n. 4, p. 607–626, 2008.

[10] LIU, W. et al. A survey of deep neural network architectures and their applications. *Neurocomputing*, Elsevier, v. 234, p. 11–26, 2017.

[11] LECUN, Y. et al. Gradient-based learning applied to document recognition. In: *PROCEEDINGS OF THE IEEE*. [S.l.: s.n.], 1998. p. 2278–2324.

[12] DUDA, R. O.; STORK, D. G.; HART, P. E. *Pattern classification*. [S.l.]: New York : John Wiley & Sons, c2001., 2001. ISBN 0471056693.

[13] FUKUNAGA, K. *Introduction to statistical pattern recognition.* [S.l.]: Elsevier, 2013.

[14] SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding machine learning: From theory to algorithms.* [S.l.]: Cambridge university press, 2014.

[15] HASTIE, T.; FRIEDMAN, J. H.; TIBSHIRANI, R. *The elements of statistical learning : data mining, inference, and prediction.* New York : Springer, c2009., 2009. (Springer Series in Statistics). ISBN 9780387848587. Disponível em: <https://web.stanford.edu/ hastie/Papers/ESLII.pdf>.

[16] SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: an introduction; 2nd ed.* Cambridge, MA: The MIT Press, 2018. (Adaptive computation and machine learning). Disponível em: <http://cds.cern.ch/record/2640193>.

[17] MARQUES, E. A. L. *Estudo sobre redes neurais de aprendizado profundo com aplicações em classificação de imagens.* 2016. Disponível em: <http://bdm.unb.br/handle/10483/15147>.

[18] WILLMOTT, C. J.; MATSUURA, K. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate research*, v. 30, n. 1, p. 79–82, 2005.

[19] CHIANG, S. L. A. P. *Avaliação do desempenho da plataforma Intel Galileo Geração 2 no treinamento de redes neurais artificiais.* 2018. Disponível em: <http://bdm.unb.br/handle/10483/20162>.

[20] BAUCHSPIESS, A. Introdução aos sistemas inteligentes. *Aplicações em Engenharia de Redes Neurais Artificiais, Lógica Fuzzy e Sistemas Neuro-Fuzzy*, 2004.

[21] BISHOP, C. M. *Pattern recognition and machine learning.* [S.l.]: Springer Science+ Business Media, 2006.

[22] LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989.

[23] FERRARIO, V. F. et al. Active range of motion of the head and cervical spine: a three-dimensional investigation in healthy young adults. *Journal of orthopaedic research*, Wiley Online Library, v. 20, n. 1, p. 122–129, 2002.

[24] CHEN, Q. et al. A robust algorithm for 3D head pose estimation. In: IEEE. *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No. 98EX170).* [S.l.], 1998. v. 2, p. 1356–1359.

[25] SEEMANN, E.; NICKEL, K.; STIEFELHAGEN, R. Head pose estimation using stereo vision for human-robot interaction. In: IEEE. *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.* [S.l.], 2004. p. 626–631.

[26] VOIT, M.; NICKEL, K.; STIEFELHAGEN, R. Multi-view head pose estimation using neural networks. In: IEEE. *The 2nd Canadian Conference on Computer and Robot Vision (CRV'05).* [S.l.], 2005. p. 347–352.

[27] KOESTINGER, M. et al. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In: IEEE. *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. [S.l.], 2011. p. 2144–2151.

[28] ZHU, X.; RAMANAN, D. Face detection, pose estimation, and landmark localization in the wild. In: IEEE. *2012 IEEE conference on computer vision and pattern recognition*. [S.l.], 2012. p. 2879–2886.

[29] ZHANG, C.; ZHANG, Z. Improving multiview face detection with multi-task deep convolutional neural networks. In: IEEE. *IEEE Winter Conference on Applications of Computer Vision*. [S.l.], 2014. p. 1036–1041.

[30] JAIN, V.; LEARNED-MILLER, E. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. [S.l.], 2010.

[31] RANJAN, R.; PATEL, V. M.; CHELLAPPA, R. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 41, n. 1, p. 121–135, 2017.

[32] RUIZ, N.; CHONG, E.; REHG, J. M. Fine-grained head pose estimation without keypoints. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. [S.l.: s.n.], 2018.

[33] PAN, S. J.; YANG, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, IEEE, v. 22, n. 10, p. 1345–1359, 2009.

[34] ROSENSTEIN, M. T. et al. To transfer or not to transfer. In: *NIPS 2005 workshop on transfer learning*. [S.l.: s.n.], 2005. v. 898, p. 3.

[35] KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] SMITH, L. N. Cyclical learning rates for training neural networks. In: IEEE. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. [S.l.], 2017. p. 464–472.

[37] ZHU, X. et al. Face alignment across large poses: A 3D solution. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 146–155.

[38] 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.

[39] GUYON, I. A scaling law for the validation-set training-set size ratio. *AT&T Bell Laboratories*, Citeseer, p. 1–11, 1997.

[40] KRIZHEVSKY, A.; HINTON, G. et al. *Learning multiple layers of features from tiny images*. [S.l.], 2009.

[41] COATES, A.; NG, A.; LEE, H. An analysis of single-layer networks in unsupervised feature learning. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. [S.l.: s.n.], 2011. p. 215–223. Dataset disponível online em: `http://cs.stanford.edu/~acoates/stl10`.

[42] CHOLLET, F. et al. *Keras*. 2015. `https://keras.io`.

[43] ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <http://tensorflow.org/>.

[44] WALT, S. van der et al. scikit-image: image processing in Python. *PeerJ*, v. 2, p. e453, 6 2014. ISSN 2167-8359. Disponível em: <https://doi.org/10.7717/peerj.453>.

[45] OLIPHANT, T. E. *A guide to NumPy*. [S.l.]: Trelgol Publishing USA, 2006.

[46] HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

[47] KENSTLER, B. *Cyclical Learning Rate (CLR)*. [S.l.]: GitHub, 2018. `https://github.com/bckenstler/CLR`.

[48] CARNEIRO, T. et al. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, IEEE, v. 6, p. 61677–61685, 2018.

[49] APELAND, S. *Intel® AI DevCloud*. 2017. `https://www.intel.ai/devcloud/`. Acesso em 23 de Novembro de 2019.

[50] CORPORATION, I. *Data-Centric Workloads for AI Training and Inference*. `https://software.intel.com/en-us/devcloud/datacenter`. Acesso em 23 de Novembro de 2019.

[51] COHEN, J. P. *Visualizing CNN architectures side by side with MXNet*. 2016. `http://josephpcohen.com/w/visualizing-cnn-architectures-side-by-side-with-mxnet/`. Acesso em 23 de Novembro de 2019.

APÊNDICE

I. CÓDIGOS

Todos os códigos necessários para a reprodução da solução apresentada se encontram em um repositório no GitHub, que pode ser acessado pelo link:

`https://github.com/RSinhoroto/UnBFace/`

É possível acessar diretamente um caderno no Google Colab contendo o treinamento de uma versão básica da UnBFace utilizando a base de dados pública AFW pelo link:

`https://colab.research.google.com/drive/1njYv8QhSisQfhuQZ_0sJ8iuMRZHwHVoS`

II. CÓDIGO EM PYTHON PARA TREINAMENTO DA REDE MULTICANAL

```
# import general and image processing libraries
from sklearn.model_selection import train_test_split
from skimage.io import imread, imshow, show
from contextlib import redirect_stdout
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import time

# import keras modules
from keras.models import Model
from keras.optimizers import SGD
from keras.applications import InceptionV3
from keras.layers import Dense, concatenate
from keras.callbacks import ModelCheckpoint, EarlyStopping

# import cyclic learning rate implementation
from clr_callback import CyclicLR

def plot_training(history, ymin=0, ymax=50):
    fig, axes = plt.subplots(1, 3, figsize=(20, 5))
    axes[0].plot(history.history['dense_2_mae'], label='Yaw Train MAE')
    axes[0].plot(history.history['val_dense_2_mae'], label='Yaw Val MAE')
    axes[0].set_xlabel('Epochs')
    ymax = 2*np.min(history.history['val_dense_2_mae'])
    axes[0].set_ylim([ymin,ymax])
    axes[0].legend()

    axes[1].plot(history.history['dense_4_mae'], label='Pitch Train MAE')
    axes[1].plot(history.history['val_dense_4_mae'], label='Pitch Val MAE')
    axes[1].set_xlabel('Epochs')
    ymax = 2*np.min(history.history['val_dense_4_mae'])
    axes[1].set_ylim([ymin,ymax])
    axes[1].legend()

    axes[2].plot(history.history['dense_6_mae'], label='Roll Train MAE')
    axes[2].plot(history.history['val_dense_6_mae'], label='Roll Val MAE')
    axes[2].set_xlabel('Epochs')
    ymax = 2*np.min(history.history['val_dense_6_mae'])
    axes[2].set_ylim([ymin,ymax])
    axes[2].legend()
```

```
    return fig


timestamp = time.strftime('%Y_%m_%d_%Hh%Mm')

data_file = open('aflw_crops_poses_radians.txt','r')
data_file_content = data_file.readlines()

filenames = []
poses = []
for line in data_file_content:
  name, pose = line.split(':')
  filenames.append(name)
  poses.append(pose)

# load AFLW
face_data = []
eye_data = []
data_target = []

for i in range(len(filenames)):
# get pose
  current_pose = poses[i][1:-2]
  angles = current_pose.split(', ')
  data_target.append([float(angles[0]),
                      float(angles[1]),
                      float(angles[2])])

# load correspondent face crop
  im_path = "crops/" + filenames[i]
  img = imread(im_path, as_gray=False, plugin='matplotlib')
  face_data.append(img)

# load correspondent eye crop
  im_path = "eyes/" + filenames[i]
  img = imread(im_path, as_gray=False, plugin='matplotlib')
  eye_data.append(img)

  if i % 1000 == 0:
      print(str(i) + " images processed.")

# convert to np array
face_data = np.asarray(face_data)
eye_data = np.asarray(eye_data)

# load AFW
```

```
data_file = open('afw_crops_poses_radians.txt','r')
data_file_content = data_file.readlines()

filenames = []
poses = []
for line in data_file_content:
  name, pose = line.split(':')
  filenames.append(name)
  poses.append(pose)


afw_face_data = []
afw_eye_data = []
afw_data_target = []

for i in range(len(filenames)):
# get pose
  current_pose = poses[i][1:-2]
  angles = current_pose.split(', ')
  afw_data_target.append([float(angles[0]),
                          float(angles[1]),
                          float(angles[2])])

# load correspondent face crop
  im_path = "crops/" + filenames[i]
  img = imread(im_path, as_gray=False, plugin='matplotlib')
  afw_face_data.append(img)

# load correspondent eye crop
  im_path = "eyes/" + filenames[i]
  img = imread(im_path, as_gray=False, plugin='matplotlib')
  afw_eye_data.append(img)


face_data = np.append(face_data, np.asarray(afw_face_data), axis=0)
eye_data = np.append(eye_data, np.asarray(afw_eye_data),axis=0)
data_target = np.append(data_target, afw_data_target, axis=0)

face_train, face_test, eye_train, eye_test, train_target, test_target = train_test_split(
    face_data, eye_data, data_target, test_size=0.1, random_state=42 )

del face_data
del eye_data
del data_target

# declare instances of base InceptionV3(GoogLeNet) model
```

```
base_model = InceptionV3(include_top=False,
                         pooling='avg', input_shape=(96,96,3))
base_model_eye = InceptionV3(include_top=False,
                             pooling='avg', input_shape=(76,152,3))

# rename to use two instances of InceptionV3
for layer in base_model_eye.layers:
    layer.name = 'eye_' + layer.name

# add fully connect regression layers
main_out = base_model.output
eye_out = base_model_eye.output
out = concatenate([main_out, eye_out])

#yaw
yaw = Dense(256, activation='relu')(out)
yaw = Dense(1, activation='linear')(yaw)

#pitch
pitch = Dense(256, activation='relu')(out)
pitch = Dense(1, activation='linear')(pitch)

#roll
roll = Dense(256, activation='relu')(out)
roll = Dense(1, activation='linear')(roll)


model = Model(inputs=[base_model.input,
                      base_model_eye.input],
              outputs=[yaw, pitch, roll])

# configure some hyper parameters
INIT_LR = 5e-3
EPOCHS = 100
BATCH_SIZE = 96
#STEPS_PER_EPOCH = 320,
VALIDATION_STEPS = 64

# add cyclical learning rate callback
MIN_LR = 1e-7
MAX_LR = 1e-2
CLR_METHOD = "triangular"
STEP_SIZE = 4

clr = CyclicLR(mode=CLR_METHOD,
               base_lr=MIN_LR,
               max_lr=MAX_LR,
```

```
                step_size=(STEP_SIZE * (np.shape(face_train)[0] // BATCH_SIZE)))

# add checkpoint to save the network and stop if training doesn't improve
filepath = "../best_weights_" + timestamp + ".hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min')
earlystop = EarlyStopping(monitor='val_loss', patience=50)
callbacks_list = [checkpoint, earlystop, clr]

# compile complete model with optmizer and print summary on screen
optim = SGD(lr=INIT_LR, momentum=0.9)
model.compile(optimizer=optim,
              loss='mean_squared_error',
              metrics=['mae'])

with open("../model_summary_" + timestamp + ".txt", 'w') as f:
    with redirect_stdout(f):
        model.summary()

# train model and plot training history
history = model.fit(x=[face_train, eye_train],
                    y=[np.transpose(train_target)[0],
                       np.transpose(train_target)[1],
                       np.transpose(train_target)[2],],
                    epochs=EPOCHS,
                    validation_data=([face_test, eye_test],
                                     [np.transpose(test_target)[0],
                                      np.transpose(test_target)[1],
                                      np.transpose(test_target)[2],]
                                     ),
                    batch_size=BATCH_SIZE,
                    callbacks=callbacks_list)

history_plot = plot_training(history)
history_plot.savefig("../training_history_" + timestamp)
```

III. CÓDIGO PARA ANÁLISE DA DIMENSÃO DOS DADOS DE ENTRADA NO AMBIENTE GOOGLE COLAB

```
"""
AFW_Facial_Pose_Recognition.py

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1njYv8QhSisQfhuQZ_0sJ8iuMRZHwHVoS
"""

!nvidia-smi -L

# download dataset files
!git clone https://github.com/RSinhoroto/AFWDataset.git
%cd AFWDataset
!ls

%tensorflow_version 1.x

# import general and image processing libraries
from skimage.io import imread, imshow, show
from skimage.transform import resize
import matplotlib.pyplot as plt
from math import degrees
import tensorflow as tf
import numpy as np

# import keras modules
from keras.models import Model
from keras.optimizers import SGD
from keras.layers import Dense, concatenate
from keras.applications import InceptionV3
from keras.callbacks import EarlyStopping

def plot_training(history, ymin=0, ymax=50):
    fig, axes = plt.subplots(1, 3, figsize=(20, 5))
    axes[0].plot(history.history['dense_2_mean_absolute_error'],
                 label='Yaw Train MAE')
    axes[0].plot(history.history['val_dense_2_mean_absolute_error'],
                 label='Yaw Val MAE')
```

```
    axes[0].set_xlabel('Epochs')
    ymax = 2*np.mean(history.history['val_dense_2_mean_absolute_error'][10:])
    axes[0].set_ylim([ymin,ymax])
    axes[0].legend()

    axes[1].plot(history.history['dense_4_mean_absolute_error'],
                 label='Pitch Train MAE')
    axes[1].plot(history.history['val_dense_4_mean_absolute_error'],
                 label='Pitch Val MAE')
    axes[1].set_xlabel('Epochs')
    ymax = 2*np.mean(history.history['val_dense_4_mean_absolute_error'][10:])
    axes[1].set_ylim([ymin,ymax])
    axes[1].legend()

    axes[2].plot(history.history['dense_6_mean_absolute_error'],
                 label='Roll Train MAE')
    axes[2].plot(history.history['val_dense_6_mean_absolute_error'],
                 label='Roll Val MAE')
    axes[2].set_xlabel('Epochs')
    ymax = 2*np.mean(history.history['val_dense_6_mean_absolute_error'][10:])
    axes[2].set_ylim([ymin,ymax])
    axes[2].legend()

imgs_file = open('filenames.txt','r')
imgs_file_content = imgs_file.read()
filenames = imgs_file_content.split('*')
if filenames[-1] == '':
  filenames = filenames[:-1]     #removes last empty element
imgs_file.close()

faces_file = open('faces.txt','r')
faces_file_content = faces_file.read()
faces = faces_file_content.split('*')
if faces[-1] == '':
  faces = faces[:-1]     #removes last empty element
faces_file.close()

poses_file = open('poses.txt','r')
poses_file_content = poses_file.read()
poses = poses_file_content.split('*')
if poses[-1] == '':
  poses = poses[:-1]     #removes last empty element
poses_file.close()

img_side = 96

afw_face_data = []
```

```
afw_data_target = []
for i in range(len(filenames)):
  name = filenames[i]
  face_locations = faces[i][1:-1]
  current_pose = poses[i][1:-1]

  if face_locations.find(’][’):
    face_locations = face_locations.split(’][’)
    current_pose = current_pose.split(’][’)

  img = imread(’images/’ + name, as_gray=False)
  for j in range(len(face_locations)):
    coords = face_locations[j]
    ypr = current_pose[j]

    points = coords.split(’;’)
    x0,y0 = points[0].split(’ ’)
    x1,y1 = points[1].split(’ ’)
    x0 = int(round(float(x0)))
    y0 = int(round(float(y0)))
    x1 = int(round(float(x1)))
    y1 = int(round(float(y1)))

    angles = ypr.split(’ ’)
    afw_data_target.append([float(angles[0]),
                            float(angles[1]),
                            float(angles[2])])

    img_crop = img[y0:y1,x0:x1]
    img_crop = resize(img_crop, (img_side,img_side, 3))  # resize to net input size
    afw_face_data.append(img_crop)

# convert to numpy array
afw_face_data = np.asarray(afw_face_data)

# declare instances of base InceptionV3(GoogLeNet) model
base_model = InceptionV3(weights=’imagenet’, include_top=False,
                         pooling=’avg’, input_shape=(img_side,img_side,3))

# add fully connect regression layers
out = base_model.output

#yaw
yaw = Dense(256, activation=’relu’)(out)
yaw = Dense(1, activation=’linear’)(yaw)

#pitch
```

```
pitch = Dense(256, activation='relu')(out)
pitch = Dense(1, activation='linear')(pitch)

#roll
roll = Dense(256, activation='relu')(out)
roll = Dense(1, activation='linear')(roll)

model = Model(inputs=[base_model.input],
              outputs=[yaw, pitch, roll])

# configure some hyper parameters
INIT_LR = 5e-3
EPOCHS = 300
BATCH_SIZE = 96

# add checkpoint to stop if training doesn't improve
earlystop = EarlyStopping(monitor='val_loss', patience=50)
callbacks_list = [earlystop]

# compile complete model with optmizer and print summary on screen
optim = SGD(lr=INIT_LR, decay=1e-2/EPOCHS)
model.compile(optimizer=optim,
              loss='mean_squared_error',
              metrics=['mean_absolute_error'])
print(model.summary())

!nvidia-smi

# train model and plot training history
history = model.fit(x=[afw_face_data],
                    y=[np.transpose(afw_data_target)[0],
                       np.transpose(afw_data_target)[1],
                       np.transpose(afw_data_target)[2],],
                    epochs=EPOCHS,
                    validation_split=0.25,
                    batch_size=BATCH_SIZE,
                    callbacks=callbacks_list)

plot_training(history)
```

ANEXOS

I. GOOGLENET (INCEPTIONV1)

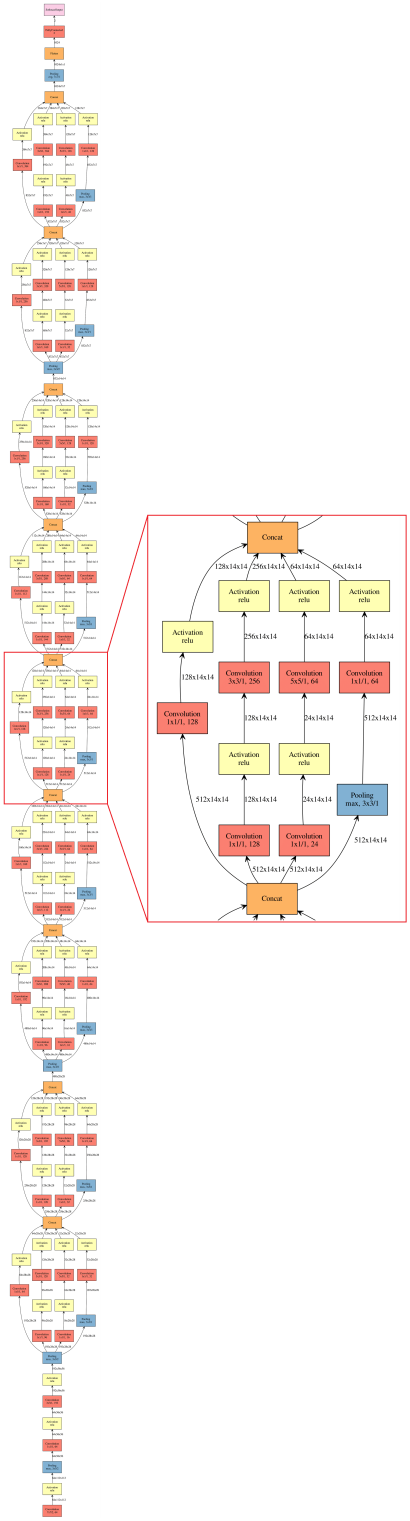


Figura I.1: Camadas da GoogLeNet (Inception V1)[5] (Fonte: [51]).

II. INCEPTIONV3

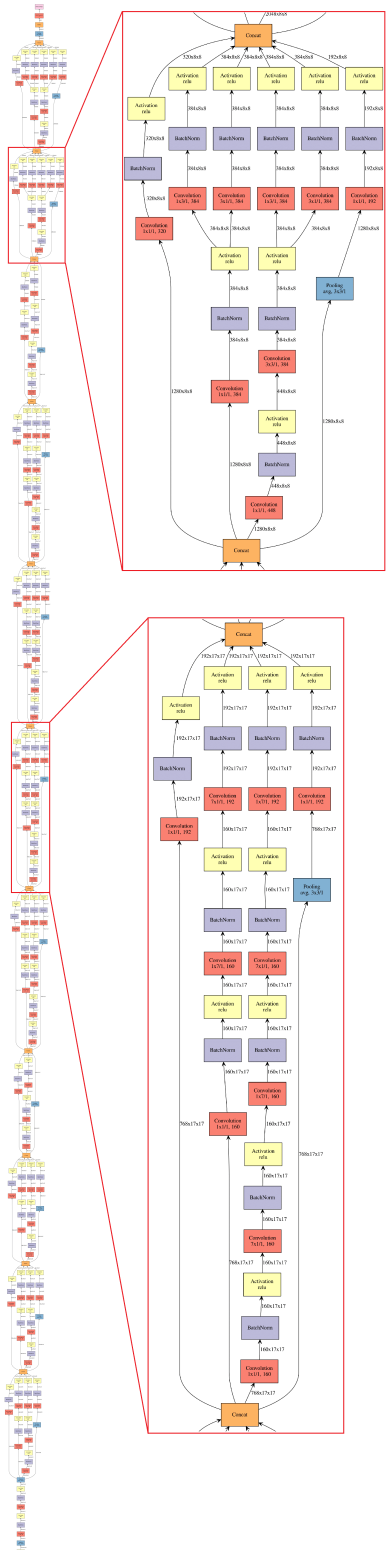


Figura II.1: Camadas da Inception V3[6] (Fonte: [51]).