UNIVERSIDADE FEDERAL DO MARANHÃO

Curso de Ciência da Computação

Jorge Luis Melo Ribeiro

Meta-aprendizado aplicado ao Problema de Reconhecimento de Expressões Faciais

São Luís - MA

2018

Jorge Luis Melo Ribeiro

Meta-aprendizado aplicado ao Problema de Reconhecimento de Expressões Faciais

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Curso de Ciência da Computação

UFMA

Orientador: Prof. Dr. Geraldo Braz Jr.

São Luís - MA

2018

Jorge Luis Melo Ribeiro

Meta-aprendizado aplicado ao Problema de Reconhecimento de Expressões Faciais

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado em São Luís - MA, 7 de janeiro de 2019:

Prof. Dr. Geraldo Braz Jr.
Orientador

Profa. Ma. Vandecia Rejane Monteiro Fernandes
Examinadora

Prof. Dr. João Dallyson Sousa de Almeida
Examinador

São Luís - MA
2018

Agradecimentos

Inicio agradecendo a aqueles que foram os principais responsáveis para que eu chegasse até aqui: Jorge Luis Pereira Ribeiro e Francisca Maria Melo Ribeiro. Sem o exemplo de meus pais eu não teria todas as virtudes que possuo e que recebi dos dois, dentre elas, as principais: respeito, responsabilidade, dedicação e amor. Ambos lutaram muito para que eu realizasse meus sonhos, e este de forma especial, a quem eu dedico inicialmente.

A meus irmãos Marco André e Lucas Raphael, que cresceram comigo, sempre foram meus melhores amigos e nunca deixaram que nenhum mal atrapalhasse a união de nossa família.

À minha namorada, Sammyra Camêlo, por apoiar minhas decisões e me dedicar palavras de força e encorajamento nos momentos em que eu fraquejava, e sempre ser uma perfeita companhia.

Aos meus recém-afilhados, Allana e Felipe, que tornaram-se parte da minha família e com quem compartilho de belos momentos e ótimas lembranças.

Aos meus grandes amigos (que são tantos que não poderia citar todos aqui), com quem consigo me sentir querido e por me ajudarem em diversos momentos que necessitei deles.

Aos colegas de graduação, com quem vivi momentos de emoções diversas, mas que sempre foram apoio nas dificuldades acadêmicas. Em especial a Julia Manayra, que apesar da distância nesse final de graduação, sempre me dava bons conselhos quando nos encontrávamos pelos corredores da UFMA. Aos professores Dr. Alexandre César, Dr. Anselmo Paiva, Dra. Simara Rocha, por serem inspiração na minha vida acadêmica e me motivarem a buscar sempre o crescimento pessoal.

Ao meu orientador, professor Dr. Geraldo Braz Junior, por também me inspirar e sempre orientar corretamente durante a graduação, principalmente nos últimos semestres, e tornar esse final de vida acadêmica mais fácil com seu apoio, dentro e fora de sala de aula.

Por último a Deus, e não por menos importância, mas sim porque foi Ele quem me permitiu conhecer e viver ao lado de todas as pessoas mencionadas acima, além de iluminar e abençoar a minha vida em todos os momentos, me dando as graças necessárias para que eu alcançasse aquilo que eu mereço.

"Escolha um trabalho
que você ame
e não terás que trabalhar
um único dia na sua vida."

Confúcio

Resumo

De acordo com o professor de psicologia Albert Mehrabian, estudioso da área de comunicação humana, 90% da expressão humana é não-verbal. O homem, ao se comunicar, expressa de diversas formas aquilo que está sentido, especialmente através de reações. As reações faciais dão contexto e significado à fala humana, sendo os gestos executados de extrema importância para a compreensão do interlocutor. Diante disso, no contexto da evolução de sistemas inteligentes que tratam da interação com humanos, o entendimento correto da emoção sentida pelo homem pode auxiliar na resposta correta ou mais adequada retornada pelo sistema. Em muitos estudos e pesquisas se tem utilizado redes neurais para o treinamento e classificação de aplicações de aprendizado de máquina. Para se construir uma rede neural precisa-se primeiro escolher a sua arquitetura, sendo esta definida a partir de alguns parâmetros específicos, chamados de hiperparâmetros. As redes neurais convolucionais (CNNs) são utilizadas especificamente para problemas que envolvem imagens como entrada de dados, e também precisam ter seus hiperparâmetros definidos. A escolha e definição dos valores desses hiperparâmetros é um problema a ser resolvido nas redes neurais, pois precisa ser feito de forma empírica. Alguns otimizadores já são utilizados para realizar a escolha dos melhores hiperparâmetros, que fazem essa validação por tentativa e erro dentro de um espaço de busca. Alguns podem ser citados, como é o caso do *Grid Search* e do *Random Search*. Diante do exposto, a proposta deste trabalho é utilizar a biblioteca de otimização *hyperopt* para otimizar os hiperparâmetros de CNNs que irão realizar o treinamento e classificação de expressões faciais humanas.

Palavras-chaves: redes neurais convolucionais, hiperparâmetros, meta-aprendizado, reconhecimento de expressões faciais

Abstract

According to the psychology professor Albert Mehrabian, scholar in the human communication field, 90% of the human expression is non-verbal. Humans, when communicating, express in several ways what he or she is feeling, specially through reactions. The facial reactions give context and meaning to the human speech, being the executed gestures of extreme importance to the comprehension of the interlocutor. With that said, in the context of evolution of intelligent systems that deal with interaction with humans, the correct comprehension of human's emotion can assist in the correct or more appropriate answer returned by the system. Many studies and researches have used neural networks for training and classification of machine learning applications. To build a neural network it is first necessary to choose its arquitecture, and this is defined with some specific parameters, called hyperparemeters. The convolutional neural networks (CNNs) are utilized specifically to problems that involve images as input, and also need to have their hyperparameters defined. The definition of these hyperparameters' values is a problem to be solved in neural networks, because it has to be done via rules-of-thumb. Some optimizers have being used to pick the best hyperparameters, and do the validation by trial and error within a search space. A few can be cited, such as Grid Search and Random Search. Based on the above, the proposal of this work is to utilize a optimization libray called hyperopt to optimize the hyperparemeters of CNNs that will train and classify human facial expressions.

Keywords: convolutional neural networks, hyperparameters, meta learning, facial expression recognition

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AAM	*Active Appearance Model*
CK+	*Extended Cohn Kanade*
CNN	*Convolutional Neural Network*
EI	*Expected Improvement*
GPU	*Graphics Processing Unit*
JAFFE	*Japanese Female Facial Expression*
LBP	*Local Binary Pattern*
LDA	*Local Discriminant Analysis*
ReLU	*Rectified Linear Unit*
RNA	*Rede Neural Artificial*
RNP	*Rede Neural Profunda*
SMBO	*Sequential Model-Based*
SVN	*Support Vector Machine*
TPE	*Tree-of-Parzen-Estimators*
UFMA	*Universidade Federal do Maranhão*

Sumário

1 Introdução

Um dos campos de pesquisa de inteligência artificial diz respeito ao aperfeiçoamento da interação das máquinas com o homem. Antes do surgimento e popularização dos Sistemas Inteligentes, era mais comum que o usuário se adaptasse ao sistema que fizesse uso, aprendendo seu funcionamento e a forma correta de utilização. Com os avanços obtidos nos estudos realizados em aprendizado de máquina, o papel de adaptação se inverteu. Os sistemas desenvolvidos para interagir com o homem buscam entender as necessidades do usuário e aprendem a partir do modo como são utilizados.

Nesse contexto, uma das formas que auxiliam no aprendizado de máquina é analisar e compreender o sentimento do homem. Para as empresas, por exemplo, é importante saber o que seus clientes pensam sobre seus serviços oferecidos (PANG; LEE et al., 2008), e por esse motivo investem em pesquisas de análise de sentimento a partir de textos extraídos principalmente da internet que dão opiniões sobre seus produtos ou serviços.

Em um campo mais amplo, o reconhecimento automático de emoções em humanos tornou-se objeto de estudo de grande interesse nas últimas décadas. Tendo aplicações que vão desde a melhora da qualidade de trabalho de funcionários até o aprimoramento da experiência de usuário em websites (KOŁAKOWSKA et al., 2014). A compreensão da emoção sentida pelo homem nas mais diversas situações é um grande passo para o aprimoramento de Sistemas Inteligentes construídos para este fim.

Nessa linha, Miskam et al. (2014) apresentou um estudo que utiliza um robô humanoide para interagir com crianças portadoras do Transtorno de Asperger, popularmente conhecido como autismo. O robô é capaz de reconhecer emoções e interagir com as crianças, que costumam apresentar dificuldades em comunicação, comportamento e interação social. O objetivo do estudo é proporcionar melhoras no desenvolvimento social e na resposta aos sentimentos de outras pessoas, tarefas as quais crianças autistas apresentam dificuldades em realizar.

De forma específica dentro do estudo de reconhecimento de emoções, surgiu grande interesse no reconhecimento de expressões faciais de forma a aprimorar a interação humano-computador. Na última década, diversos trabalhos foram feitos nesse campo, como por exemplo (SARODE; BHATIA, 2010), (MANGLIK et al., 2004) e (BARTLETT et al., 2003). Nos três estudos é citada a dificuldade de se obter bons resultados, devido aos desafios que existem na tarefa de classificar emoções humanas. Alguns desses desafios são: a carência de *datasets* extensos; a ambiguidade que uma expressão facial pode apresentar; a dificuldade dos computadores reconhecerem emoções faciais em situações naturais.

Além dos trabalhos citados no parágrafo anterior, outros autores buscaram diferentes métodos para resolver o problema do reconhecimento de expressões faciais. Antes da popularização das redes neurais profundas, foram propostas diferentes técnicas para realizar a classificação das emoções. O trabalho de Shan, Gong e McOwan (2009) baseia-se na análise das expressões através de características *Local Binary Pattern (LBP)*, e o aprendizado das emoções por meio de diferentes métodos como *Template Matching*, *Support Vector Machine (SVN)* e *Linear Discriminant Analysis (LDA)*. Nesse trabalho os autores utilizaram o *dataset Cohn-Kanade* (KANADE; TIAN; COHN, 2000) gerado em laboratório sob condições controladas, obtendo acurácia acima de 90% em algumas das classes.

Uma outra técnica foi proposta por Dhavalikar e Kulkarni (2014) e consiste em três fases: detecção de face, extração de características e reconhecimento da expressão. A extração de características segmentava os principais elementos do rosto humano como olhos, nariz e boca, utilizando o método *Active Appearance Model (AAM)*. Com os elementos segmentados, a expressão era reconhecida calculando-se a distância euclidiana entre os pontos dos segmentos, sendo esse valor comparado com a distância euclidiana de imagens de treinamento. A emoção que apresentasse menor diferença na comparação é então escolhida.

Trabalhos mais recentes (LEVI; HASSNER, 2015), (MOLLAHOSSEINI; CHAN; MAHOOR, 2016) e (PRAMERDORFER; KAMPEL, 2016a) fizeram uso das redes neurais convolucionais (*Convolutional Neural Networks* - CNN), uma classe de redes neurais que recebem imagens como entrada de dadoss. Esse método de *Deep Learning* tem demonstrado grande capacidade de classificação em problemas de aprendizado de máquina. Essa demonstração pode ser verificada nos últimos resultados do *ImageNet Challenge*, desafio que ocorre todos os anos e disponibiliza um dos maiores *datasets* de imagens anotadas, que possui mais de 14 milhões de imagens (DENG et al., 2014). Nas últimas edições as redes neurais convolucionais alcançaram resultados cada vez melhores, com a construção de variadas arquiteturas de CNNs em cada edição.

Para se construir uma CNN, alguns aspectos importantes precisam ser levados em conta: dados que serão utilizados como entrada, tamanho do *dataset*, quantidade de camadas, tamanho dos filtros das camadas convolucionais, porcentagem de *dropout*, entre outros. Logo, para cada problema, a escolha desses parâmetros deve ser bem definida para que se consiga obter um bom resultado. No contexto das CNNs, existem alguns parâmetros de suma importância, além dos já citados anteriormente, que são definidos antes do treinamento da rede. Tais parâmetros estão estritamente ligados à arquitetura da CNN, e por isso são chamados de hiperparâmetros. Após o grande crescimento das arquiteturas das CNNs em relação ao número de camadas, existe também a preocupação de se escolher os hiperparâmetros de forma eficaz. Isso pode ser feito por algumas técnicas

já conhecidas, como o *Grid Search* e o *Random Search* (BERGSTRA et al., 2011). O *Grid Search* é uma técnica de força-bruta, logo demanda muito tempo para ser executada. O *Random Search* é semelhante ao *Grid Search*, porém executa de forma estocástica e tem como proposta retornar os melhores hiperparâmetros em menos tempo quando comparado ao *Grid Search*.

Estudos de Pinto et al. (2009) e Coates e Ng (2011) demonstram que o desafio de otimizar os hiperparâmetros em modelos profundos têm impedido o progresso científico. Por isso, seria adequado utilizar uma técnica em torno do processo de aprendizagem, de forma a realizar a escolha em um espaço de busca definido, que contenha intervalos específicos para cada hiperparâmetro. Nesse contexto, uma biblioteca chamada de *hyperopt* (BERGSTRA et al., 2013) tem se popularizado. Nela, um espaço de busca é construído de forma estruturada e dois algoritmos de busca são fornecidos: *Random Search* e *Tree-of-Parzen-Estimators (TPE)* (BERGSTRA et al., 2011).

Diante da problemática do reconhecimento automático de expressões faciais, somada à dificuldade de escolha dos hiperparâmetros das Redes Neurais Convolucionais, este trabalho propõe realizar a otimização dos hiperparâmetros para arquiteturas de CNNs, as quais serão utilizadas para o treinamento e classificação de expressões faciais em imagens de rostos. Tanto o algoritmo de otimização quanto as arquiteturas escolhidas são discutidas no Capítulo 2.

1.1 Objetivo

Com base no que foi introduzido, o principal objetivo deste trabalho é estimar os melhores hiperparâmetros para arquiteturas de Redes Neurais Convolucionais aplicadas ao problema de reconhecimento de expressões faciais.

1.1.1 Objetivos Específicos

Especificamente este trabalho tem como objetivos:

- Utilização da biblioteca de otimização *hyperopt* para realizar a busca dos melhores hiperparâmetros para as arquiteturas de CNN escolhidas;
- Treinamento e classificação de emoções faciais por CNNs em sete classes: alegria, tristeza, desgosto, medo, neutro, raiva e surpresa (no caso dos *datasets CK+* e *FERPlus* há uma oitava classe: desprezo);
- Verificação dos hiperparâmetros que mais influenciam no treinamento das CNNs;
- Avaliar o desempenho de diferentes arquiteturas de CNN no problema de reconhecimento de expressões faciais;

- Utilização de três diferentes *datasets* de forma a validar a metodologia;
- Comparar os resultados com os apresentados no estado da arte para trabalhos que utilizem os mesmos *datasets*.

1.2 Contribuição

As principais contribuições deste trabalho são:

- Verificação comparativa do desempenho de diferentes arquiteturas de Redes Neurais Convolucionais e seus hiperparâmetros;
- Método de meta-aprendizado em *Deep Learning* pela seleção automática dos hiperparâmetros das CNNs, para o problema de reconhecimento de expressões faciais.

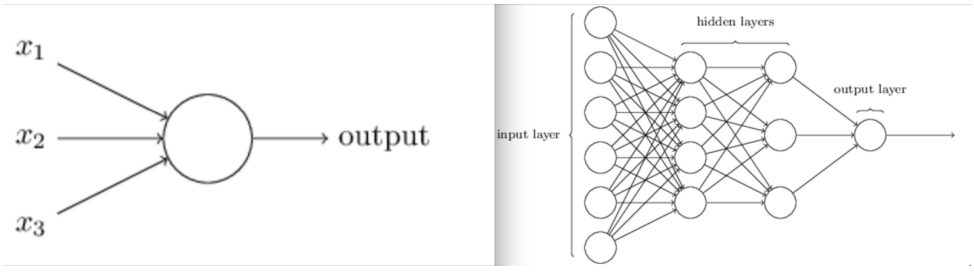
2 Fundamentação Teórica

Neste capítulo são discutidos os principais conceitos utilizados na realização deste trabalho. Todos estão relacionados com *Deep Learning*, como: Redes Neurais Convolucionais, Aprendizado de Máquina e hiperparâmetros que compõem arquiteturas de Redes Neurais. Também será abordado o conceito de Meta-aprendizagem por meio da otimização automática dos hiperparâmetros de CNNs. Dentro do conceito de Meta-aprendizagem, será feita uma explicação da biblioteca *hyperopt* e de seus algoritmos de otimização.

2.1 Deep Learning

Deep Learning ou Aprendizado Profundo é um subcampo de Aprendizado de Máquina que consiste em algoritmos chamados de Redes Neurais Artificiais, pois são inspirados na estrutura e funcionamento do cérebro. O termo *Deep* diz respeito à estrutura da Rede Neural Artificial (RNA), que é construída em camadas. Normalmente essas camadas são separadas entre camada de entrada, camada de saída e camada(s) escondida(s) (que ficam entre a entrada e a saída). Quanto mais camadas uma RNA possuir, mais profunda ela é. Essa estrutura pode ser visualizada na representação da Figura 1. Cada círculo representa um neurônio, e um conjunto de neurônios compõe uma camada.

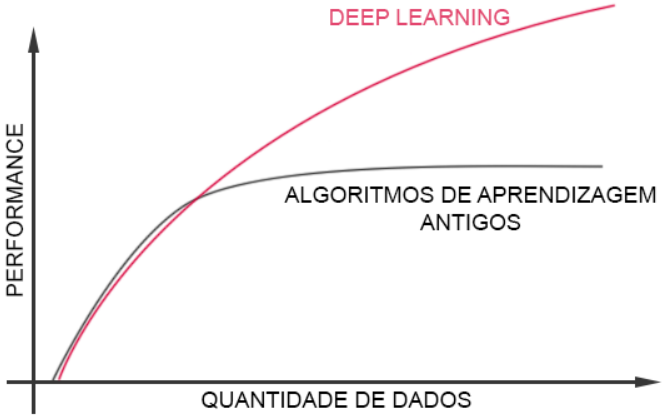
Figura 1 – Representação da estrutura de um Neurônio (esquerda) e uma Rede Neural Artificial (direita).



Fonte: Adaptado de (NIELSEN, 2018)

O uso de RNAs para realizar tarefas de Aprendizado de Máquina se popularizou com o crescimento expressivo de dados disponíveis. Os modelos anteriores às RNAs lidavam com conjuntos pequenos de dados, logo, com o crescimento ocorrido nas últimas décadas, novos modelos eram necessários para processar o grande volume de dados existente. Outro fator que contribuiu para o crescimento de uso das RNAs foi o aumento do poder computacional, principalmente pelo uso de placas gráficas (*GPUs*) para realizar o processamento dos dados de entrada. Essa mudança de paradigma é descrita no gráfico da Figura 2.

Figura 2 – Gráfico comparativo de Deep Learning versus métodos anteriores de aprendizado.



Fonte: Adaptado de (NG, 2015)

Outro grande benefício das RNAs é a sua habilidade de realizar a extração automática de características dos dados (*Feature Learning*) (BENGIO, 2012). Em um outro trabalho de Bengio et al. (2009), a extração de características é explicada: métodos de *Deep Learning* aprendem as características de forma hierárquica; as características são extraídas dos níveis mais altos até os níveis mais baixos. Dessa forma, as RNAs aprendem tais características em diferentes níveis de abstração, permitindo ao sistema aprender funções complexas sem depender de características extraídas de forma manual.

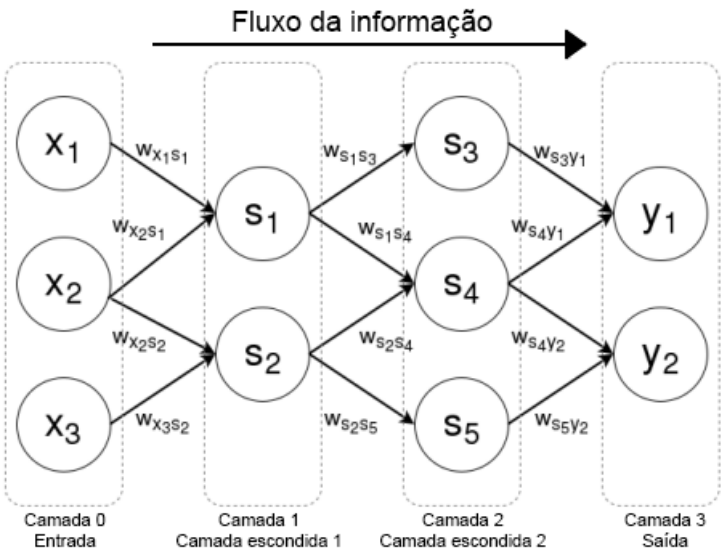
Os bons resultados alcançados pelas RNAs às popularizaram como um dos melhores métodos de Aprendizado de Máquina. Diversos problemas anteriores de Inteligência Artificial que persistiam em ser problemáticos de resolver têm apresentado bons avanços quando aplicados em *Deep Learning*. De forma especial, as RNAs produzem resultados promissores em tarefas de processamento de linguagem natural, e por este motivo, foram escolhidas para resolver o problema de classificação de emoções neste trabalho.

2.1.1 *Feedforward* e *Backpropagation*

Para melhor compreensão de como as Redes Neurais Profundas (RNP) realizam o Aprendizado de Máquina, é necessário primeiramente que dois conceitos sejam explicados: o *Feedforward* e o *Backpropagation*. Alguns autores chamam as RNPs de *Feedforward Neural Networks*, porque os dados seguem na mesma direção por todas as camadas da rede, nas quais diversas computações são feitas sobre esses dados até chegar na última camada retornando uma saída. A Figura 3 exemplifica o fluxo de dados em uma RNP.

Cada ligação entre neurônios possui um peso associado. Cada neurônio recebe a informação dos neurônios que possua ligação na camada anterior, e processa esses dados

Figura 3 – Dados fluem em uma RNP da entrada [x1, x2, x3] para a saída [y1, y2], passando pelos neurônios [s1, s2, s3, s4, s5] das camadas centrais.



Fonte: Adaptado de (BRILLIANT.ORG, 2018)

junto ao peso da ligação. O resultado do cálculo obtido por esse neurônio é redirecionado aos neurônios que possui ligação na camada seguinte. Esse procedimento é feito por cada neurônio a partir da primeira camada escondida até a camada de saída, na qual o resultado é comparado à saída esperada por meio do cálculo do erro. O objetivo de uma RNP é diminuir esse erro, de forma a obter na saída o resultado mais próximo da classificação desejada. Esse procedimento é chamado de *Feedforward*.

Um erro associado é calculado para cada iteração de uma RNP. Esse erro é transferido de volta ao início da rede, para que os pesos de cada neurônio sejam corrigidos de acordo com a taxa de erro do neurônio relacionado, sendo a taxa calculada a partir da saída esperada. Esse processo de retorno do erro para correção dos pesos é chamado de *Backpropagation*. O *Feedforward* é o processo de transferência dos dados à frente, enquanto o *Backpropagation* faz o caminho inverso, realizando a correção dos pesos.

Os dados de entrada são processados pela RNP por um número finito de vezes (quantidade de épocas), até se alcançar uma acurácia desejada ou a taxa de erro ser igual ou menor a uma taxa mínima definida.

2.1.2 CNN

Redes Neurais Convolucionais (CNN do inglês *Convolutional Neural Networks*) (LECUN et al., 1989) são um tipo especializado de Rede Neural projetadas para processar dados que estão na forma de múltiplos *arrays*. Alguns exemplos desses dados são: sinais e sequências (uma dimensão), imagem ou áudio (duas dimensões), vídeos ou imagens

volumétricas (três dimensões). A imagem é o tipo de dado mais comum utilizado como entrada em uma CNN.

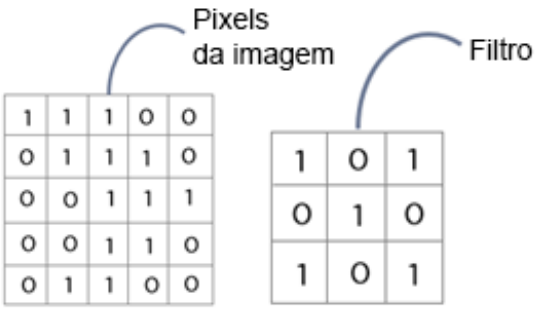
O termo "Rede Neural Convolucional" indica que a rede faz uso de uma operação matemática chamada convolução. A convolução é um tipo específico de operação linear, utilizada no lugar das multiplicações de matrizes que são realizadas em uma Rede Neural comum.

A arquitetura de uma CNN é estruturada como uma série de estágios, assim como a distribuição em camadas de uma RNP. Os primeiros estágios geralmente são compostos de dois tipos de camadas: camadas convolucionais e camadas *pooling*. O objetivo da camada convolucional é detectar conjuntos de características dos dados recebidos da camada imediatamente anterior a ela, enquanto que a camada *pooling* une as características detectadas que são semanticamente similares, realizando uma sub-amostragem dessas características. A seguir, veremos como as operações de convolução e *pooling* são feitas em imagens.

2.1.2.1 Convolução

Como explicado na Subseção 2.1.1, uma Rede Neural aprende características de cada classe a partir da correção dos pesos associados às ligações entre neurônios. No caso das CNNs, os pesos da rede estão associados nas camadas convolucionais por meio de conjuntos de filtros. Cada filtro é passado sobre a imagem recebida da camada anterior, resultando em um mapa de ativação de características daquele filtro. Intuitivamente, a rede irá aprender filtros que são ativados quando percebem algum tipo de característica visual na imagem como bordas, cores e padrões. Na Figura 4 é possível visualizar uma imagem de entrada e um filtro de ativação em forma de "X" da camada convolucional.

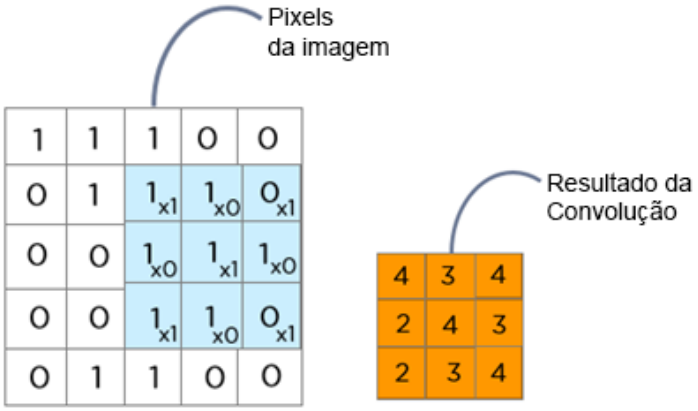
Figura 4 – Esquerda: imagem de entrada em uma CNN. Direita: exemplo de filtro de ativação.



Fonte: Adaptado de (SHARMA, 2018)

Como notado na Figura 5, a convolução é o resultado de uma soma ponderada local entre o filtro e a imagem, após esta ter sido percorrida totalmente pelo filtro. Definindo o

Figura 5 – Processo de convolução realizado pelo filtro (azul) sobre a imagem, resultando no mapa de características (laranja).

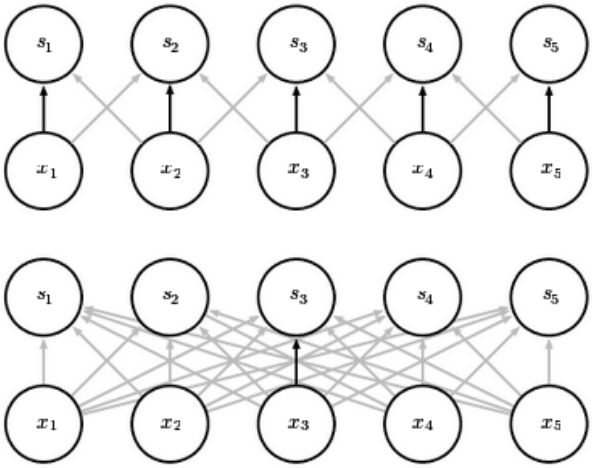


Fonte: Adaptado de (SHARMA, 2018)

tamanho do filtro (*kernel*) menor que o tamanho da imagem permite que haja economia computacional, dado que apenas computações locais limitadas à área do filtro são feitas.

Outra propriedade que garante a viabilidade do processamento de imagens nas CNNs é o compartilhamento de parâmetros. Em uma Rede Neural tradicional, cada elemento da matriz de pesos é usado exatamente uma vez quando se computa a saída de uma camada. No caso das CNNs os pesos são considerados interligados, pois o valor de um peso aplicado a uma entrada é ligado ao valor de outro peso aplicado em outro local da rede.

Figura 6 – Visualização do compartilhamento de parâmetros de uma CNN.



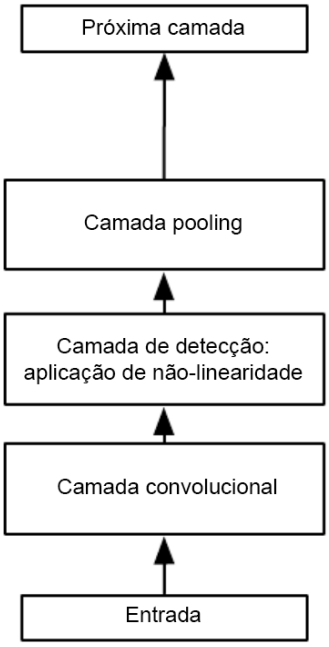
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

Vê-se na Figura 6 uma descrição visual do funcionamento do compartilhamento de parâmetros. As setas pretas em destaque indicam conexões que utilizam um parâmetro em particular em dois diferentes modelos. Na imagem do topo, as setas pretas indicam o uso do elemento central de um *kernel* de três elementos em um modelo convolucional. Observa-se que ele é utilizado repetidas vezes para diferentes entradas. Na imagem inferior, a seta preta indica o uso do elemento central de uma matriz de pesos em um modelo totalmente conectado. Esse último modelo não possui compartilhamento de parâmetros, por isso, o parâmetro é utilizado apenas uma única vez.

2.1.2.2 *Pooling*

Uma CNN é tipicamente composta de três estágios (Figura 7). No primeiro estágio diversas convoluções são realizadas para produzir um conjunto de ativações lineares. Em seguida, cada ativação linear é passada por uma função de ativação não-linear, como por exemplo a ativação *Rectifed Linear Unit (ReLU)*. Por último, a rede faz uso de uma função *pooling* para modificar a saída da camada.

Figura 7 – Estágios de uma CNN típica.



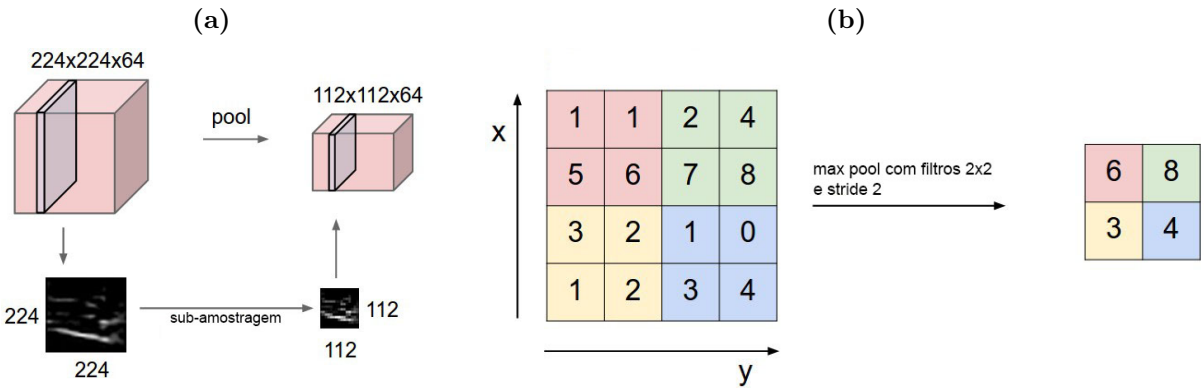
Fonte: Adaptado de (GOODFELLOW; BENGIO; COURVILLE, 2016)

Uma função *pooling* substitui a saída da rede em certo local por uma representação estatística. Por exemplo, a operação *max pooling* (ZHOU et al., 1988) retorna uma saída máxima em uma vizinhança retangular. Outra operação *pooling* conhecida é *average pooling* que retorna a média de uma vizinhança retangular.

A operação *max pooling* é a mais comum de ser utilizada em arquiteturas de CNN, e pode ser vista na Figura 8. É possível notar em ambas as Figuras 8a e 8b que ao

final, uma sub-amostragem da imagem dada como entrada é retornada pela operação *pooling*. É importante mencionar que a quantidade de dados devolvida após a operação é significantemente reduzida sem grandes perdas, tornando viável a utilização dessa operação para realização de sub-amostragem dos mapas de ativação.

Figura 8 – Demonstração da operação *max pooling*.



Fonte: Adaptado de (VISION; LAB, 2018)

2.1.3 Hiperparâmetros

Modelos de Redes Neurais, tanto comuns quanto convolucionais, são parametrizados por um conjunto de hiperparâmetros que precisam ser escolhidos apropriadamente de forma a maximizar o processo de aprendizagem. Os hiperparâmetros são utilizados para configurar diversos aspectos do algoritmo de aprendizagem e podem ter efeitos variados no modelo resultante e na sua performance (CLAESEN; MOOR, 2015).

Para que fique mais claro, é válido primeiramente diferenciar o que são parâmetros e hiperparâmetros de um modelo. Parâmetros são valores que precisam ser estimados dos dados utilizados como entrada. Tais valores não costumam ser definidos manualmente e podem ser salvos ao final do processo de aprendizagem. Os pesos aprendidos por uma rede neural ao final de um treinamento, por exemplo, são definidos como parâmetros de um modelo.

Já os hiperparâmetros são valores externos ao modelo que não podem ser estimados a partir dos dados de entrada. Eles influenciam diretamente nos processos que realizam a estimativa dos parâmetros de um modelo, e são definidos manualmente por quem realiza a modelagem, a partir de conhecimento empírico ou pela realização de testes em conjuntos de hiperparâmetros.

Alguns exemplos de hiperparâmetros de redes neurais já foram mencionados na Seção 1, e para a modelagem de arquiteturas vários outros precisam ser considerados. Os hiperparâmetros mais comuns para Redes Neurais Convolucionais (objeto desse trabalho) são:

- Quantidade de filtros convolucionais
- Tamanho dos filtros convolucionais
- Tipo de ativação (*relu*, *elu*, *tanh*, *sigmoid*)
- Tamanho do *pooling*
- Salto do *Stride* (passo)
- Porcentagem de *dropout*
- Uso de *Batch Normalization*
- Tamanho do *batch* de treinamento

2.2 Meta-aprendizagem

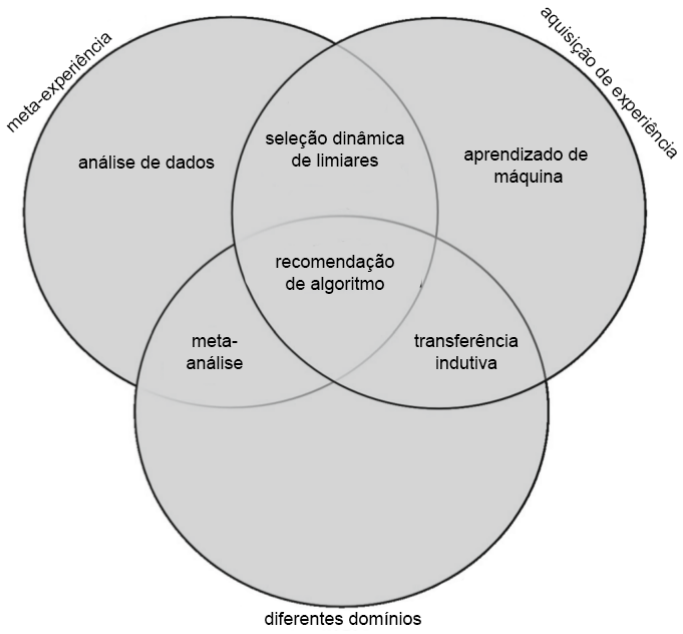
O paradigma tradicional em aprendizado de máquina é adquirir um grande *dataset* e treinar um novo modelo utilizando esse *dataset*. A decisão de escolha do *dataset*, modelo a ser utilizado e seus hiperparâmetros está totalmente relacionada a quem realiza a modelagem, a partir de suas experiências e aprendizado anteriores. Meta-aprendizado (do inglês *meta-learning*) é um tópico recente de pesquisa direcionado ao problema de "aprender a aprender". Seu objetivo é aperfeiçoar a performance de algoritmos de aprendizagem existentes por meio da auto-decisão de variáveis que compõem o problema.

O conceito chave de sistemas de aprendizagem é o aperfeiçoamento por experiência. Como explicado por Vanschoren (2010), o "meta-aprendizado monitora o processo de aprendizagem automático, no contexto dos problemas de aprendizado que ele encontra, e tenta adaptar seu comportamento para se aperfeiçoar." O trabalho de Lemke, Budka e Gabrys (2015) enumera dois requisitos que definem um sistema de meta-aprendizagem:

- Um sistema de meta-aprendizagem precisa incluir um subsistema de aprendizagem.
- A experiência é obtida pela exploração da metainformação extraída:
  - em um episódio anterior de aprendizagem em um único dataset e/ou
  - de diferentes domínios ou problemas.

A meta-aprendizagem pode ser empregada em uma variedade de configurações, com certa discordância na literatura sobre quais configurações constituem ou não problemas de meta-aprendizado. Este trabalho, no entanto, seguirá os requisitos citados anteriormente, e que estão agrupados no diagrama da Figura 9. Cada um dos três círculos relaciona-se aos

Figura 9 – Componentes de um sistema de meta-aprendizado.



Fonte: Adaptado de (LEMKE; BUDKA; GABRYS, 2015)

pontos dos requisitos (obtenção de experiência, meta-aprendizado com um único *dataset*, meta-aprendizado em diferentes domínios).

Para este trabalho, o sistema de meta-aprendizagem se dará pela estimação automática de valores para hiperparâmetros de Redes Neurais Convolucionais, sendo as CNNs o subsistema de aprendizagem a ser utilizado. A experiência será obtida pela realização de diversos treinamentos de CNN, onde variados valores de hiperparâmetros serão testados buscando o melhor resultado (menor perda e maior acurácia) para os datasets escolhidos. Esse sistema fará uso da biblioteca de otimização automática *Hyperopt*, objeto da subseção a seguir.

2.2.1 *Hyperopt*

Avanços recentes no estado da arte de resultados em classificação de imagens têm sido obtidos pelo aperfeiçoamento de configurações de técnicas existentes, no lugar da criação de novas abordagens que realizem o aprendizado de características (BERGSTRA et al., 2011). Tradicionalmente, a busca dos hiperparâmetros que compõem um modelo é feita manualmente, por meio de visualizações anteriores (HINTON, 2012), (HSU; CHANG; LIN, 2003) ou pelo teste de conjuntos de hiperparâmetros em uma grade de opções predefinidas (PEDREGOSA et al., 2011).

Tais abordagens, no entanto, deixam a desejar em termos de reprodução e são impraticáveis quando o número de hiperparâmetros é extenso (CLAESEN; MOOR, 2015).

Devido tais falhas, a ideia de automatizar a busca de hiperparâmetros tem recebido grande atenção em aprendizado de máquina recentemente. Processos automatizados de busca já têm demonstrado melhor capacidade quando comparados à busca manual em diversos problemas (BERGSTRA et al., 2011), (BERGSTRA; BENGIO, 2012).

No contexto desse problema surge uma biblioteca chamada *Hyperopt* (BERGSTRA et al., 2013) com o objetivo de realizar a busca otimizada de hiperparâmetros sobre espaços de busca predefinidos, que podem incluir valores de dimensão real, discreta ou condicional. A biblioteca utiliza a otimização *Sequential model-based (SMBO)*, também conhecida como otimização Bayesiana (PELIKAN; GOLDBERG; CANTÚ-PAZ, 1999), que é, segundo os criadores do *Hyperopt*, um dos métodos mais eficientes de otimização existente.

O ponto chave que diferencia a busca realizada pelo *SMBO* do *Grid Search* e do *Random Search* é que estes últimos desconsideram qualquer busca feita anteriormente pelo fato de não serem otimizados. Métodos *SMBO* funcionam a partir da busca do próximo conjunto de hiperparâmetros a ser avaliado na função objetivo pela seleção dos hiperparâmetros que sobressaíram em uma função probabilística substituta (chamada de função *surrogate*), que é menos custosa de ser avaliada. Caso os hiperparâmetros avaliados apresentem bons resultados também na função objetivo, eles são incorporados ao conjunto de melhores hiperparâmetros.

Existem cinco aspectos que fazem parte de um método de otimização *SMBO* (alguns já citados no parágrafo anterior):

- Um domínio de hiperparâmetros no qual a busca é realizada
- Uma função objetivo que utiliza os valores dos hiperparâmetros e retorna uma saída que pode ser maximizada ou minimizada
- Uma função *surrogate* (substituta) da função objetivo
- Um critério de avaliação de quais hiperparâmetros devem ser escolhidos em seguida a partir da função *surrogate*
- Um histórico que consiste em um par {score, hiperparâmetros} utilizado pelo algoritmo para atualizar a função *surrogate*

Em seu artigo, Bergstra et al. (2013) explicam que métodos *SMBO* são aplicados em cenários nos quais o usuário deseja minimizar o valor de uma função que possui avaliação custosa, seja em termos de tempo ou custo. Algumas vantagens do *SMBO* são enumeradas:

- Lida com variáveis reais, discretas ou condicionais
- Capaz de realizar avaliações paralelas de uma função

- Lida com centenas de variáveis, mesmo que precise realizar também centenas de avaliações de funções

A biblioteca *Hyperopt* fornece dois algoritmos de busca de hiperparâmetros: *Random Search* (BERGSTRA; BENGIO, 2012) e *Tree-structured Parzen Estimator (TPE)* (BERGSTRA et al., 2011), que será explicado na Subseção 2.2.1.1 pois foi o algoritmo escolhido para realizar as buscas dos melhores espaços de hiperparâmetros neste trabalho.

A biblioteca é, em geral, indicada para resolver qualquer problema *SMBO*, no entanto, seus autores afirmam que a desenvolveram com o intuito de otimizar os hiperparâmetros de Redes Neurais Profundas e Redes Neurais Convolucionais. A sua utilização se dá por: a) definição de uma função objetivo a ser minimizada, b) definição de um espaço de busca de hiperparâmetros, c) decisão do algoritmo de busca (*random* ou *tpe*). Com esses três pontos definidos, basta realizar a chamada do método próprio de minimização para iniciar a busca. A biblioteca ainda é capaz de executar paralelamente em *cluster* para realizar buscas mais rápidas, por meio de um banco de dados *MongoDB*.

Além do *Hyperopt*, existem outras bibliotecas que realizam tarefas de meta-aprendizado em Redes Neurais como é o caso da ferramenta *Neural Network Intelligence* (NNI, 2018), que além de realizar buscas otimizadas em espaços de hiperparâmetros, também faz a busca de melhores arquiteturas de Redes Neurais para determinados problemas. Tarefa semelhante é realizada por Liu, Simonyan e Yang (2018), que trabalha na modelagem de arquiteturas especificamente para CNNs.

2.2.1.1 *Tree-structured Parzen Estimator*

Tree-structured Parzen Estimator (*TPE*) (BERGSTRA et al., 2011) é um método *SMBO* que em suas iterações realiza a coleta de novas observações, e ao fim da iteração decide de forma otimizada qual conjunto de parâmetros ele deve testar na sequência. Sendo um método *SMBO*, ele segue os cinco aspectos citados na Subseção 2.2.1, empregando uma função *surrogate* própria e tendo como critério de avaliação o Aperfeiçoamento Esperado (do inglês *Expected Improvement - EI*).

Para iniciá-lo, é necessário primeiramente definir uma distribuição inicial de hiperparâmetros. Esse conjunto normalmente é uma distribuição uniforme, porém também é possível associar qualquer hiperparâmetro com uma distribuição estocástica. Em suas primeiras iterações, como não há nenhuma informação conhecida do espaço de busca, é realizada uma busca aleatória usando intervalos de hiperparâmetros definidos, para assim obter resultados que podem ser avaliados.

Após obter alguns resultados com a busca aleatória, o *TPE* é então iniciado. As observações coletadas até então são divididas em dois grupos: o primeiro grupo contém as observações com os melhores *scores* enquanto que o segundo grupo contém o restante das

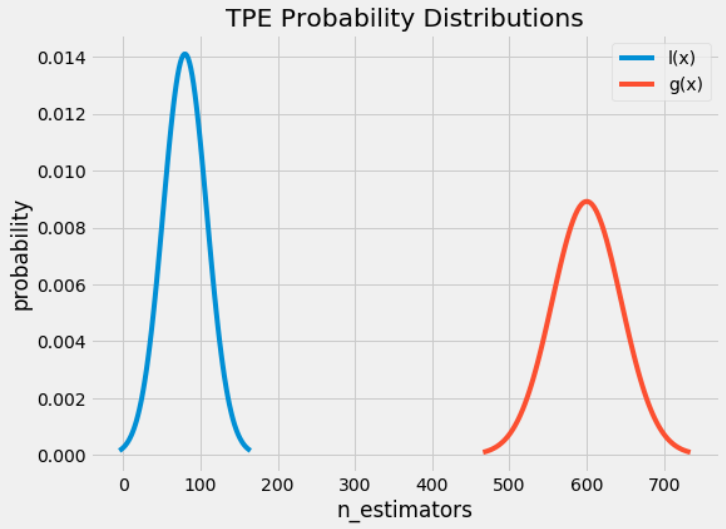
observações. O objetivo é encontrar um conjunto de hiperparâmetros que é mais provável de estar no primeiro grupo e menos provável de estar no segundo grupo.

O próximo passo é calcular a probabilidade de um candidato estar em cada um dos dois grupos. Dos candidatos já obtidos, o algoritmo tenta encontrar um candidato que é mais provável de ser encontrado no primeiro grupo e menos provável de estar no segundo grupo. A equação a seguir define o *Expected Improvement* para cada um dos candidatos:

$$EI(x) = \frac{l(x)}{g(x)}$$

Onde $l(x)$ é a probabilidade de estar no primeiro grupo e $g(x)$ a probabilidade de estar no segundo grupo. A cada iteração do algoritmo, as distribuições tornam-se mais esparsas, como pode ser visto na Figura 10. A seleção dos melhores hiperparâmetros formam um espaço em árvore (o que dá o nome ao algoritmo), na qual a descida dos caminhos no grafo formam o caminho de decisão percorrido pelo algoritmo.

Figura 10 – Distribuição de probabilidades do algoritmo *TPE*.

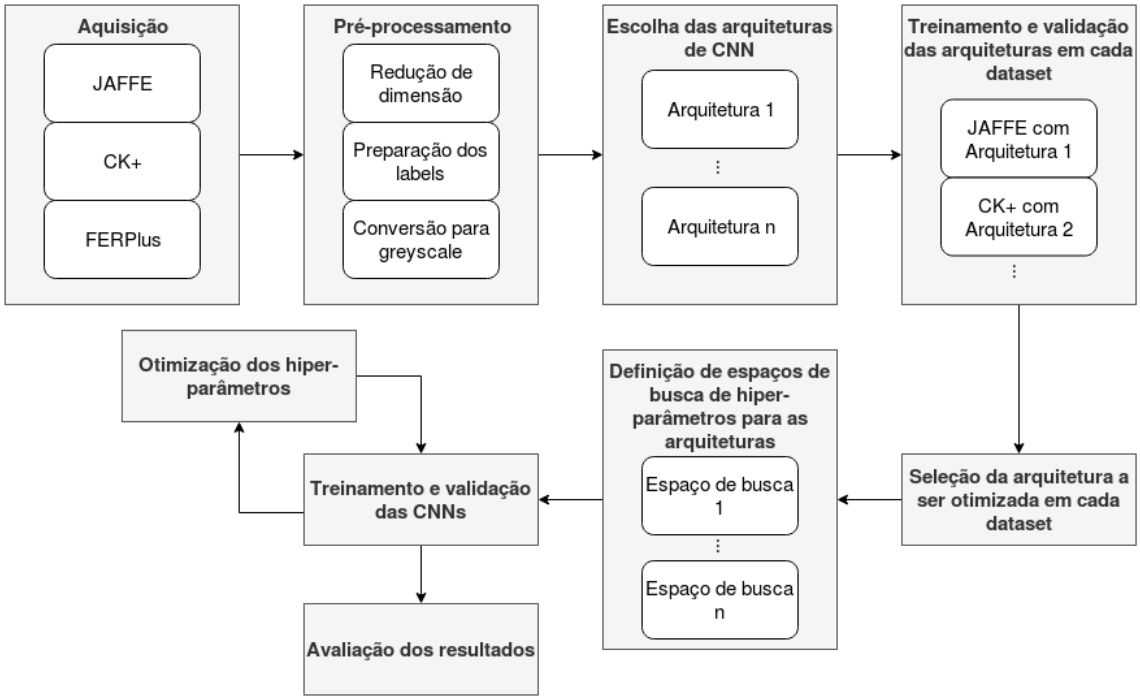


Fonte: (KOEHRSEN, 2018)

3 Metodologia

Neste capítulo serão apresentadas as técnicas e métodos que serão utilizados para a realização deste trabalho. A metodologia consistirá em etapas bem definidas, que são: (1) Aquisição de *datasets*; (2) Pré-processamento dos *datasets* adquiridos; (3) Escolha das arquiteturas de CNN para realização dos treinamentos; (4) Treinamento e validação de cada arquitetura escolhida em cada *dataset*; (5) Seleção de arquitetura a realizar otimização de hiperparâmetros a partir dos resultados da etapa anterior; (6) Definição de espaço de busca de hiperparâmetros para cada arquitetura selecionada; (7) Treinamento, validação e otimização (execução) das CNNs. A Figura 11 apresenta as etapas citadas e descritas nas seções a seguir.

Figura 11 – Metodologia proposta.



Fonte: Elaborado pelo autor

Todos os experimentos foram realizados em uma máquina configurada com duas placas gráficas NVIDIA GeForce GTX 1080 Ti. Não foi necessário utilizar ambas as placas paralelamente, pois apenas uma era capaz de processar os dados recebidos pelas CNNs em tempo hábil.

3.1 Aquisição de datasets

A primeira etapa desta metodologia foi dedicada à aquisição dos três *datasets* públicos que serão utilizados nesse trabalho: JAFFE (LYONS et al., 1998), Extended Cohn-Kanade (CK+) (LUCEY et al., 2010) e FERPlus (BARSOUM et al., 2016).

A escolha destes *datasets* está relacionada com o vasto uso dos mesmos em trabalhos de reconhecimento de expressões faciais, e dessa forma é possível comparar seus resultados com os obtidos neste trabalho. Os *datasets* são descritos nas subseções a seguir.

3.1.1 JAFFE: *The Japanese Female Facial Expression*

Este *dataset* é um dos mais utilizados em trabalhos de reconhecimento de expressões faciais (SHAN; GONG; MCOWAN, 2009; SARODE; BHATIA, 2010; LI; DENG, 2018). Ele contém 213 imagens de 10 pessoas de etnia japonesa do sexo feminino, e foi gerado em laboratório para a validação da metodologia do trabalho de Lyons et al. (1998).

Cada imagem possui apenas um canal de cor e tem dimensões 256x256 pixels. As sete emoções presentes nesse *dataset* são: neutro, alegria, tristeza, surpresa, raiva, desgosto e medo. Na Figura 12 podem ser vistos exemplos de imagens de expressões faciais do *dataset* de um mesmo indivíduo.

Figura 12 – Exemplos de imagens do *dataset* JAFFE.

Fonte: Adaptado de (LYONS et al., 1998)

3.1.2 CK+: *The Extended Cohn-Kanade Dataset*

Esta é a segunda versão do *dataset* Cohn-Kanade, publicado em 2000 (KANADE; TIAN; COHN, 2000), e que foi liberado para pesquisadores com o intuito de promover pesquisas de detecção automática de expressões faciais. Este primeiro *dataset* tornou-se um dos utilizados neste campo de pesquisa por conta das limitações apresentadas pelos *datasets* existentes na época da publicação. Ele contém 2105 sequências de imagens de 182 indivíduos adultos de diferentes etnias.

A popularidade deste primeiro *dataset* estimulou seus autores a construírem uma extensão dele: o Extended Cohn-Kanade (CK+) (LUCEY et al., 2010). Esta segunda versão apresenta um aumento de 22% no número de sequências de imagens e 27% no

número de indivíduos. As oito emoções presentes nesse *dataset* são: neutro, raiva, desprezo, desgosto, medo, alegria, tristeza e surpresa.

Diferentemente do JAFFE, o CK+ não apresenta imagens individuais de cada emoção, e sim sequências de imagens que iniciam da emoção neutra até o pico de expressão apresentado na última imagem, como pode ser visto na Figura 13. A quantidade de imagens (*frames*) que cada sequência possui é variável. A forma como cada sequência de *frames* foi processada será discutida na Seção 3.2. As sequências diferem na dimensão dos *frames*, sendo algumas de dimensão 640x480 pixels e outras 640x490 pixels. Também há diferença quanto à quantidade de canais, pois a maioria apresenta apenas um canal, enquanto que outras possuem três canais.

Figura 13 – Exemplo de sequência de *frames* do *dataset* CK+.

Fonte: (JIA et al., 2013)

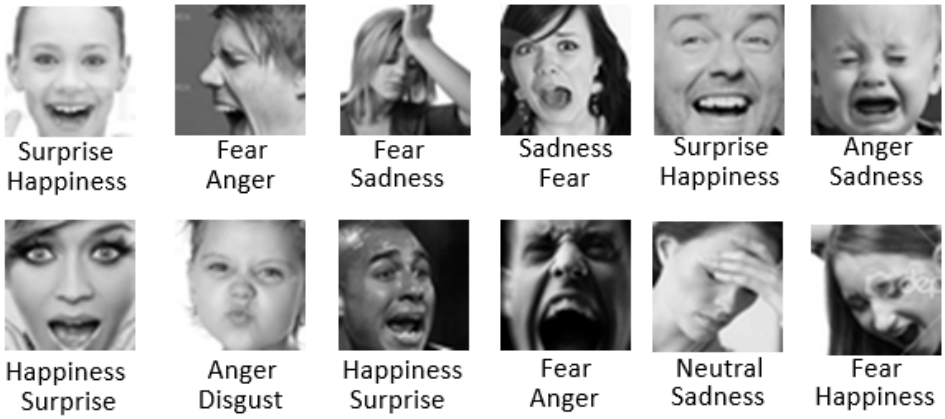
3.1.3 FERPlus

O *dataset* FER2013 (GOODFELLOW et al., 2013) foi disponibilizado a partir de um *challenge* da plataforma de desafios Kaggle, no ano de 2013 (KAGGLE, 2018). Ele possui mais de 30 mil imagens de rostos *in-the-wild* divididas em conjuntos de treino, teste e validação, e já foi utilizado em diversos trabalhos relacionados com o reconhecimento de expressões faciais (PRAMERDORFER; KAMPEL, 2016b; LI; DENG, 2018). Atualmente é um dos maiores *datasets* anotados disponíveis publicamente.

Uma versão aprimorada do FER2013 foi publicada pela Microsoft no ano de 2016, chamado de FERPlus (BARSOUM et al., 2016). A proposta dos autores foi aperfeiçoar as anotações do FER2013, fornecendo uma distribuição de probabilidade de emoções para cada imagem. A forma como essa distribuição foi trabalhada para a coleta das anotações será discutida na Seção 3.2. Esse *dataset* contém oito emoções: neutro, alegria, surpresa, tristeza, raiva, desgosto, medo e desprezo.

Os autores do FERPlus afirmam que o *dataset* FER2013 possui diversos erros de anotação, indicando de forma errada a emoção presente em determinadas imagens. Na Figura 14 é possível visualizar as novas anotações que o FERPlus apresenta para diferentes emoções. Todas as imagens deste *dataset* possuem um canal e dimensão 48x48 pixels.

Figura 14 – Comparação entre anotações dos *datasets* FER2013 (acima) e FERPlus (abaixo).



Fonte: (BARSOUM et al., 2016)

3.2 Pré-processamento

Nesta seção serão descritos os processos realizados em cada *dataset* de forma a prepará-los para os treinamentos das CNNs e otimização dos seus hiperparâmetros. Cada processo será explicado separadamente para cada um dos *datasets* selecionados para este trabalho.

3.2.1 JAFFE

O *dataset* JAFFE não fornece um arquivo de anotações específico para a organização das anotações. A emoção de cada imagem é dada pelos nomes dos arquivos presentes no *dataset*, como pode ser visto na Figura 15. Logo, o primeiro passo foi gerar as anotações para cada uma das 213 imagens.

O nome do arquivo é separado por três pontos, e a sigla lida após o primeiro ponto indica qual a emoção de determinada imagem. Logo, na primeira fileira da Figura 15 as anotações das imagens são: medo (FE: *Fear*), alegria (HA: *Happy*) e alegria novamente. Um simples *script* de leitura de arquivos facilitou a extração das 213 anotações do *dataset*.

A etapa seguinte do pré-processamento foi realizar um corte na imagem, de forma a extrair apenas a região do rosto dos indivíduos, excluindo o fundo e regiões do cabelo. Esse processo foi feito por meio de um algoritmo de detecção de faces, sendo utilizada neste trabalho a biblioteca MTCNN (*Multi-task Cascaded Convolutional Networks*) Zhang et al. (2016), que utiliza CNNs para a detectar uma região retangular que compreende as faces encontradas na imagem.

Por fim, com todas as imagens devidamente anotadas e recortadas na região do rosto, foi feito um redimensionamento das mesmas para o tamanho 150x112 pixels

Figura 15 – Visualização dos nomes das imagens presentes no dataset JAFFE.

Fonte: Elaborado pelo autor

(originalmente eram 256x256 pixels). Na Figura 16 é possível visualizar três imagens após o pré-processamento. Ao fim, o *dataset* foi dividido entre conjunto de treino (80%) e conjunto de teste (20%).

Figura 16 – Exemplos de imagens do JAFFE após o pré-processamento.

Fonte: Elaborado pelo autor

3.2.2 CK+

O *dataset* CK+ fornece as anotações para cada sequência de imagens em arquivos de texto. Será feita a leitura desses arquivos para extrair a emoção correspondente à sua respectiva sequência. De acordo com a documentação do *dataset*, algumas sequências não possuem uma anotação específica (ou seja, não apresentam um arquivo de anotação correspondente), e estas serão descartadas nessa etapa.

As demais sequências que possuem uma anotação correspondente serão tratadas da seguinte forma:

- O primeiro *frame* da sequência será armazenado como emoção neutra.

- Pelo fato das sequências possuírem quantidades variadas de *frames*, a partir do *frame* (tamanho da sequência / 2) + 2 em diante serão armazenados de acordo com a emoção correspondente.
- Entre o primeiro *frame* e o *frame* (tamanho da sequência / 2) + 2 sobrarão algumas imagens de transição. Estes serão descartados.

Por exemplo: uma sequência possui 11 *frames*. O *frame* 1 será emoção neutra, enquanto que a partir do *frame* 7 até o último *frame* (pico de expressão), serão armazenados com a emoção correspondente no seu arquivo de anotação. A Figura 17 abaixo demonstra a divisão de uma sequência de *frames* específica.

Figura 17 – Seleção da primeira imagem (acima, emoção neutra) e demais imagens selecionadas de uma sequência (abaixo, emoção correspondente).

Fonte: Elaborado pelo autor

O algoritmo MTCNN também é utilizado neste *dataset* para realizar o corte da região do rosto de cada imagem, realizando em seguida o redimensionamento para 150x112 pixels e conversão para apenas um canal (*grayscale*), nas imagens que apresentarem três canais. O CK+, assim como o JAFFE, foi dividido entre conjunto de treino e teste na mesma proporção: 80% e 20% respectivamente.

3.2.3 FERPlus

O FERPlus já é disponibilizado pelos seus autores dividido entre conjunto de treino e conjunto de teste. O conjunto de treino possui 28.709 imagens, enquanto que o conjunto de teste possui 3.589 imagens.

Como já mencionado Subseção 3.1.3, as anotações são disponibilizadas na forma de uma distribuição de emoção apresentada em cada imagem. O arquivo de anotação é no formato .csv, e cada linha é relacionada a uma imagem do *dataset*. Uma linha deste arquivo de anotação indica em suas colunas, respectivamente: (1) o nome da imagem correspondente; (2) a dimensão da imagem (todas são 48x48 pixels); (3) das colunas 3 a 10, a pontuação que indica o nível de emoção sentida na imagem, seguindo a ordem

neutro, alegria, surpresa, tristeza, raiva, novo, medo e desprezo; (4) a coluna 11 indica se a imagem não possui uma face ou o rosto não está bem enquadrado. Na Figura 18 demonstra-se o conteúdo do arquivo de anotações para o conjunto de teste.

Figura 18 – Visualização do arquivo .csv que contém a distribuição de emoções para o FERPlus.

	A	B	C	D	E	F	G	H	I	J	K
1	fer0032220.png	(0, 0, 48, 48)	0	0	0	0	2	1	0	7	0
2	fer0032222.png	(0, 0, 48, 48)	3	0	0	5	0	0	0	0	0
3	fer0032223.png	(0, 0, 48, 48)	6	0	1	2	0	0	0	0	0
4	fer0032224.png	(0, 0, 48, 48)	0	0	4	0	1	2	3	0	0
5	fer0032225.png	(0, 0, 48, 48)	0	0	0	0	9	0	1	0	0
6	fer0032226.png	(0, 0, 48, 48)	6	0	0	4	0	0	0	0	0
7	fer0032227.png	(0, 0, 48, 48)	0	10	0	0	0	0	0	0	0
8	fer0032228.png	(0, 0, 48, 48)	0	0	0	0	8	0	1	0	0
9	fer0032229.png	(0, 0, 48, 48)	0	9	0	0	0	1	0	0	0
10	fer0032230.png	(0, 0, 48, 48)	0	10	0	0	0	0	0	0	0
11	fer0032231.png	(0, 0, 48, 48)	0	10	0	0	0	0	0	0	0

Fonte: Elaborado pelo autor

A emoção anotada para cada imagem é a que apresentar maior pontuação, fazendo a leitura das colunas C a J do arquivo de anotações. Caso a maior pontuação se repita em alguma outra coluna (por exemplo, 5 para surpresa, 5 para medo), a imagem será descartada. Também será descartada caso seja indicado na coluna K que a imagem não possui rosto bem definido.

Esse *dataset*, assim como os anteriores, também é submetido ao procedimento de detecção de faces via MTCNN, para remoção de imagens que não forem detectadas faces bem enquadradas, além de realizar um corte na região do rosto das imagens em que a face for encontrada. Por fim, será realizado o redimensionamento das imagens para 44x34 pixels. Após todo o pré-processamento descrito acima, o conjunto de treino foi armazenado com 22.125 imagens e o conjunto de teste com 2.752 imagens.

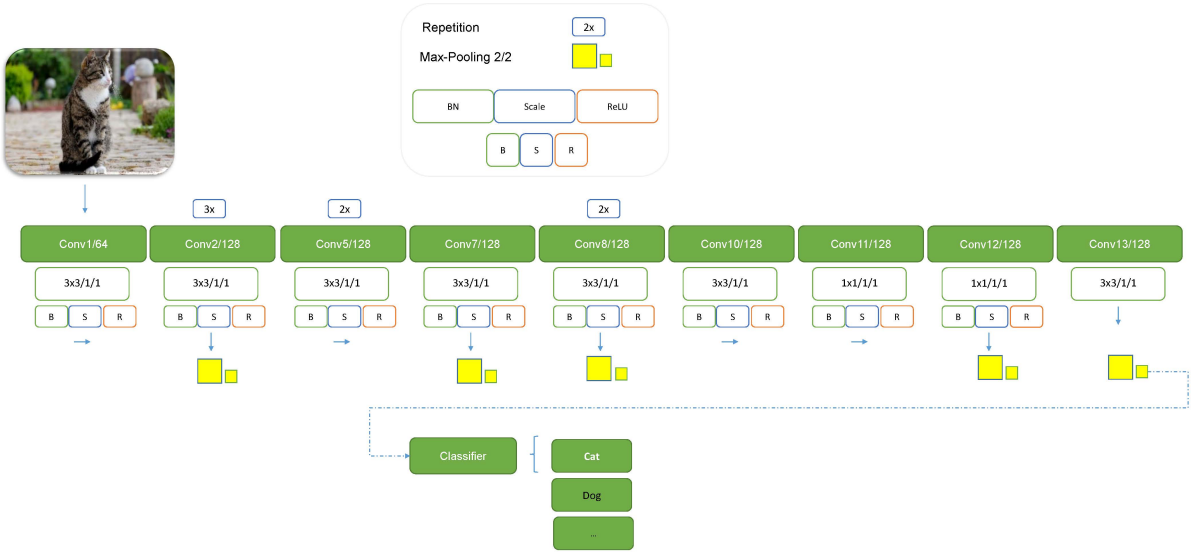
Por conta da sua extensão e prevendo o longo tempo de espera para realizar treinamentos neste *dataset*, foi optado neste trabalho a criação de versões menores do mesmo. Foram gerados dois novos *datasets*: um de tamanho médio, com 11.062 imagens de treino e 1.376 imagens de teste e um de tamanho pequeno, com 5.531 imagens de treino e 688 imagens de teste. Estes dois *datasets* serão úteis quando for necessário realizar treinamentos mais rápidos.

3.3 Escolha das arquiteturas de CNN

O processo de definição das arquiteturas foi baseado em conhecimento empírico. Três arquiteturas de CNN foram selecionadas para este trabalho: SimpleNet (HASANPOUR et al., 2016), VGG (SIMONYAN; ZISSERMAN, 2014) e ResNet (HE et al., 2015). A escolha foi baseada em uma ordem de grandeza dessas arquiteturas, objetivando utilizar as com menos camadas para os menores *datasets* e as com mais camadas para os maiores *datasets*.

A rede neural convolucional SimpleNet é a mais recente entre as escolhidas, e seus autores propuseram uma arquitetura simples, como o próprio nome da CNN indica, e com menos camadas, que pudesse ter resultados melhores ou equiparáveis às arquiteturas populares como VGG, ResNet e GoogleNet (SZEGEDY et al., 2014). É possível visualizar a estrutura da SimpleNet na Figura 19. Como resultado, obtém-se menor número de parâmetros e operações. Em seu artigo, HasanPour et al. (2016) realizaram testes nos *datasets* CIFAR10, CIFAR100 e MNIST, obtendo resultados melhores que os obtidos com arquiteturas consideradas mais robustas.

Figura 19 – Visualização das camadas da arquitetura da SimpleNet.



Fonte: (HASANPOUR et al., 2016)

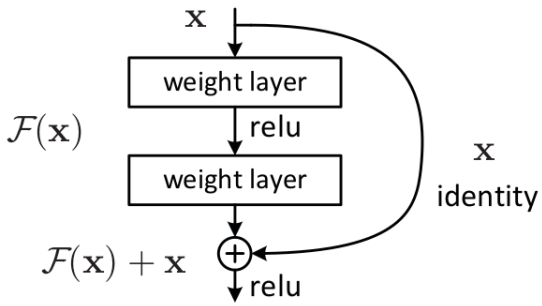
As arquiteturas VGG-16 e VGG-19, desenvolvidas pelo *Visual Geometry Group* em 2014 apresentam bons resultados em variadas aplicações de *Deep Learning*, e têm como principal contribuição o aumento de profundidade de arquitetura, quando comparadas às arquiteturas de anos anteriores, e também pela utilização de filtros convolucionais 3x3. A numeração 16 e 19 indica o nível de profundidade de ambas as arquiteturas, sendo que a arquitetura de profundidade 19 possui três camadas convolucionais extras.

A arquitetura da ResNet talvez seja a que apresente maior peculiaridade dentre as escolhidas para este trabalho. Ela é considerada mais profunda quando comparada

com a SimpleNet e VGG, possuindo até 152 camadas: 8 vezes mais que a VGG. Sua grande profundidade foi devida a observação dos seus autores de que quanto mais profunda a arquitetura, maior a acurácia obtida no treinamento. O diferencial da ResNet é a implementação do conceito de *Deep Residual Learning*. Isso quer dizer que no lugar de empilhar camadas uma a uma, as camadas são encaixadas formando um mapeamento residual.

Essa nova implementação gerou o que se chama de aprendizado residual (*residual learning*). Ao invés de enviar para a próxima camada o resultado obtido do processamento da camada anterior, é enviado a concatenação desse processamento com a identidade, que seria o que foi recebido pela camada antes do processamento. Para melhor entendimento, um bloco residual que realiza essa operação é demonstrado na Figura 20.

Figura 20 – Um bloco residual da ResNet.



Fonte: (HE et al., 2015)

Neste trabalho, foram utilizadas as seguintes versões da ResNet: ResNet-18, ResNet-34, ResNet-50 e ResNet-152.

3.4 Treinamento e validação das arquiteturas

Nessa etapa foram realizados diversos treinamentos entre as arquiteturas selecionadas e os *datasets* escolhidos, para que fosse possível selecionar qual arquitetura será escolhida para ter seus hiperparâmetros otimizados para determinado *dataset*.

Algumas arquiteturas foram treinadas mais de uma vez para um mesmo *dataset*, com modificações no número de épocas e no tamanho do *batch*. Para fins de simplificação, apenas os melhores resultados de cada arquitetura em cada *dataset* serão considerados e mostrados na Tabela 1.

Nota-se também pela Tabela 1 que algumas arquiteturas não foram testadas em todos os *datasets*, como é o caso da VGG-19 e ResNet de tamanhos 34, 50 e 152. A VGG-19 foi treinada uma vez com o *dataset* FERPlus para verificar se era possível obter resultados melhores que os obtidos com a VGG-16, o que não ocorreu.

No caso da ResNet, como foi possível notar que o JAFFE e o CK+ não foram bem sucedidos na ResNet-18, obtendo baixas porcentagens de teste, foi decidido não realizar treinamentos nas arquiteturas mais profundas com 34, 50 e 152 camadas.

Tabela 1 – Resultados dos treinamentos de cada arquitetura em cada *dataset*. (A) corresponde à acurácia de treinamento e (T) corresponde à acurácia de teste.

	JAFFE	**CK+**	**FERPlus**
SimpleNet	A: 99% T: 87%	A: 98% T: 95%	A: 96% T: 75%
VGG-16	A: 87% T: 65%	A: 99% T: 95%	A: 98% T: 78%
VGG-19	Não realizado	Não realizado	A: 98% T: 77%
ResNet-18	A: 98% T: 53%	A: 99% T: 87%	A: 96% T: 72%
ResNet-34	Não realizado	Não realizado	A: 96% T: 75%
ResNet-50	Não realizado	Não realizado	A: 90% T: 66%
ResNet-152	Não realizado	Não realizado	A: 93% T: 71%

Importante mencionar que a metodologia de escolha das arquiteturas por ordem de grandeza, de modo que as menores fossem utilizadas com os menores *datasets*, e as maiores fossem utilizadas com os maiores *datasets* foi confirmada por meio dos treinamentos realizados nesta etapa. A partir destes resultados, foi possível realizar a seleção das arquiteturas que terão seus hiperparâmetros otimizados, processo que será feito e descrito na Seção 3.5 a seguir.

3.5 Seleção de arquitetura a ser otimizada em cada *dataset*

Pelos resultados obtidos nos treinamentos realizados e que podem ser vistos na Tabela 1, foi possível escolher qual a arquitetura será otimizada para cada um dos *datasets*. A escolha foi feita simplesmente pela verificação do melhor resultado de acurácia de treinamento e teste.

A Tabela 2 resume as arquiteturas que terão seus hiperparâmetros otimizados, esperando melhores resultados dos que os apresentados na Tabela 1. Era esperado que a arquitetura ResNet retornasse o melhor resultado para o *dataset* FERPlus, devido sua grande quantidade de imagens, combinada à extensão desta arquitetura. No entanto, a arquitetura VGG-16 apresentou melhor acurácia tanto no treinamento como no teste, e foi escolhida para ser otimizada em dois *datasets*: CK+ e FERPlus.

Tabela 2 – Arquitetura selecionada para otimização dos hiperparâmetros para cada *dataset*.

Dataset	Arquitetura
JAFFE	SimpleNet
CK+	VGG-16
FERPlus	VGG-16

3.6 Definição de espaço de busca de hiperparâmetros

O espaço de busca de hiperparâmetros precisa ser construído obedecendo a estrutura do *hyperopt*. Essa tarefa pode ser feita de várias formas, porém a forma mais comum é instanciar uma lista indexada em Python, nomeando cada hiperparâmetro e definindo seu intervalo de busca.

Um espaço de busca distinto precisará ser construído para cada arquitetura utilizada, devido ao fato de que cada arquitetura possuir estrutura própria. Logo, as arquiteturas escolhidas precisam ser bem compreendidas, de forma a selecionar os melhores hiperparâmetros que irão compor o espaço de busca e seus respectivos intervalos. Um exemplo de espaço de busca para o *hyperopt* pode ser visto na Figura 21.

Figura 21 – Exemplo de um espaço de busca definido para uma *Multilayer Perceptron*.

```
space = {'layer_size':hp.quniform('layer_size', 25, 100, 1),
         'alpha':hp.lognormal('alpha', mu=np.log(1e-4), sigma=1),
         'solver':hp.choice('solver', ['lbfgs', 'sgd', 'adam']),
         'activation':hp.choice('activation', ['logistic', 'tanh', 'relu']),
         'learning_rate':hp.loguniform('learning_rate', low=np.log(1e-4), high=np.log(1.)),
        }
```

Fonte: Elaborado pelo autor

Ambas as arquiteturas SimpleNet e VGG-16 possuem hiperparâmetros semelhantes, no entanto foi necessária a criação de espaços de busca distintos para cada uma. Como foi utilizada a biblioteca Keras para construção de cada CNN, os valores ou intervalos que poderiam ser utilizados em cada hiperparâmetro foram extraídos da documentação da biblioteca, obedecendo as opções que a mesma disponibiliza.

Para a SimpleNet, os seguintes hiperparâmetros foram inseridos no espaço de busca: tipo de ativação, tamanho do *batch* de treinamento, um multiplicador para o tamanho de filtro de convolução, o tipo de inicializador para o *kernel* de convolução, tamanho do *kernel* de convolução, porcentagem de *dropout*, tipo de otimizador, tamanho do filtro de *pooling* e salto do *pooling*. O espaço de busca criado pode ser visualizado na Figura 22.

Figura 22 – Espaço de busca de hiperparâmetros definido para a SimpleNet.

```
space_simplenet = {
    'activation': hp.choice('activation', ['relu', 'elu', 'tanh']),
    'batch_size': hp.quniform('batch_size', 10, 30, 5),
    'conv_filter_size_mult': hp.uniform('conv_filter_size_mult', 0.7, 1.4),
    'conv_kernel_initializer': hp.choice('conv_kernel_initializer', [
        'glorot_normal', 'glorot_uniform', 'random_normal', 'random_uniform'
        ]),
    'conv_kernel_size': hp.quniform('conv_kernel_size', 2, 4, 1),
    'conv_padding': hp.choice('conv_padding', ['same']),
    'data_aug': hp.choice('data_aug', [True]), # sempre usar data augmentation
    'dropout': hp.uniform('dropout', 0.0, 0.35),
    'optimizer': hp.choice('optimizer', ['adam', 'nadam', 'adadelta', 'adamax']),
    'pool_padding': hp.choice('pool_padding', ['same']),
    'pool_size': hp.quniform('pool_size', 2, 4, 1),
    'pool_strides': hp.quniform('pool_strides', 2, 4, 1),
    'use_BN': hp.choice('use_BN', [True]) # sempre usar batch normalization
}
```

Fonte: Elaborado pelo autor

Neste espaço de busca, alguns hiperparâmetros foram definidos com um valor fixo, como é o caso do preenchimento (*padding*) da convolução e do *pooling*, uso de aumento de dados (*data augmentation*) e uso de *batch normalization*. Essa decisão foi tomada baseada na conveniência da arquitetura, pois notou-se em treinamentos prévios que a variação desses hiperparâmetros afetava de forma negativa os resultados obtidos.

O espaço de busca da VGG-16 visto na Figura 23 apesar de semelhante ao da SimpleNet, apresenta diferenças nos intervalos em alguns hiperparâmetros: tamanho do *batch* de treino, opção de utilizar ou não aumento de dados, porcentagem de *dropout*, tipo de otimizador, e não há uma opção definida para *batch normalization*, pois a arquitetura não utiliza esta técnica.

Figura 23 – Espaço de busca de hiperparâmetros definido para a VGG-16.

```
space_vgg_16 = {
    'activation': hp.choice('activation', ['relu', 'elu', 'tanh']),
    'batch_size': hp.quniform('batch_size', 20, 100, 10),
    'conv_filter_size_mult': hp.uniform('conv_filter_size_mult', 0.7, 1.4),
    'conv_kernel_initializer': hp.choice('conv_kernel_initializer', [
        'glorot_normal', 'glorot_uniform', 'random_normal', 'random_uniform'
        ]),
    'conv_kernel_size': hp.quniform('conv_kernel_size', 2, 4, 1),
    'conv_padding': hp.choice('conv_padding', ['same']),
    'data_aug': hp.choice('data_aug', [True, False]),
    'dropout': hp.uniform('dropout', 0.0, 0.5),
    'optimizer': hp.choice('optimizer', ['adam', 'adadelta', 'sgd']),
    'pool_padding': hp.choice('pool_padding', ['same']),
    'pool_size': hp.quniform('pool_size', 2, 4, 1),
    'pool_strides': hp.quniform('pool_strides', 2, 4, 1)
}
```

Fonte: Elaborado pelo autor

Com estes dois espaços de busca construídos, passamos para a última etapa desta metodologia, que é a otimização e seleção dos hiperparâmetros para cada uma das arquiteturas, utilizando os três *datasets* como entrada nas mesmas. Este processo é descrito na Seção a seguir.

3.7 Treinamento, validação e otimização dos hiperparâmetros das CNNs

Na última etapa desta metodologia serão realizados os testes propostos, nos quais a biblioteca *hyperopt* será utilizada para otimizar os hiperparâmetros de cada arquitetura, e a mesma também irá realizar as chamadas aos treinamentos que serão feitos pela biblioteca *Keras*, que também é responsável pela construção das arquiteturas.

O *hyperopt* tem como parâmetros no seu método de minimização *fmin*: o método que constrói e realiza o treinamento da CNN, o espaço de busca a ser utilizado, o algoritmo de otimização (no caso deste trabalho, o TPE foi utilizado), um objeto *trials* que armazena os resultados e hiperparâmetros selecionados de cada avaliação e um inteiro que representa quantas avaliações serão feitas.

A quantidade de épocas dos treinamentos nos *datasets* CK+ e FERPlus foi fixada em 100, e o número de avaliações em 10. No JAFFE foi definido que o treinamento seria feito em 500 épocas, com 20 avaliações. O tempo que levará para otimização irá depender

da complexidade da arquitetura e do tamanho dos dados de entrada. Ao final da otimização, os melhores hiperparâmetros serão armazenados em um arquivo JSON. No Capítulo a seguir serão reunidos e apresentados os resultados obtidos com a metodologia adotada.

4 Resultados

Este capítulo apresenta os resultados obtidos pela aplicação da metodologia proposta. O principal objetivo é comparar os resultados obtidos pela otimização automatizada dos hiperparâmetros por meio da meta-aprendizagem com os resultados obtidos sem otimização.

Nas seções seguintes os resultados alcançados em cada *dataset* são discutidos e detalhados. Cada seção seguirá a mesma organização, exibindo e comparando os resultados das arquiteturas padrões de cada CNN com a arquitetura otimizada, além de apresentar tabelas que irão diferenciar os modelos originais dos modelos otimizados.

4.1 Resultados obtidos com a base JAFFE

Primeiramente é válido relembrar que este *dataset* obteve melhores resultados em treinamentos feitos na arquitetura SimpleNet. Pelo estudo da arquitetura e explicação de HasanPour et al. (2016), o espaço de busca foi definido e apresentado na Figura 22.

Como já mencionado na Seção 3.7, foi definido que a otimização seria limitada em 20 avaliações e o treinamento em 500 épocas. Nestas configurações, o processo completo de otimização durou cerca de 6 horas, com auxílio do hardware descrito no início do Capítulo 3.

Nota-se na Tabela 3, que a acurácia no treinamento não obteve mudanças, permanecendo na mesma porcentagem de 99%. A acurácia de teste, no entanto, cresceu em 8 pontos percentuais, demonstrando que o processo de otimização dos hiperparâmetros da SimpleNet foi válido e garantiu melhora no resultado.

Tabela 3 – Resultados obtidos para o treinamento do JAFFE com a SimpleNet regular e SimpleNet com arquitetura otimizada.

Hiperparâmetros	**Acurácia de Treinamento**	**Acurácia de Teste**
Sem otimização	99%	87%
Com otimização	99%	95%

A Tabela 4 apresenta um esboço da arquitetura final otimizada para *Jaffe*. A ordem das camadas é a mesma tanto para a arquitetura regular quanto para a arquitetura otimizada. As mudanças estão nos hiperparâmetros de cada camada. Nota-se a diferença na quantidade de filtros e o tamanho do *kernel* nas camadas convolucionais; na porcentagem de dropout; no tamanho da sub-amostragem e *strides* nas camadas *pooling*.

Tabela 4 – Visualização das camadas que compõem a arquitetura regular (esquerda) e arquitetura otimizada (direita) da SimpleNet para o *dataset* JAFFE.

Arquitetura regular	**Arquitetura otimizada**
Conv2D (64, 3x3)	Conv2D (54, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (128, 3x3)	Conv2D (108, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (128, 3x3)	Conv2D (108, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (128, 3x3)	Conv2D (108, 2x2)
BatchNorm	BatchNorm
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 4)
Dropout (0.2)	Dropout (0.2253)
Conv2D (128, 3x3)	Conv2D (108, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (128, 3x3)	Conv2D (108, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (256, 3x3)	Conv2D (216, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 4)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (256, 3x3)	Conv2D (216, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (256, 3x3)	Conv2D (216, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 4)
Conv2D (512, 3x3)	Conv2D (433, 2x2)
BatchNorm	BatchNorm
Dropout (0.2)	Dropout (0.2253)
Conv2D (2048, 3x3)	Conv2D (1735, 2x2)
Dropout (0.2)	Dropout (0.2253)
Conv2D (256, 3x3)	Conv2D (216, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 4)
Dropout (0.2)	Dropout (0.2253)
Conv2D (256, 3x3)	Conv2D (216, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 4)
Dense (3072)	Dense (5440)

Outros hiperparâmetros escolhidos pelo otimizador são: tipo de ativação (*ReLu*), tamanho do *batch* (20), inicializador do *kernel* (*random uniform*) e otimizador (*adadelta*). Todos os hiperparâmetros escolhidos podem ser vistos na Figura 24. Houve também uma grande diferença na quantidade total de parâmetros processados, devido a redução de quantidade de filtros convolucionais, e que são apresentados na Tabela 5.

Tabela 5 – Comparação entre quantidade de parâmetros calculados para SimpleNet regular e SimpleNet otimizada para treinamentos com o *dataset* JAFFE.

Tipo	Quantidade de parâmetros
Arquitetura regular	5.515.015
Arquitetura otimizada	2.375.540

Figura 24 – Melhores hiperparâmetros escolhidos para treinamento do JAFFE com a SimpleNet.


```
{
    "activation": "relu",
    "batch_size": 20.0,
    "conv_filter_size_mult": 0.847433617401051,
    "conv_kernel_initializer": "random_uniform",
    "conv_kernel_size": 2.0,
    "conv_padding": "same",
    "data_aug": true,
    "dropout": 0.2253066544231664,
    "optimizer": "adadelta",
    "pool_padding": "same",
    "pool_size": 3.0,
    "pool_strides": 4.0,
    "use_BN": true
}
```


Fonte: Elaborado pelo autor

4.2 Resultados obtidos com a base CK+

Dentre os *datasets* selecionados, o CK+ foi o que obteve os melhores resultados de acurácia, tanto para a arquitetura regular quanto para a arquitetura otimizada. A arquitetura VGG-16 foi escolhida para ter seus hiperparâmetros otimizados para este *dataset*, dado que se sobressaiu sobre as outras, como pode ser revisto na Tabela 1.

O processo de meta-aprendizagem para o CK+ durou cerca de 10 horas, com treinamentos de 100 épocas e o otimizador realizando 10 avaliações. A Tabela 6 apresenta a comparação entre os treinamentos para a VGG-16 regular e VGG-16 otimizada. Apesar

dos bons resultados já obtidos sem a otimização, foi possível alcançar alguma melhora tanto na acurácia de treinamento quanto na acurácia de teste com a arquitetura otimizada, obtendo evolução de 1% e 2%, respectivamente.

Tabela 6 – Resultados obtidos para o treinamento do CK+ com a VGG-16 regular e VGG-16 com arquitetura otimizada.

Hiperparâmetros	**Acurácia de Treinamento**	**Acurácia de Teste**
Sem otimização	99%	95%
Com otimização	100%	97%

Na Tabela 7 é feita a comparação entre a arquitetura regular e a arquitetura otimizada da VGG-16 para o CK+. É possível notar as diferenças entre quantidade de filtros convolucionais e tamanho do *kernel* dos mesmos, além do tamanho das sub-amostragens e *strides* nas camadas *pooling*. Também há mudança na porcentagem de *dropout* na arquitetura otimizada.

Tabela 7 – Visualização das camadas que compõem a arquitetura regular (esquerda) e arquitetura otimizada (direita) da VGG-16 para o *dataset* CK+.

Arquitetura regular	**Arquitetura otimizada**
Conv2D(64, 3x3)	Conv2D (53, 2x2)
Conv2D (64, 3x3)	Conv2D (53, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 3)
Conv2D (128, 3x3)	Conv2D (106, 2x2)
Conv2D (128, 3x3)	Conv2D (106, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 3)
Conv2D (256, 3x3)	Conv2D (213, 2x2)
Conv2D (256, 3x3)	Conv2D (213, 2x2)
Conv2D (256, 3x3)	Conv2D (213, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 3)
Conv2D (512, 3x3)	Conv2D (427, 2x2)
Conv2D (512, 3x3)	Conv2D (427, 2x2)
Conv2D (512, 3x3)	Conv2D (427, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 3)
Conv2D (512, 3x3)	Conv2D (427, 2x2)
Conv2D (512, 3x3)	Conv2D (427, 2x2)
Conv2D (512, 3x3)	Conv2D (427, 2x2)
MaxPool (2x2, strides 2)	MaxPool (3x3, strides 3)
Dense (4096)	Dense (3421)
Dropout (0.5)	Dropout (0.1873)
Dense (4096)	Dense (3421)
Dropout (0.5)	Dropout (0.1873)

Os hiperparâmetros selecionados para a arquitetura otimizada são apresentados na Figura 25. O *hyperopt* selecionou *adadelta* como otimizador, opção diferente do *sgd*, que é comumente utilizado para a VGG-16. Também para o CK+ obteve-se menor número de parâmetros processados com a arquitetura otimizada. Isto ocorre também pelo fato de se utilizar menor quantidade de filtros convolucionais ao longo da CNN. Essa diferença é mostrada na Tabela 8.

Figura 25 – Melhores hiperparâmetros escolhidos para treinamento do CK+ com a VGG-16.


```
{
    "activation": "relu",
    "batch_size": 15.0,
    "conv_filter_size_mult": 0.8352187384147807,
    "conv_kernel_initializer": "glorot_uniform",
    "conv_kernel_size": 2.0,
    "conv_padding": "same",
    "data_aug": false,
    "dropout": 0.18734336426175052,
    "optimizer": "adadelta",
    "pool_padding": "same",
    "pool_size": 3.0,
    "pool_strides": 3.0
}
```


Fonte: Elaborado pelo autor

Tabela 8 – Comparação entre quantidade de parâmetros calculados para VGG-16 regular e VGG-16 otimizada para treinamentos com o *dataset* CK+.

Tipo	Quantidade de parâmetros
Arquitetura regular	56.697.544
Arquitetura otimizada	17.744.257

4.3 Resultados obtidos com a base FERPlus

O último *dataset*, que é o mais extenso dentre os três, resultou em melhores acurácias de treinamento e teste também na arquitetura VGG-16, e por isso, teve seus hiperparâmetros otimizados para o FERPlus. Na Tabela 9 são comparados ambos os treinamentos.

Para este *dataset*, o processo de meta-aprendizagem durou em torno de 13 horas, com treinamentos de 100 épocas e 10 avaliações feitas pelo otimizador. Pela sua grande extensão e variedade de imagens, os treinamentos neste *dataset* foram os que apresentaram

menores taxas de acurácia. No entanto, os resultados são satisfatórios pois são equiparáveis a resultados de outros trabalhos que serão citados no Capítulo 5.

Tabela 9 – Resultados obtidos para o treinamento do FERPlus com a VGG-16 regular e VGG-16 com arquitetura otimizada.

Hiperparâmetros	**Acurácia de Treinamento**	**Acurácia de Teste**
Sem otimização	98%	78%
Com otimização	99%	80%

A arquitetura otimizada é apresentada na Tabela 10, comparada lado a lado com a arquitetura padrão da VGG-16. O otimizador selecionou *ReLu* como método ativação e optou por não utilizar *data augmentation*. Para este *dataset*, o otimizador selecionado foi o *sgd*, padrão da arquitetura VGG-16, e a porcentagem de *dropout* foi definida em 35%.

Tabela 10 – Visualização das camadas que compõem a arquitetura regular (esquerda) e arquitetura otimizada (direita) da VGG-16 para o *dataset* FERplus.

Arquitetura regular	**Arquitetura otimizada**
Conv2D(64, 3x3)	Conv2D (65, 4x4)
Conv2D (64, 3x3)	Conv2D (65, 4x4)
MaxPool (2x2, strides 2)	MaxPool (4x4, strides 3)
Conv2D (128, 3x3)	Conv2D (130, 4x4)
Conv2D (128, 3x3)	Conv2D (130, 4x4)
MaxPool (2x2, strides 2)	MaxPool (4x4, strides 3)
Conv2D (256, 3x3)	Conv2D (261, 4x4)
Conv2D (256, 3x3)	Conv2D (261, 4x4)
Conv2D (256, 3x3)	Conv2D (261, 4x4)
MaxPool (2x2, strides 2)	MaxPool (4x4, strides 3)
Conv2D (512, 3x3)	Conv2D (522, 4x4)
Conv2D (512, 3x3)	Conv2D (522, 4x4)
Conv2D (512, 3x3)	Conv2D (522, 4x4)
MaxPool (2x2, strides 2)	MaxPool (4x4, strides 3)
Conv2D (512, 3x3)	Conv2D (522, 4x4)
Conv2D (512, 3x3)	Conv2D (522, 4x4)
Conv2D (512, 3x3)	Conv2D (522, 4x4)
MaxPool (2x2, strides 2)	MaxPool (4x4, strides 3)
Dense (4096)	Dense (4176)
Dropout (0.5)	Dropout (0.3500)
Dense (4096)	Dense (4176)
Dropout (0.5)	Dropout (0.3500)

O espaço de hiperparâmetros completo da VGG-16 para o FERPlus pode ser visto na Figura 26. Na Tabela 11 é feita a comparação entre o número de parâmetros processados para as arquiteturas regular e otimizada. Importante mencionar que essa

foi a única arquitetura otimizada na qual o número de parâmetros cresceu em relação à arquitetura original. Isso ocorreu por conta do aumento (mesmo que pequeno) de filtros das camadas convolucionais.

Figura 26 – Melhores hiperparâmetros escolhidos para treinamento do FERPlus com a VGG-16.


```
{
    "activation": "relu",
    "batch_size": 30.0,
    "conv_filter_size_mult": 1.274183152684306,
    "conv_kernel_initializer": "glorot_uniform",
    "conv_kernel_size": 4.0,
    "conv_padding": "same",
    "data_aug": false,
    "dropout": 0.3500133416151149,
    "optimizer": "sgd",
    "pool_padding": "same",
    "pool_size": 4.0,
    "pool_strides": 3.0
}
```


Fonte: Elaborado pelo autor

Tabela 11 – Comparação entre quantidade de parâmetros calculados para VGG-16 regular e VGG-16 otimizada para treinamentos com o *dataset* FERPlus.

Tipo	Quantidade de parâmetros
Arquitetura regular	33.628.872
Arquitetura otimizada	46.840.505

4.4 Comparação com trabalhos relacionados

Esta Seção é destinada a comparar os resultados obtidos neste trabalho com os resultados de Li e Deng (2018), trabalho que reúne os resultados de diversos autores na resolução do problema de reconhecimento de expressões faciais. Ele abrange diversas bases de treinamento, dentre elas, as três que foram utilizadas na metodologia deste trabalho.

Para um mesmo *dataset* diversas performances foram citadas, cada uma utilizando sua própria técnica de aprendizagem. Para fins de simplificação, apenas a melhor acurácia em cada base será referenciada. Vale mencionar que Li e Deng (2018) destacam que o valor de acurácia é na verdade a acurácia média, calculada a partir da matriz de confusão de cada resultado.

Um quadro comparativo pode ser visto na Tabela 12. Nota-se que os resultados da metodologia deste trabalho foram satisfatórios, pois como desejado, equiparam-se a resultados de outros trabalhos do mesmo campo de pesquisa. Para referência completa, os resultados resumidos pertencem a: (HAMESTER; BARROS; WERMTER, 2015) (JAFFE), (YU; LIU; LIU, 2017) (CK+) e (PRAMERDORFER; KAMPEL, 2016a) (FER2013).

Tabela 12 – Comparação entre resultados de trabalhos relacionados e os resultados obtidos neste trabalho.

Dataset	Melhor resultado	Resultado neste trabalho
JAFFE	95.8%	99%
CK+	99.6%	100%
FERPlus (FER2013)	75.2% (Teste)	80% (Teste)

5 Conclusão

Este trabalho apresentou abordagens de meta-aprendizagem para a resolução de problemas de reconhecimento de expressões faciais. Buscou-se aprimorar as técnicas já existentes, que na sua grande maioria utiliza Redes Neurais Convolucionais, que foi o subsistema de aprendizagem utilizado na metodologia. O objetivo foi encontrar valores de hiperparâmetros específicos para arquiteturas já conhecidas, para realização de treinamento em três bases diferentes.

A metodologia aplicada foi bem sucedida, pois obteve resultados semelhantes ao estado da arte nas bases selecionadas, comprovando que o bom ajuste das arquiteturas aos dados de entrada favorece o desempenho das CNNs. Convenciona-se entre os pesquisadores a utilização de grandes conjuntos de dados de entrada (grandes bases) em arquiteturas muito profundas ou complexas. Esperava-se neste trabalho que o melhor resultado para a base FERPlus fosse obtido no treinamento com a arquitetura ResNet, o que se provou errado, pois a arquitetura VGG-16 acabou por ser escolhida para ser otimizada neste *dataset*.

A metodologia apresentada neste trabalho pode ser estendida a outros sistemas de Aprendizado de Máquina, favorecendo diferentes problemas existentes por meio de otimizações de arquiteturas de Redes Neurais aplicadas em diferentes esferas. Espera-se que o mesmo abra espaço para trabalhos futuros que envolvam a seleção automática de hiperparâmetros. O código-fonte do mesmo pode ser encontrado em Ribeiro (2018).

Referências

BARSOUM, E. et al. Training deep networks for facial expression recognition with crowd-sourced label distribution. In: *Proceedings of the 18th ACM International Conference on Multimodal Interaction*. New York, NY, USA: ACM, 2016. (ICMI 2016), p. 279–283. ISBN 978-1-4503-4556-9. Disponível em: <http://doi.acm.org/10.1145/2993148.2993165>. Citado 3 vezes nas páginas 30, 31 e 32.

BARTLETT, M. S. et al. Real time face detection and facial expression recognition: Development and applications to human computer interaction. In: IEEE. *Conference on Computer Vision and Pattern Recognition Workshop, 2003. CVPRW'03*. Madison, WI, USA, 2003. v. 5, p. 53–53. Citado na página 13.

BENGIO, Y. Deep learning of representations for unsupervised and transfer learning. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Bellevue, Washington, USA: PMLR, 2012. (Proceedings of Machine Learning Research, v. 27), p. 17–36. Disponível em: <http://proceedings.mlr.press/v27/bengio12a.html>. Citado na página 18.

BENGIO, Y. et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, Now Publishers, Inc., v. 2, n. 1, p. 1–127, 2009. Citado na página 18.

BERGSTRA, J. et al. Algorithms for hyper-parameter optimization. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. Granada, Spain: Curran Associates Inc., 2011. (NIPS'11), p. 2546–2554. ISBN 978-1-61839-599-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2986459.2986743>. Citado 3 vezes nas páginas 15, 25 e 27.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, JMLR.org, v. 13, p. 281–305, fev. 2012. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=2188385.2188395>. Citado 2 vezes nas páginas 26 e 27.

BERGSTRA, J. et al. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, v. 8, n. 1, p. 014008, 2013. Disponível em: <http://stacks.iop.org/1749-4699/8/i=1/a=014008>. Citado 2 vezes nas páginas 15 e 26.

BERGSTRA, J. S. et al. Algorithms for hyper-parameter optimization. In: SHAWE-TAYLOR, J. et al. (Ed.). *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., 2011. p. 2546–2554. Disponível em: <http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf>. Citado na página 26.

BRILLIANT.ORG. *Feedforward Neural Networks*. 2018. <https://brilliant.org/wiki/feedforward-neural-networks/>. Online; acessado em 15 de Setembro de 2018. Citado na página 19.

CLAESEN, M.; MOOR, B. D. Hyperparameter search in machine learning. *CoRR*, abs/1502.02127, 2015. Citado 2 vezes nas páginas 23 e 25.

COATES, A.; NG, A. Y. The importance of encoding versus training with sparse coding and vector quantization. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Bellevue, WA, USA: Omnipress, 2011. (ICML'11), p. 921–928. ISBN 978-1-4503-0619-5. Disponível em: <http://dl.acm.org/citation.cfm?id=3104482.3104598>. Citado na página 15.

DENG, J. et al. Scalable multi-label annotation. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2014. (CHI '14), p. 3099–3102. ISBN 978-1-4503-2473-1. Disponível em: <http://doi.acm.org/10.1145/2556288.2557011>. Citado na página 14.

DHAVALIKAR, A. S.; KULKARNI, R. Face detection and facial expression recognition system. In: IEEE. *2014 International Conference on Electronics and Communication Systems (ICECS)*. Coimbatore, India, 2014. p. 1–7. Citado na página 14.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, Massachusetts, USA: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 2 vezes nas páginas 21 e 22.

GOODFELLOW, I. J. et al. Challenges in representation learning: A report on three machine learning contests. In: SPRINGER. *International Conference on Neural Information Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 117–124. Citado na página 31.

HAMESTER, D.; BARROS, P.; WERMTER, S. Face expression recognition with a 2-channel convolutional neural network. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. Killarney, Ireland: IEEE, 2015. p. 1–8. ISSN 2161-4407. Citado na página 50.

HASANPOUR, S. H. et al. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037*, 2016. Citado 2 vezes nas páginas 36 e 43.

HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>. Citado 2 vezes nas páginas 36 e 37.

HINTON, G. E. A practical guide to training restricted boltzmann machines. In: _____. *Neural Networks: Tricks of the Trade: Second Edition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 599–619. ISBN 978-3-642-35289-8. Disponível em: <https://doi.org/10.1007/978-3-642-35289-8_32>. Citado na página 25.

HSU, C.-W.; CHANG, C.-C.; LIN, C.-J. A practical guide to support vector classification. 2003. Citado na página 25.

JIA, J. et al. Head and facial gestures synthesis using pad model for an expressive talking avatar. *Multimedia Tools and Applications*, 08 2013. Citado na página 31.

KAGGLE. *Kaggle Inc.* 2018. <https://www.kaggle.com/>. Online; acessado em 18 de Outubro de 2018. Citado na página 31.

KANADE, T.; TIAN, Y.; COHN, J. F. Comprehensive database for facial expression analysis. In: IEEE. *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*. Grenoble, France, 2000. p. 46. Citado 2 vezes nas páginas 14 e 30.

KOEHRSEN, W. *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*. 2018. <https://bit.ly/2lYVvXU>. Online; acessado em 15 de Novembro de 2018. Citado na página 28.

KOŁAKOWSKA, A. et al. Emotion recognition and its applications. v. 300, p. 51–62, 07 2014. Citado na página 13.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, v. 1, n. 4, p. 541–551, 1989. Citado na página 19.

LEMKE, C.; BUDKA, M.; GABRYS, B. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*, v. 44, n. 1, p. 117–130, Jun 2015. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-013-9406-y>. Citado 2 vezes nas páginas 24 e 25.

LEVI, G.; HASSNER, T. Emotion recognition in the wild via convolutional neural networks and mapped binary patterns. In: ACM. *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. New York, NY, USA, 2015. p. 503–510. Citado na página 14.

LI, S.; DENG, W. Deep facial expression recognition: A survey. *CoRR*, abs/1804.08348, 2018. Disponível em: <http://arxiv.org/abs/1804.08348>. Citado 3 vezes nas páginas 30, 31 e 49.

LIU, H.; SIMONYAN, K.; YANG, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. Citado na página 27.

LUCEY, P. et al. The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. In: IEEE. *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society*. San Francisco, CA, USA, 2010. p. 94–101. Citado na página 30.

LYONS, M. et al. Coding facial expressions with gabor wavelets. In: IEEE. *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*. Nara, Japan, 1998. p. 200–205. Citado na página 30.

MANGLIK, P. K. et al. Facial expression recognition. In: IEEE. *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*. The Hague, Netherlands, 2004. v. 3, p. 2220–2224. Citado na página 13.

MISKAM, M. A. et al. Humanoid robot nao as a teaching tool of emotion recognition for children with autism using the android app. In: IEEE. *2014 International Symposium on Micro-NanoMechatronics and Human Science (MHS)*. Nagoya, Japan, 2014. p. 1–5. Citado na página 13.

MOLLAHOSSEINI, A.; CHAN, D.; MAHOOR, M. H. Going deeper in facial expression recognition using deep neural networks. In: IEEE. *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Lake Placid, NY, USA, 2016. p. 1–10. Citado na página 14.

NG, A. Y.-T. *What data scientists should know about deep learning*. 2015. <https://www.slideshare.net/ExtractConf>. Online; acessado em 10 de Setembro de 2018. Citado na página 18.

NIELSEN, M. *Neural Networks and Deep Learning*. 2018. <http://neuralnetworksanddeeplearning.com/index.html>. Online; acessado em 18 de Outubro de 2018. Citado na página 17.

NNI, M. *Neural Network Intelligence*. 2018. <https://microsoft.github.io/nni/>. Online; acessado em 07 de Novembro de 2018. Citado na página 27.

PANG, B.; LEE, L. et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, Now Publishers, Inc., v. 2, n. 1–2, p. 1–135, 2008. Citado na página 13.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, JMLR.org, v. 12, p. 2825–2830, nov. 2011. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=1953048.2078195>. Citado na página 25.

PELIKAN, M.; GOLDBERG, D. E.; CANTÚ-PAZ, E. Boa: The bayesian optimization algorithm. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (GECCO'99), p. 525–532. ISBN 1-55860-611-4. Disponível em: <http://dl.acm.org/citation.cfm?id=2933923.2933973>. Citado na página 26.

PINTO, N. et al. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, Public Library of Science, v. 5, n. 11, p. e1000579, 2009. Citado na página 15.

PRAMERDORFER, C.; KAMPEL, M. Facial expression recognition using convolutional neural networks: state of the art. *arXiv preprint arXiv:1612.02903*, 2016. Citado 2 vezes nas páginas 14 e 50.

PRAMERDORFER, C.; KAMPEL, M. Facial expression recognition using convolutional neural networks: State of the art. *CoRR*, abs/1612.02903, 2016. Disponível em: <http://arxiv.org/abs/1612.02903>. Citado na página 31.

RIBEIRO, J. *Código-fonte deste trabalho*. 2018. <https://github.com/jorgimello/meta-learning-facial-expression-recognition>. Online; acessado em 14 de Dezembro de 2018. Citado na página 51.

SARODE, N.; BHATIA, S. Facial expression recognition. *International Journal on computer science and Engineering*, Engg Journals Publications, v. 2, n. 5, p. 1552–1557, 2010. Citado 2 vezes nas páginas 13 e 30.

SHAN, C.; GONG, S.; MCOWAN, P. W. Facial expression recognition based on local binary patterns: A comprehensive study. *Image and vision Computing*, Elsevier, v. 27, n. 6, p. 803–816, 2009. Citado 2 vezes nas páginas 14 e 30.

SHARMA, A. *Is there a difference between neural networks and convolutional neural networks?* 2018. <https://www.quora.com/Is-there-a-difference-between-neural-networks-and-convolutional-neural-networks>. Online; acessado em 16 de Outubro de 2018. Citado 2 vezes nas páginas 20 e 21.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado na página 36.

SZEGEDY, C. et al. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. Disponível em: <http://arxiv.org/abs/1409.4842>. Citado na página 36.

VANSCHOREN, J. Understanding machine learning performance with experiment databases. *lirias. kuleuven. be, no. May*, 2010. Citado na página 24.

VISION, S.; LAB, L. *Convolutional Neural Networks for Visual Recognition*. 2018. <http://cs231n.github.io/convolutional-networks/>. Online; acessado em 20 de Outubro de 2018. Citado na página 23.

YU, Z.; LIU, Q.; LIU, G. Deeper cascaded peak-piloted network for weak expression recognition. *The Visual Computer*, v. 34, p. 1691–1699, 2017. Citado na página 50.

ZHANG, K. et al. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, v. 23, n. 10, p. 1499–1503, Oct 2016. ISSN 1070-9908. Citado na página 32.

ZHOU, Y. . et al. Image restoration using a neural network. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 36, n. 7, p. 1141–1151, July 1988. ISSN 0096-3518. Citado na página 22.