UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

LUÍS GUSTAVO VITORINO DE SOUZA

SISTEMA PARA DETECÇÃO DO USO DE MÁSCARA EM TRANSEUNTES UTILIZANDO TÉCNICAS DE PROCESSAMENTO DE IMAGENS E APRENDIZADO DE MÁQUINA

OURO PRETO
2021

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO

UFOP

ATA DE SESSÃO DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

No segundo dia do mês de setembro de dois mil e vinte um, realizou-se às 10 horas, de forma não presencial, por meio do aplicativo Google Meet (meet.google.com/gjs-wmbw-pdr), a sessão de defesa de Trabalho de Conclusão de Curso do candidato ao grau de Engenheiro de Controle e Automação, **Luís Gustavo Vitorino de Souza**, intitulada "Sistema para Detecção do Uso de Máscara em Transeuntes Utilizando Técnicas de Processamento de Imagens e Aprendizado de Máquina". A Banca Examinadora foi constituída pelo Pesquisador André Almeida dos Santos (Orientador, ITV), pelo Prof. Agnaldo José da Rocha Reis (Universidade Federal de Ouro Preto, Coorientador), pelo Prof. Eduardo José da Silva Luz (Universidade Federal de Ouro Preto, Examinador Externo) e pela Profa. Adrielle de Carvalho Santana (Universidade Federal de Ouro Preto, Examinadora Interna). O pesquisador André abriu a sessão agradecendo a participação dos examinadores supracitados e passou a palavra ao candidato, que fez a exposição do seu trabalho. Em seguida, foi realizada a arguição pelos examinadores, com a respectiva defesa do candidato. Finalizada a arguição, a Banca Examinadora, sem a presença do candidato, deliberou pela sua **Aprovação**. Nada mais havendo para constar, lavrou-se a presente ata que será assinada eletronicamente pelo Coorientador via SEI!UFOP, com anuência dos demais examinadores e do aluno.

André Almeida Santos - ITV -Orientador
Agnaldo José da Rocha Reis - UFOP - Coorientador
Eduardo José da Silva Luz - UFOP - Examinador
Adrielle Carvalho de Santana - UFOP - Examinadora
Luís Gustavo Vitorino de Souza - Discente

Documento assinado eletronicamente por **Agnaldo Jose da Rocha Reis**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 02/09/2021, às 12:39, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0216574** e o código CRC **525A59A0**.

Referência: Caso responda este Memorando, indicar expressamente o Processo nº 23109.009192/2021-40 SEI nº 0216574

R. Diogo de Vasconcelos, 122, - Bairro Pilar - Ouro Preto/MG, CEP 35400-000
Telefone: 3135591533 - www.ufop.br

AGRADECIMENTOS

Agradeço a Deus pela saúde e iluminação.

Aos meus pais por todo incentivo e apoio.

Aos meus irmãos pela compreensão e apoio.

A Universidade Federal de Ouro Preto e professores, pelo estudo de qualidade.

Aos professores André Almeida Santos e Agnaldo José da Rocha Reis pela orientação e todos os ensinamentos.

Ao Gabriel Carvalho Garcia pela sugestão de tema deste trabalho.

RESUMO

Foi estudado e aprimorado um sistema de detecção do uso de máscara antiviral em imagens com a função de notificar o resultado ao usuário administrador via e-mail. Foi estudado um banco de dados já existente e a ele incrementado, pelo autor deste trabalho, imagens de pessoas utilizando máscaras reais. Foram estudadas redes neurais convolucionais e todos os conceitos mais relevantes para o desenvolvimento deste tipo de rede, tais como: camadas de convolução, neurônio, função de ativação e funções de otimização. Para o desenvolvimento do treinamento da rede foram estudadas as ferramentas Keras e Tensorflow, que foram utilizadas para arquitetar as camadas da rede. Em seguida, foram definidas, executadas e comparadas 9 formas diferentes de treinamento que se diferenciam em épocas e função de otimização. A configuração de treinamento com 20 épocas e função de ativação adam atingiu melhor performance, essa foi testada utilizando o método de data augmentation, configuração que não melhorou os resultados do treinamento. Após a finalização do treinamento da rede, foram expostas imagens de pessoas a serem classificadas como utilizando máscara ou não. Os resultados de precisão obtidos com as classificações são maiores que 90%, sendo a notificação enviada instantaneamente para o e-mail cadastrado.

Palavras-chaves: Reconhecimento do uso de máscara, Visão Computacional, Processamento de Imagens, Aprendizado de Máquina, Redes Neurais Artificiais, COVID-19.

ABSTRACT

A system for detection of antiviral mask use in images was studied and enhanced with a result notification to the administrator user of the email address registered in the system. An existent database was studied and enhanced by the author of this work with images of people using real masks. Convolutional neural networks and all important concepts for the development of this kind of neural network were studied, such as convolutional layers, neuron, activation function and optimization functions. For the development of network training, Keras and Tensorflow tools were studied, which were used to architect the network layers. Afterwards, 9 different forms of training that differ in epochs and optimization function were defined, executed and compared. The training configuration with 20 epochs and adam activation function achieved the best performance, it was tested using the data augmentation method which did not improve the training results. After completing the training of the network, were exposed images of people to be classified as wearing a mask or not. The accurate results obtained are greater than 90%, with notification being sent instantly to the email address registered.

Keywords: Recognition of mask use, Computer Vision, Image Processing, Machine Learning, Artificial Neural Networks, COVID-19.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Em meio à pandemia de Covid-19, a sociedade necessita estabelecer novos hábitos para reduzir o número de contaminados. A elevada taxa de infectividade do vírus da Covid, aliada à ausência de imunidade prévia da população e à inexistência de vacina, fazem com que o crescimento do número de casos seja exponencial caso medidas não sejam tomadas para deter sua transmissão (Kucharski AJ, 2020). Neste contexto, são indicadas Intervenções Não Farmacológicas (INF), que incluem medidas com alcance individual, ambiental e comunitário, tais como: a lavagem das mãos, a etiqueta respiratória, o distanciamento social e o uso de máscara facial (Garcia LP, 2020).

No dia 17 de março de 2020 foi registrada a primeira morte por infecção do novo vírus no Brasil e até o dia 10 de agosto de 2021 são 563.562 óbitos, sendo 140.809 só no estado de São Paulo. O número de casos já chega a 20.177.757, sendo que o pico de 115.230 casos registrados ocorreu no dia 23 de junho de 2021. A pandemia, em 10 de agosto de 2021, encontra-se em tendência de baixa, registrando-se 11.843 casos. Minas Gerais que registrou um total de 1.757.649 de casos acumulados no dia 10 de agosto de 2021, é o segundo estado com maior número de pessoas infectadas no país, sendo 45.036 o número total de óbitos registrados no dia 10 de agosto de 2021 (Saúde, 2021).

O presidente da república Jair Bolsonaro sancionou no dia 03 de Julho de 2020 a Lei nº 14.019/2020 que torna obrigatório o uso de máscaras de proteção individual em espaços públicos e privados durante a pandemia, incluindo vias públicas e transportes públicos coletivos, como ônibus aeronaves ou embarcações de uso coletivo fretados. De acordo com lei, as máscaras podem ser artesanais ou industriais, porém a fiscalização da utilização é importante para todos os setores, sendo passível de multa o não cumprimento (Planalto, 2020). Considerando a grande presença de câmeras de segurança em locais públicos e o grande número de pessoas que frequentam diversos ambientes a serem fiscalizados, torna-se muito relevante o uso da tecnologia na detecção do uso de máscaras com o objetivo de redução de custo da função e de prevenir penalizações contidas na lei.

A visão artificial vem sendo desenvolvida em paralelo a modelos matemáticos para identificar a forma tridimensional em imagens (Szeliski, 2010). Disponibilizando uma base de dados contendo uma quantidade suficiente de exemplos do objeto ou das formas que se deseja detectar, consegue-se construir modelos de identificação de alta precisão. É possível detectar pessoas em movimento em imagens com fundo complexo, e até mesmo distinguir pessoas por meio de características do rosto, cabelo ou vestimenta.

Comparada com a computação gráfica, a visão computacional pode ser considerada como o sistema inverso, onde tenta-se descrever o mundo que vemos em uma ou mais imagens e reconstruir suas propriedades, tais como o formato, iluminação e distribuição de cores (Szeliski, 2010).

Segundo Szeliski (2010), a visão computacional tem sido usada em uma vasta variedade de aplicações reais que incluem:

- Reconhecimento Óptico de Caracteres: Leitura de manuscritos e reconhecimento de número de placas;
- Inspeção de Máquinas: Inspeção rápida de qualidade de peças de aço utilizando visão raio-x;
- Segurança automotiva: Detecção de obstáculos inesperados como pedestres na rua;
- Vigilância: Detecção de intrusos, tráfego intenso ou monitoramento em piscinas para evitar afogamentos;
- Reconhecimento Biométrico: Detecção de padrão de biometria para acesso automático e demandas forenses;
- Detecção de face: Detecção de faces para ajuste de foco em câmeras ou pesquisa de imagens.

Redes Neurais Artificiais (RNA) são muito utilizadas em tarefas de visão computacional. Essas possuem capacidade de aprender a realizar uma certa tarefa, ou comportamento, a partir de um conjunto de exemplos dados. Aliadas a técnicas de processamento de imagens, podem desenvolver tarefas com robustez e tolerância à ruído, como reconhecimento de padrões, classificação de padrões, tratamento de imagens e visão artificial (Fernando Osório, 2000).

Neste trabalho um modelo matemático de rede neural será treinado utilizando imagens de pessoas utilizando ou não máscaras, a fim de que este modelo se torne apto a detectar com alta precisão quando uma imagem contém o rosto de uma pessoa utilizando máscara ou não, enviando o resultado da análise ao endereço de e-mail predeterminado.

1.1 Objetivo Geral

Desenvolver um sistema de visão computacional para detecção automática do uso de máscara utilizando técnicas de processamento de imagens e aprendizado de máquina.

1.2 Objetivo Específicos

Este trabalho tem como objetivos específicos os seguintes itens:

1. Definir a base de dados;
2. Definir as técnicas de pré-processamento a serem empregadas;
3. Escolher e configurar a rede neural artificial para a detecção do uso de máscara;
4. Testar modelo com diferentes hiperparâmetros;
5. Acrescentar ao sistema função para envio da classificação por e-mail;
6. Apresentar e analisar os resultados obtidos.

1.3 Estrutura do trabalho

No Capítulo 1 foi apresentada a atual situação da pandemia de COVID-19 no Brasil, demonstrando dados, protocolos de segurança e normatizações provenientes da crise sanitária, que tem como objetivo amenizar seus prejuízos à população. Também foi apresentada a ideia de como a tecnologia associada à inteligência artificial pode nos ajudar no combate ao vírus. Já no segundo capítulo, serão referenciados teoricamente todos os métodos, conceitos, técnicas e ferramentas utilizadas para o desenvolvimento do sistema de detecção de máscaras em imagens. Os tópicos lá apresentados são fundamentados em obras bibliográficas renomadas no campo de processamento de imagens e redes neurais. Além dos livros, serão utilizados artigos de grande importância para o desenvolvimento da inteligência artificial e a própria documentação dos softwares. A metodologia aplicada para o desenvolvimento do trabalho é apresentada no Capítulo 3. Nele são enumerados e explicados por ordem de execução todos os passos necessários para o desenvolvimento, funcionamento e apuração de desempenho do sistema. No Capítulo 4 são apresentados todos os resultados obtidos. Sob a forma de gráficos, tabelas e imagens são analisados os valores de saída e de performance tanto da etapa de treinamento da rede quanto da etapa de detecção do uso ou não de máscara nos rostos das

imagens utilizadas como exemplo. Após apurados os resultados, apresenta-se a conclusão do trabalho no Capítulo 5, que leva em consideração a comparação entre os resultados obtidos neste trabalho e de trabalhos semelhantes. Também são avaliadas duas formas diferentes de aplicação da rede MobileNetV2. Além disso, são propostas melhorias e possíveis aplicações para trabalhos futuros. Ao final, no Capítulo 6, encontra-se a bibliografia, com todas as obras usadas como base para o desenvolvimento deste trabalho, além dos apêndices, onde se tem acesso a todos os códigos contidos no sistema.

2 REFERENCIAL TEÓRICO

Nesta seção encontra-se a revisão bibliográfica das ferramentas e técnicas utilizadas para o desenvolvimento do trabalho. Serão conceituados rede neural e suas características, aprendizado de máquina, bibliotecas dedicadas às funções e a utilizada para o processamento de imagens. Além disso, será descrito e explicado o banco de dados utilizado no treinamento da rede.

2.1 Rede Neural Artificial

O trabalho com Redes Neurais Artificiais (RNA), comumente chamadas de Redes Neurais, vem sendo motivado pelo fato de o cérebro humano realizar cálculos de uma maneira completamente diferente dos computadores digitais convencionais. O cérebro é um complexo computador paralelo e não linear. Um sistema de processamento de informação que possui capacidade de organizar seus componentes estruturais, conhecidos como neurônios, para realizar tarefas como o reconhecimento de padrões e controle de motores de maneira mais rápida que os computadores mais eficientes existentes (Haykin S. , 2008).

Um cérebro possui naturalmente estrutura e habilidade de construir suas próprias regras de comportamento, o que é chamado experiência. Essa evolui com o passar do tempo, sendo sua maior fase de desenvolvimento do nascimento até os dois anos de idade. A plasticidade do cérebro é essencial para que os neurônios funcionem como unidades de processamento de informação. Ela permite que o sistema nervoso se adapte às situações que o cercam, característica também aplicada a neurônios artificiais que, para alcançar boa performance necessitam de uma robusta conexão entre si.

Segundo Haykin S. (2008), Redes Neurais Artificiais são definidas como um robusto processador paralelo e distribuído feito de simples unidades de processamento que tem a característica natural de guardar conhecimento adquirido por experiência e disponibilizá-lo para uso se assemelhando ao cérebro em dois aspectos:

1. O conhecimento é adquirido do ambiente da rede por meio de um processo de aprendizagem;
2. A intensidade das conexões entre os neurônios conhecidas como pesos sinápticos são utilizadas para guardar o conhecimento.

O procedimento utilizado para executar o processo de aprendizado, chamado de Algoritmo de Aprendizado, possui a função de modificar os pesos sinápticos da rede de maneira que se ajustem para atingir o objetivo do projeto. Uma Rede Neural tem também a capacidade de modificar sua topologia relacionando-se ao fato de que no cérebro humano neurônios podem morrer, gerando assim uma nova conexão sináptica (Haykin S. , 2008).

Um neurônio é uma unidade de processamento de informação fundamental no funcionamento de uma Rede Neural. A Figura 1 mostra o modelo de um neurônio composto pelos seguintes elementos:

1. Um conjunto de Sinapses, conexões, que se diferenciam uma das outras por seus pesos individuais. Cada sinal x_j na entrada de uma sinapse j conectada a um neurônio k é multiplicado pelo peso da sinapse w_{kj}.
2. Uma junção somadora que soma os valores de cada entrada x_j multiplicada por seu respectivo peso w_{kj}.
3. Um valor b_k chamado Bias que aumenta ou diminui a entrada da função de ativação v_k sendo seu valor positivo ou negativo.
4. Uma função de ativação que limita a amplitude da saída do neurônio geralmente à intervalos [0,1] ou [-1,1].

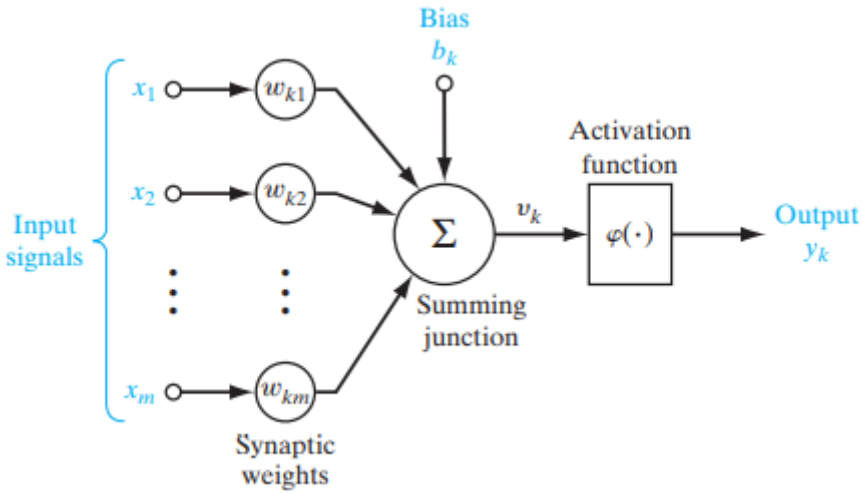

Figura 1 - Modelo de Neurônio não linear.

Fonte: (Haykin S. , 2008)

Matematicamente podemos representar o neurônio utilizando a Equação 1 e Equação 2:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

$$y_k = \varphi(u_k + b_k) \tag{2}$$

Onde x_1, x_2, ..., x_m são sinais de entrada, w_{k1}, w_{k2}, ..., w_{km} são os respectivos pesos sinápticos do neurônio k. u_k é a combinação linear de saída referente aos sinais de entrada e seus pesos, b_k o bias, $\varphi(.)$ a função de ativação e y_k é o sinal de saída do neurônio. Na Figura 1 o efeito do Bias nos sinais de entrada é representado por v_k que corresponde a Equação 3:

$$v_k = u_k + b_k \tag{3}$$

A função de ativação $\varphi(.)$ determina a saída de um neurônio com relação ao valor de v_k. A seguir serão explicados três tipos de função de ativação relevantes para este trabalho:

1. Função de Heaviside: Também conhecida como função degrau pelo formato de seu gráfico demonstrado na Figura 2. Um neurônio que possui esse tipo de função de ativação possui saída 0 caso v_k for negativo e 1 nos demais casos como definida abaixo pela Equação 4.

$$\varphi(v) = \begin{cases} 1 \; se \; v \geq 0 \\ 0 \; se \; v < 0 \end{cases} \tag{4}$$

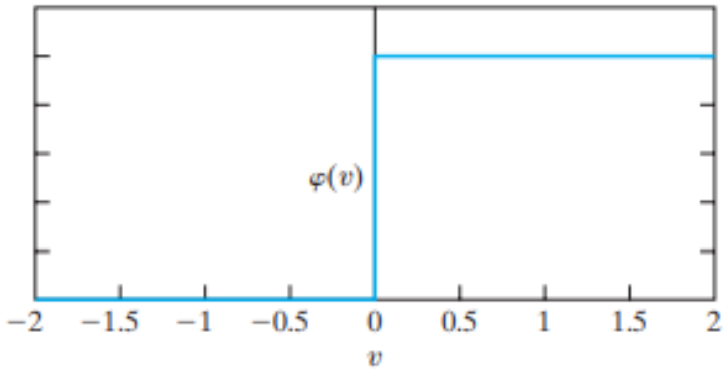

Figura 2 - Gráfico função de Heaviside.

Fonte: (Haykin S. , 2008)

2. Sigmoide: Funções Sigmoides, demonstrada na Figura 3, são a forma mais comum de função de ativação utilizada em redes neurais por ser uma função crescente com um bom equilíbrio entre linearidade e não linearidade. Um exemplo desse tipo de função é a função hiperbólica definida pela Equação 5:

$$\varphi(v) = tanh\,(v) \tag{5}$$

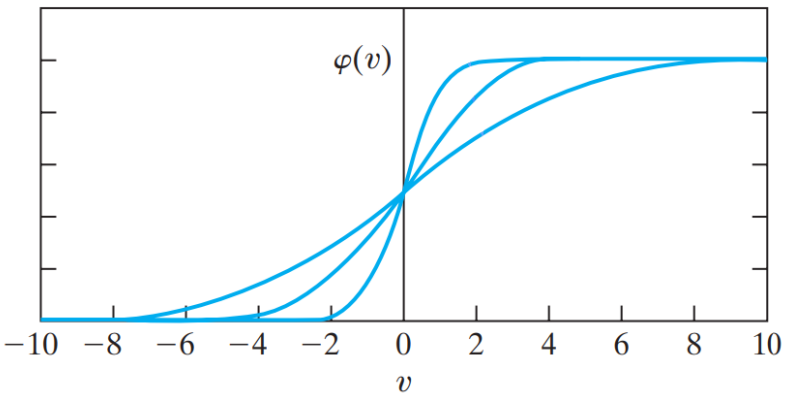

Figura 3 - Gráfico funções Sigmoide.

Fonte: (Haykin S. , 2008)

3. ReLU: Do inglês Restricted Linear Unit (Unidade linear restrita), a aplicação da função ReLU como função de ativação em redes neurais é atualmente o padrão se tratando de redes que desempenhem tarefas complexas como a de detecção de objetos e fala. Uma das características da função que levaram a essa padronização é que a mesma apresenta saída sempre linear. Abaixo encontram-se sua definição pela Equação 6 e seu comportamento no plano cartesiano representado na Figura 4 (Konstantin Eckle, 2018).

$$f(z) = max\ (0, z) \tag{6}$$

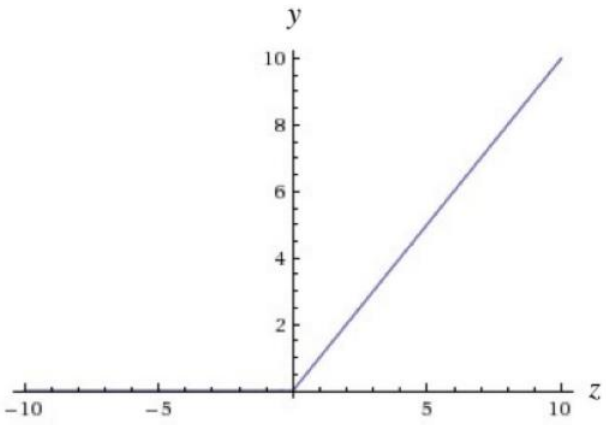

Figura 4 - Função ReLU.

Fonte: (Konstantin Eckle, 2018).

3.1. ReLU6: A função de ativação ReLU6 (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018) é uma modificação do valor máximo da função ReLU à 6 como pode-se observar na Figura 5. O objetivo dessa modificação é aumentar sua robustez quando utilizada em aplicações computacionais de baixa precisão.

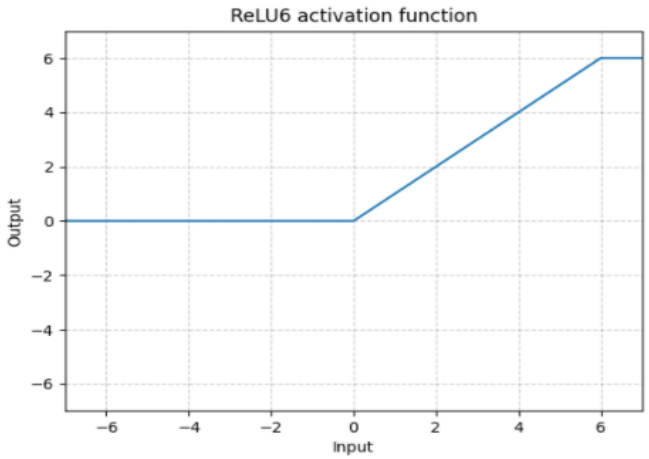

Figura 5 - Função de ativação ReLU6.

Fonte: (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018)

Em uma Rede Neural os neurônios são organizados em camadas sendo a forma mais simples de arquitetura uma camada de entradas conectada diretamente em uma camada de neurônios de saída, processando informação somente nessa direção. Esse tipo de arquitetura é conhecida como *Single Layer Feedfoward* e é representada na Figura 6 com quatro nós de entrada e saída de dados (Haykin S. , 2008).

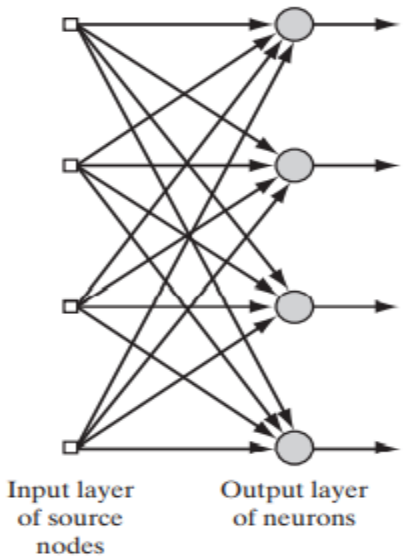

Figura 6 - Rede neural Feedfoward de uma camada.

Fonte: (Haykin S. , 2008).

Outro modelo de arquitetura da mesma classe, conhecido como *Multiple Feedfoward Networks*, se diferencia por possuir uma ou mais camadas (Figura 7) formadas por neurônios que possuem a finalidade de interferir nos dados entre as camadas de entrada e saída. Essas representam o conceito de *Deep Learning.*

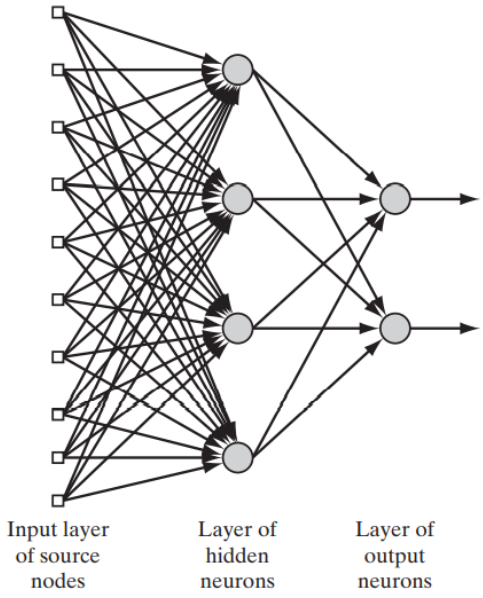

Figura 7 - Rede neural Feedfoward com uma camada de entrada e uma camada de saída.

Fonte: (Haykin S. , 2008)

Em um dos maiores eventos de visão computacional do mundo até 2017, ImageNet Large Scale Visual Recognition Challenge (ILSVRC), foram aplicadas pesquisas de inteligência artificial em uma competição de performance em classificação de imagens. A atividade se resumia em classificar cinco imagens entre duzentas classes possíveis contando com uma base de dados de quatrocentas e cinquenta mil imagens para treinamento. O objetivo da competição é apontar o estado da arte em visão computacional. Em 2011 o vencedor apresentou uma taxa de erro de 25,7%, um bom resultado, porém não o suficiente para aplicações comerciais. Em 2012, Alex Krizhevsky, utilizando uma Rede Neural Convolucional, apresentou uma taxa de erro de 26,1% durante a competição, diminuindo para 16% meses depois. Esse resultado rompeu um ciclo de 50 anos de pesquisa e revolucionou o campo de visão computacional (Buduma, 2017). Essa é a arquitetura utilizada para a realização deste trabalho e será introduzida no próximo tópico.

2.2 Redes Neurais Convolucionais

Redes Neurais Convolucionais (CNNs) ou Redes Convolucionais são uma classe de redes neurais de múltiplas camadas utilizadas em tarefas de reconhecimento de padrões. Em 1959 David Hubel e Torsten Wiesel projetaram padrões preto e branco em uma tela e, utilizando eletrodos conectados no cérebro de um gato, constataram que dependendo dos padrões mostrados (linhas verticais, horizontais ou inclinadas) diferentes neurônios foram ativados. Trabalhos posteriores concluíram que o córtex é organizado em camadas, sendo cada uma responsável por trabalhar em cima das informações extraídas de sua camada anterior, indo de linhas para colunas e de formas para objetos, replicando o trabalho por toda a área da imagem (Buduma, 2017).

Na Figura 8 podemos visualizar as divisões de camadas em um exemplo de rede neural convolucional de múltiplas camadas. As funções de cada uma delas serão explicadas a seguir.

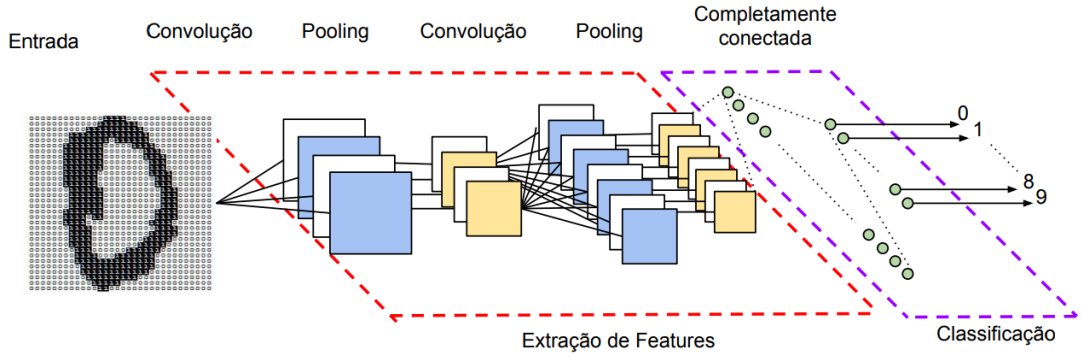

Figura 8 - Exemplo de uma rede neural convolucional e suas diferentes camadas.

Fonte: (Ana Caroline Gomes Vargas, 2016)

2.2.1 Filtro

Um filtro é uma ferramenta que pode ser utilizada para detectar informações. Em um exemplo em que se deseja detectar linhas verticais e horizontais em uma imagem, uma abordagem seria utilizar um filtro para linhas verticais e outro para linhas horizontais posicionados inicialmente no topo esquerdo da imagem sendo transportados por toda sua extensão, verificando-se em cada passo se são encontradas semelhanças. As respostas são alocadas em matrizes reservadas, como mostrado à direita da Figura 9, onde cada correspondência com os filtros encontrada na imagem é simbolizada por um quadrado

preto e as não correspondências simbolizadas por um quadrado branco. Ao conjunto de matrizes que armazenam as respostas das operações realizadas com os filtros, dá-se o nome de *Feature Map* (Buduma, 2017).

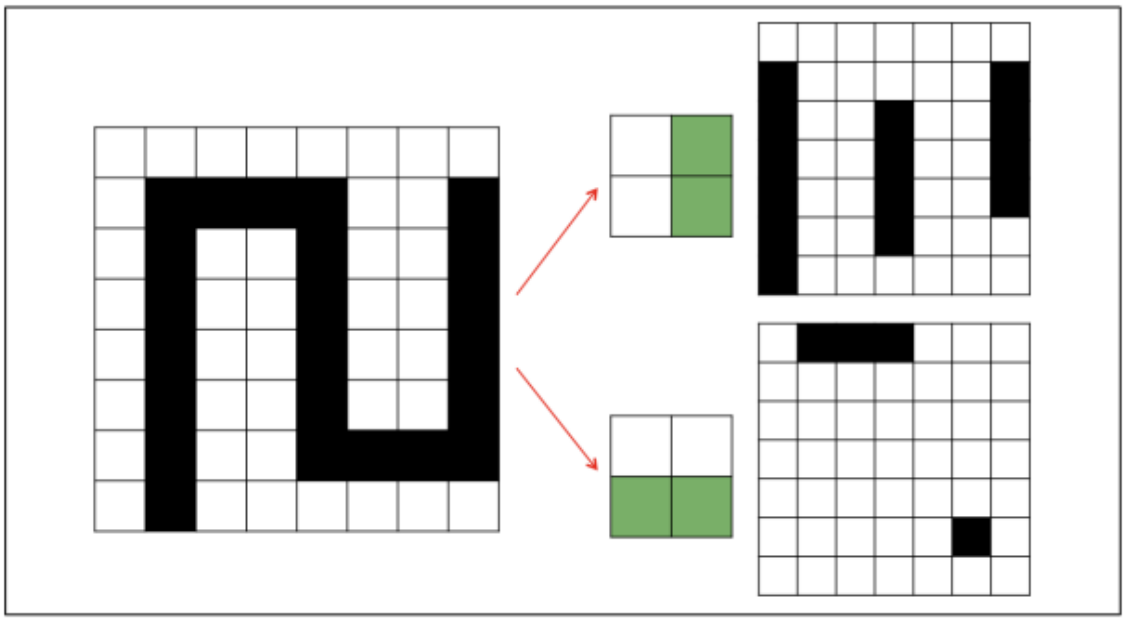

Figura 9 - Exemplo de aplicação de filtro para detecção de linhas horizontais e verticais.

Fonte: (Buduma, 2017)

2.2.2 Convolução

Uma convolução é uma integral que expressa a quantidade de sobreposição de uma função *g* conforme ela é deslocada sobre outra função *f* que é definida pela Equação 7 em um domínio [0, t] (Weissten, 2021).

$$[f * g] = \int_o^t f(\tau)g(t-\tau)d\tau \qquad (7)$$

Representado pela Figura 10 na forma de uma rede neural Feedfoward, a primeira camada é a imagem de entrada e a segunda são as matrizes de saída das operações realizadas pelos filtros (Feature Map), onde um neurônio é ativado caso o filtro conectado a ele detecte a característica especificada. Os filtros são representados pelas combinações das conexões que possuem pesos diferentes de acordo com sua cor (Buduma, 2017).

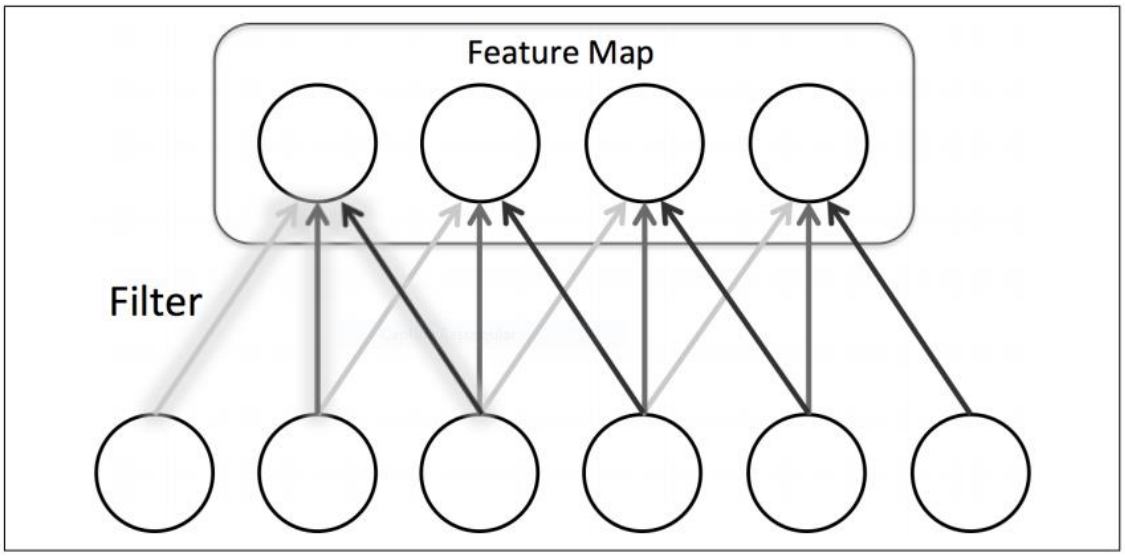

Figura 10 - Representação de filtros e feature map em uma rede neural convolucional.

Fonte: (Buduma, 2017)

Em uma rede neural convolucional os filtros podem operar em mais de um Feature Map na mesma camada. Sendo m_ko k-ésimo Feature Map da camada *m*, *W* o peso do filtro correspondente e b_k o bias, podemos representar o Feature Map com a Equação 8:

$$m_{ij}^{k} = f\left((W^{*}x)_{ij} + b^{k}\right) \tag{8}$$

Quando aplicados em CNNs os filtros operam em todo o conjunto (volume) de Feature Maps gerados em uma camada anterior. Por exemplo, em uma camada em que se deseja detectar uma face temos um para os olhos, um para o nariz e outro para a boca. Sabemos que a imagem contém uma face se são encontrados dois olhos, um nariz e uma boca na mesma região, ou seja, combinando a informação de três Feature Maps. Também são necessárias três Feature Maps para realizar convoluções em imagens coloridas, essas possuem três pixels (vermelho, verde e azul) que se misturam para formar a cor final de cada pixel. Como representado na Figura 11, uma porção da imagem de entrada em que cada pixel representa um neurônio, é multiplicada por um filtro de mesmo volume resultando em um neurônio da próxima camada de neurônios (Buduma, 2017).

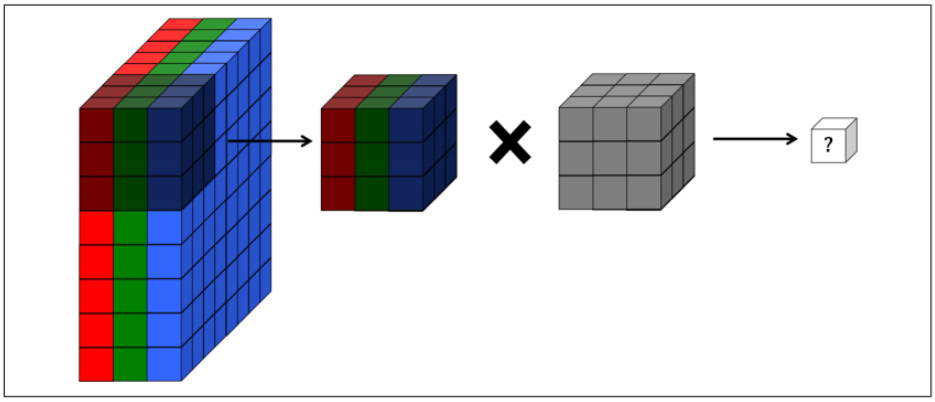

Figura 11 - Exemplo de aplicação de um filtro em imagem RGB.

Fonte: (Buduma, 2017)

Uma camada de convolução consiste em um conjunto de filtros que aplicados a entrada resultam em um volume de saída equivalente ao número de filtros utilizados. Como pode-se observar na Figura 12, a convolução de cada filtro com a entrada resulta em uma fatia da saída (Buduma, 2017).

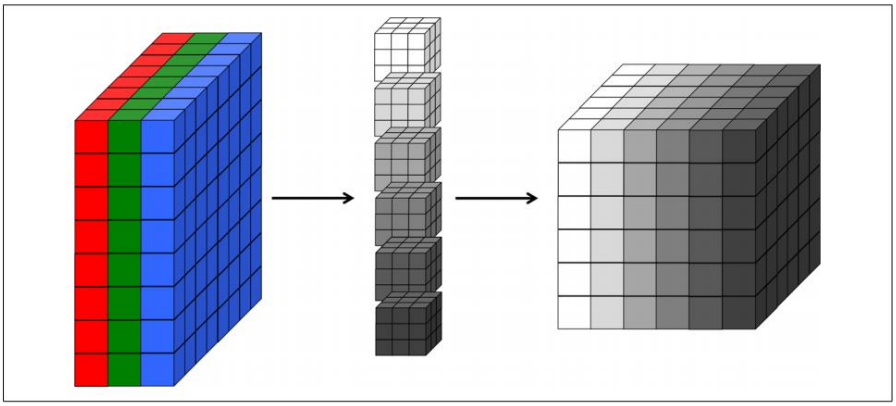

Figura 12 - Visualização de uma camada de convolução tridimensional onde cada filtro corresponde a uma dimensão do volume de saída.

Fonte: (Buduma, 2017)

Segundo (Hashemi, 2019), as imagens de entrada de uma rede devem possuir a mesma largura w_{in}, altura h_{in}, e profundidade d_{in}. Como em uma amostra de imagens em um banco de dados geralmente encontram-se imagens de largura e altura diferentes, utiliza-se uma técnica para padronizar o tamanho de entrada das imagens chama-se *Zero-Padding*. Esta consiste em acrescentar às imagens menores que o determinado como entrada da rede uma borda de pixels de valor zero e largura *p*, proporcional ao necessário

para atingir o padrão. Na Figura 13 temos um exemplo onde a entrada possui *Zero-Padding* p igual a 1.

As entradas são processadas por todos os k filtros da camada. Estes são definidos por sua extensão espacial e, que corresponde à altura e largura, pelo stride s, que é a distância entre as aplicações do filtro na entrada conforme demonstrado na Figura 13, e pelo bias b que é adicionado a cada componente da convolução.

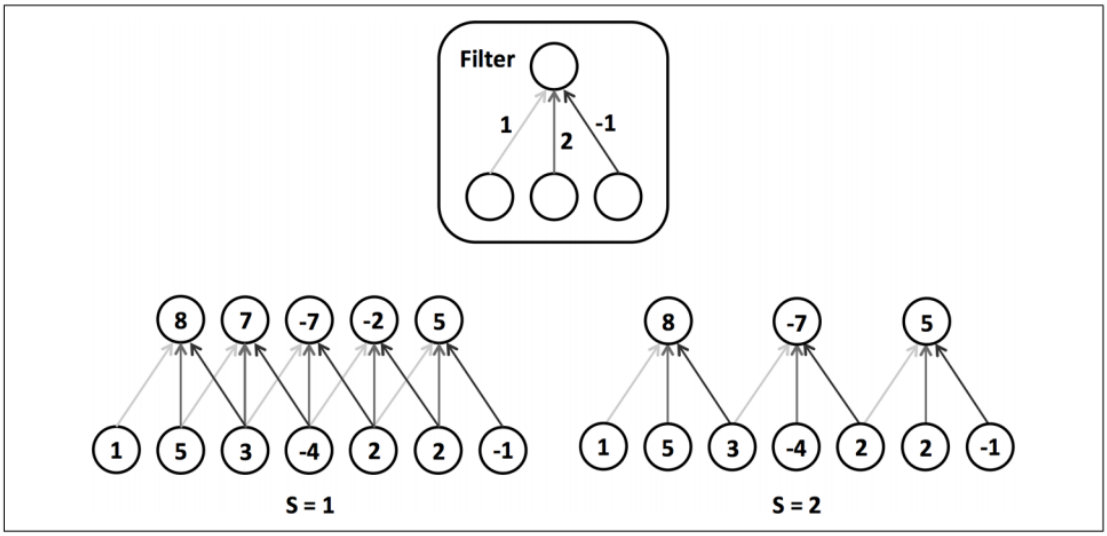

Figura 13 - Representação em forma de neurônios de filtros com stride $s = 1$ e $s = 2$.

Fonte: (Hashemi, 2019)

Utilizando os parâmetros apresentados acima e a função de ativação dos neurônios, podemos determinar as dimensões da matriz de saída por meio das equações 9, 10 e 11 citadas por (Buduma, 2017):

- Largura: $$w_{out} = \left\lceil \frac{w_{in} - e + 2p}{s} \right\rceil + 1 \quad (9)$$

- Altura: $$h_{out} = \left\lceil \frac{h_{in} - e + 2p}{s} \right\rceil + 1 \quad (10)$$

- Profundidade: $$d_{out} = k \quad (11)$$

A m-ésima fatia do volume de saída, sendo $1 \leq m \leq k$, corresponde a função de ativação aplicada ao m-ésimo filtro aplicado ao volume de saída e somado ao bias b_m. Desta forma conclui-se que existem $d_{in} \times e^2$ parâmetros por filtro, $k \times d_{in} \times e^2$ parâmetros e *k* bias por camada. Nas figuras 14 e 15 apresenta-se um exemplo de camada convolucional de entrada $5 \times 5 \times 3$, zero-padding $p = 1$, dois filtros $3 \times 3 \times 3$ com stride $s = 2$ e função de ativação linear, sendo assim a dimensão da saída $3 \times 3 \times 2$. Nelas encontra-se um exemplo de camada de convolução com volume de entrada de 5 de largura, 5 de altura, 3 de profundidade e zero padding 1. Este encontra-se exposto a 2 filtros 3×3 e stride $s = 2$. O volume de saída possui 3 de largura, 3 de altura e 2 de profundidade. A primeira peça superior esquerda da primeira camada de saída é resultado do primeiro filtro aplicado ao primeiro conjunto de dimensão 3×3 em cada uma das camadas de entrada.

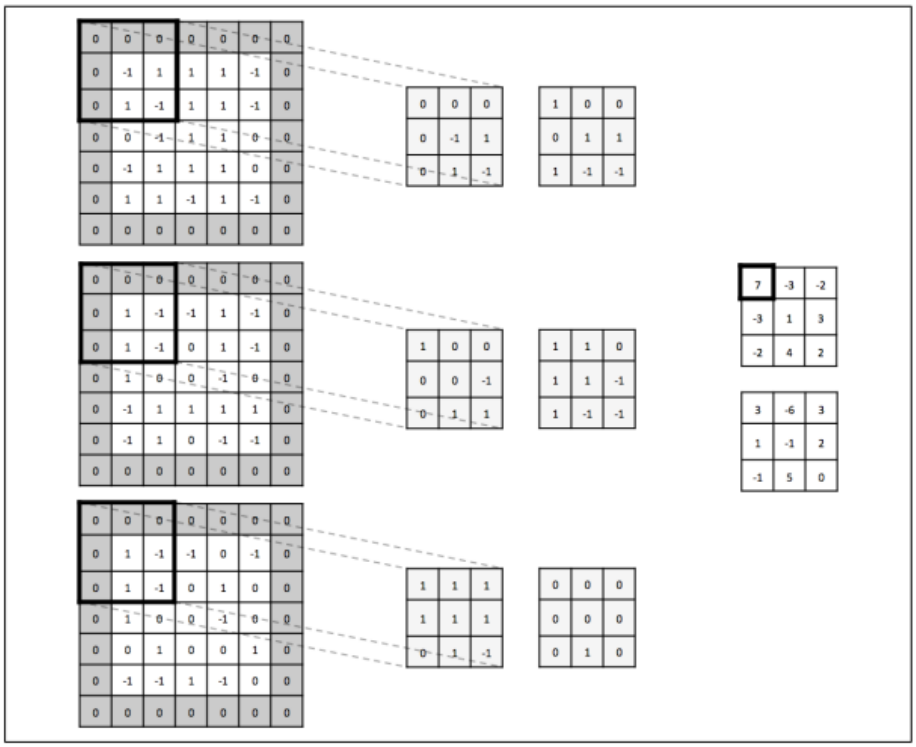

Figura 14 - Representação da segunda operação da convolução.

Fonte: (Buduma, 2017)

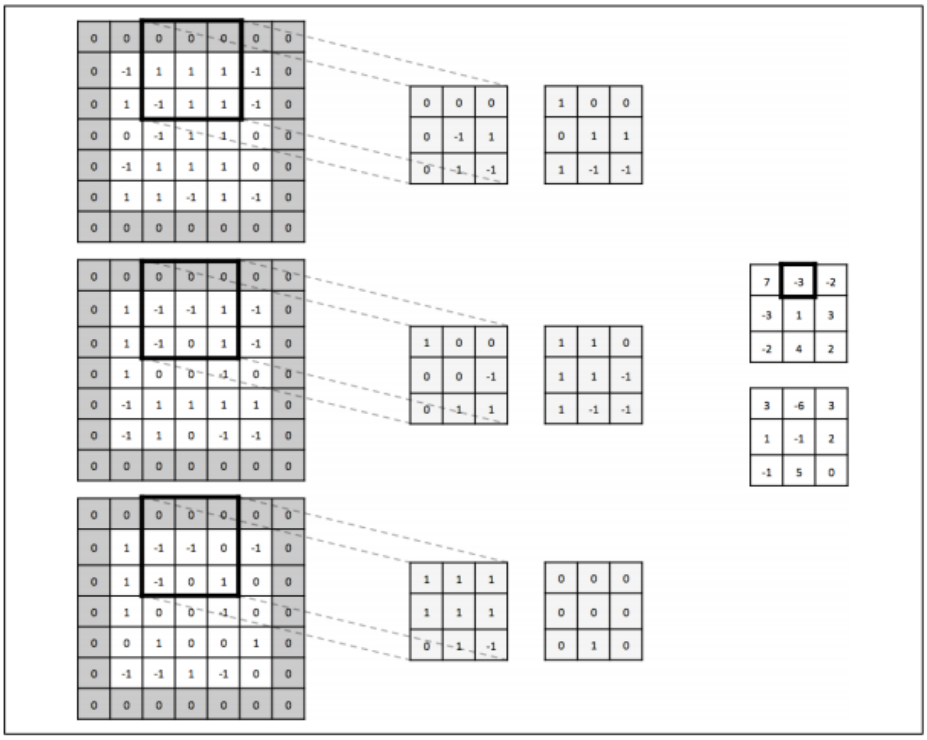

Figura 15 - Representação da segunda operação da convolução realizada na mesma imagem da Figura 14.

Fonte: (Buduma, 2017)

Segundo Buduma (2017), os filtros comumente utilizados possuem tamanho 3×3 ou 5×5. De forma menos frequente, utiliza-se filtros maiores de tamanho 7×7 na primeira camada. O motivo é que filtros menores, em camadas com stride $s = 1$, conseguem alcançar boa representação utilizando menos parâmetros.

2.2.3 Pooling

A camada de convolução é comumente acompanhada de uma camada de agrupamento ou *Pooling*. Esta tem como objetivo reduzir a dimensão do Feature map, sendo chamada de Max Pooling quando a função aplicada na camada divide a imagem em tamanhos iguais e extrai o maior valor de cada pedaço criando um novo Feature Map como ilustrado na Figura 16.

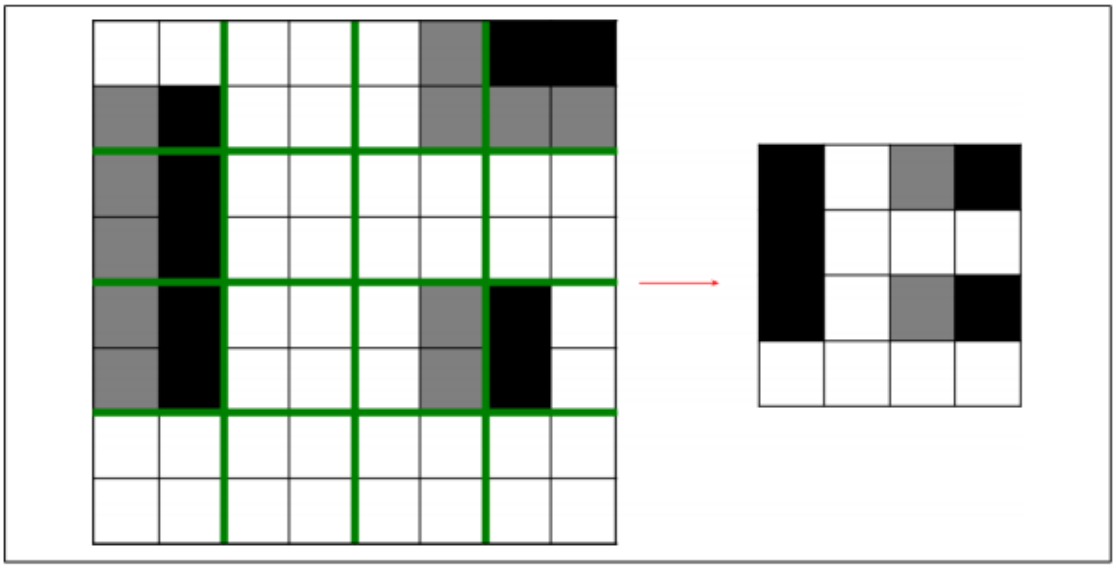

Figura 16 - Representação da redução de uma imagem efetuada por meio de uma operação de Max Pooling.

Fonte: (Buduma, 2017)

Conforme citado por (Buduma, 2017), podemos também descrever a camada de Pooling utilizando sua extensão espacial e e seu stride s, que aplicados nas equações 12 e 13 resultam na largura e altura da camada:

- Largura: $$w_{out} = \left\lceil \frac{w_{in} - e}{s} \right\rceil + 1 \quad (12)$$

- Altura: $$h_{out} = \left\lceil \frac{h_{in} - e}{s} \right\rceil + 1 \quad (13)$$

É importante ressaltar que somente duas variações da camada Pooling são utilizadas, onde na primeira $e = 2$ e $s = 2$, e na segunda $e = 3$ e $s = 2$.

2.2.4 Aprendizado Residual

Segundo o artigo Deep Residual Learning for Image Recognition (kaiming He, 2016) redes convolucionais a cada camada de convolução o tamanho do feature map diminui, fazendo com que se alcance um limite para essa redução. É possível adicionar novas camadas e continuar efetuando convoluções sem diminuir a dimensão, utilizando padding com stride igual a 1. Porém, adicionar camadas a uma rede neural pode introduzir

um problema de degradação que diminui a performance da rede. Quando tenta-se inserir um bloco a um modelo já suficiente o novo bloco resulta em uma cópia do bloco anterior, o que pela degradação não traz evolução no aprendizado no novo bloco idêntico, gerando custos de processamento.

Aprendizado residual são blocos introduzidos a redes neurais de múltiplas camadas com a finalidade de evitar a perda de acurácia que ocorre devido a grande quantidade de camas. Proposto pela equipe da Microsoft (kaiming He, 2016), ao invés de somente acrescentar um novo bloco de mesma dimensão utiliza-se passar a informação da última camada além da nova convolução e adicionar o valor original transportado à saída da convolução como é possível visualizar na Figura 17.

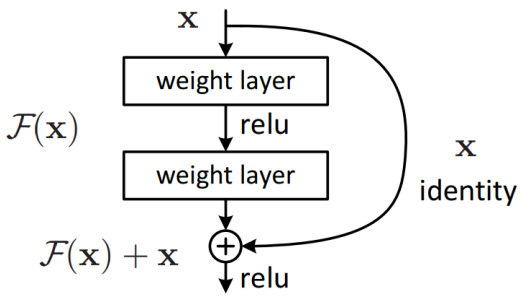

Figura 17 – Aprendizado residual.

Fonte: Deep Residual Learning for Image Recognition (kaiming He, 2016).

2.3 MobileNetV2

RNAs vêm revolucionando muitas áreas da inteligência artificial proporcionando precisão sobre-humana aos computadores, contudo, esse desenvolvimento possui custo computacional além da capacidade de dispositivos móveis e embarcados. A MobileNetV2 é uma arquitetura de rede neural convolucional mobile que tem como objetivo, atingir o estado da arte de desempenho em aplicações desenvolvidas para dispositivos móveis com processamento reduzido. Esta atua de forma que, comparada às redes convolucionais tradicionais, reduz significativamente o número de operações e memória necessária mantendo-se a acurácia (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018).

A principal diferença da MobileNetV2 em relação às redes tradicionais é um novo módulo de camadas chamadas convolução em profundidade e residual invertido. Este módulo recebe como entrada uma representação comprimida (com poucas dimensões)

que é expandida a dimensões maiores para passar por uma convolução em profundidade utilizando filtros com pesos reduzidos. A saída é então redimensionada a poucas dimensões por uma convolução linear. Esse procedimento será detalhado baseado no artigo MobileNetV2: Inverted Residuals and Linear Bottlenecks (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018).

2.3.1 Convolução em profundidade

A ideia deste tipo de convolução é separar a convolução tradicional em duas camadas como representado na Figura 18. A primeira camada, chamada de convolução em profundidade, aplica um filtro de convolução para cada canal (dimensão) de entrada. A segunda, chamada de convolução 1×1 ou ponto a ponto, constrói novas informações por meio de combinações lineares das entradas (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018).

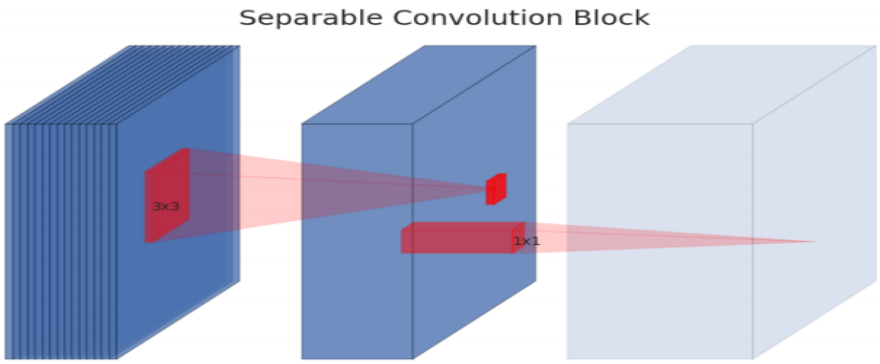

Figura 18 - Representação de uma convolução em profundidade.

Fonte: (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018)

Segundo o artigo MobileNetV2: Inverted Residuals and Linear Bottlenecks (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018), uma convolução tradicional tem seu custo representado pela Equação 14, que é diferente do de uma convolução em profundidade representado pela Equação 15. Nessas h_i representa a altura da entrada, w_i a largura de entrada e d_i a dimensão de entrada. A dimensão da saída é representada por d_j, ou seja, a quantidade de filtros de dimensão $k \times k$ utilizados.

$$h_i \times w_i \times d_i \times d_j \times k \times k \tag{14}$$

$$h_i \times w_i \times d_i\left(k^2 + d_j\right) \quad (15)$$

Por meio das equações conclui-se que uma convolução em profundidade reduz em k^2 o número de operações realizadas. Sendo em uma MobileNetV2 $k = 3$, temos que seu custo é de 8 a 9 vezes menor comparado ao de uma convolução tradicional.

2.3.2 Residual Invertido

Esta camada recebe como entrada Feature Maps de pequena dimensão que são expandidos por meio de uma convolução 1×1. Em seguida opera-se uma convolução em profundidade utilizando filtros de dimensão 3×3 e função de ativação ReLU6, a saída da segunda convolução é novamente reduzida a poucas dimensões por meio de uma convolução ponto a ponto. No Quadro 1 encontra-se um resumo dos passos executados neste bloco.

Entrada	**Operador**	**Saída**
$h \times w \times k$	1x1 conv2d, RelU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, RelU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	Linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Quadro 1 - Bloco de residual invertido (bottleneck) com função de transformar uma entrada de dimensão h x w e k canais em k' canais sendo s o fator de transposição e t o fator de expansão que representa a proporção em que a quantidade de canais será alterada.

Fonte: (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018).

A arquitetura MobileNetV2 contém 32 filtros seguidos de 19 blocos bottleneck (residual invertido) conforme detalhado no Quadro 2 onde t é o fator de expansão, c a quantidade de canais de saída, n o número de repetições da camada, e s o valor do stride. (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018).

Entrada	**Operador**	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	1280	1	1
$7^2 \times 1280$	avgpool 7×7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1×1	-	k	-	

Quadro 2 - Arquitetura MobileNetV2.

Fonte: (Sandler, Andrew, Menglong, Adrey, & Liang-Chieh, 2018).

2.4 Keras

Keras (Manaswi, 2018) é uma biblioteca escrita em Python (Manaswi, 2018), compacta e usada para o aprendizado de máquina de múltiplas camadas com o auxílio do TensorFlow (Manaswi, 2018). Essa contém dois frameworks principais: API sequencial e API funcional. Com ela é possível otimizar sua rede adicionando camadas com diversas funções de acordo com as necessidades de aprendizado, podendo conter convoluções, pooling e funções de ativação e normalização. O principal motivo de utilizá-la é permitir ao desenvolvedor focar nos conceitos principais de aprendizado profundo e na criação das camadas da rede (Manaswi, 2018).

Segundo Manaswi (2018), para se criar um modelo de aprendizado profundo são necessários quatro passos principais:

1. Definir o modelo: cria-se um modelo definindo-se a sequência de camadas da rede, podendo ser camadas de convolução, pooling ou de ativação. O número de camadas e de neurônios em cada camada é definido empiricamente por meio da experiência do

desenvolvedor e tentativa e erro. Os pesos e bias da rede são inicializados com valores aleatórios pequenos de 0 a 1 por meio de distribuições definidas no Keras;

2. Compilar o modelo: Tendo definido o modelo em termos de camadas é necessário declarar as funções de perda e otimização antes que a rede seja compilada. Os pesos e bias aleatórios utilizados para criar a rede não são capazes de alcançar um nível aceitável de acerto nas previsões, dessa forma, para encontrar os valores ideais durante o procedimento de treinamento, esses valores são alterados de acordo com a função de otimização com o objetivo de minimizar a função de perda. No projeto desenvolvido neste trabalho utiliza-se a função *Binary crossentropy* como função de perda e as funções *Adam* e *SGD* como funções de otimização.

2.1. Binary crossentropy (Rebecka Hogevall, 2021) é uma função de perda utilizada em tarefas de classificação binária, ou seja, que possuem duas possíveis respostas como o uso de máscara ou não. Esta pode ser descrita pela Equação 15, onde y_i é o valor previsto e t_i o valor esperado.

$$Loss = \sum_{i=1}^{2} -t_i log\,(y_i)\; -(1-t_i)\,log\,(1-y_i) \tag{16}$$

2.2. Adam é uma função de otimização baseada em modelo estocástico de fácil implementação, computacionalmente eficiente e de baixo custo de memória. No artigo ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION pode-se ter acesso a seu algoritmo e explicação detalhada de seus hiperparâmetros (Kingma et al., 2014).

2.3. SGD (Stochastic Gradient Descent) é uma função de otimização em que, dado um conjunto de dados $D = (z_i)_{i=1}^{n}$, um lote mínimo $d_t \subset D$, um parâmetro de aprendizado $\hat{x}_t \in R^l$ de f na interação t, e taxa de aprendizado η, possui regra de atualização representada pela Equação 16.

$$\hat{x}_{t+1} = \hat{x}_t - \eta \nabla f(d_t; \hat{x}_t) \tag{17}$$

3. Introduzir os dados de treinamento no modelo: Após o modelo compilado deve-se realizar previsões utilizando exemplos de imagens. Neste passo é necessário especificar dois parâmetros que serão utilizados no processo de treinamento: as Épocas, que são o número de iterações com as imagens do banco de dados, e o tamanho do lote (*Bach size*), quantidade de imagens a serem expostas à rede antes de se atualizar os pesos e Bias. Esses também devem ser estabelecidos de maneira empírica utilizando tentativa e erro.

4. Fazer previsões: Uma vez compilado e treinado o modelo deve-se aplicar imagens desconhecidas ao modelo, essas serão trabalhadas pelas camadas da rede resultando em um valor de saída. Neste trabalho são expostas à rede imagens de pessoas, obtendo-se como saída "*Mask"* para pessoas que são detectadas utilizando máscara e "*No mask"* para pessoas que não utilizam máscara nas imagens.

2.5 TensorFlow

Um dos motivos da evolução do aprendizado de máquina nos últimos anos deve-se ao desenvolvimento de plataformas de software que facilitam o uso de grande recurso computacional para treinamento de modelos utilizando grande volume de dados (Martín Abadi, 2016).

O TensorFlow é um sistema desenvolvido pela equipe Google Brain para o aprendizado em larga escala, utilizando gráficos de fluxo de dados para representar o cálculo, o estado compartilhado, e as operações que alteram o estado. Esse tem a capacidade de mapear os nós dos gráficos de vários processadores que trabalham simultaneamente dando flexibilidade e possibilidade de otimização para o desenvolvedor (Martín Abadi, 2016).

Este sistema possibilita o uso de centenas de servidores para treinamento rápido sendo executável em várias plataformas variando de grandes grupos de dados em datacenters à dispositivos mobile. Ao mesmo tempo é flexível para executar experimentos e pesquisas de novos modelos de aprendizado e otimizações (Martín Abadi, 2016).

2.6 Scikit-Learn

A biblioteca Scikit-Learn Fabian Pedregosa (2011) contém uma grande variedade de algoritmos de aprendizado de máquina de fácil utilização em linguagem Python, que

se consolida como a linguagem mais popular na computação científica devido seu alto nível de interatividade e seu estruturado ecossistema de bibliotecas. Muito utilizada no meio acadêmico e industrial, possui um arsenal que proporciona implementações no estado da arte, correspondendo com as necessidades de análises estatísticas de dados realizadas por estudantes de todas as áreas (Fabian Pedregosa, 2011).

Segundo Fabian Pedregosa (2011), encontram-se quatro diferenças entre o Scikit-learn e outras bibliotecas em Python, são elas:

- Possui licença de código aberto BSD;
- Incorpora código compilado para eficiência;
- Depende somente das bibliotecas NumPy e Scipy;
- Possui foco em programação imperativa, paradigma em que comandos alteram o estado das variáveis.

2.7 NumPy

Para executar os cálculos com vetores multidimensionais é utilizada a biblioteca NumPy (Oliphant, 2006), principal pilar existente para computação científica devido ao seu grande conjunto de operações e funções. Desenvolvida pela necessidade da utilização de funções em linguagem C e Fortran não contidas na biblioteca Numeric Python, a SciPy, codificada por contribuição de vários cientistas, objetivou fornecer em linguagem Python ferramentas para entradas e saídas robustas, funções especiais, integração, resolução de equações diferenciais ordinárias, otimização de mínimos quadrados e resolução de matrizes esparsas (Oliphant, 2006).

Em 2005 com a SciPy ainda em desenvolvimento foi criada a Numarray, novamente utilizando Python e C, para suprir necessidades como mapeamento de memória e formas de verificação de erros. A nova biblioteca, apesar de acrescentar ferramentas limitadas ou ainda não existentes na SciPy, tornou lenta a interação com funções complexas da Numeric, o que fragmentou a comunidade e causou rejeição à novos usuários do Python. Para promover novamente a interação foi desenvolvida a NumPy, que conta com infraestrutura para rotinas básicas de álgebra linear, transformada de Fourier e gerador de números aleatórios (Oliphant, 2006).

2.8 OpenCV

Visão computacional é o campo de análise, modificação e interpretação de imagens a fim de proporcionar um melhor entendimento para os humanos. OpenCV (Krupali Mistry, 2016) é uma biblioteca aberta para análise de imagem e vídeo desenvolvida pela Intel disponibilizada nas linguagens C, C++, Java e Python, desenvolvida com foco em eficiência computacional e aplicações em tempo real. Sua primeira versão foi em linguagem C, tornando popular com a escrita em C++. Segundo Krupali Mistry (2016) são enumeradas as seguintes características:

- APIs de baixo e alto nível;
- Manipulação de dados de imagens (alocação, liberação, cópia, configuração, conversão);
- Entrada e saída de imagem e vídeo;
- Manipulação de matrizes e vetores e rotinas de álgebra linear;
- Dinâmicas de estrutura de dados (listas, filas, conjuntos, árvores e gráficos);
- Processamento básico de imagens (filtros, detecção de borda, detecção de quina, interpolação, conversão de cores, operações morfológicas, histogramas e pirâmide de imagens);
- Análise estrutural (componentes conectados, processamento de contorno, transformação de distância, correspondência de modelo, aproximação poligonal, ajuste de linha);
- Calibração de câmera;
- Análise de movimento;
- Reconhecimento de objetos.

2.9 Matplotlib

Os resultados de treinamento da rede neural são demonstrados em gráficos plotados utilizando o Matplotlib (Hunter, 2007). Trata-se de uma ferramenta desenvolvida para gerar gráficos 2D em aplicações Python e substituir funções antes executadas no software Matlab. Essa contém um ambiente de emulação do Matlab chamado PyLab que facilita a utilização por estudantes com conhecimentos básicos sobre programação. Além de simples gráficos de linha, a ferramenta permite de forma simples a criação de gráficos sofisticados e imagens com mapeamento de cores. Outra facilidade

é a possibilidade de criação de resposta a eventos como o clique do mouse ou tecla pressionada (Hunter, 2007).

2.10 VMware

Com a finalidade de proporcionar maior confiabilidade, isolamento e escalabilidade, o sistema deste trabalho foi executado em ambiente virtual criado por meio do aplicativo VMware (Mattos, 2008). Esse foi formatado com a versão 20.04 do sistema operacional Ubuntu, onde foram instaladas todas as ferramentas utilizadas neste trabalho (Mattos, 2008).

3 METODOLOGIA

Neste capítulo serão descritos todos os passos seguidos para o desenvolvimento deste trabalho. Com o objetivo de testar o comportamento da rede em diferentes configurações, foram executados treinamentos utilizando diferentes hiperparâmetros para comparação dos resultados no próximo capítulo. A rede foi treinada com 8, 15 e 20 épocas, utilizando para otimização as funções adam e sgd. Os resultados de cada função que apresentaram os maiores valores de Precisão, Recall e F1 Score foram treinados novamente inserindo a função channel_shift_range no data augmentation. Esta proporciona uma maior generalização ao modelo inserindo dados para treinamento com valores alterados aleatórios dentro do range especificado para cada canal de entrada, alterando as cores das imagens. Para testar a eficiência dos treinamentos, as imagens a serem classificadas pelo uso de máscara foram expostas ao sistema com maiores resultados de Precisão, Recall e F1 Score e que tenha sido treinado utilizando menos épocas. Avaliar um modelo por meio dos dados de treinamento ou de validação requer a definição de métricas de classificação que serão definidas a seguir.

3.1 Métricas

3.1.1 Acurácia

A acurácia é definida como o cálculo da fração da amostra que foi classificada corretamente. Dado o conjunto $X' = \{(x_1, y_1), \dots, (x_N, y_N)\}$ contendo *N* pares de amostras, o valor de classificação é calculado pela Equação 18 em que y_i e $\widehat{y_I}$ são a entrada e saída da i^{th} amostra em X' respectivamente, e w_i o peso da amostra que é calculado pela Equação 19 (Hamed Habibi Aghdam, 2017).

$$acurácia = \frac{1}{N}\sum_{i=1}^{N} \quad w_i \times 1[y_i == \widehat{y_i}] \tag{18}$$

$$\frac{1}{C \times quantidade\ de\ amostras\ na\ classe\ A} \tag{19}$$

Na função 18, o fator $1[y_i == \widehat{y_i}]$ retorna 1 caso o valor do argumento for verdadeiro e 0 caso for falso. O valor da acurácia varia no conjunto $[0,1]$. Assim caso seja igual a 1 significa que a amostra de X' foi classificada corretamente. Ao contrário, se a acurácia for 0 significa que a amostra do conjunto X' foi classificada erroneamente (Hamed Habibi Aghdam, 2017).

3.1.2 Matriz de Confusão

Matriz de confusão é uma métrica utilizada para avaliar a precisão de modelos de classificação. Dado um problema que contenha C possíveis classes, uma matriz de confusão M é uma matriz $C \times C$ em que o elemento M_{ij} mostra a quantidade de amostras em X' de classe i classificados como j, e M_{ii} a quantidade de amostras classificadas corretamente.

Em um problema de classificação binária existem apenas duas classes possíveis. Desta forma a matriz confusão será uma matriz 2×2 em que o elemento M_{11} mostra a quantidade de amostras de classe 1 classificadas corretamente (TP), M_{12} as amostras de classe 1 classificadas como -1 (FN), M_{21} amostras de classe -1 classificadas como 1 (FP), e M_{22} amostras de classe -1 classificadas como –1 (TN) conforme representado na Figura 20 (Hamed Habibi Aghdam, 2017).

	1	-1
1	TP	FN
-1	FP	TN

Figura 19 - Matriz de confusão 2x2.

Fonte: Autor.

Com a matriz de confusão podemos calcular a acurácia utilizando a Equação 20 demonstrada abaixo:

$$acurácia = \frac{TP + TN}{TP + TN + FP + FN} \quad (20)$$

3.1.3 Precisão e Recall

Precisão e Recall (Hamed Habibi Aghdam, 2017) são duas medidas quantitativas importantes para avaliar um classificador. Precisão calcula a fração de amostras previstas positivas e Recall a fração de amostras realmente positivas conforme demonstrado nas equações 21 e 22.

$$precisão = \frac{TP}{TP + FP} \quad (21)$$

$$recall = \frac{TP}{TP + FN} \quad (22)$$

Em um classificador perfeito FP e FN devem ser zero para que precisão e recall sejam 1, neste caso podemos dizer que é um classificador perfeito. Caso precisão e recall sejam próximos de zero significa que o classificador é impreciso (Hamed Habibi Aghdam, 2017).

3.1.4 F1 Score

F1 Score (Hamed Habibi Aghdam, 2017) é uma forma eficiente de avaliar uma CNN utilizando a combinação de precisão e recall conforme definido pela Equação 23. O valor de $F1$ varia em $[0,1]$ sendo 1 o resultado de um classificador perfeito.

$$F1 = \frac{2}{\frac{1}{precisão} + \frac{1}{recall}} = \frac{2TP}{2TP + FP + FN} \quad (23)$$

3.1.5 Macro-average

O Macro-average (S. Vani, 2019) é uma medida de performance que pode ser calculada por uma das formas definidas abaixo:

- Por meio da precisão (S. Vani, 2019) o Macro-average é calculado como a média dos valores de precisão de cada classe como demonstrado pela Equação 24, onde n é a quantidade de classes e P_i é a precisão da classe i.

$$Macro - average\ da\ Precisão = \frac{\sum_{i=1}^{n} P_i}{n} \quad (24)$$

- Por meio do Recall (S. Vani, 2019) o Macro-average é calculado como a média dos valores de recall calculados para cada classe como demonstrado pela Equação 25, onde n é a quantidade de classes e R_i é a precisão da classe i.

$$Macro - average\ da\ Recall = \frac{\sum_{i=1}^{n} R_i}{n} \quad (25)$$

- Por meio do F1 Score (S. Vani, 2019) o Macro-average M_h é calculado utilizando a média harmônica do Macro-average do Recall $\underline{R}$ e da Precisão como representado na Equação 26.

$$Macro - average\ de\ F1\ Score = \frac{\sum_{i=1}^{n} F1_i}{n} \quad (26)$$

3.1.6 Weighted-average

O cálculo do Weighted-average (S. Vani, 2019) por meio do F1 Score é definido pela Equação 27 em que F_i é o F1 Score calculado para a classe i em n classes, e N_i a quantidade de instâncias da classe i. O Weighted-average é também calculado por meio da precisão e recall de forma semelhante.

$$Weighted - average\ de\ F1\ Score = \frac{\sum_{i=1}^{n} F_i * N_i}{N} \quad (27)$$

3.2 Banco de dados

O banco de dados utilizado neste trabalho é composto por 1.386 imagens, sendo que contém 686 imagens de pessoas sem máscara (Figura 20) extraídas do banco de dados disponibilizado por Prajnasb (2020), e 700 imagens extraídas do Google de pessoas utilizando diferentes tipos de máscaras reais (Figura 21). Além das imagens utilizadas

para treinamento, foram classificadas pelo sistema 140 novas imagens (70 de pessoas com máscara e 70 de pessoas sem máscara), essas também extraídas do Google.

Figura 20 - Exemplo de imagem de pessoa sem máscara.

Fonte: (Prajnasb, 2020)

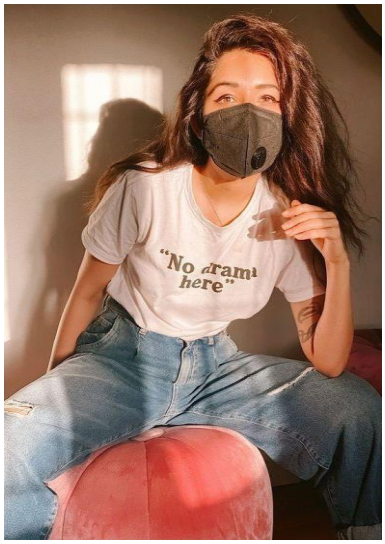

Figura 21 - Exemplo de imagem com máscara real.

Fonte: Google Imagens.

3.3 Treinamento da Rede

O treinamento da rede MobileNetV2 é executado pelo script *train_mask_detector.py* (Apêndice A), este importa ferramentas incluídas nas bibliotecas Keras, TensorFlow e Scikit-learn que são base para a funcionamento do código. Para executar o script de treinamento, apontando-se para a pasta raiz do projeto, utiliza-se o seguinte comando no console nativo do Ubuntu: *$ python3 train_mask_detector.py --dataset dataset* (Rosebrock, Pyimagesearch, 2020).

O detalhamento da rede é realizado através desse script, onde são especificados a taxa inicial de aprendizado como 1e-4, o número de épocas de treinamento, que irá variar entre 8, 15 e 20, e tamanho de lote 32. Após são inicializados os vetores de imagens e labels que serão utilizados para efetuar os cálculos (Rosebrock, Pyimagesearch, 2020).

É necessário que todas as imagens sejam padronizadas. Para isso é realizado um loop para que todas as imagens sejam enquadradas em 224x224 pixels e convertidas para formato de matriz onde cada pixel está na faixa de -1 a 1. Para finalizar, verifica-se se os vetores de dados então no formato NumPy, após os elementos do vetor *labels* passam pela transformação *one-hote* que transforma cada elemento em um novo vetor binário onde apenas um elemento é 1 (Rosebrock, Pyimagesearch, 2020).

Na construção de um modelo de aprendizado é importante a divisão dos dados em dados de treinamento e de validação, para que o modelo seja treinado e avaliado quanto a seu potencial de generalização. O que ocorre é que após um dado influenciar nos pesos dos neurônios na fase de treinamento, torna-se impossível saber o desempenho de classificação da rede utilizando esse mesmo dado, já que a rede se encontra adaptada para a classificação deste.

Utilizando o método *train_test_split* do Scikit-learn são separados 80% dos dados para aprendizado e 20% para validação. Em seguida utiliza-se a função *ImagadeDataGenerator* para alterar a rotação, zoom, recorte, deslocamento e direção de cada imagem com o objetivo de aumentar a generalização do modelo (Rosebrock, Pyimagesearch, 2020). Essa função compõe as ferramentas do framework Keras que auxiliam na aplicação do conceito de *Data Augmentation*. Traduzido como aumento de dados, esse se refere às técnicas aplicadas ao conjunto de dados com a finalidade de aumentar a acurácia da rede por meio do aumento do número de dados sendo que os novos dados inseridos no treinamento são versões modificadas das próprias imagens originais

do grupo de treinamento. A quantidade de novas imagens acrescentadas é determinada de acordo com a Equação 28 (Tawsin Uddin Ahmed, 20019).

$$Imagens\ acrescentadas\ por\ época = \frac{Imagens\ no\ dataset}{Bath\ Size} \qquad (28)$$

Com o objetivo de preparar a MobileNetV2 para treinamento, são ajustados e fixados seus parâmetros de entrada, os pesos de treinamento pré-estabelecidos para imagens pelo ImageNet e os parâmetros do topo do modelo base (Rosebrock, Pyimagesearch, 2020).

Com os dados já preparados e a rede ajustada, os próximos passos são: compilar o modelo, executar o treinamento, fazer previsões utilizando as imagens de teste, imprimir o relatório de classificação e publicar o modelo de classificação em disco. Por último são plotadas as curvas de precisão e perda (Rosebrock, Pyimagesearch, 2020). Com o objetivo de escolher a melhor configuração da rede, serão realizados 9 treinamentos com configurações diferentes.

3.4 Detecção do uso de máscara em imagem

Para classificar as imagens com máscara e sem máscara será utilizado o script detect_mask_image.py (Apêndice B) que além do Keras e TensorFlow serão importadas ferramentas da biblioteca OpenCV. Para executar o script de classificação é executado o seguinte comando no console de comando nativo do sistema operacional na pasta raiz onde se encontra o projeto:
$ python3 detect_mask_image.py --image 0-with-mask (Rosebrock, Pyimagesearch, 2020).

Primeiro são carregados os modelos de detecção de face e de máscara. A imagem que será utilizada como entrada (0-with-mask) é carregada, copiada e reduzida à metade de sua escala, pré-processada e redimensionada para 300x300 pixels utilizando a ferramenta *blobFromImage* da biblioteca OpenCV, e utilizando o modelo de detecção facial são localizadas as faces na imagem (Rosebrock, Pyimagesearch, 2020).

Depois de verificado o fator de confiança, é extraída da imagem a região de interesse, convertida para o padrão RGB, redimensionada para 244x244 pixels e processada, e assim é previsto do uso de máscara (Rosebrock, Pyimagesearch, 2020).

Para apresentar o resultado para o usuário, é determinada a classe *label* de acordo com a probabilidade da região de interesse na imagem conter máscara ou não, associada a cor verde para com máscara e vermelha para sem máscara, é desenhado um retângulo ao redor da face, impresso o texto da classe *label* resultante, ambos na imagem e na cor da referida classe, e após isso, a imagem é mostrada para o usuário (Rosebrock, Pyimagesearch, 2020).

Foi implementado um sistema que, após obter o resultado da utilização ou não de máscara, envia uma mensagem de e-mail para o endereço cadastrado. Para teste do funcionamento são utilizadas como entrada a Figura 22, Figura 23 e Figura 24 capturadas pelo autor sem máscara e com máscara real.

Figura 22 - Fotografia do autor sem máscara.

Fonte: Autor.

Figura 23 - Fotografia do autor máscara branca.

Fonte: Autor.

Figura 24 - Fotografia do autor máscara azul.

Fonte: Autor.

4 RESULTADOS

Neste tópico serão apresentados os resultados obtidos após ser preparado o ambiente no aplicativo VMware, finalizado o treinamento da rede com as diferentes configurações pré estabelecidas e executado o código de detecção de máscara.

4.1 Resultado do treinamento da rede

Após a execução do comando *python detect_mask_image.py --image nome_da_imagem*, o treinamento da rede é iniciado sendo este executado de acordo com os parâmetros definidos pelo script detect_mask_image.py. A cada época do treinamento são impressos no console os valores de perda e precisão, os quais demonstram melhoria contínua aumentando a precisão e reduzindo a perda. Após o treinamento finalizado é impressa no console uma tabela contendo os resultados de Precisão, Recall, F1-Score da validação das imagens com e sem máscara, e também os resultados das métricas Acurácia, Macro-average e Weighted-average. Os tópicos a seguir contêm separadamente os resultados de cada configuração utilizada para treinamento.

4.1.1 Treinamento com 8 épocas e função de otimização *sgd* sem data augmentation

O treinamento executado com 8 épocas e função de otimização *sgd* iniciou apresentando perda de 0,7932 e acurácia de 0,5288 para os dados de treinamento e perda de 0,7222 e acurácia de 0,5271 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou perda de 0,7395 e acurácia de 0,5604 para os dados de treinamento, e perda de 0,6502 e acurácia de 0,6679 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 1.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 1 e podem ser interpretados como o de um sistema não preciso.

Tabela 1 - Resultado treinamento com 8 épocas e função de otimização sgd.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,63	0,83	0,72	140
Sem máscara	0,74	0,50	0,67	137
Acurácia			0,60	277
Macro-average	0,69	0,67	0,66	277
Weighted-average	0,69	0,67	0,66	277

Fonte: Autor.

Gráfico 1 - Resultado treinamento com 8 épocas e função de otimização sgd.

Fonte: Autor.

4.1.2 Treinamento com 8 Épocas e função de otimização *adam* sem data augmentation

O treinamento executado com 8 épocas e função de otimização *adam* iniciou apresentando perda de 0,5573 e acurácia de 0,7723 para os dados de treinamento e perda de 0,3054 e acurácia de 0,9711 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou, perda de 0,0710 e acurácia de 0,9823 para os dados de treinamento e perda de 0,0368 e acurácia de 0,9964 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 2.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 2 e podem ser interpretados como o de um sistema preciso.

Tabela 2 - Resultado treinamento com 8 épocas e função de otimização adam.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,99	1,00	1,00	140
Sem máscara	1,00	0,99	1,00	137
Acurácia			1,00	277
Macro-average	1,00	1,00	1,00	277
Weighted-average	1,00	1,00	1,00	277

Fonte: Autor.

Gráfico 2 - Resultado treinamento com 8 épocas e função de otimização adam.

Fonte: Autor.

4.1.3 Treinamento com 8 épocas, função de otimização *adam* com data augmentation

O treinamento executado com 8 épocas, função de otimização *adam* e função de data augmentation *channel_shift_range* iniciou apresentando perda de 0,5575 e acurácia de 0,7435 para os dados de treinamento e perda de 0,3183 e acurácia de 0,9531 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou perda de 0,0831 e acurácia de 0,9796 para os dados de treinamento, e perda de 0,0349 e acurácia de 0,9964 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 3.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 3 e podem ser interpretados como o de um sistema preciso.

Tabela 3 - Resultado treinamento com 8 Épocas, função de otimização adam e função channel_shift_range.

Métrica	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	1,00	0,99	1,00	140
Sem máscara	0,99	1,00	1,00	137
Acurácia			1,00	277
Macro-average	1,00	1,00	1,00	277
Weighted-average	1,00	1,00	1,00	277

Fonte: Autor.

Gráfico 3 - Resultado treinamento com 8 Épocas, função de otimização adam e função channel_shift_range.

Fonte: Autor.

4.1.4 Treinamento com 15 épocas e função de otimização *sgd* sem data augmentation

O treinamento executado em 15 épocas e função de otimização *sgd* iniciou apresentando perda de 0,8002 e acurácia de 0,5353 para os dados de treinamento e perda de 0,7060 e acurácia de 0,5632 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou, perda de 0,6622 e acurácia de 0,6664 para os dados de treinamento e perda de 0,5605 e acurácia de 0,8087 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 4.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average está apresentado na Tabela 4, estes podem ser interpretados como o de um sistema não preciso.

Tabela 4 - Resultado treinamento com 15 Épocas e função de otimização sgd.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,85	0,75	0,80	140
Sem máscara	0,77	0,87	0,82	137
Acurácia			0,81	277
Macro-average	0,81	0,81	0,81	277
Weighted-average	0,81	0,81	0,81	277

Fonte: Autor.

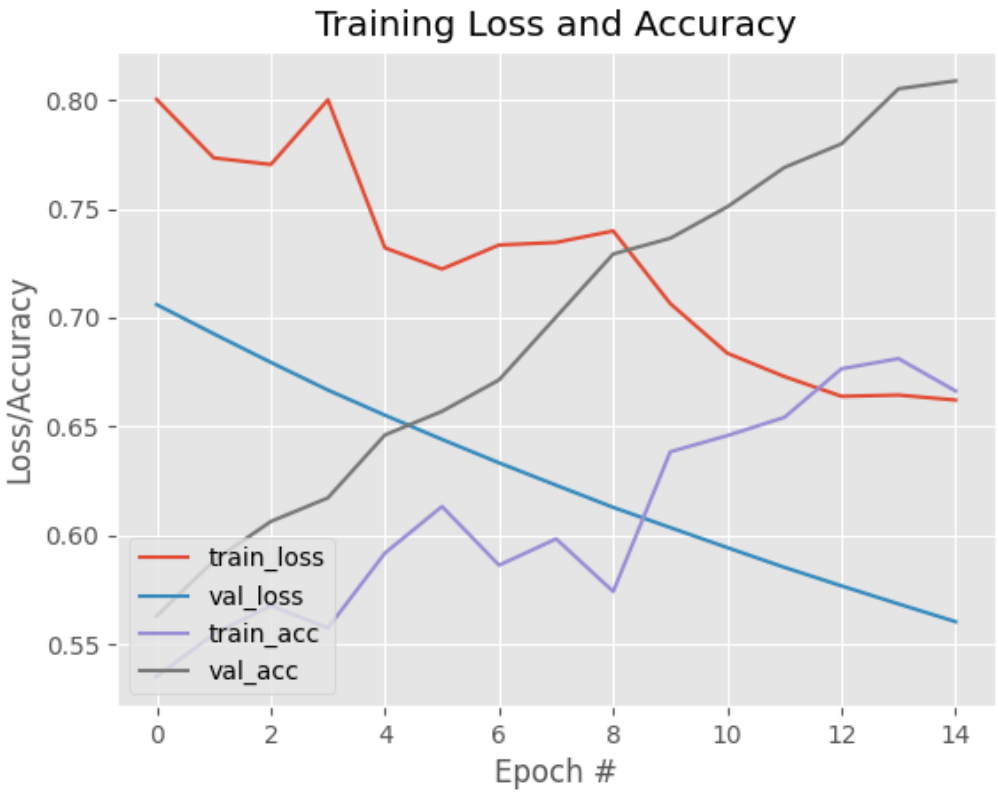

Gráfico 4 - Treinamento com 15 Épocas e função de otimização sgd.

Fonte: Autor.

4.1.5 Treinamento com 15 épocas e função de otimização *adam* sem data augmentation

O treinamento executado em 15 épocas e função de otimização *adam* iniciou apresentando perda de 0,5189 e acurácia de 0,7900 para os dados de treinamento e perda de 0,2577 e acurácia de 0,9747 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas a última apresentou, perda de 0,0293 e acurácia de 0,9926 para os dados de treinamento e perda de 0,0177 e acurácia de 0,9928 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 5.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 5 e podem ser interpretados como o de um sistema preciso.

Tabela 5 - Resultado treinamento com 15 épocas e função de otimização adam.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,99	1,00	0,99	140
Sem máscara	1,00	0,99	0,99	137
Acurácia			0,99	277
Macro-average	0,99	0,99	0,99	277
Weighted-average	0,99	0,99	0,99	277

Fonte: Autor.

Gráfico 5 - Resultado treinamento com 15 épocas e função de otimização adam.

Fonte: Autor.

4.1.6 Treinamento com 15 Épocas, função de otimização *adam* com data augmentation

O treinamento executado em 15 épocas, função de otimização *adam* e função de data augmentation *channel_shift_range* iniciou apresentando perda de 0,5720 e acurácia de 0,7546 para os dados de treinamento e perda de 0,3177 e acurácia de 0,9783 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou, perda de 0,0569 e acurácia de 0,9851 para os dados de treinamento e perda de 0,0303 e acurácia de 0,9928 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 6.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 6 e podem ser interpretados como o de um sistema preciso.

Tabela 6 - Resultado treinamento com 15 Épocas, função de otimização adam e função channel_shift_range.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,99	1,00	0,99	140
Sem máscara	1,00	0,99	0,99	137
Acurácia			0,99	277
Macro-average	0,99	0,99	0,99	277
Weighted-average	0,99	0,99	0,99	277

Fonte: Autor.

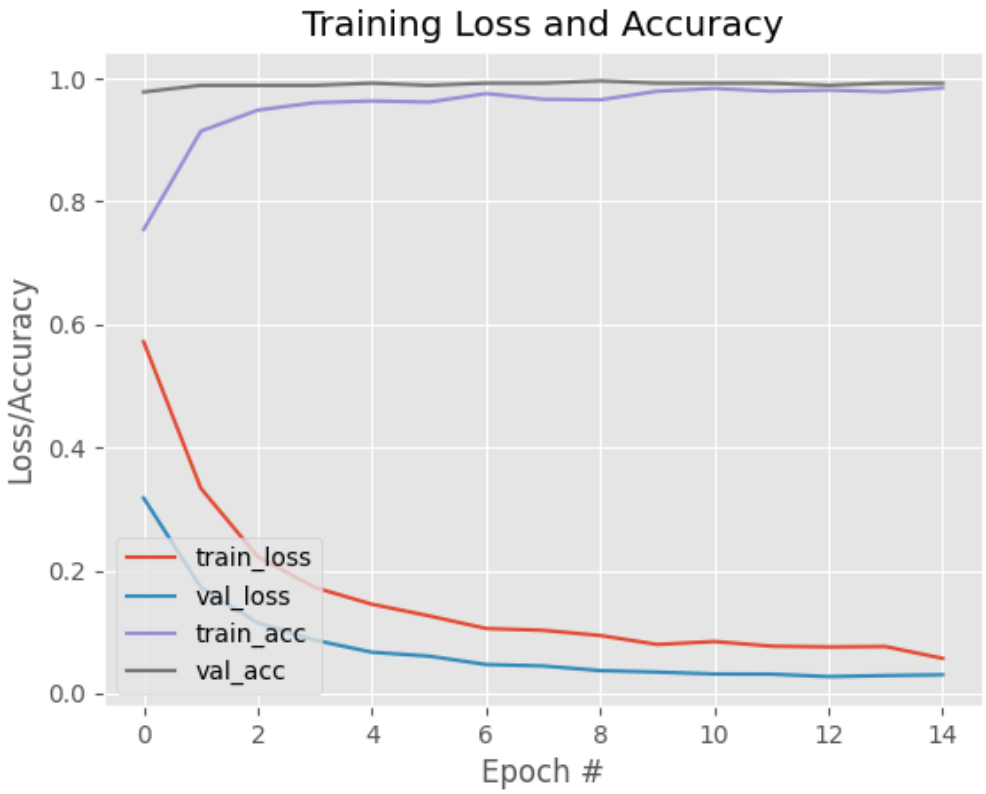

Gráfico 6 - Resultado treinamento com 15 Épocas, função de otimização adam e função channel_shift_range.

Fonte: Autor.

4.1.7 Treinamento com 20 Épocas e função de otimização *sgd* sem data augmentation

O treinamento executado em 20 épocas e função de otimização *sgd* iniciou apresentando perda de 0,8493 e acurácia de 0,5362 para os dados de treinamento e perda de 0,7703 e acurácia de 0,5235 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou, perda de 0,6515 e acurácia de 0,6970 para os dados de treinamento e perda de 0,5735 e acurácia de 0,7942 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 7.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 7 e podem ser interpretados como o de um sistema não preciso.

Tabela 7 - Resultado treinamento com 20 Épocas e função de otimização sgd.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,74	0,91	0,82	140
Sem máscara	0,88	0,68	0,77	137
Acurácia			0,79	277
Macro-average	0,81	0,79	0,79	277
Weighted-average	0,81	0,79	0,79	277

Fonte: Autor.

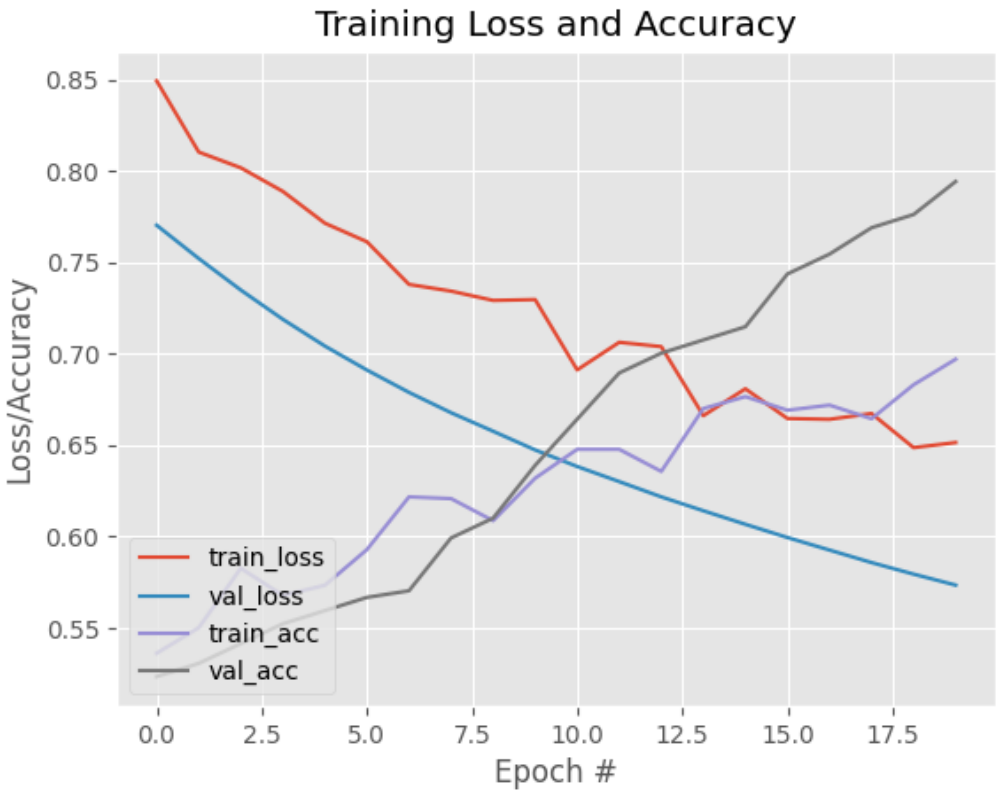

Gráfico 7 - Resultado treinamento com 20 Épocas e função de otimização sgd.

Fonte: Autor.

4.1.8 Treinamento com 20 Épocas e função de otimização *adam* sem data augmentation

O treinamento executado em 20 épocas e função de otimização *adam* iniciou apresentando perda de 0,6176 e acurácia de 0,7361 para os dados de treinamento e perda de 0,3206 e acurácia de 0,9783 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou, perda de 0,0241 e acurácia de 0,9954 para os dados de treinamento e perda de 0,0169 e acurácia de 0,9928 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 8.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 8 e podem ser interpretados como o de um sistema preciso.

Tabela 8 - Resultado treinamento com 20 Épocas e função de otimização adam.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,99	1,00	0,99	140
Sem máscara	1,00	0,99	0,99	137
Acurácia			0,99	277
Macro-average	0,99	0,99	0,99	277
Weighted-average	0,99	0,99	0,99	277

Fonte: Autor.

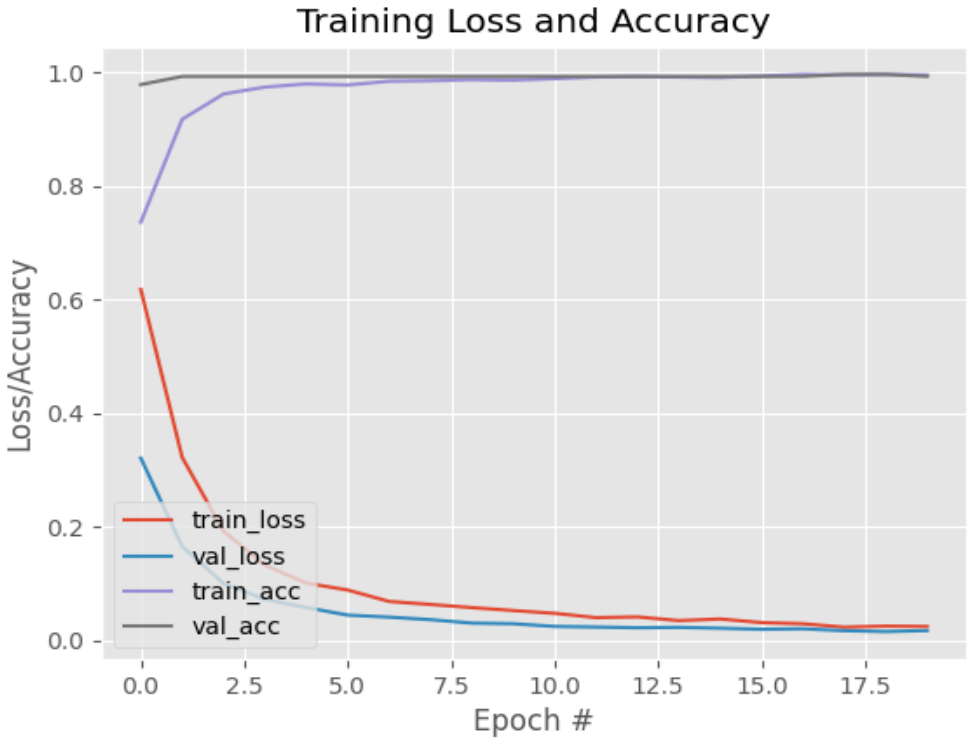

Gráfico 8 - Resultado treinamento com 20 Épocas e função de otimização adam.

Fonte: Autor.

4.1.9 Treinamento com 20 Épocas, função de otimização *adam* com data augmentation

O treinamento executado em 20 épocas, função de otimização *adam* e função de data augmentation *channel_shift_range* iniciou apresentando perda de 0,5994 e acurácia de 0,6924 para os dados de treinamento e perda de 0,3399 e acurácia de 0,9675 para os dados de validação. Na oitava época os resultados finalizaram com valores melhores que os da primeira, após uma sequência de crescimento nos valores de acurácia e decrescimento nos valores de perda nas demais épocas, a última apresentou, perda de 0,0447 e acurácia de 0,9870 para os dados de treinamento e perda de 0,0200 e acurácia de 0,9928 para os dados de validação conforme pode-se visualizar nas curvas do Gráfico 9.

Os resultados obtidos do cálculo das métricas Acurácia, Macro-average e Weighted-average estão apresentados na Tabela 9 e podem ser interpretados como o de um sistema preciso.

Tabela 9 - Resultado treinamento com 20 Épocas, função de otimização adam e função channel_shift_range.

Métricas	Precisão	Recall	F1 Score	Quantidade de dados
Com máscara	0,99	1,00	0,99	140
Sem máscara	1,00	0,99	0,99	137
Acurácia			0,99	277
Macro-average	0,99	0,99	0,99	277
Weighted-average	0,99	0,99	0,99	277

Fonte: Autor.

Gráfico 9 - Resultado treinamento com 20 Épocas, função de otimização adam e função channel_shift_range.

Fonte: Autor.

4.2 Resultado da previsão do uso de máscara

Para identificar o resultado é impresso na imagem de entrada um retângulo ao redor da face. É utilizada a cor verde quando a imagem apresenta face com máscara e vermelho quando apresenta face sem máscara. Abaixo são apresentados os resultados das três imagens testadas pelo autor nas figuras 25, 26 e 27. Com o objetivo de sinalizar a não utilização de máscara em alguma imagem exposta ao sistema, são apresentados nas figuras 28, 29 e 30 o formato de e-mail enviado contendo os resultados.

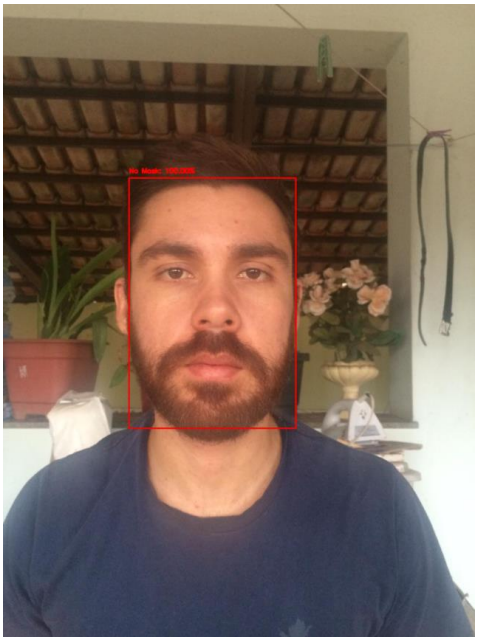

Figura 25 - Resultado para fotografia do autor sem máscara.

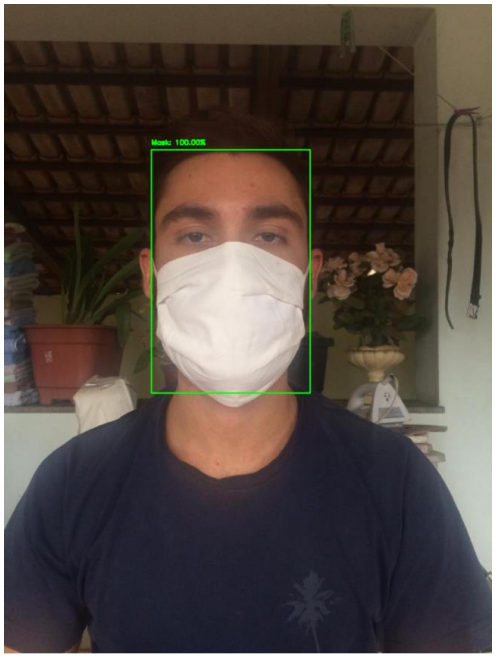

Figura 26 - Resultado para fotografia do autor com máscara branca.

Figura 27 - Resultado para fotografia do autor com máscara azul.

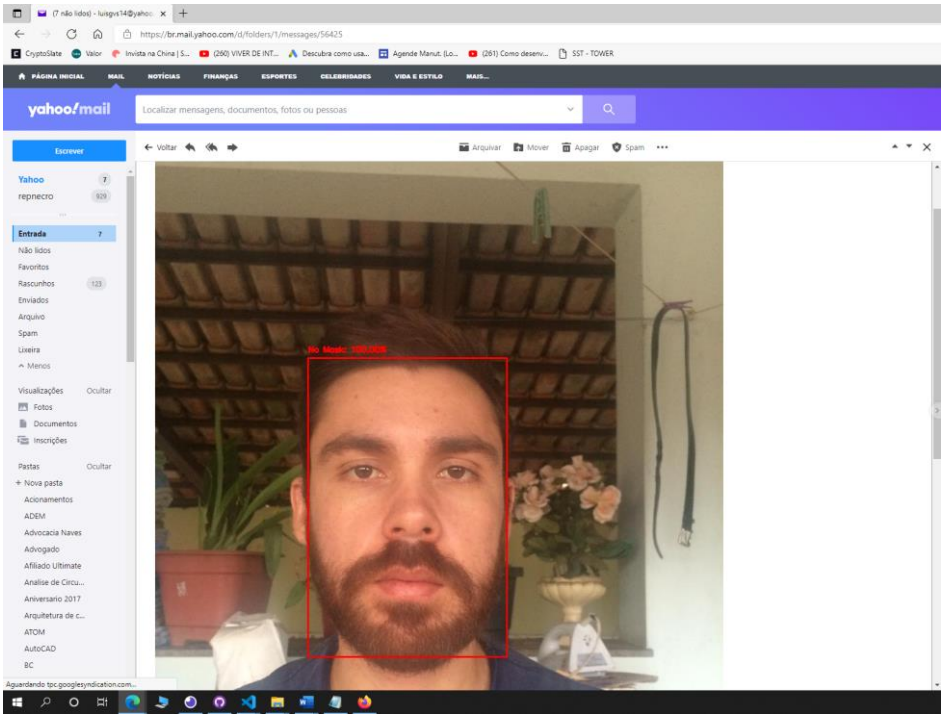

Figura 28 - E-mail enviado com o resultado da Figura 24.

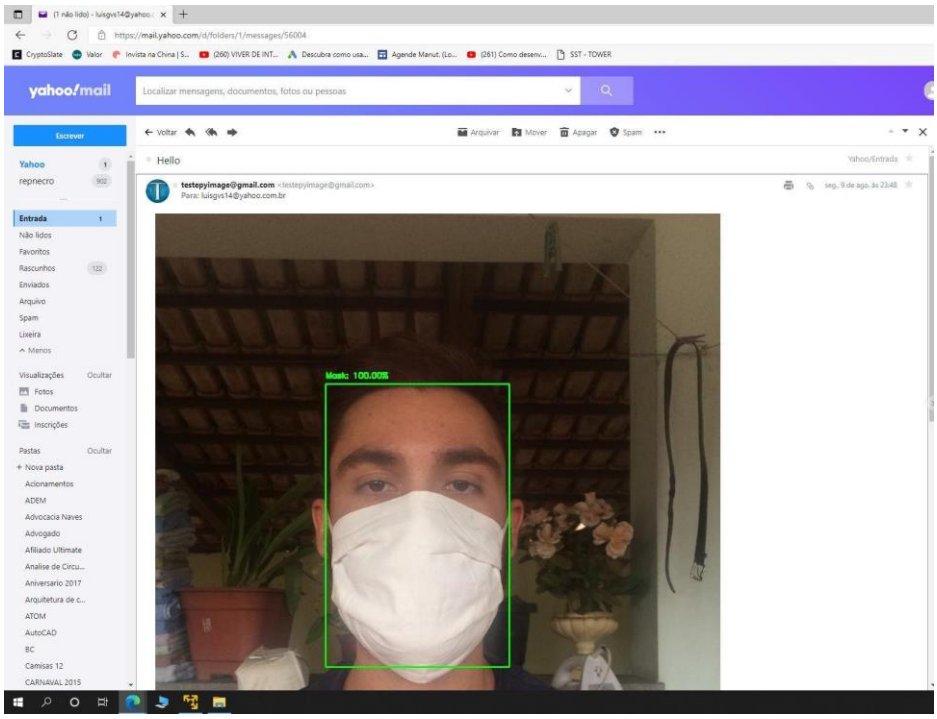

Figura 29 - E-mail enviado com o resultado da Figura 25.

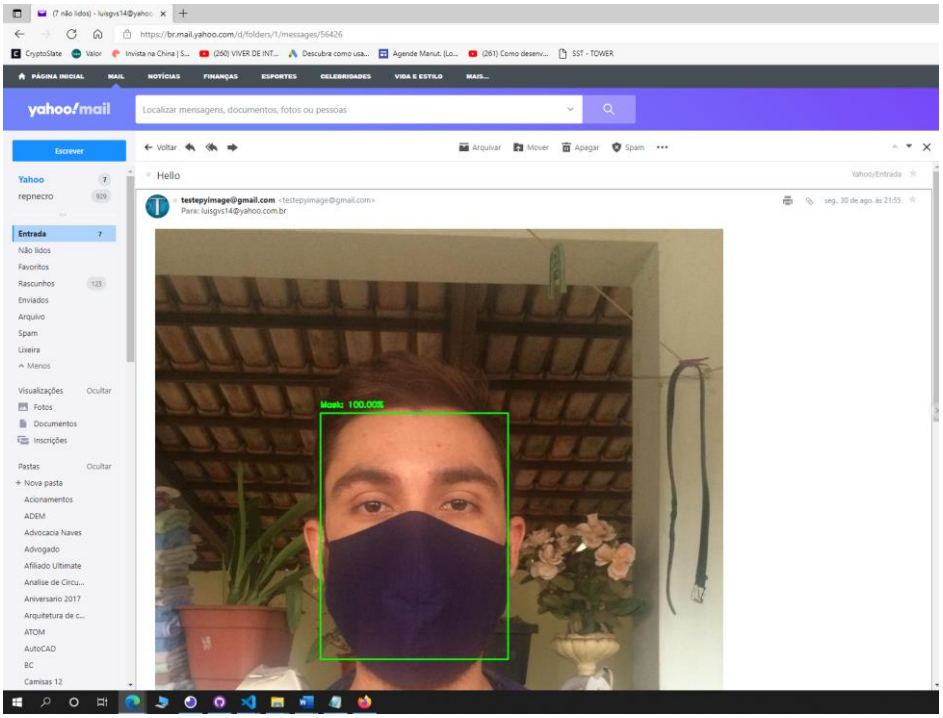

Figura 30 - E-mail enviado com o resultado da Figura 26

4.3 Matriz Confusão

Para construção da matriz de confusão, após o sistema treinado utilizando 8 épocas e função de ativação adam sem data augmentation, foram testadas 140 novas imagens (70 de pessoas com máscara e 70 de pessoas sem máscara), essas foram extraídas do Google e não foram utilizadas no treinamento. Os resultados obtidos da matriz de confusão e métricas dessa provenientes estão apresentados nas Figura 31 e Tabela 10.

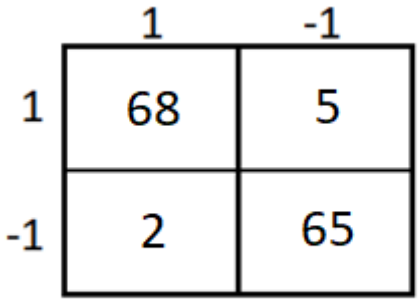

Figura 31 – Matriz de confusão obtida após o teste de 140 novas imagens.

Tabela 10 – Resultado dos testes realizados com 140 novas imagens.

Precisão	0,97
Recall	0,93
F1-Score	0,95

5 CONCLUSÃO

Após análise dos resultados observou-se a grande eficiência de aprendizado da rede que, mesmo em condições menos favoráveis, como utilizando 8 épocas e função de otimização sgd, conseguiu evoluir em acurácia e perda. Os testes mostraram uma grande diferença de resultados entre treinamento e validação realizados com função de otimização sdg e adam, sendo a função adam melhor em todas as opções de quantidade de épocas testadas.

Os melhores resultados de treinamento foram obtidos quando executado com a rede configurada para 20 épocas e função de ativação adam sem utilizar a função de data augmentation channel_shift_range. Neste, a rede apresentou perda de 0,0241 e acurácia de 0,9954 para os dados de treinamento e perda de 0,0169 e acurácia de 0,9928 para os dados de validação.

Os melhores resultados nas métricas de validação foram obtidos quando executada com a rede configurada para 8 épocas e função de ativação adam, sendo os mesmos resultados com e sem a função de data augmentation channel_shift_range. Levando em consideração os dados de teste destas duas configurações, a rede sem channel_shift_range obteve um melhor resultado em sua última época de treinamento, em que apresentou perda de 0,0710 e acurácia de 0,9823 para os dados de treinamento e perda de 0,0368 e acurácia de 0,9964 para os dados de validação.

5.1 SUGESTÕES PARA TRABALHOS FUTUROS

Para trabalhos futuros se sugere o desenvolvimento de um sistema que trabalhe fazendo a detecção de imagens em vídeos. Desta forma seria possível termos um sistema trabalhando em tempo real, aumentando a eficiência no controle do uso de máscara.

Visto a ampla diversidade de dispositivos móveis que possuímos atualmente sugere-se a implantação do sistema em equipamentos de menor custo que um notebook. Desta forma o projeto pode se tornar mais abrangente para comercialização.

6 BIBLIOGRAFIA

Academy, D. S. (2020). *Deep Learning Book Brasil.*

Ana Caroline Gomes Vargas, A. P. (2016). Um Estudo sobre Redes Neurais Convolucionais e sua aplicção em detecção de pedestres.

Buduma, N. (2017). *Fundamentals of Deep Learning Designing Next-Generation Machine.* Oreilly.

Culjak, I., Abram, D., Pribanic, T., Dzapo, H., & Cifrek, M. (2012). A brief introduction to OpenCV. *IEEE*.

Fabian Pedregosa, G. V. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2825-2830.

Fernando Osório, J. R. (Junho de 2000). Sistemas Inteligentes baseados em Redes Neurais Artificiais aplicadas ao Processamento de Imágens. *I WORKSHOP DE INTELIGÊNCIA ARTIFICIAL UNISC - Universidade de Santa Cruz do Sul Departamento de Informática*.

Ferneda, E. (2006). Redese neurais e sua aplicação em sistemas de recuperação de informação. *Ciência da Informação*.

Garcia LP, D. E. (2020). Intervenções não farmacológicas para o enfrentamento à epidemia da Covid-19 no Brasil. *Epidemiol Serv Saúde*.

Hamed Habibi Aghdam, E. J. (2017). *Guide to Covolutional Newral Networks A Practical Application to Traffic-Sign Detection and Classification.* Springer.

Hashemi, M. (2019). Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. *Springer Open*.

Haykin, S. (2008). *Neural Networks and Learning Machines .* Pearson.

Haykin, S. S. (2001). *Redes Neurais.* Bookman.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Enviroment. *IEEE*.

Konstantin Eckle, J. S.-H. (13 de Abril de 2018). A comparison of deep networks with ReLU activation function and linear spline-type methods. *Elsevier*.

Krupali Mistry, A. S. (02 de Outubro de 2016). An Introduction to OpenCV using Python with Ubuntu. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 65-68.

Kucharski AJ, R. T. (2020). Early dynamics of transmission and control of COVID-19: a mathematical modelling study. *Lancet Infect Dis*.

Manaswi, N. K. (2018). *Understanding and Working with Keras.*

Martín Abadi, P. B. (2016). TensorFlow: A System for Large-Scale. *USENIX*.

Mattos, D. M. (2008). *Virtualização: VMWare e Xen.* Fonte: researchgate: https://www.researchgate.net/profile/Diogo_Menezes3/publication/255657320_Virtualizacao_VMWare_e_Xen/links/5617ce6f08aeccf998eae4eb/Virtualizacao-VMWare-e-Xen.pdf

Oliphant, T. E. (2006). *Guide to NumPy.*

Planalto. (04 de 11 de 2020). *Presidência da República*. Fonte: www.gov.br: https://www.gov.br/planalto/pt-br/acompanhe-o-planalto/noticias/2020/07/lei-que-torna-obrigatorio-o-uso-de-mascara-e-sancionada

prajnasb. (01 de 07 de 2020). *github*. Fonte: github: https://github.com/prajnasb/observations

Rebecka Hogevall, P. R. (10 de Agosto de 2021). *Peltarion*. Fonte: Peltarion: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy

Rosebrock, A. (09 de Abril de 2020). *Pyimagesearch*. Fonte: Pyimagesearch: https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/

Rosebrock, A. (09 de 10 de 2020). *Pyimagesearch*. Fonte: Pyimagesearch: https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

S. Vani, D. T. (2019). An Experimental Approach towards the Performance Assessment of Various Optimizers on Convolutional Neural Network. *IEEE.*

Sandler, M., Andrew, H., Menglong, Z., Adrey, Z., & Liang-Chieh, C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *IEEE*.

Saúde, M. d. (10 de Agosto de 2021). *Coronavírus Brasil.* Fonte: Coronavírus Brasil: https://covid.saude.gov.br/

Szeliski, R. (2010). *Computer Vision Algorithms and Applications.* Springer.

Tawsin Uddin Ahmed, S. H. (20019). Facial Expression Recognition using Convolutional Neural Network with Data Augmentation.

APÊNDICES

APÊNDICE A – ALGORÍTIMO DE TREINAMENTO DA REDE COM 8 ÉPOCAS E FUNÇÃO DE OTIMIZAÇÃO *ADAM*

```
# USAGE
# python train_mask_detector.py --dataset dataset

# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
        help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
        help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
        default="mask_detector.model",
        help="path to output face mask detector model")
args = vars(ap.parse_args())
```

```
38
39 # initialize the initial learning rate, number of epochs to train for,
40 # and batch size
41 INIT_LR = 1e-4
42 EPOCHS = 20
43 BS = 32
44
45 # grab the list of images in our dataset directory, then initialize
46 # the list of data (i.e., images) and class images
47 print("[INFO] loading images...")
48 imagePaths = list(paths.list_images(args["dataset"]))
49 data = []
50 labels = []
51
52 # loop over the image paths
53 for imagePath in imagePaths:
54         # extract the class label from the filename
55         label = imagePath.split(os.path.sep)[-2]
56
57         # load the input image (224x224) and preprocess it
58         image = load_img(imagePath, target_size=(224, 224))
59         image = img_to_array(image)
60         image = preprocess_input(image)
61
62         # update the data and labels lists, respectively
63         data.append(image)
64         labels.append(label)
65
66 # convert the data and labels to NumPy arrays
67 data = np.array(data, dtype="float32")
68 labels = np.array(labels)
69
70 # perform one-hot encoding on the labels
71 lb = LabelBinarizer()
72 labels = lb.fit_transform(labels)
73 labels = to_categorical(labels)
74
75 # partition the data into training and testing splits using 75% of
76 # the data for training and the remaining 25% for testing
77 (trainX, testX, trainY, testY) = train_test_split(data, labels,
78     test_size=0.20, stratify=labels, random_state=42)
```

```
79
80 # construct the training image generator for data augmentation
81 aug = ImageDataGenerator(
82         rotation_range=20,
83         zoom_range=0.15,
84         width_shift_range=0.2,
85         height_shift_range=0.2,
86         shear_range=0.15,
87         horizontal_flip=True,
88         fill_mode="nearest")
89
90 # load the MobileNetV2 network, ensuring the head FC layer sets are
91 # left off
92 baseModel = MobileNetV2(weights="imagenet", include_top=False,
93         input_tensor=Input(shape=(224, 224, 3)))
94
95 # construct the head of the model that will be placed on top of the
96 # the base model
97 headModel = baseModel.output
98 headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
99 headModel = Flatten(name="flatten")(headModel)
100 headModel = Dense(128, activation="relu")(headModel)
101 headModel = Dropout(0.5)(headModel)
102 headModel = Dense(2, activation="softmax")(headModel)
103
104 # place the head FC model on top of the base model (this will become
105 # the actual model we will train)
106 model = Model(inputs=baseModel.input, outputs=headModel)
107
108 # loop over all layers in the base model and freeze them so they will
109 # *not* be updated during the first training process
110 for layer in baseModel.layers:
111         layer.trainable = False
112
113 # compile our model
114 print("[INFO] compiling model...")
115 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
116 model.compile(loss="binary_crossentropy", optimizer=opt,
117         metrics=["accuracy"])
118
119 # train the head of the network
```

```
120 print("[INFO] training head...")
121 H = model.fit(
122         aug.flow(trainX, trainY, batch_size=BS),
123         steps_per_epoch=len(trainX) // BS,
124         validation_data=(testX, testY),
125         validation_steps=len(testX) // BS,
126         epochs=EPOCHS)
127
128 # make predictions on the testing set
129 print("[INFO] evaluating network...")
130 predIdxs = model.predict(testX, batch_size=BS)
131
132 # for each image in the testing set we need to find the index of the
133 # label with corresponding largest predicted probability
134 predIdxs = np.argmax(predIdxs, axis=1)
135
136 # show a nicely formatted classification report
137 print(classification_report(testY.argmax(axis=1), predIdxs,
138        target_names=lb.classes_))
139
140 # serialize the model to disk
141 print("[INFO] saving mask detector model...")
142 model.save(args["model"], save_format="h5")
143
144 # plot the training loss and accuracy
145 N = EPOCHS
146 plt.style.use("ggplot")
147 plt.figure()
148 plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
149 plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
150 plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
151 plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
152 plt.title("Training Loss and Accuracy")
153 plt.xlabel("Epoch #")
154 plt.ylabel("Loss/Accuracy")
155 plt.legend(loc="lower left")
156 plt.savefig(args["plot"])
```

APÊNDICE B – ALGORÍTIMO DE DETECÇÃO DE MÁSCARA E ENVIO DE NOTIFICAÇÃO POR E-MAIL

```
# USAGE
# python detect_mask_image.py --image examples/example_01.png
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart
import numpy as np
import smtplib
import argparse
import cv2
import os
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
        help="path to input image")
ap.add_argument("-f", "--face", type=str,
        default="face_detector",
        help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
        default="mask_detector.model",
        help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
        help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
        "res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
```

```
model = load_model(args["model"])

# load the input image from disk, clone it, and grab the image spatial
# dimensions
image = cv2.imread(args["image"])
orig = image.copy()
(h, w) = image.shape[:2]

# construct a blob from the image
blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
        (104.0, 177.0, 123.0))

# pass the blob through the network and obtain the face detections
print("[INFO] computing face detections...")
net.setInput(blob)
detections = net.forward()

# loop over the detections
for i in range(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the detection
confidence = detections[0, 0, i, 2]

# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > args["confidence"]:
# compute the (x, y)-coordinates of the bounding box for
# the object
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")

# ensure the bounding boxes fall within the dimensions of
# the frame
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))

# extract the face ROI, convert it from BGR to RGB channel
# ordering, resize it to 224x224, and preprocess it
face = image[startY:endY, startX:endX]
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
```

```
face = img_to_array(face)
face = preprocess_input(face)
face = np.expand_dims(face, axis=0)

# pass the face through the model to determine if the face
# has a mask or not
(mask, withoutMask) = model.predict(face)[0]

# determine the class label and color we'll use to draw
# the bounding box and text
label = "Mask" if mask > withoutMask else "No Mask"
color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

# include the probability in the label
label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

# display the label and bounding box rectangle on the output
# frame
cv2.putText(image, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
cv2.imwrite("Output.png", image)


print("[INFO] email...")

#e-mail
# conexão com os servidores do google
smtp_ssl_host = 'smtp.gmail.com'
smtp_ssl_port = 465
# username ou email para logar no servidor
username = 'testepyimage@gmail.com'
password = 'password'

from_addr = 'testepyimage@gmail.com'
to_addrs = ['luisgvs14@yahoo.com.br']

# a biblioteca email possuí vários templates
# para diferentes formatos de mensagem
# neste caso usaremos MIMEText para enviar
# somente texto
```

```
120
121 message = MIMEMultipart('alternative')
122 Text = MIMEText('<img src="cid:Output">', 'html')
123 message.attach(text)
134
135 image = MIMEImage(open('Output.png', 'rb').read())
136
137 image.add_header('Content-ID', '<Output>')
138 message.attach(image)
139
140 message['subject'] = 'Hello'
141 message['from'] = from_addr
142 message['to'] = ', '.join(to_addrs)
143
144 # conectaremos de forma segura usando SSL
145 server = smtplib.SMTP_SSL(smtp_ssl_host, smtp_ssl_port)
146
147 # para interagir com um servidor externo precisaremos
148 # fazer login nele
149 server.login(username, password)
150 server.sendmail(from_addr, to_addrs, message.as_string())
151 server.quit()
152
153 # show the output image
154 cv2.imshow("Output", image)
155 cv2.waitKey(0)
```