Trabalho de Conclusão de Curso

Desenvolvimento de um Modelo para Detecção de Uso de Máscara Facial e sua Adaptação para um Dispositivo Embarcado

Matheus Ferreira Gêda
mfg@ic.ufal.br

Orientador:

Prof. Dr. Erick de Andrade Barboza

Maceió, dezembro de 2023

Matheus Ferreira Gêda

Desenvolvimento de um Modelo para Detecção de Uso de Máscara Facial e sua Adaptação para um Dispositivo Embarcado

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação pelo Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Prof. Dr. Erick de Andrade Barboza

Maceió, dezembro de 2023

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Taciana Sousa dos Santos – CRB-4 – 2062

G295d Gêda, Matheus Ferreira.
Desenvolvimento de um modelo para detecção de uso de máscara facial e sua adaptação para um dispositivo embarcado / Matheus Ferreira Gêda. – 2023.
67 f. : il. color.

Orientador: Erick de Andrade Barboza.
Monografia (Trabalho de Conclusão de Curso em Engenharia da Computação) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2023.

Bibliografia: f. 66-67.

1. Aprendizagem de máquina. 2. Sistemas embarcados. 3. Máscara facial. I. Título.

CDU: 004.8

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação pelo Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Prof. Dr. Erick de Andrade Barboza - Orientador
Universidade Federal de Alagoas

Prof. Dr. Baldoino Fonseca dos Santos Neto - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Prof. Dr. Bruno Almeida Pimentel - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Maceió, dezembro de 2023

Agradecimentos

Agradeço primeiramente a Deus, por me guiar ao longo desta jornada acadêmica e me proporcionar força e inspiração. Aos meus pais, Cláudio e Patrícia, aos meus avós, tios, irmãos e à minha namorada, quero expressar minha profunda gratidão por todo o amor, apoio e compreensão que vocês sempre me proporcionaram ao longo desta jornada acadêmica. Vocês são a minha base, minha inspiração e minha motivação constante. Sem o suporte incondicional de vocês, este trabalho não teria sido possível. Obrigado por acreditarem em mim e por serem a minha força em todos os momentos. Aos meus amigos e amigas, que sempre estiveram dispostos a ouvir, apoiar e encorajar, tornando essa jornada mais leve e divertida. Ao meu orientador, Erick Barboza, pela orientação valiosa, paciência e sabedoria compartilhada ao longo deste trabalho. Suas contribuições foram fundamentais para o meu crescimento acadêmico. Aos professores e funcionários do Instituto de Computação e da Universidade Federal de Alagoas, que forneceram a estrutura necessária para a realização deste trabalho e que contribuíram para o meu aprendizado ao longo dos anos. A todos que, de alguma forma, contribuíram para o meu desenvolvimento acadêmico e pessoal, o meu sincero agradecimento.

Resumo

Essa monografia aborda a elaboração de modelos de aprendizado de máquina destinados a sistemas embarcados. No âmbito deste estudo, foram treinados diversos modelos com o propósito de classificar o uso de máscaras, sendo esses modelos posteriormente adaptados para integração em um sistema embarcado. O treinamento do modelo baseou-se em uma base de dados preexistente, composta por cinco mil imagens distribuídas em duas classes distintas. Os resultados experimentais obtidos indicam a viabilidade de incorporar esse modelo, desde que sejam observadas as restrições específicas de cada hardware, como limitações de memória, capacidade de processamento reduzida, baixo consumo de energia, entre outros aspectos. Durante a fase de teste, a avaliação dos modelos destacou a superioridade dos modelos em escala de cinza em termos de acurácia e perda, sendo notável o desempenho do modelo 64×64 em escala de cinza, que alcançou uma acurácia impressionante de 99.66%.

Palavras-chave: Sistemas Embarcados; TinyML; Inteligência artificial; Aprendizagem de máquina; Aprendizado Profundo; Visão Computacional; Processamento de Imagem; Redes Neurais; Redes Neurais Convolucionais; Segurança no trabalho; Equipamentos de proteção individual.

Abstract

This dissertation explores the development of machine learning models tailored for embedded systems. Within this study, diverse models were trained with the objective of classifying mask usage, and subsequently adapted for integration into an embedded system. Model training relied on an existing dataset comprising five thousand images categorized into two distinct classes. Experimental results suggest the feasibility of incorporating this model, provided specific hardware constraints such as limited memory, reduced processing capacity, low power consumption, among other aspects, are carefully considered. This research contributes to the understanding of deploying machine learning in resource-constrained embedded environments, with a focus on real-world applications like mask detection.

Key-words: Embedded Systems; TinyML; Artificial inteligence; Machine learning; Deep Learning; Computer Vision; Image Processing; Neural Networks; Convolutional Neural Networks; Workplace safety; Personal protective equipment.

Lista de Figuras

Conteúdo

1

Introdução

No cenário atual, a saúde pública ganhou uma importância sem precedentes, com desafios globais como a pandemia de COVID-19 destacando a necessidade de medidas preventivas eficazes. Entre as várias medidas adotadas, o uso de máscaras de proteção facial emergiu como uma estratégia fundamental na redução da propagação de doenças transmitidas pelo ar. O uso generalizado de máscaras pode não apenas proteger a saúde individual, mas também contribuir para a proteção coletiva em ambientes públicos e comunitários, bem como em ambientes controlados, como instalações de saúde, escolas e empresas.

Nesse contexto, este trabalho se concentra no desenvolvimento de um sistema de detecção, com o propósito de identificar se uma pessoa está utilizando máscara facial. A motivação para este projeto é clara e urgente, dadas as implicações para a saúde pública e a necessidade de soluções tecnológicas eficazes para promover e monitorar o uso de máscaras de proteção em todos os tipos de ambientes.

O objetivo primordial deste estudo é criar um modelo de aprendizado de máquina capaz de reconhecer de maneira precisa e eficiente se um indivíduo está ou não utilizando máscara. O modelo de detecção é desenvolvido a partir de um conjunto de dados previamente coletado e rotulado, aproveitando técnicas avançadas de visão computacional e aprendizado profundo, notadamente redes neurais convolucionais. A eficácia do modelo é demonstrada em tarefas de detecção de máscaras em imagens e vídeos, com resultados promissores.

Um dos pontos chave deste trabalho é a conversão do modelo de detecção para o formato TensorFlow Lite (TFLite), tornando-o apto para ser executado em dispositivos embarcados com recursos computacionais limitados. Isso viabiliza a incorporação do modelo em sistemas embarcados, como câmeras de vigilância, possibilitando a detecção em tempo real em ambientes públicos e controlados, sem depender de uma conexão constante com a internet.

A combinação de modelo de detecção e dispositivo embarcado apresenta uma solução viável e escalável para monitorar o uso de máscaras, contribuindo assim para a promoção da saúde

pública e o controle da disseminação de doenças infecciosas, não apenas em ambientes públicos, mas também em ambientes controlados.

Este projeto não apenas representa uma contribuição importante no contexto da vigilância de saúde pública, mas também destaca o potencial das tecnologias de aprendizado de máquina em dispositivos embarcados para enfrentar desafios de segurança e saúde em nossa sociedade. Este trabalho oferece uma abordagem prática e inovadora para a promoção da saúde pública e a redução de riscos em cenários de pandemia e além, abrangendo ambientes variados, controlados e públicos.

O texto segue uma estrutura organizada que facilita a compreensão do desenvolvimento do sistema. Começando com uma introdução que destaca a importância do tema no contexto da saúde pública, o texto motiva a necessidade de soluções tecnológicas, especificamente o desenvolvimento de um modelo de aprendizado de máquina. Em seguida, é mostrado alguns detalhes sobre os assuntos e técnicas que serão utilizadas nesse projeto. Após, detalha-se a implementação desse modelo, destacando sua eficácia na detecção de máscaras em diferentes ambientes. A adaptação para o formato TensorFlow Lite é discutida, permitindo a execução do modelo em dispositivos embarcados. O parágrafo final ressalta a relevância da combinação entre o modelo de detecção e dispositivos embarcados, oferecendo uma solução prática e escalável para monitorar o uso de máscaras e contribuir para a saúde pública. Dessa forma, temos uma organização simples que guia o leitor de forma sequencial, desde a motivação até as aplicações práticas do projeto.

2

Fundamentação Teórica

2.1 Uso de Máscaras

O uso de máscaras tornou-se uma prática fundamental em todo o mundo como uma medida eficaz de prevenção contra a propagação do vírus SARS-CoV-2, causador da doença COVID-19. As máscaras atuam como uma barreira física, reduzindo a transmissão de partículas respiratórias que podem conter o vírus. Essa medida preventiva é particularmente crucial em situações em que o distanciamento social pode ser desafiador.

A pandemia de COVID-19, declarada pela Organização Mundial da Saúde (OMS) em março de 2020, teve um impacto sem precedentes na saúde global e nas dinâmicas sociais. O vírus se espalhou rapidamente, exigindo a implementação de medidas rigorosas para conter sua disseminação. O uso de máscaras emergiu como uma estratégia essencial para mitigar a transmissão do vírus, sendo incentivado e, em muitos casos, obrigatório em diversos contextos, desde espaços públicos até ambientes de trabalho.

2.1.1 Ambientes Controlados

O uso de máscaras não se restringe apenas a espaços públicos, mas também se estende a ambientes controlados, como hospitais, clínicas e laboratórios. Nestes locais, onde a vulnerabilidade à infecção é elevada, as medidas de precaução são especialmente rigorosas. Profissionais de saúde e outros trabalhadores essenciais frequentemente utilizam máscaras como parte de um conjunto abrangente de protocolos de segurança. Além disso, ambientes de pesquisa e produção, nos quais a continuidade das atividades é crucial, adotaram medidas similares para proteger os trabalhadores e manter a operacionalidade.

A utilização de máscaras em ambientes controlados não apenas visa proteger os indivíduos dentro desses espaços, mas também impede a disseminação potencial do vírus para a comuni-

Figura 2.1: População usando máscara após campanhas de incentivo, imagem retirada de: [1]

dade. A abordagem preventiva é essencial para garantir a segurança de todos os envolvidos, e a implementação rigorosa dessas práticas em ambientes controlados demonstra a adaptação contínua das medidas de saúde pública em resposta à pandemia.

2.2 Aprendizado de Máquina

O campo da inteligência artificial é impulsionado por dois pilares fundamentais: aprendizado de máquina e deep learning. Essas disciplinas desempenham papéis cruciais na capacidade dos sistemas computacionais de realizar tarefas complexas e extrair padrões a partir de dados.[5]

2.2.1 Introdução ao Aprendizado de Máquina

O Aprendizado de Máquina é um ramo da inteligência artificial que como o nome já diz, capacita os sistemas a "aprender"a partir de dados, permitindo-lhes realizar tarefas sem serem explicitamente programados. No aprendizado supervisionado, o algoritmo é treinado com um conjunto de dados rotulado, no qual a saída desejada é conhecida. O modelo, então, generaliza esse conhecimento para fazer previsões ou classificações em dados novos ou não rotulados.

Por outro lado, no aprendizado não supervisionado, os algoritmos exploram dados sem rótulos, buscando padrões. Agrupamento e redução de dimensionalidade são exemplos de técnicas não supervisionadas. Essas abordagens têm aplicação em uma variedade de campos, desde reconhecimento de padrões até análise de dados complexos.

2.2.2 Introdução ao Deep Learning

O Deep Learning é uma subcategoria do aprendizado de máquina que utiliza redes neurais profundas para realizar tarefas complexas de aprendizado e reconhecimento de padrões. Ao contrário de modelos mais rasos, as redes neurais profundas são capazes de aprender representações hierárquicas de dados, o que as torna particularmente eficazes em tarefas como reconhecimento de imagem, processamento de linguagem natural e reconhecimento de voz.

A arquitetura de redes neurais profundas é composta por várias camadas, por isso do profundo, cada uma delas processando características específicas dos dados de entrada. Redes neurais convolucionais são comuns em tarefas de visão computacional, enquanto redes neurais recorrentes são aplicadas em sequências temporais, como no processamento de linguagem natural.

O treinamento de modelos de deep learning geralmente requer grandes conjuntos de dados e poder computacional significativo, mas os avanços recentes e o aumento na disponibilidade de recursos têm impulsionado a aplicação prática dessas técnicas em uma variedade de setores, incluindo saúde, finanças, e nesse projeto, na identificação do uso de máscaras. A capacidade do deep learning em extrair características complexas de imagens o torna uma escolha ideal para abordar desafios como a variação na aparência das máscaras e condições de iluminação adversas.[6]

2.2.3 Redes Neurais Convolucionais

No campo do *deep learning*, existe uma técnica denominada RNC (rede neural convolucional) é considerado um tipo de rede neural artificial. Desde o surgimento desse tipo de rede, foi notada a eficácia no emprego dessas RNCs na solução de desafios de categorização. Portanto, neste estudo, optaremos por empregar RNCs para avaliar o uso de máscaras. A arquitetura de rede neural de convolução tem emergido recentemente como uma das opções mais pragmáticas em comparação com abordagens de categorização mais convencionais.

Uma dificuldade ao treinar um algoritmo supervisionado surge da necessidade de dispor de uma grande base de dados categorizados para estabelecer essas relações e, assim, extrair as características desejadas. Para extrair essas características com uma RNC, são essenciais três elementos básicos: a camada de convolução, a camada de agrupamento ou *pooling* e a camada totalmente conectada. Independentemente da simplicidade da arquitetura de uma RNC, esses três elementos são fundamentais. Importante notar que a arquitetura e a configuração das RNCs podem variar de acordo com a natureza do problema e as exigências específicas do contexto abordado.[7]

A camada convolucional é um componente central nas RNCs, desempenhando um papel crucial na extração de características intrínsecas na base de dados. A operação básica da camada

convolucional envolve a aplicação de filtros (kernels) a regiões específicas da entrada. No estágio inicial do treinamento, os filtros são ajustados inicialmente para o conjunto de dados, e à medida que as épocas de treinamento se desdobram, esses filtros evoluem e se aprimoram para otimizar a acurácia na identificação das propriedades. Esses filtros são responsáveis por detectar padrões locais, como bordas, texturas ou formas, permitindo que a rede aprenda representações hierárquicas complexas. Os benefícios da camada convolucional incluem a capacidade de aprender representações locais discriminativas, reduzindo a quantidade de parâmetros em comparação com camadas totalmente conectadas. Essa eficiência é crucial para processar dados de alta dimensionalidade, como imagens, de maneira eficaz.

A camada de *pooling* desempenha um papel fundamental nas RNCs. Essa camada é responsável por reduzir a dimensionalidade espacial da representação da imagem, preservando características e aprimorando a eficiência computacional. A operação básica de pooling envolve a subdivisão da entrada em regiões locais, seguida pela aplicação de uma função específica, como por exemplo, a extração do valor máximo *max pooling* ou a média *average pooling* que podemos como funciona na figura 2.2. Os benefícios da camada de pooling são vastos, incluindo a redução da carga computacional, a promoção da generalização da rede e a prevenção do overfitting. No entanto, é importante considerar os desafios associados, como a perda de informações detalhadas. Existem estratégias como o ajuste adequado dos parâmetros podem ser empregadas para mitigar esses desafios e otimizar o desempenho nessa camada.

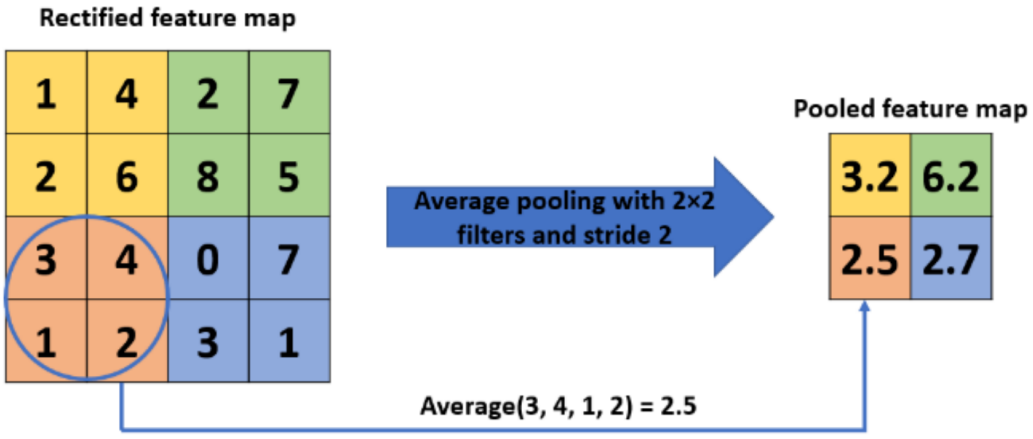

Figura 2.2: Average pooling, imagem retirada de: [2]

A camada totalmente conectada representa um elemento central nas Redes Neurais, desempenhando um papel crucial na integração de informações e na tomada de decisões finais. Diferentemente da camada convolucional, que se concentra na extração de características, a camada totalmente conectada operam na representação global, consolidando as informações para produzir uma saída final. Essa camada é caracterizada por conexões entre todos os neurônios da camada anterior e a camada totalmente conectada, resultando em uma representação densa da

informação. Os benefícios dessa camada incluem a capacidade de aprender representações hierárquicas complexas e de lidar com dados altamente não lineares. No entanto, o uso indiscriminado de camadas totalmente conectadas pode resultar em modelos propensos a overfitting, devido à alta dimensionalidade dos parâmetros. Porém, existem algumas estratégias que podem minimizar o overfitting no modelo, vamos falar um pouco sobre isso a seguir.

2.2.4 Overfitting

O *overfitting*, ou ajuste excessivo, é um fenômeno no aprendizado de máquina supervisionado onde um modelo se adapta tão bem aos dados de treinamento que começa a capturar também o ruído presente nesses dados, em vez de aprender os padrões subjacentes e generalizáveis. Em outras palavras, o modelo se torna excessivamente específico para os exemplos de treinamento, perdendo a capacidade de generalizar corretamente para novos dados não vistos durante o treinamento como podemos perceber na figura 2.3.[8]

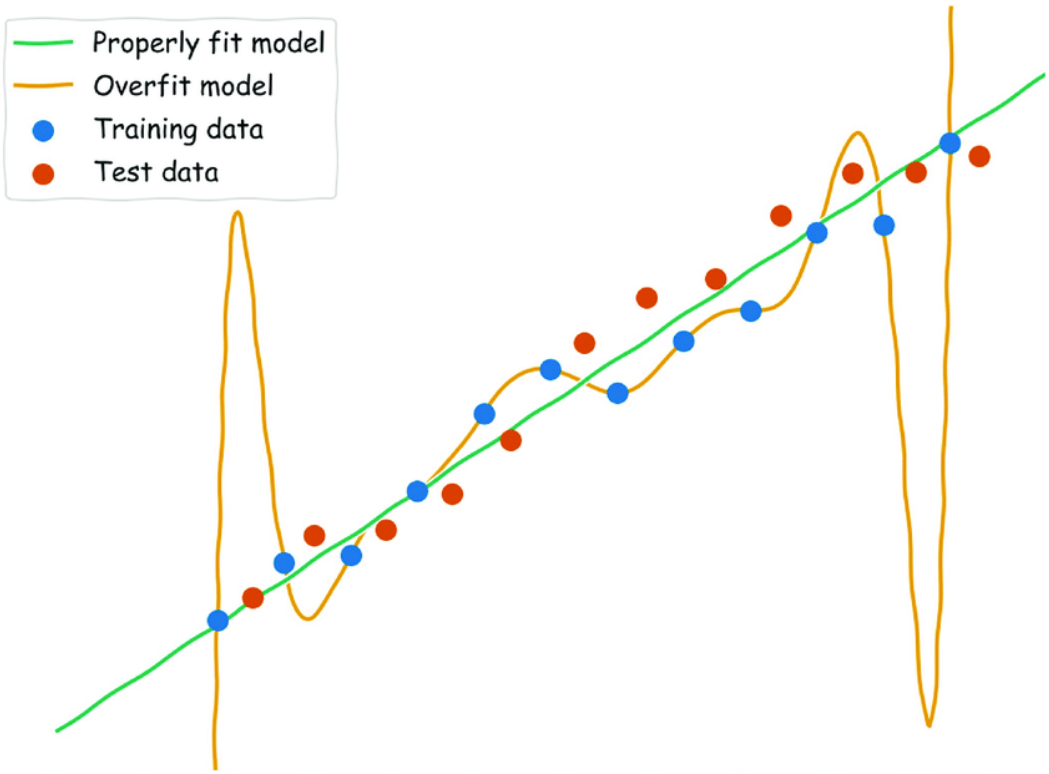

Figura 2.3: Overfitting, imagem retirada de: [3]

Existem várias razões pelas quais o overfitting pode ocorrer, algumas delas são:

- **Ruído nos Dados:** Se os dados de treinamento contêm ruído ou variações aleatórias, o modelo pode aprender essas variações como padrões significativos.
- **Tamanho Limitado do Conjunto de Treinamento:** Um conjunto de treinamento pequeno pode não ser representativo o suficiente da distribuição real dos dados, levando o modelo a ajustar-se demasiadamente a padrões específicos do conjunto de treinamento.

- **Complexidade do Modelo:** Modelos muito complexos, com muitos parâmetros, têm maior probabilidade de se adaptarem demais aos dados de treinamento.

Para mitigar o *overfitting*, algumas estratégias podem ser empregadas:

- *Early Stopping* (Parada Antecipada)
- *Dropout Regularization*
- Redução da Complexidade do Modelo
- *Data Augmentation* (Aumento de Dados)

Em seguida abordaremos um pouco mais das estratégias que foram utilizadas durante o treinamento do modelo, o *Dropout Regularization* e o *Early Stopping*.

Como foi dito anteriormente, o *overfitting* pode ser um desafio quando nós falamos sobre redes neurais. Portanto, *Dropout Regularization* é uma técnica eficaz em redes neurais para mitigar esse ajuste excessivo, consiste de uma técnica simples, que envolve a desativação temporária de neurônios, de forma aleatória durante o treinamento. Durante o treinamento, em cada iteração de dados, uma fração aleatória de unidades é temporariamente removida ou "desligada". Isso significa que a informação não é passada para a frente durante a fase de treinamento e, portanto, a rede é forçada a aprender de maneira mais robusta, evitando depender excessivamente de neurônios específicos. [9]

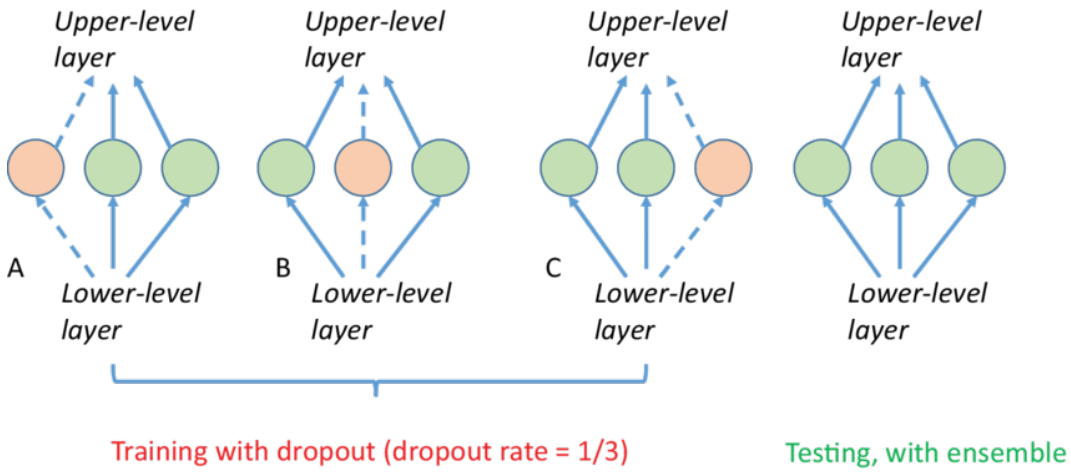

Figura 2.4: Dropout, imagem retirada de: [4]

Na técnica de *early stopping*, o objetivo é interromper o treinamento do modelo assim que o desempenho em um conjunto de validação começa a piorar, indicando que o modelo atingiu um ponto onde está começando a se ajustar demais aos dados de treinamento. A aplicação do *early*

stopping envolve o monitoramento contínuo do desempenho do modelo em um conjunto de validação. Um critério de parada é definido, como a ausência de melhoria após um determinado número de épocas consecutivas. No entanto, é crucial ajustar adequadamente os parâmetros do *early stopping*, como o número de épocas sem melhoria necessário para interromper o treinamento. Utilizar um valor muito pequeno pode interromper o treinamento prematuramente, enquanto um valor muito grande pode permitir *overfitting* antes da interrupção.[10]

2.3 Sistemas Embarcados

Os sistemas embarcados são fundamentais na integração de hardware e software, projetados para realizar tarefas específicas em tempo real, com restrições de recursos. Esses sistemas desempenham um papel crucial em uma ampla variedade de aplicações, desde dispositivos domésticos inteligentes até sistemas críticos em áreas como saúde, automotiva e industrial.[11]

2.3.1 Aplicações de Sistemas Embarcados

As aplicações de sistemas embarcados são vastas e abrangem os mais diversos setores. Em dispositivos domésticos, os sistemas embarcados estão presentes nos eletrodomésticos inteligentes, como geladeiras e termostatos, facilitando a automação residencial. Na área de saúde, esses sistemas são usados em dispositivos médicos, como monitores de sinais vitais e equipamentos de diagnóstico por imagem.

No setor automotivo, os sistemas embarcados controlam funções críticas, como injeção de combustível, sistemas de freios ABS e assistência à condução. Além disso, em ambientes industriais, sistemas embarcados são empregados em controle de processos, automação de fábricas e sistemas de monitoramento de condições.

A ascensão da Internet das Coisas também impulsionou a presença de sistemas embarcados em dispositivos conectados, como câmeras de segurança inteligentes, sensores de ambiente e sistemas de rastreamento.

2.3.2 Desafios em Sistemas Embarcados

Apesar de suas numerosas aplicações, os sistemas embarcados enfrentam diversos desafios. As restrições de recursos, como poder computacional limitado e espaço de armazenamento reduzido, demandam otimização eficiente de software e hardware. A garantia de tempo real é crucial em muitos casos, tornando a previsibilidade e a eficiência nas operações essenciais.

A segurança é uma preocupação constante, especialmente em sistemas críticos, onde a integridade e a confiabilidade são imperativas. Além disso, evolução rápida da tecnologia e a necessidade de atualizações de software em dispositivos embarcados após sua implantação adicionam uma camada adicional de complexidade. Estratégias eficazes de gerenciamento de ciclo

de vida são essenciais para garantir a segurança contínua e a funcionalidade dos sistemas embarcados ao longo do tempo.

Portanto, apesar dos desafios, os sistemas embarcados desempenham um papel crucial em inúmeras aplicações, impulsionando a inovação em diversos setores e contribuindo para a criação de soluções tecnológicas avançadas.[12]

2.3.3 TinyML

TinyML, ou *Tiny Machine Learning*, refere-se à implementação de modelos de aprendizado de máquina em dispositivos de recursos limitados, como microcontroladores e sistemas embarcados. O objetivo é trazer capacidades de machine learning para dispositivos IoT, sensores e outros dispositivos incorporados, permitindo a execução de inferência de modelos em tempo real sem depender de uma conexão com a nuvem. Essa abordagem visa otimizar a eficiência computacional e promover a autonomia de dispositivos com recursos limitados.[13]

Algumas características do TinyML são:

1. **Eficiência em Recursos:** O TinyML visa a eficiência em termos de recursos computacionais, sendo especialmente projetado para dispositivos com limitações significativas, como microcontroladores e sistemas embarcados de baixa potência.

2. **Processamento Local:** Um dos objetivos centrais do TinyML é realizar o processamento de modelos de aprendizado de máquina diretamente no dispositivo onde os dados são gerados, eliminando a necessidade de transmitir esses dados para processamento em servidores remotos.

3. **Autonomia de Dispositivos:** Ao permitir que os dispositivos executem tarefas de aprendizado de máquina internamente, o TinyML contribui para a autonomia desses dispositivos, reduzindo a dependência de conexões de rede constantes.

4. **Privacidade e Segurança:** A execução local de modelos diminui a exposição dos dados sensíveis, contribuindo para a privacidade do usuário. Além disso, reduz as vulnerabilidades associadas à transmissão de dados pela rede.

5. **Menor Latência:** Ao processar dados no local, o TinyML minimiza a latência, permitindo respostas mais rápidas e eficientes em tempo real, crucial para cenários como IoT e sistemas embarcados.

6. **Otimização de Energia:** A eficiência energética é uma consideração primordial, especialmente para dispositivos alimentados por baterias. O TinyML visa otimizar o consumo de energia, prolongando a vida útil da bateria e reduzindo a necessidade de recargas frequentes.

7. **Escalabilidade e Distribuição:** A aplicação de técnicas de aprendizado de máquina em dispositivos de menor escala permite a criação de redes distribuídas, contribuindo para a escalabilidade e adaptabilidade em ambientes onde a comunicação entre dispositivos é essencial.

8. **Desenvolvimento Ágil e Simplificado:** O TinyML proporciona um ambiente de desenvolvimento simplificado, facilitando a implementação rápida de modelos em dispositivos com restrições de recursos, tornando o ciclo de desenvolvimento mais ágil.

3

Metodologia

3.1 Banco de dados

O banco de dados utilizado para treinar os modelos de aprendizado de máquina neste estudo já existia previamente e foi construído capturando múltiplas imagens de pessoas com diferentes configurações (com máscara e sem máscara), garantindo diversidade e representatividade para treinar modelos eficazes de detecção de máscara. Esse conjunto de dados consiste em cinco mil imagens distribuídas em duas classes distintas: uma indicando a presença de máscaras e a outra indicando a ausência de máscaras. Essa base de dados é fundamental para capacitar os modelos a reconhecerem padrões associados ao uso de máscaras faciais e pode ser encontrada em: Link para o banco de imagens.

A divisão em duas classes sugere que as imagens estão rotuladas de acordo com a presença ou ausência de máscaras. Esse tipo de rotulação é crucial para o treinamento supervisionado dos modelos, permitindo que eles aprendam a distinguir automaticamente entre rostos com e sem máscaras.[14] [1]

Uma pequena amostra do banco de dados pode ser vista abaixo, na figura 3.1.

O banco de dados é composto por:

- 5592 imagens em resolução 256×256 com três canais de cor
  - 2796 imagens com o uso de máscara
  - 2796 imagens sem o uso de máscara

Sendo estas imagens divididas da seguinte forma:

- 4000 imagens para treinamento

[1] https://zenodo.org/records/6408603

  - 2000 imagens com o uso de máscara
  - 2000 imagens sem o uso de máscara
- 1000 imagens para validação
  - 500 imagens com o uso de máscara
  - 500 imagens sem o uso de máscara
- 592 imagens para teste dos modelos
  - 296 imagens com o uso de máscara
  - 296 imagens sem o uso de máscara

Uma vez estabelecido o conjunto de dados, a preparação para o treinamento de cada modelo envolve ajustes na resolução e nos canais de cor. Isso implica redimensionar as imagens para a resolução desejada. Para modelos em escala de cinza, além do redimensionamento, há também a necessidade de converter as imagens para escala de cinza.

Figura 3.1: Amostra de imagens do banco de dados.

3.2 Exploração dos Modelos

Desenvolvemos 8 modelos de redes neurais convolucionais, distribuídos em 4 modelos coloridos e 4 em escala de cinza. Para a implementação dessas arquiteturas, aproveitamos o ambiente fornecido pelo Google Colaboratory, executando código em Python3. O ambiente conta com recursos robustos, incluindo 12.7GB de RAM e 15GB de VRAM em uma GPU Tesla T4. O notebook pode ser acessado em: Notebook Google Colaboratory. No processo de criação e treinamento dos modelos, utilizamos principalmente as bibliotecas TensorFlow e Keras. Essas ferramentas desempenharam um papel central desde a concepção dos modelos até a fase de conversão para serem empregados em sistemas embarcados.

Os modelos foram treinados com o objetivo de atender às exigências específicas de sistemas embarcados, considerando as limitações desses dispositivos, como restrições de memória, capacidade de processamento limitada e baixo consumo de energia. A adaptação dos modelos para a integração em sistemas embarcados é crucial para garantir sua eficácia em ambientes com recursos limitados.

Inicialmente, para a importação das imagens e a segmentação do conjunto de dados em conjuntos de treinamento e validação, empregamos a funcionalidade do TensorFlow Keras, notadamente o **tensorflow.keras.ImageDataGenerator**, com seu método **flow_from_directory**[2]. [15][16]

Por meio dessas ferramentas, configuramos diversos parâmetros, incluindo:

- Diretório que contém as imagens
- Resolução das imagens
  - 32×32
  - 48×48
  - 64×64
  - 128×128
- Tamanho do batch
  - 20
- Modo de cor

[2] https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator#flow_from_directory

  - Colorido
  - Escala de cinza

- Modo de classe
  - Binária

- Subset
  - Treino
  - Validação
  - Teste

O objeto *ImageDataGenerator* é uma instância configurável que facilita o processo de carregamento de imagens usando o método *flow_from_directory* , o qual não apenas carrega os dados, mas também ajusta as imagens para o formato desejado. Isso envolve redimensionar as imagens para a resolução predefinida e converter para o tipo de dados configurado, tornando o banco de dados pronto para a aplicação nos modelos.

Para visualizar essas imagens, adotamos o OpenCV[3] em conjunto com o MatPlotLib[4], permitindo-nos plotar essas amostras de maneira eficaz.

O MatPlotLib desempenha um papel fundamental em todo o nosso processo, sendo usado para criar todos os tipos de visualizações presentes neste trabalho.

Quanto à construção dos modelos, empregamos a funcionalidade **Sequential** do *tensorflow*. Esta nos permite especificar as camadas internas da rede neural de maneira sequencial. Para isso, importamos a ferramenta **layers**, que abrange diversos tipos de camadas úteis. As camadas utilizadas incluíram: entrada, convolução 2D, maxpooling 2D, flatten, dropout, e por último a camada densa.

- **Camada de Entrada:** A camada de entrada define o formato dos dados que serão alimentados na rede neural. Em uma imagem, o formato típico seria especificado por `input_shape=(altura, largura, canais)`, onde altura e largura são as dimensões da imagem, e canais indicam se a imagem é em escala de cinza (1 canal) ou colorida (3 canais).

- **Convolução 2D:** A camada de convolução 2D aplica filtros à entrada, identificando padrões espaciais e características importantes nas imagens.

- **Maxpooling 2D:** O maxpooling 2D realiza subamostragem, reduzindo a dimensionalidade da representação da imagem. Isso é feito ao selecionar o valor máximo de uma região específica da entrada, preservando as características mais proeminentes.

[3] https://pypi.org/project/opencv-python/
[4] https://matplotlib.org

- **Flatten:** A camada Flatten transforma os dados de entrada em um vetor unidimensional. Ela é frequentemente utilizada para preparar os dados para serem alimentados em camadas totalmente conectadas, removendo as estruturas de matriz.
- **Dropout:** A camada de Dropout introduz aleatoriamente a desativação de um conjunto definido de unidades durante o treinamento. Isso ajuda a prevenir o *overfitting*, tornando a rede mais robusta, ao impedir que unidades específicas se tornem muito dependentes umas das outras.
- **Camada Densa (totalmente conectada):** A camada densa é uma camada totalmente conectada, onde cada unidade está conectada a todas as unidades da camada anterior. Essa camada finaliza a rede neural, produzindo a saída desejada.

Implementamos o **Early Stopping**, como foi mencionado anteriormente. Os argumentos fornecidos para criar a instância desse recurso incluem: a variável a ser acompanhada, uma tolerância de 10 épocas e a configuração para recuperar os melhores pesos. Desta forma, mitigamos o sobreajuste e recuperamos os pesos mais eficazes treinados em nossas arquiteturas.

3.3 Transformação dos Modelos

Para realizar a transformação dos modelos, utilizamos o TensorFlow Lite[5], que é usado para converter um modelo TensorFlow em um formato mais leve e eficiente para implementação em sistemas embarcados, como microcontroladores.[17]

A transformação dos modelos foi realizada por meio de um método chamado **quantização**. Essa técnica busca diminuir a quantidade de bits exigida para expressar os parâmetros do modelo, incluindo pesos e ativações, com o propósito de conservar espaço de armazenamento e intensificar a velocidade da inferência. Os pesos do modelo, originalmente representados como números de ponto flutuante de 32 bits, são convertidos para inteiros de 8 bits.

Então, convertemos os modelos para esse formato e, a partir disso, realizaremos uma análise em relação ao espaço ocupado por cada um desses modelos. Além disso, verificaremos a viabilidade desses modelos para serem implementados no Arduino Nano.

3.4 Avaliação dos Modelos

Para a análise dos modelos, empregamos algumas ferramentas de visualização de dados, como o Matplotlib. Inicialmente, examinamos os resultados durante o treinamento e observamos que todos os modelos não apresentaram *overfitting*, graças às técnicas utilizadas para prevenir esse problema. Posteriormente, utilizamos o TensorFlow para realizar inferências de teste

[5] https://www.tensorflow.org/lite

com o conjunto de dados de teste, composto por 592 imagens, sendo 296 da classe **with_mask** e 296 da classe **without_mask**.

Após as inferências, analisamos a acurácia desses modelos nos testes e relizamos os testes nos modelos convertidos para o TensorFlow Lite e fizemos a comparação dos modelos.

A avaliação do consumo de memória dos modelos convertidos é especialmente relevante quando consideramos as limitações do Arduino Nano. Este microcontrolador utiliza o nRF52840, que possui 256 kilobytes de RAM. [18]

3.5 Definições

A seguir, apresentamos as definições iniciais com as quais foram treinados os modelos. Estas informações detalham aspectos cruciais, desde o tamanho do lote até a arquitetura específica da rede neural, evidenciando as escolhas cuidadosas feitas para otimizar o desempenho e evitar o sobreajuste.

1. **Batch Size:** O Batch Size(Tamanho do Lote) utilizado em todos os modelos é de 20 imagens por lote. Isso afeta a dimensão de entrada dos modelos durante o treinamento, que serão executadas simultâneamente.

2. **Resoluções Utilizadas:** As resoluções das imagens de entrada usadas são: 32x32, 48x48, 64x64 e 128x128. Além disso, tanto imagens coloridas (3 canais) quanto imagens em escala de cinza (1 canal) foram utilizadas.

3. **Limite de Épocas e EarlyStopping:** O treinamento do modelo foi configurado com um limite máximo de 100 épocas. No entanto, a técnica de EarlyStopping foi empregada para interromper o treinamento se a perda de validação (val_loss) não diminuir por 10 épocas consecutivas. Se a perda diminuir, são dadas mais 10 épocas para avaliar se o modelo melhora. A opção restore_best_weights = True garante que os melhores pesos sejam restaurados.

4. **Arquitetura da Rede Neural** Foram utilizadas camadas convolucionais bidimensionais (*Conv2D*) e camadas de *MaxPooling2D* para redução da dimensionalidade. Após as camadas de convolução e *maxpooling*, há uma camada *Flatten* para achatar as dimensões, que nos retorna um vetor unidimensional. Uma camada de *Dropout* é aplicada antes da camada de saída para evitar overfitting. O *Dropout* desativa aleatoriamente 50% dos neurônios nesta camada. E por fim, a camada densa conecta os neurônios anteriores a cada neurônio da camada atual e realiza uma classificação final.

4

Resultados e Discussões

4.1 Modelo 32×32 RGB

4.1.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional proposta, o modelo possui uma entrada no formato (Batch_Size, 32, 32, 3), onde Batch_Size é 20, representando o número de imagens processadas simultaneamente. A resolução de cada imagem é de 32×32 pixels, e a última dimensão é 3, indicando que as imagens são coloridas (RGB).

As camadas mais internas incluem camadas convolucionais 2D, seguidas por camadas de MaxPooling2D. Essas camadas convolucionais são responsáveis por extrair características importantes das imagens, enquanto as camadas de MaxPooling2D reduzem a dimensionalidade da representação.

Posteriormente, há uma camada Flatten que transforma a saída das camadas convolucionais em um vetor unidimensional. Essa operação é necessária antes de alimentar os dados para as camadas densas.

Uma camada Dropout com uma taxa de 50% é introduzida para ajudar a prevenir o overfitting. O Dropout desativa aleatoriamente metade dos neurônios durante o treinamento.

A última camada do modelo é uma camada densa com uma única unidade de saída, indicando uma tarefa de classificação binária. A dimensão final da saída é (Batch_Size, 1), onde Batch_Size é novamente 20. A ativação dessa camada é sigmoid, sugerindo que o modelo está sendo treinado para classificar se as imagens apresentam o uso ou não de máscara.

A estrutura completa do modelo pode ser visualizada na figura referenciada como 4.1.

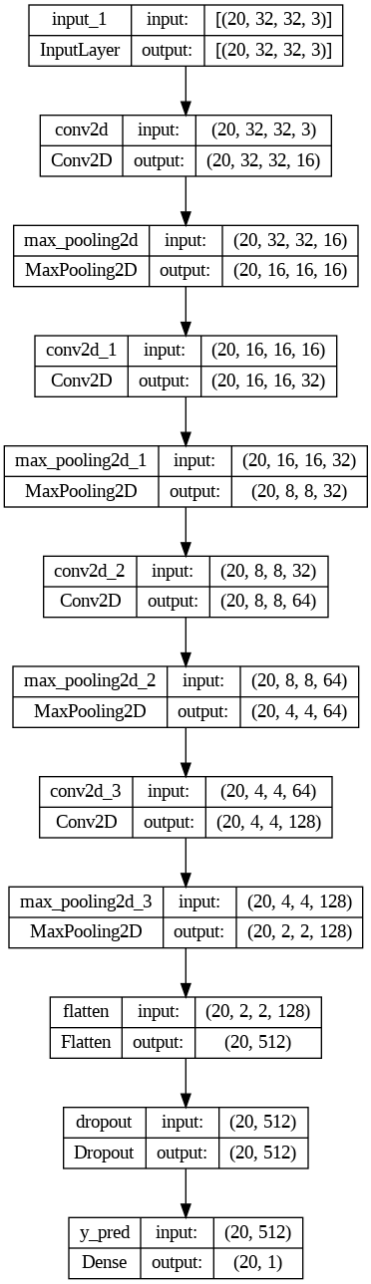

Figura 4.1: Arquitetura do modelo 32×32 RGB.

4.1.2 Entrada do Modelo

Para este modelo, empregamos imagens com dimensão 32×32 e três canais (RGB). Algumas dessas imagens, acompanhadas de suas classes, podem ser visualizadas abaixo, conforme ilustra a figura 4.2.

Figura 4.2: Amostra de imagens com resolução 32×32 RGB.

4.1.3 Treinamento do Modelo

Foi inicialmente estabelecido um total de 100 épocas para o treinamento. No entanto, optamos por utilizar o *EarlyStopping* configurado com uma paciência de 10 épocas. Dessa forma, conforme observado na Figura 4.3, o modelo, treinado com imagens de dimensão 32×32 RGB, alcançou seu desempenho máximo na época 34, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.1 e na figura 4.3.

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.9087	0.2383	0.9750	0.0528
2	0.9643	0.0987	0.9800	0.0541
3	0.9745	0.0827	0.9760	0.0643
4	0.9768	0.0676	0.9860	0.0384
5	0.9827	0.0629	0.9820	0.0468
6	0.9845	0.0525	0.9890	0.0307
7	0.9845	0.0502	0.9900	0.0295
8	0.9893	0.0413	0.9800	0.0580
9	0.9872	0.0346	0.9890	0.0266
10	0.9918	0.0293	0.9870	0.0309
11	0.9912	0.0294	0.9920	0.0202
12	0.9920	0.0261	0.9970	0.0179
13	0.9918	0.0206	0.9950	0.0215
14	0.9915	0.0203	0.9910	0.0193
15	0.9955	0.0160	0.9830	0.0576
16	0.9912	0.0248	0.9950	0.0179
17	0.9955	0.0158	0.9920	0.0173
18	0.9965	0.0114	0.9910	0.0255
19	0.9948	0.0147	0.9910	0.0183
20	0.9970	0.0092	0.9870	0.0467
21	0.9962	0.0137	0.9920	0.0148
22	0.9945	0.0151	0.9930	0.0207
23	0.9965	0.0101	0.9710	0.0818
24	0.9950	0.0137	0.9970	0.0076
25	0.9965	0.0096	0.9910	0.0237
26	0.9965	0.0135	0.9910	0.0294
27	0.9977	0.0067	0.9950	0.0109
28	0.9975	0.0073	0.9960	0.0131
29	0.9970	0.0095	0.9860	0.0361
30	0.9970	0.0085	0.9950	0.0195
31	0.9970	0.0084	0.9970	0.0077
32	0.9940	0.0132	0.9840	0.0527
33	0.9967	0.0099	0.9940	0.0241
34	0.9983	0.0053	0.9860	0.0406

Tabela 4.1: Tabela de treinamento e validação ao longo das épocas do modelo 32×32 RGB.

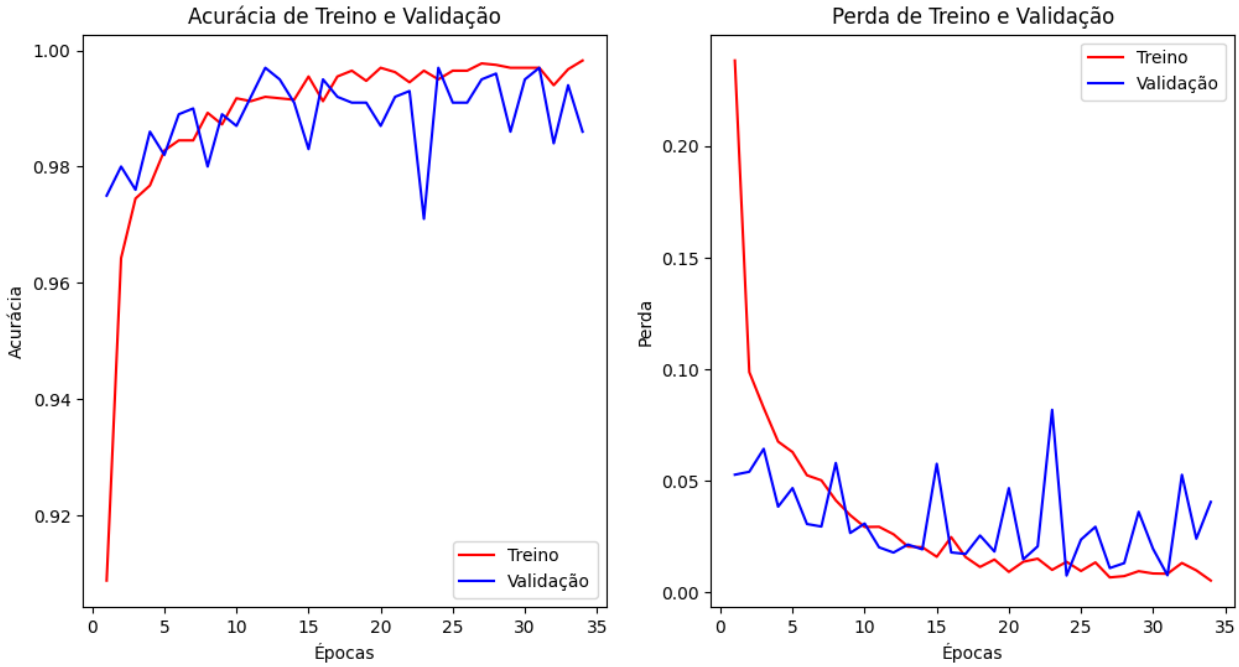

Figura 4.3: Evolução da acurácia e perda durante o treino e validação do modelo 32×32 RGB.

4.1.4 Teste do Modelo

Durante a fase de teste do modelo, os resultados foram os seguintes:

- **Acurácia no Teste:** 98.14%
- **Perda no Teste:** 0.0481

4.2 Modelo 32×32 Grayscale

4.2.1 Arquitetura do Modelo

Acerca da estrutura da rede neural convolucional proposta, o modelo apresenta uma arquitetura semelhante à da seção anterior, com uma pequena modificação. A entrada é definida no formato (Batch_Size, 32, 32, 1), onde Batch_Size é fixado em 20, representando o número de imagens processadas simultaneamente. Cada imagem possui uma resolução de 32×32 pixels, sendo a principal diferença observada na última dimensão, que é igual a 1, indicando que as imagens estão em escala de cinza.

A estrutura completa do modelo pode ser visualizada na figura referenciada como 4.4.

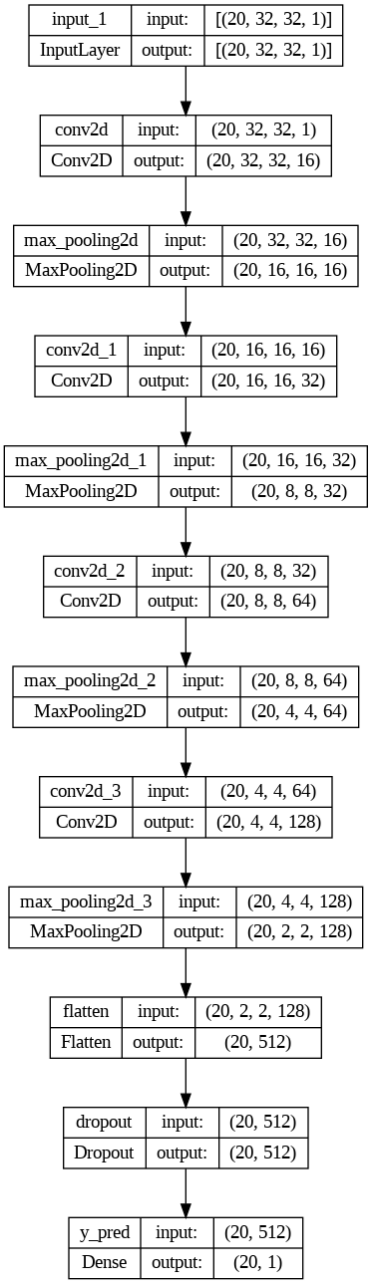

Figura 4.4: Arquitetura do modelo 32×32 Grayscale.

4.2.2 Entrada do Modelo

Para este modelo, as imagens possuem resolução 32×32 em escala de cinza, como visto na figura 4.5.

Figura 4.5: Amostra de imagens com resolução 32×32 Grayscale.

4.2.3 Treinamento do Modelo

Foi utilizado os mesmos parâmetros da seção anterior, porém com as imagens com resolução de 32×32 Grayscale. O modelo alcançou o desempenho máximo na época 23, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.2 e na figura 4.6.

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.7253	0.5385	0.9010	0.2833
2	0.8740	0.3031	0.9060	0.2252
3	0.9097	0.2252	0.9330	0.1814
4	0.9212	0.1850	0.9340	0.1776
5	0.9388	0.1605	0.9400	0.1674
6	0.9408	0.1463	0.9430	0.1542
7	0.9523	0.1182	0.9420	0.1491
8	0.9588	0.1107	0.9390	0.1552
9	0.9635	0.0979	0.9520	0.1269
10	0.9582	0.1005	0.9360	0.1673
11	0.9722	0.0804	0.9530	0.1402
12	0.9675	0.0882	0.9490	0.1210
13	0.9707	0.0789	0.9610	0.0986
14	0.9745	0.0670	0.9500	0.1179
15	0.9728	0.0756	0.9560	0.1326
16	0.9730	0.0753	0.9510	0.1117
17	0.9775	0.0613	0.9540	0.1437
18	0.9770	0.0586	0.9480	0.1707
19	0.9793	0.0597	0.9570	0.1147
20	0.9803	0.0518	0.9590	0.1115
21	0.9800	0.0564	0.9560	0.1141
22	0.9818	0.0512	0.9590	0.1256
23	0.9785	0.0557	0.9600	0.1053

Tabela 4.2: Tabela de treinamento e validação ao longo das épocas do modelo 32×32 Grayscale.

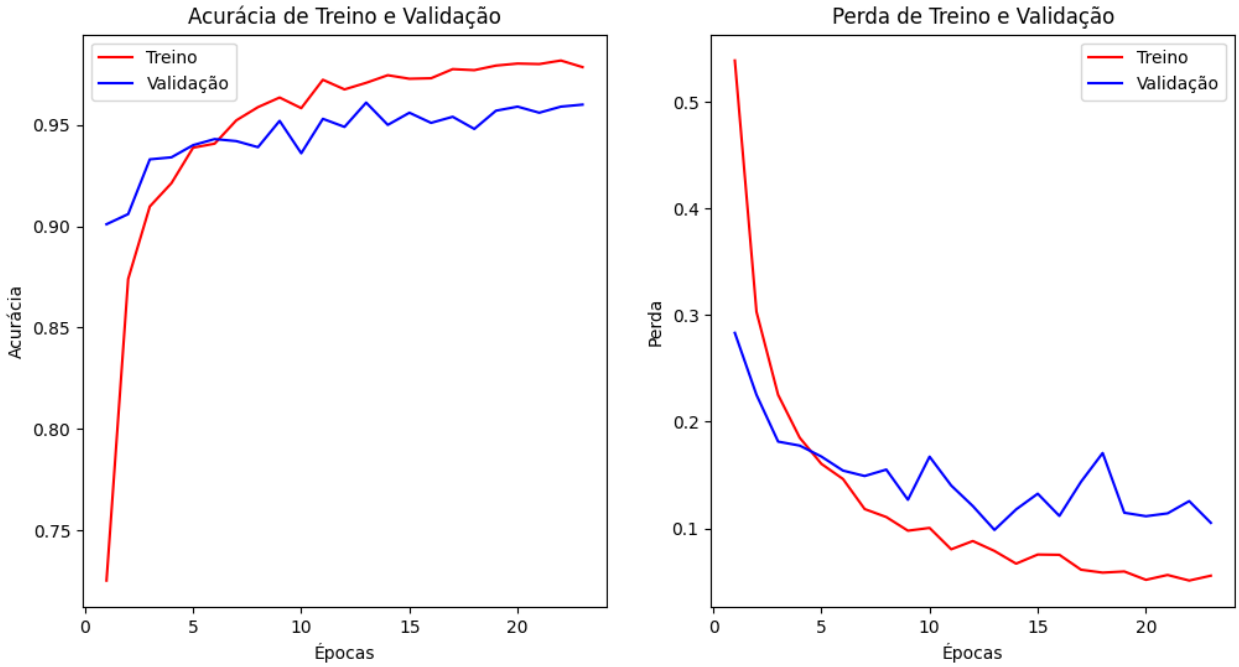

Figura 4.6: Evolução da acurácia e perda durante o treino e validação do modelo 32×32 Grayscale.

4.2.4 Teste do Modelo

Durante a avaliação do modelo, os resultados obtidos foram os seguintes:

- **Acurácia no Teste:** 96.11%
- **Perda no Teste:** 0.0942

4.3 Modelo 48×48 RGB

4.3.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é igual à arquitetura dos modelos acima com cada imagem com resolução de 48×48 e três camadas que fazem com que a imagem seja colorida (RGB), podemos ver isso na figura 4.7.

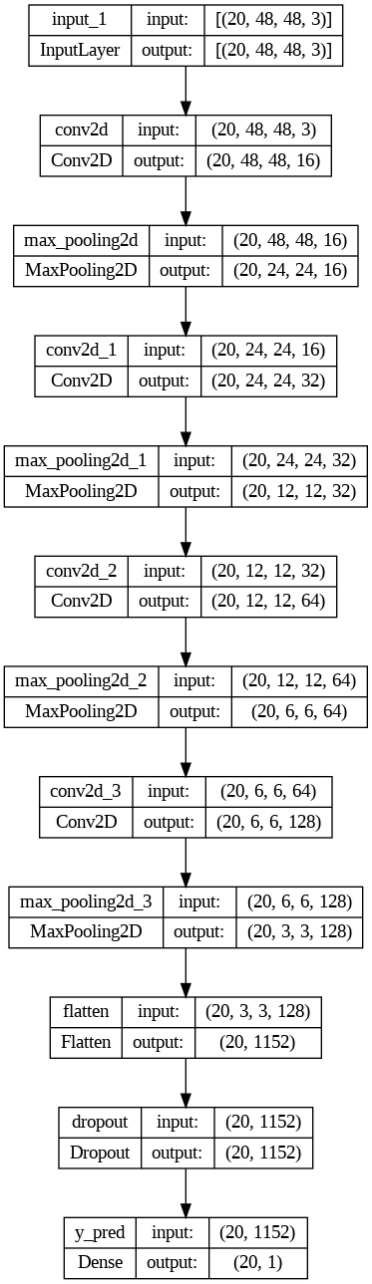

Figura 4.7: Arquitetura do modelo 48×48 RGB.

4.3.2 Entrada do Modelo

Para este modelo, as imagens possuem resolução 48×48 em RGB, como visto na figura 4.8.

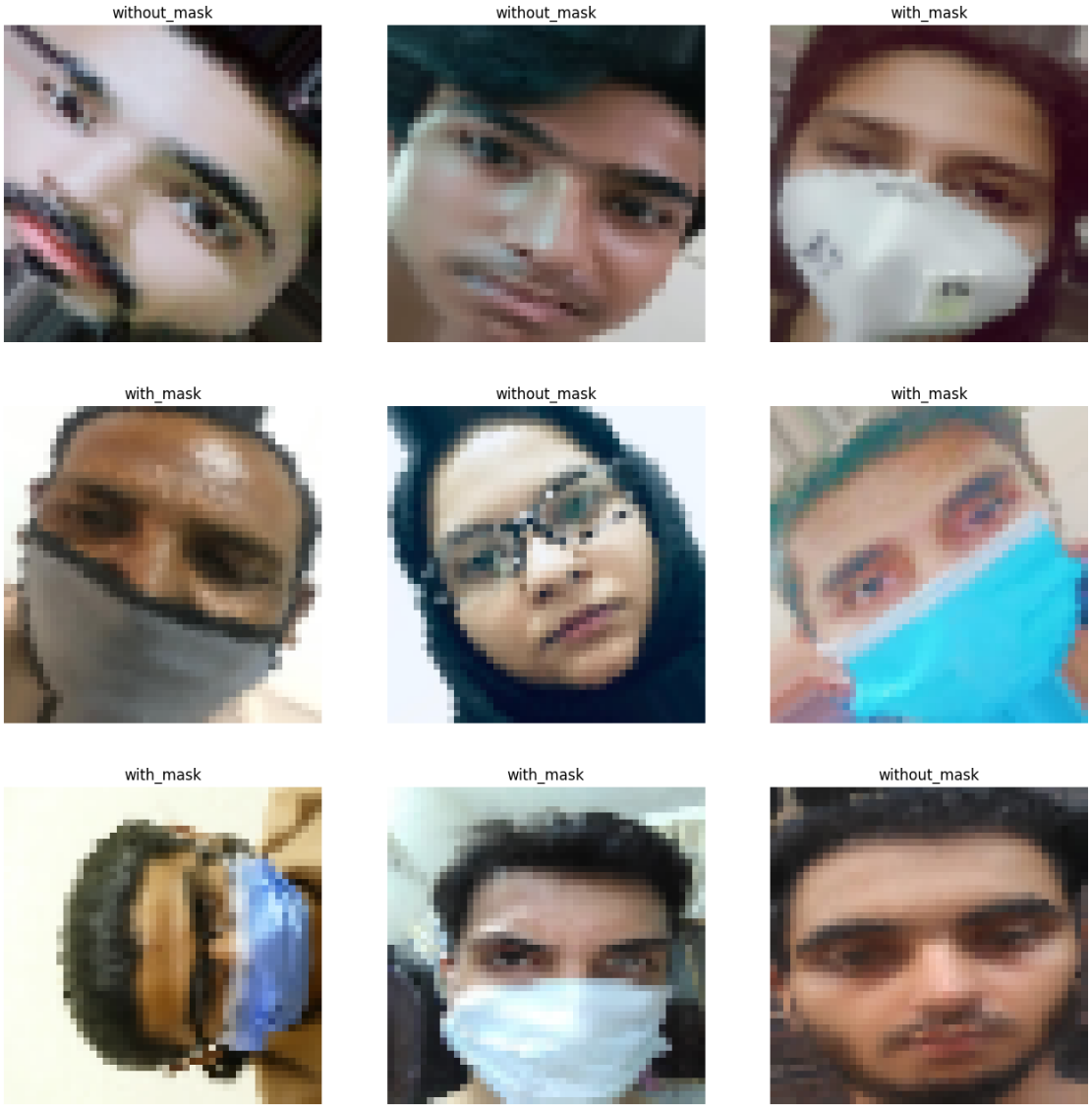

Figura 4.8: Amostra de imagens com resolução 48×48 RGB.

4.3.3 Treinamento do Modelo

Foi utilizado os mesmos parâmetros das seções anteriores, porém com as imagens com resolução de 48×48 RGB. O modelo alcançou o desempenho máximo na época 37, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.3 e na figura 4.9

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.9110	0.2117	0.9680	0.1134
2	0.9710	0.0883	0.9810	0.0543
3	0.9768	0.0732	0.9720	0.0642
4	0.9833	0.0623	0.9860	0.0361
5	0.9847	0.0510	0.9870	0.0324
6	0.9850	0.0506	0.9860	0.0290
7	0.9847	0.0489	0.9890	0.0308
8	0.9865	0.0453	0.9830	0.0464
9	0.9877	0.0396	0.9880	0.0309
10	0.9885	0.0362	0.9750	0.0798
11	0.9893	0.0377	0.9890	0.0281
12	0.9898	0.0288	0.9930	0.0196
13	0.9893	0.0328	0.9910	0.0259
14	0.9915	0.0260	0.9920	0.0215
15	0.9927	0.0240	0.9930	0.0221
16	0.9908	0.0233	0.9910	0.0304
17	0.9923	0.0265	0.9920	0.0266
18	0.9935	0.0185	0.9900	0.0271
19	0.9918	0.0217	0.9930	0.0271
20	0.9935	0.0174	0.9950	0.0114
21	0.9948	0.0156	0.9870	0.0261
22	0.9942	0.0166	0.9870	0.0239
23	0.9948	0.0152	0.9910	0.0368
24	0.9955	0.0131	0.9890	0.0350
25	0.9965	0.0109	0.9890	0.0447
26	0.9945	0.0160	0.9900	0.0238
27	0.9948	0.0142	0.9970	0.0094
28	0.9952	0.0132	0.9950	0.0166
29	0.9962	0.0084	0.9940	0.0197
30	0.9952	0.0146	0.9920	0.0200
31	0.9967	0.0098	0.9840	0.0498
32	0.9948	0.0121	0.9890	0.0217
33	0.9950	0.0134	0.9970	0.0133
34	0.9985	0.0046	0.9930	0.0200
35	0.9952	0.0184	0.9920	0.0183
36	0.9965	0.0096	0.9940	0.0261
37	0.9942	0.0155	0.9980	0.0122

Tabela 4.3: Tabela de treinamento e validação ao longo das épocas do modelo 48×48 RGB.

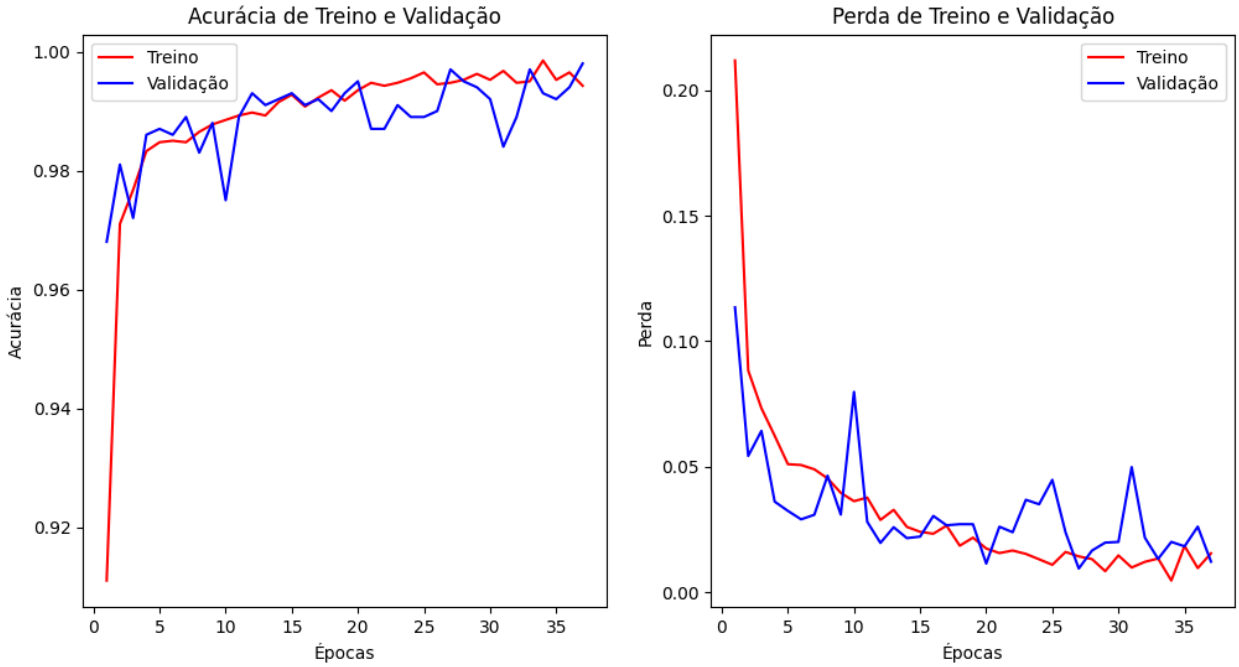

Figura 4.9: Evolução da acurácia e perda durante o treino e validação do modelo 48×48 RGB.

4.3.4 Teste do Modelo

Durante a fase de teste, o modelo apresentou os seguintes resultados:

- **Acurácia no Teste:** 97.30%
- **Perda no Teste:** 0.0940

4.4 Modelo 48×48 Grayscale

4.4.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é igual a arquitetura das seções anteriores, porém, com resolução 48×48 e em escala cinza. Podemos ver mais detalhes na figura 4.10.

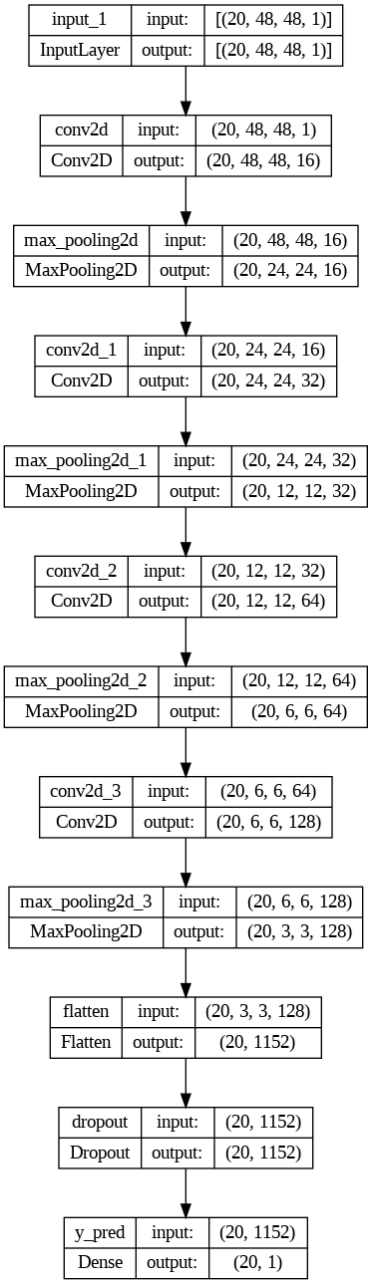

Figura 4.10: Arquitetura do modelo 48×48 Grayscale.

4.4.2 Entrada do Modelo

Para este modelo, as imagens possuem resolução 48×48 em escala de cinza, como visto na figura 4.11.

Figura 4.11: Amostra de imagens com resolução 48×48 Grayscale.

4.4.3 Treinamento do Modelo

Foi utilizado os mesmos parâmetros das seções anteriores, porém com as imagens com resolução de 48×48 em escala de cinza. O modelo alcançou o desempenho máximo na época 77, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.4 e na figura 4.12

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.7253	0.5385	0.9010	0.2833
2	0.8740	0.3031	0.9060	0.2252
3	0.9097	0.2252	0.9330	0.1814
4	0.9212	0.1850	0.9340	0.1776
5	0.9388	0.1605	0.9400	0.1674
6	0.9408	0.1463	0.9430	0.1542
7	0.9523	0.1182	0.9420	0.1491
8	0.9588	0.1107	0.9390	0.1552
9	0.9635	0.0979	0.9520	0.1269
10	0.9582	0.1005	0.9360	0.1673
...	...	...	...	...
61	0.0171	0.9923	0.0721	0.9720
62	0.0123	0.9950	0.0546	0.9810
63	0.0201	0.9923	0.0670	0.9720
64	0.0159	0.9945	0.0388	0.9810
65	0.0108	0.9967	0.0389	0.9880
66	0.0139	0.9950	0.0440	0.9850
67	0.0198	0.9935	0.0333	0.9840
68	0.0128	0.9958	0.0380	0.9840
69	0.0118	0.9962	0.0372	0.9890
70	0.0159	0.9958	0.0507	0.9770
71	0.0181	0.9942	0.0484	0.9850
72	0.0171	0.9930	0.0455	0.9810
73	0.0112	0.9967	0.0537	0.9770
74	0.0143	0.9945	0.0369	0.9870
75	0.0107	0.9960	0.0417	0.9830
76	0.0104	0.9962	0.0342	0.9860
77	0.0162	0.9942	0.0771	0.9710

Tabela 4.4: Tabela de treinamento e validação ao longo das épocas do modelo 48×48 Grayscale.

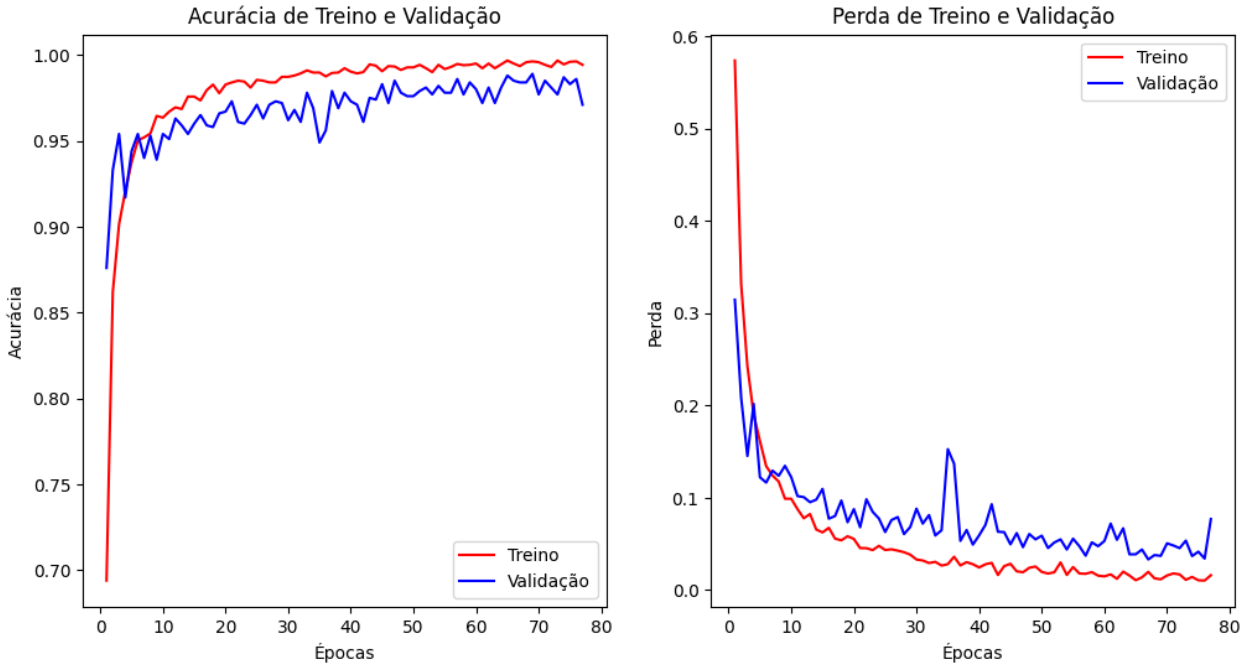

Figura 4.12: Evolução da acurácia e perda durante o treino e validação do modelo 48×48 Grayscale.

4.4.4 Teste do Modelo

Durante a fase de teste, o modelo apresentou os seguintes resultados:

- **Acurácia no Teste:** 98.82%
- **Perda no Teste:** 0.0387

4.5 Modelo 64×64 RGB

4.5.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é igual a arquitetura das seções anteriores, porém, com resolução 64×64 e em RGB. Podemos ver mais detalhes na figura 4.13.

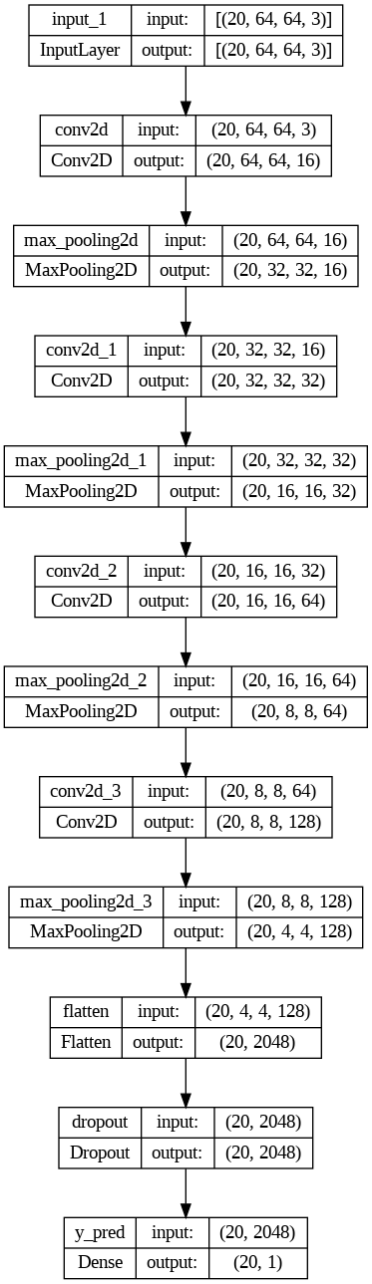

Figura 4.13: Arquitetura do modelo 64×64 RGB.

4.5.2 Entrada do Modelo

Para este modelo, as imagens possuem resolução 64×64 em RGB, como visto na figura 4.14.

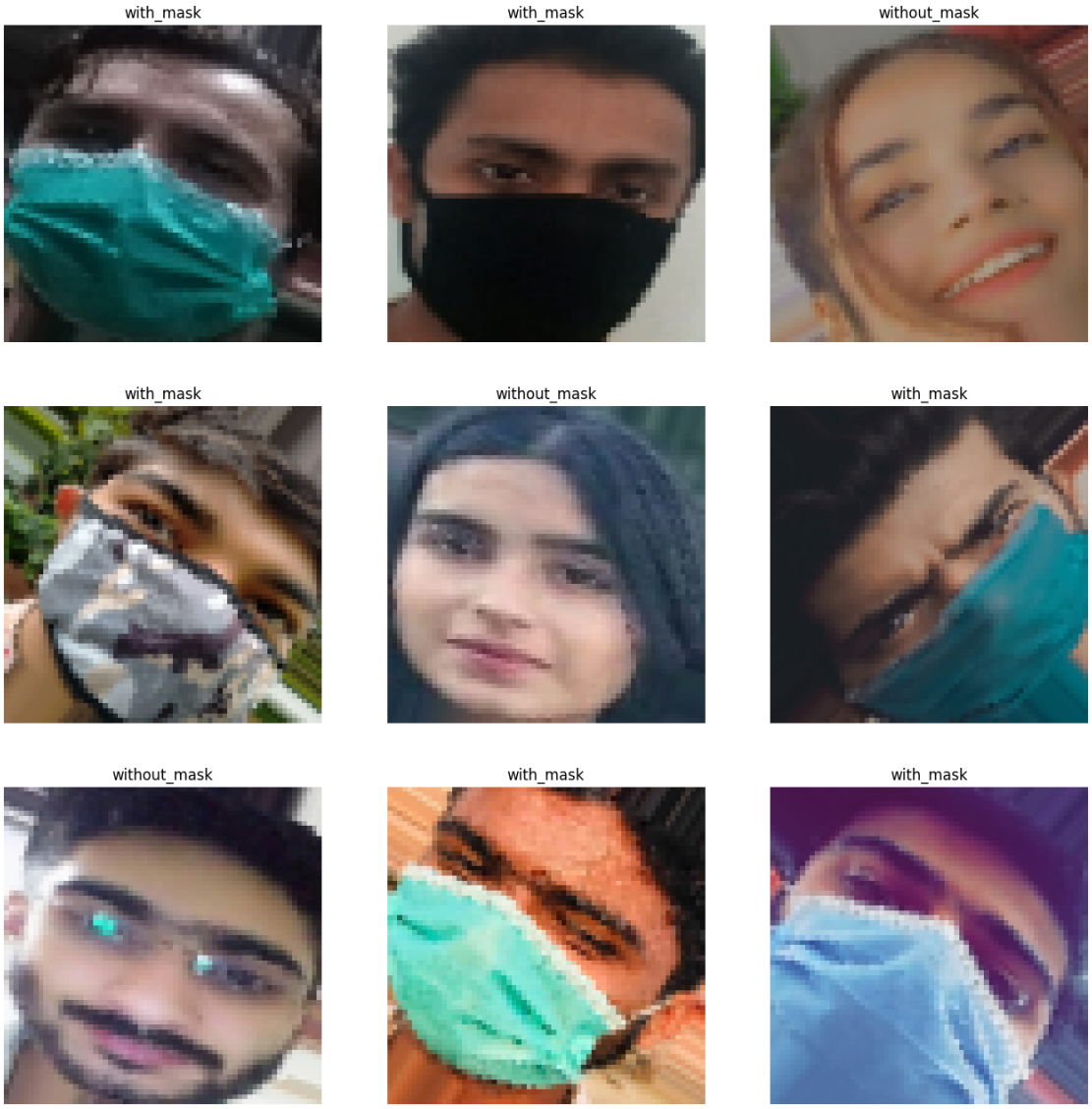

Figura 4.14: Amostra de imagens com resolução 64×64 RGB.

4.5.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, porém com as imagens com resolução de 64×64 em RGB. O modelo alcançou o desempenho máximo na época 34, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.5 e na figura 4.15.

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.7253	0.5385	0.9010	0.2833
2	0.8740	0.3031	0.9060	0.2252
3	0.9097	0.2252	0.9330	0.1814
4	0.9212	0.1850	0.9340	0.1776
5	0.9388	0.1605	0.9400	0.1674
6	0.9408	0.1463	0.9430	0.1542
7	0.9523	0.1182	0.9420	0.1491
8	0.9588	0.1107	0.9390	0.1552
9	0.9635	0.0979	0.9520	0.1269
10	0.9582	0.1005	0.9360	0.1673
11	0.9722	0.0804	0.9530	0.1402
12	0.9675	0.0882	0.9490	0.1210
13	0.9707	0.0789	0.9610	0.0986
14	0.9745	0.0670	0.9500	0.1179
15	0.9728	0.0756	0.9560	0.1326
16	0.9730	0.0753	0.9510	0.1117
17	0.9775	0.0613	0.9540	0.1437
18	0.9770	0.0586	0.9480	0.1707
19	0.9793	0.0597	0.9570	0.1147
20	0.9803	0.0518	0.9590	0.1115
21	0.9800	0.0564	0.9560	0.1141
22	0.9818	0.0512	0.9590	0.1256
23	0.9785	0.0557	0.9600	0.1053
24	0.9922	0.0120	0.9930	0.0149
25	0.9922	0.0134	0.9830	0.0641
26	0.9885	0.0170	0.9940	0.0270
27	0.9910	0.0153	0.9930	0.0273
28	0.9945	0.0102	0.9790	0.0855
29	0.9973	0.0078	0.9860	0.0469
30	0.9937	0.0158	0.9920	0.0174
31	0.9958	0.0127	0.9930	0.0206
32	0.9948	0.0144	0.9930	0.0215
33	0.9970	0.0096	0.9920	0.0186
34	0.9990	0.0035	0.9960	0.0247

Tabela 4.5: Tabela de treinamento e validação ao longo das épocas do modelo 64×64 RGB.

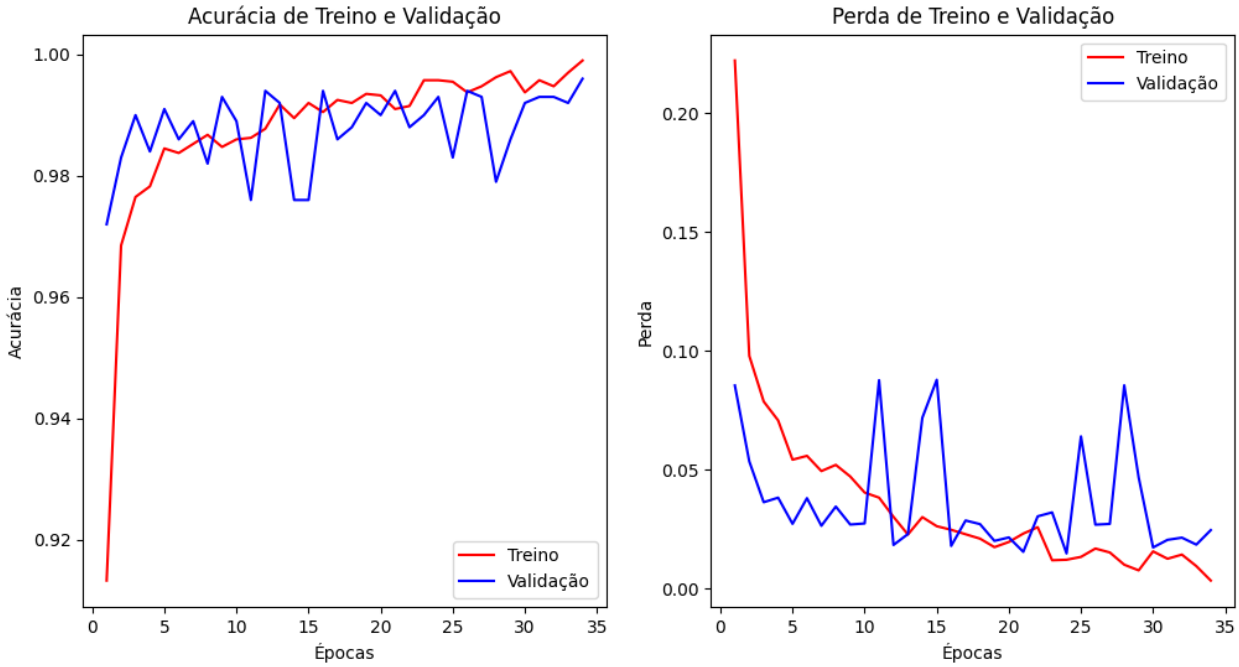

Figura 4.15: Evolução da acurácia e perda durante o treino e validação do modelo 64×64 RGB.

4.5.4 Teste do Modelo

Durante a fase de teste, o modelo apresentou os seguintes resultados:

- **Acurácia no Teste:** 97.13%
- **Perda no Teste:** 0.1002

4.6 Modelo 64×64 Grayscale

4.6.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é igual a arquitetura das seções anteriores, porém, com resolução de 64×64 e em escala cinza. Podemos ter uma visão mais detalhada do modelo na figura 4.16.

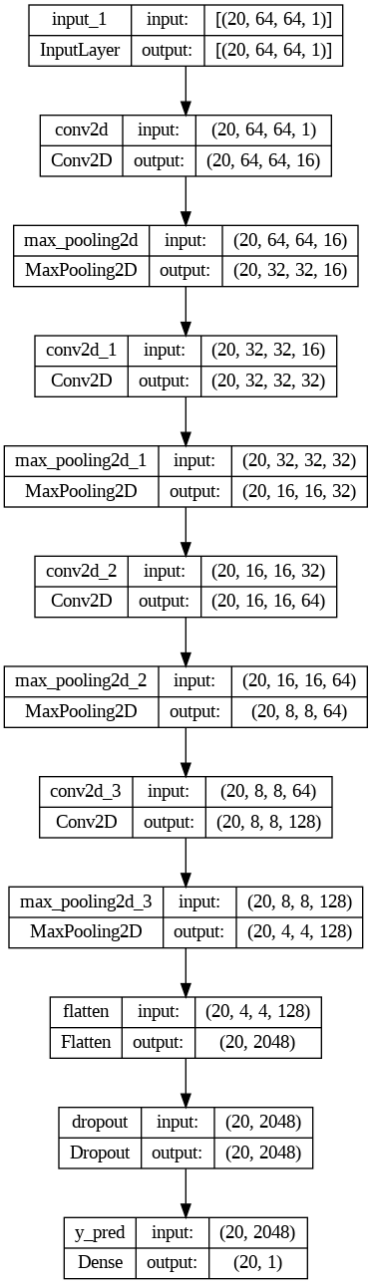

Figura 4.16: Arquitetura do modelo 64×64 Grayscale.

4.6.2 Entrada do Modelo

Para este modelo, as imagens tem resolução de 64×64 e estão em escala de cinza, como visto na figura 4.17.

Figura 4.17: Amostra de imagens com resolução 64×64 Grayscale.

4.6.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, porém com as imagens com resolução de 64×64 e em escala de cinza. O modelo alcançou o desempenho máximo na época 41, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.6 e na figura 4.18.

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.6810	0.5821	0.8740	0.3528
2	0.8702	0.3357	0.9300	0.2221
3	0.9053	0.2310	0.9290	0.1509
4	0.9302	0.1770	0.9230	0.1947
5	0.9362	0.1513	0.9320	0.1920
6	0.9440	0.1364	0.9520	0.1056
7	0.9492	0.1233	0.9460	0.1363
8	0.9603	0.1091	0.9640	0.0879
9	0.9617	0.0981	0.9620	0.0979
10	0.9672	0.0892	0.9570	0.1063
11	0.9693	0.0897	0.9580	0.1059
12	0.9695	0.0765	0.9760	0.0653
13	0.9668	0.0914	0.9520	0.1256
14	0.9758	0.0685	0.9590	0.0917
15	0.9750	0.0652	0.9610	0.0848
16	0.9778	0.0597	0.9620	0.0925
17	0.9783	0.0573	0.9620	0.0937
18	0.9790	0.0547	0.9600	0.1221
19	0.9803	0.0570	0.9710	0.0791
20	0.9835	0.0470	0.9720	0.0748
21	0.9810	0.0520	0.9750	0.0640
22	0.9877	0.0375	0.9430	0.1335
23	0.9855	0.0468	0.9590	0.0817
24	0.9855	0.0404	0.9670	0.0750
25	0.9862	0.0426	0.9720	0.0700
26	0.9858	0.0420	0.9690	0.0820
27	0.9893	0.0366	0.9790	0.0542
28	0.9868	0.0364	0.9660	0.0697
29	0.9865	0.0371	0.9740	0.0564
30	0.9887	0.0284	0.9590	0.1091
31	0.9900	0.0268	0.9870	0.0398
32	0.9870	0.0341	0.9740	0.0639
33	0.9877	0.0316	0.9810	0.0482
34	0.9905	0.0285	0.9750	0.0623
35	0.9872	0.0337	0.9670	0.0801
36	0.9902	0.0262	0.9620	0.1090
37	0.9915	0.0268	0.9680	0.0718
38	0.9920	0.0268	0.9770	0.0675
39	0.9933	0.0195	0.9840	0.0431
40	0.9927	0.0205	0.9810	0.0541
41	0.9933	0.0189	0.9710	0.0790

Tabela 4.6: Tabela de treinamento e validação ao longo das épocas do modelo 64×64 Grayscale.

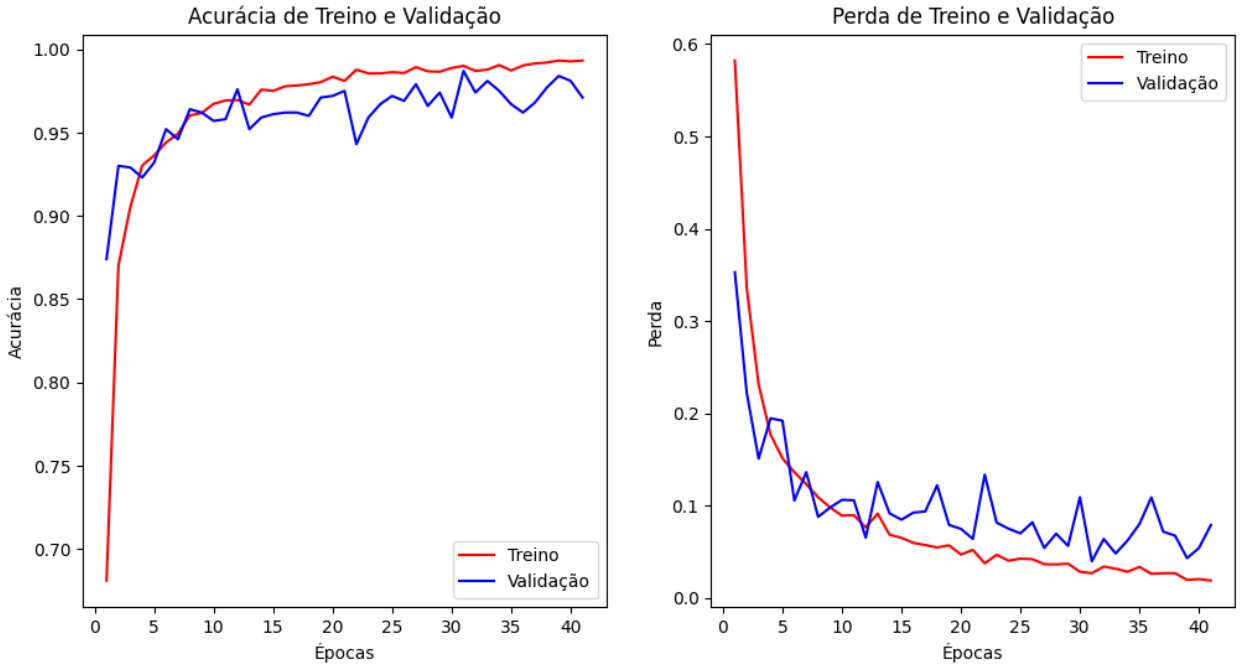

Figura 4.18: Evolução da acurácia e perda durante o treino e validação do modelo 64×64 Grayscale.

4.6.4 Teste do Modelo

Durante a fase de teste, o modelo apresentou os seguintes resultados:

- **Acurácia no Teste:** 98.65%
- **Perda no Teste:** 0.0356

4.7 Modelo 128×128 RGB

4.7.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é igual a arquitetura das seções anteriores, porém, com resolução de 128×128 em RGB. Podemos ter uma visão mais detalhada do modelo na figura 4.19.

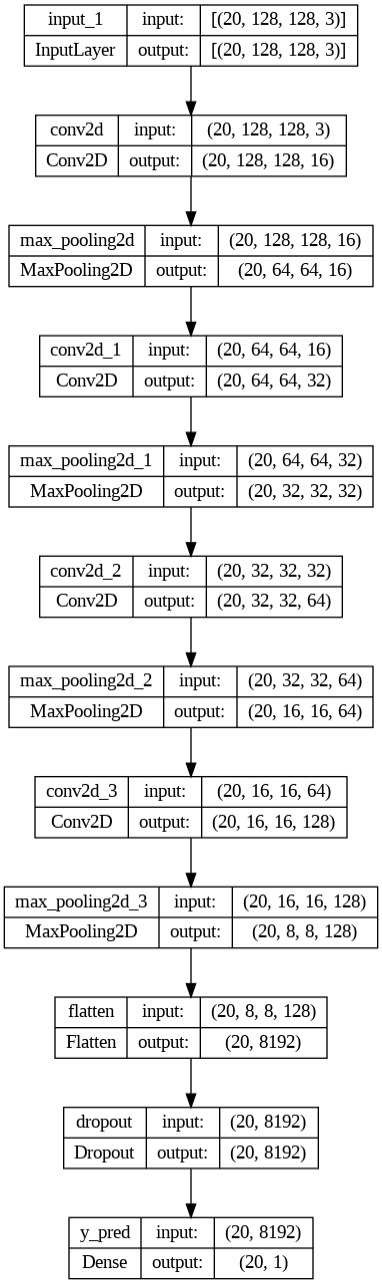

Figura 4.19: Arquitetura do modelo 128×128 RGB.

4.7.2 Entrada do Modelo

Para este modelo, as imagens possuem resolução 128×128 em RGB, como visto na figura 4.20.

Figura 4.20: Amostra de imagens com resolução 128×128 RGB.

4.7.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, porém com as imagens com resolução de 128×128 em RGB. O modelo alcançou o desempenho máximo na época 42, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.7 e na figura 4.21.

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.9283	0.1788	0.9820	0.0540
2	0.9783	0.0829	0.9820	0.0372
3	0.9833	0.0611	0.9750	0.0545
4	0.9805	0.0613	0.9780	0.0491
5	0.9840	0.0568	0.9910	0.0314
6	0.9847	0.0518	0.9900	0.0311
7	0.9855	0.0471	0.9900	0.0249
8	0.9855	0.0468	0.9900	0.0400
9	0.9887	0.0367	0.9710	0.0738
10	0.9887	0.0328	0.9690	0.0496
11	0.9883	0.0377	0.9930	0.0358
12	0.9855	0.0361	0.9750	0.0557
13	0.9887	0.0322	0.9980	0.0195
14	0.9918	0.0284	0.9910	0.0269
15	0.9910	0.0273	0.9910	0.0215
16	0.9908	0.0270	0.9940	0.0280
17	0.9925	0.0238	0.9910	0.0209
18	0.9948	0.0180	0.9950	0.0265
19	0.9923	0.0207	0.9900	0.0252
20	0.9925	0.0262	0.9920	0.0199
21	0.9940	0.0196	0.9830	0.0415
22	0.9950	0.0161	0.9930	0.0175
23	0.9940	0.0226	0.9920	0.0226
24	0.9958	0.0151	0.9960	0.0208
25	0.9940	0.0171	0.9940	0.0242
26	0.9930	0.0213	0.9890	0.0264
27	0.9952	0.0170	0.9880	0.0246
28	0.9948	0.0155	0.9920	0.0298
29	0.9945	0.0174	0.9880	0.0252
30	0.9965	0.0094	0.9810	0.0395
31	0.9945	0.0145	0.9860	0.0341
32	0.9965	0.0119	0.9960	0.0135
33	0.9930	0.0204	0.9900	0.0288
34	0.9980	0.0074	0.9960	0.0223
35	0.9937	0.0164	0.9950	0.0220
36	0.9977	0.0096	0.9930	0.0258
37	0.9958	0.0151	0.9930	0.0177
38	0.9973	0.0100	0.9880	0.0345
39	0.9962	0.0105	0.9980	0.0171
40	0.9967	0.0105	0.9920	0.0328
41	0.9942	0.0187	0.9910	0.0290
42	0.9948	0.0147	0.9920	0.0238

Tabela 4.7: Tabela de treinamento e validação ao longo das épocas do modelo 128×128 RGB.

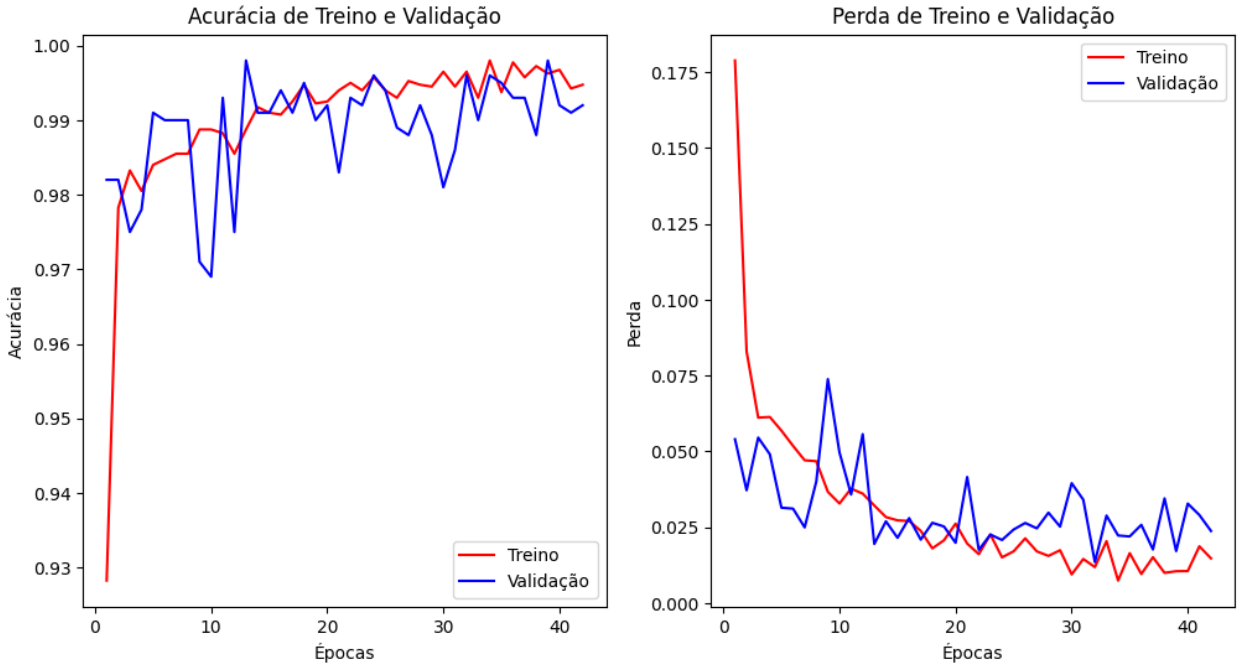

Figura 4.21: Evolução da acurácia e perda durante o treino e validação do modelo 128×128 RGB.

4.7.4 Teste do Modelo

Durante a fase de teste, o modelo apresentou os seguintes resultados:

- **Acurácia no Teste:** 96.96%
- **Perda no Teste:** 0.0872

4.8 Modelo 128×128 Grayscale

4.8.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é igual a arquitetura das seções anteriores, porém, com resolução de 128×128 e em escala de cinza. Podemos ter uma visão mais detalhada do modelo na figura 4.22.

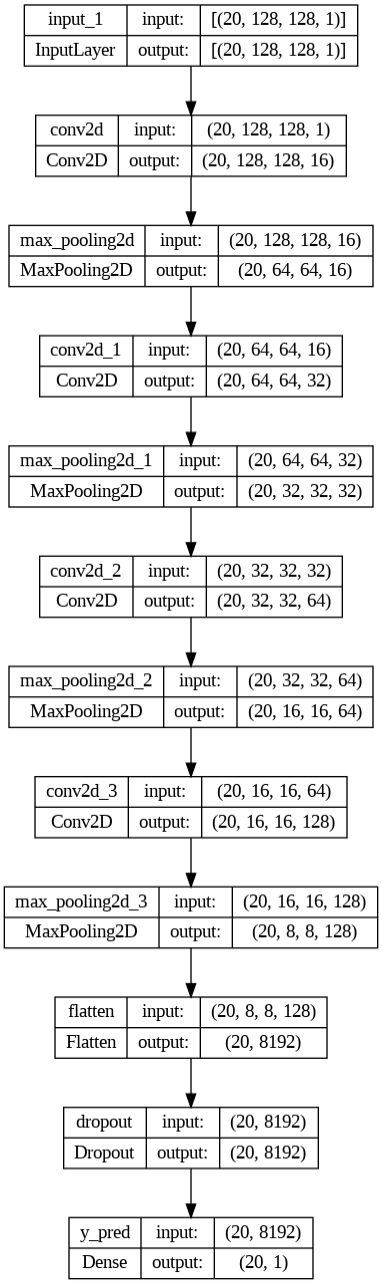

Figura 4.22: Arquitetura do modelo 128×128 Grayscale.

4.8.2 Entrada do Modelo

Para este modelo, as imagens possuem resolução 128×128 em escala de cinza, como visto na figura 4.23.

Figura 4.23: Amostra de imagens com resolução 128×128 Grayscale.

4.8.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, porém com as imagens com resolução de 128×128 em escala de cinza. O modelo alcançou o desempenho máximo na época 32, segue os valores correspondentes a acurácia e perda no treinamento e validação do modelo na tabela 4.8 e na figura 4.24.

Época	Acurácia (Treinamento)	Perda (Treinamento)	Acurácia (Validação)	Perda (Validação)
1	0.7423	0.5131	0.8920	0.3142
2	0.8777	0.3151	0.9120	0.2255
3	0.9035	0.2425	0.9320	0.1740
4	0.9320	0.1798	0.9470	0.1394
5	0.9417	0.1477	0.9510	0.1215
6	0.9550	0.1297	0.9610	0.0959
7	0.9653	0.0997	0.9670	0.0958
8	0.9672	0.0944	0.9480	0.1101
9	0.9710	0.0792	0.9720	0.0796
10	0.9728	0.0750	0.9380	0.1459
11	0.9690	0.0825	0.9730	0.0715
12	0.9837	0.0551	0.9750	0.0624
13	0.9732	0.0686	0.9750	0.0659
14	0.9810	0.0513	0.9740	0.0600
15	0.9762	0.0625	0.9710	0.0804
16	0.9818	0.0568	0.9700	0.0759
17	0.9790	0.0593	0.9370	0.1607
18	0.9840	0.0474	0.9800	0.0568
19	0.9840	0.0450	0.9750	0.0514
20	0.9875	0.0386	0.9750	0.0546
21	0.9875	0.0342	0.9820	0.0443
22	0.9810	0.0451	0.9830	0.0398
23	0.9883	0.0370	0.9790	0.0533
24	0.9868	0.0417	0.9780	0.0549
25	0.9875	0.0348	0.9810	0.0461
26	0.9862	0.0384	0.9800	0.0495
27	0.9905	0.0281	0.9780	0.0440
28	0.9937	0.0216	0.9820	0.0455
29	0.9895	0.0293	0.9730	0.0626
30	0.9883	0.0330	0.9810	0.0455
31	0.9883	0.0304	0.9830	0.0482
32	0.9898	0.0311	0.9810	0.0471

Tabela 4.8: Tabela de treinamento e validação ao longo das épocas do modelo 128×128 Grayscale.

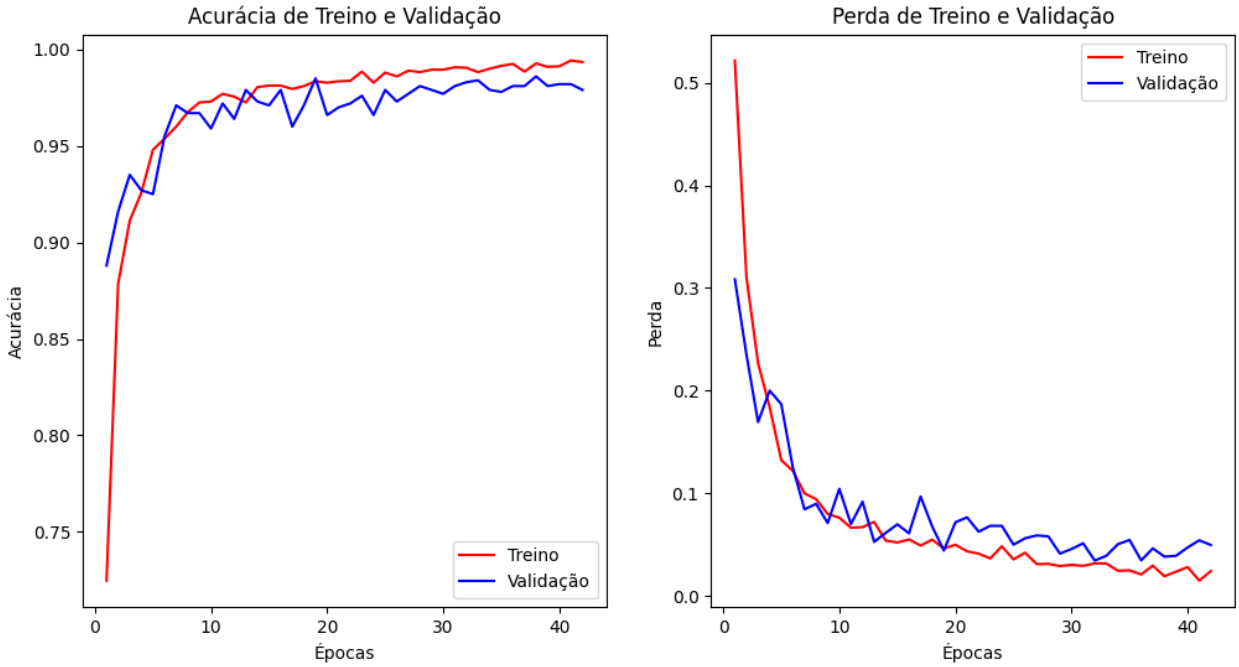

Figura 4.24: Evolução da acurácia e perda durante o treino e validação do modelo 128×128 Grayscale.

4.8.4 Teste do Modelo

Durante a fase de teste, o modelo apresentou os seguintes resultados:

- **Acurácia no Teste:** 97.97%
- **Perda no Teste:** 0.0794

4.9 Comparativo entre os modelos durante o teste

4.9.1 Acurácia

De forma geral, observamos que os modelos em escala de cinza (*Grayscale*) apresentaram uma acurácia superior em comparação com os modelos RGB nas resoluções testadas, como ilustrado na Figura 4.25. O modelo com a melhor acurácia foi o 48×48 *Grayscale*, atingindo 98.82%.

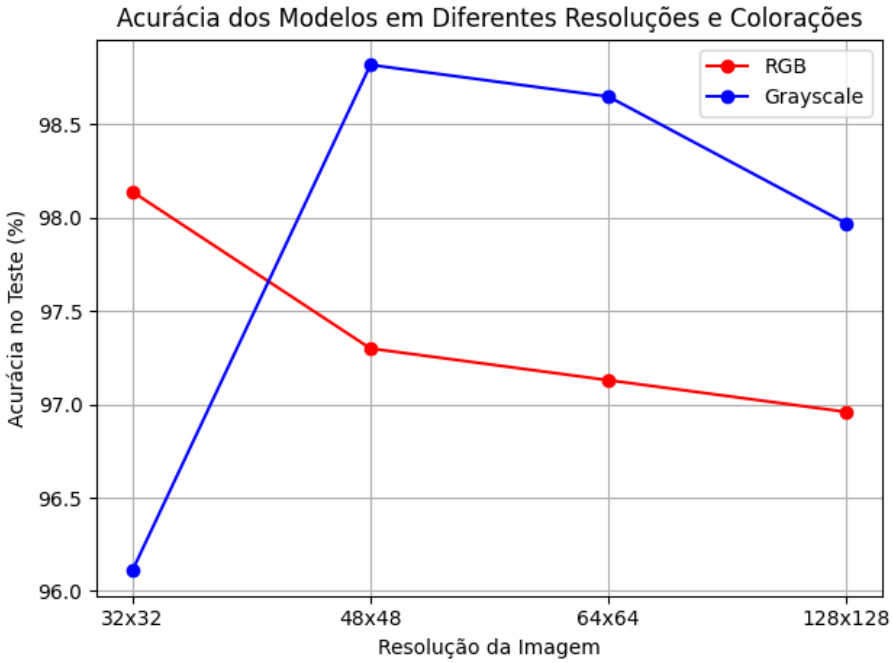

Figura 4.25: Gráfico de Acurácia dos modelos na fase de teste.

4.9.2 Perda

No que diz respeito à perda, os modelos *Grayscale* também demonstraram vantagens, sugerindo uma melhor capacidade de generalização, conforme evidenciado na Figura 4.26. O modelo com a menor perda foi o 64×64 *Grayscale*, registrando 0.0356.

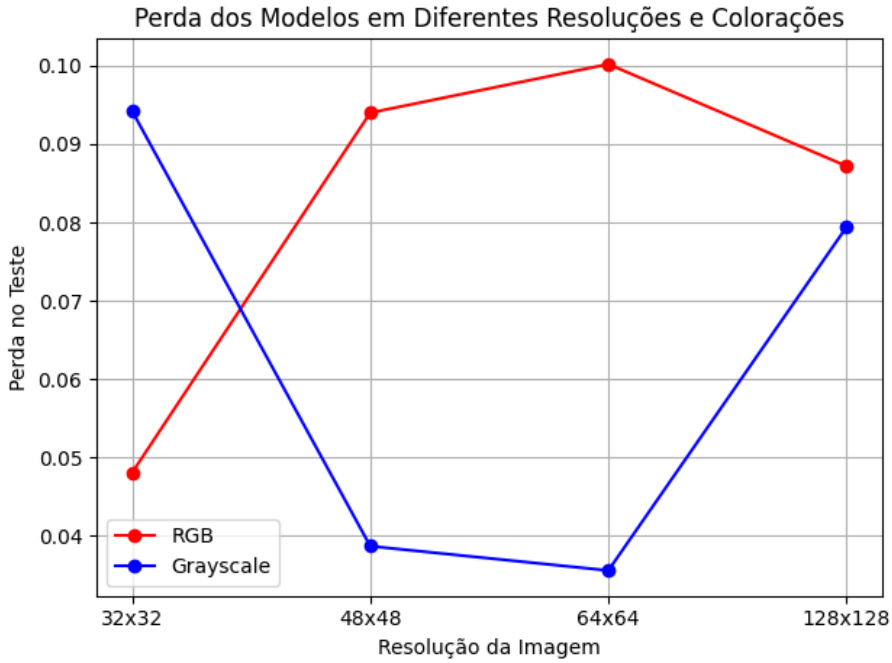

Figura 4.26: Gráfico de Perda dos modelos na fase de teste.

4.10 Comparação de desempenho dos modelos embarcados

Abaixo, apresentamos as acurácias de cada modelo em sua versão padrão e na versão quantizada tflite. Podemos observar a comparação na figura 4.27 e na figura 4.28.

4.10.1 32×32 RGB:

- Acurácia do Modelo Tensor Flow: 97.13%
- Acurácia do Modelo TFLite Quantizado: 96.45%
- Tamanho do Modelo Tensor Flow: 374.981 KB
- Tamanho do Modelo TFLite Quantizado: 105.758 KB

4.10.2 32×32 Grayscale:

- Acurácia do Modelo Tensor Flow: 96.96%
- Acurácia do Modelo TFLite Quantizado: 98.48%
- Tamanho do Modelo Tensor Flow: 374.452 KB
- Tamanho do Modelo TFLite Quantizado: 104.633 KB

4.10.3 48×48 RGB:

- Acurácia do Modelo Tensor Flow: 95.27%
- Acurácia do Modelo TFLite Quantizado: 95.78%
- Tamanho do Modelo Tensor Flow: 377.094 KB
- Tamanho do Modelo TFLite Quantizado: 104.938 KB

4.10.4 48×48 Grayscale:

- Acurácia do Modelo Tensor Flow: 98.82%
- Acurácia do Modelo TFLite Quantizado: 99.16%
- Tamanho do Modelo Tensor Flow: 377.166 KB
- Tamanho do Modelo TFLite Quantizado: 103.813 KB

4.10.5 64×64 RGB:

- Acurácia do Modelo Tensor Flow: 99.16%
- Acurácia do Modelo TFLite Quantizado: 98.65%
- Tamanho do Modelo Tensor Flow: 379.956 KB
- Tamanho do Modelo TFLite Quantizado: 105.813 KB

4.10.6 64×64 Grayscale:

- Acurácia do Modelo Tensor Flow: 99.49%
- Acurácia do Modelo TFLite Quantizado: 99.66%
- Tamanho do Modelo Tensor Flow: 379.998 KB
- Tamanho do Modelo TFLite Quantizado: 104.688 KB

4.10.7 128×128 RGB:

- Acurácia do Modelo Tensor Flow: 99.16%
- Acurácia do Modelo TFLite Quantizado: 98.99%
- Tamanho do Modelo Tensor Flow: 403.552 KB
- Tamanho do Modelo TFLite Quantizado: 111.813 KB

4.10.8 128×128 Grayscale:

- Acurácia do Modelo Tensor Flow: 97.47%
- Acurácia do Modelo TFLite Quantizado: 98.14%
- Tamanho do Modelo Tensor Flow: 402.761 KB
- Tamanho do Modelo TFLite Quantizado: 110.688 KB

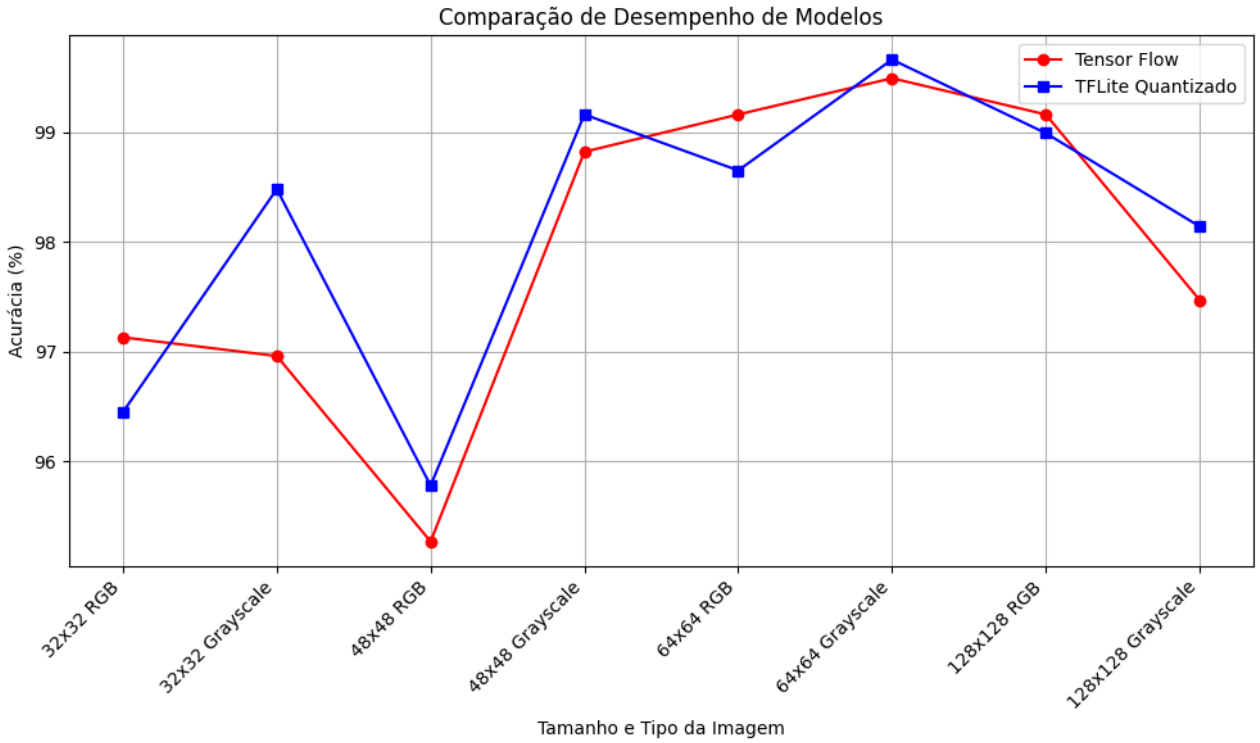

Figura 4.27: Gráfico Comparativo das Acurácias dos modelos antes e depois da conversão.

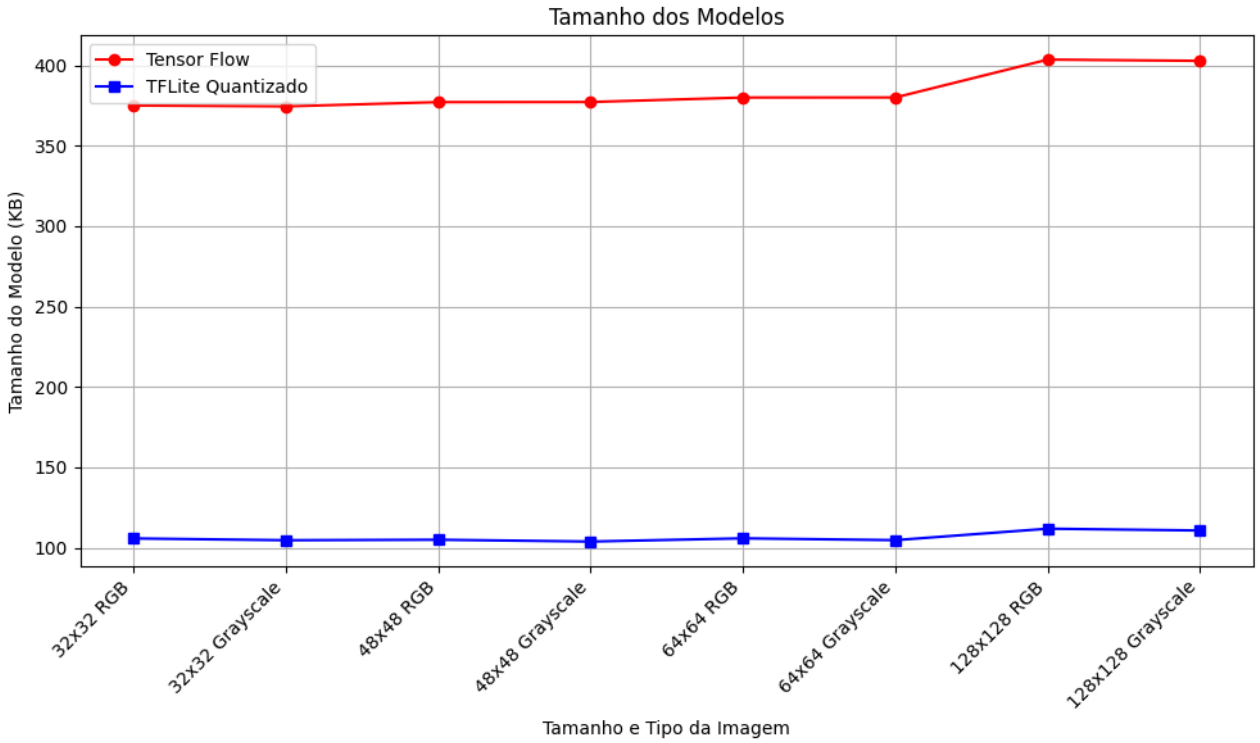

Figura 4.28: Gráfico Comparativo dos Tamanhos dos modelos antes e depois da conversão.

4.11 Análise de modelos

O hardware escolhido foi o Arduino Nano 33 BLE, que conta com o microcontrolador nRF52840. Ele foi o escolhido principalmente por possuir:

1. **Facilidade de Acesso**
2. **Versatilidade**
3. **Comunidade Ativa**
4. **Preço Acessível**
5. **Desempenho Satisfatório**
6. **Portabilidade**
7. **Compatibilidade com Shields**
8. **Ampla Disponibilidade de Componentes**

Com uma limitação de 256 KB de RAM no Arduino Nano com nRF52840, é crucial garantir que o modelo de aprendizado de máquina escolhido seja compatível com essa restrição. Todos os modelos TFLite Quantizados estão abaixo de 112 KB, o que é uma boa notícia, considerando a restrição de 256 KB de RAM. Isso sugere que o tamanho do modelo não excederá a metade da RAM disponível. Então, com base nas informações vistas acima, os melhores modelos para serem embarcados são o 48x48 Grayscale ou o 64x64 Grayscale, os quais apresentaram um equilíbrio entre acurácia, tamanho do modelo e eficiência de inferência.

5

Conclusão

Ao longo deste estudo, desenvolvemos um sistema eficaz de detecção de máscaras faciais, abordando aspectos cruciais desde a escolha e preparação da base de dados até a adaptação final para sistemas embarcados. A abordagem cuidadosa e sistemática adotada em cada fase do projeto resultou em conclusões significativas.

A base de dados, composta por imagens rotuladas, desempenhou um papel muito importante no treinamento dos modelos de aprendizado de máquina. Os resultados experimentais indicaram a capacidade dos modelos em reconhecer padrões associados ao uso de máscaras, sendo viáveis para integração em sistemas embarcados, desde que observadas as restrições específicas de hardware.

A exploração dos modelos envolveu o desenvolvimento de redes neurais convolucionais, considerando diferentes resoluções e modos de cor. A configuração dos parâmetros, o uso de técnicas como *ImageDataGenerator* e a implementação do *Early Stopping* contribuíram para um treinamento eficaz e a prevenção do *overfitting*.

A transformação dos modelos para o formato TensorFlow Lite, utilizando a técnica de quantização, mostrou-se essencial para a implementação em sistemas embarcados. A avaliação dos modelos durante a fase de teste destacou a superioridade dos modelos em escala de cinza em termos de acurácia e perda, com o modelo 64×64 em escala de cinza alcançando uma acurácia notável de 99.66%.

Em consideração aos requisitos específicos do Arduino Nano, a análise do consumo de memória dos modelos convertidos foi essencial. A escolha cuidadosa de configurações iniciais, como tamanho do lote e resoluções utilizadas, demonstrou uma abordagem criteriosa para otimizar o desempenho e evitar o *overfitting*.

Este estudo não apenas contribui para a implementação de soluções eficazes na detecção de máscaras faciais, mas também destaca a importância da otimização e adaptação para sistemas embarcados. A aplicabilidade prática dos modelos desenvolvidos representa uma contribuição significativa diante da atual preocupação com a saúde pública.

Referências bibliográficas

[1] Covid-19: pesquisa indica que 3 em cada 10 brasileiros usam máscara em locais abertos. *O Globo*, Abril 2022. URL https://oglobo.globo.com/saude/noticia/2022/04/covid-19-pesquisa-indica-que-3-em-cada-10-brasileiros-usam-mascara-em-locais-ab ghtml.

[2] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.

[3] Benjamin Lee, Anthony Gitter, Casey Greene, Sebastian Raschka, Finlay Maguire, Alexander Titus, Michael Kessler, Alexandra Lee, Marc Chevrette, Paul Stewart, Thiago Britto-Borges, Evan Cofer, Kun-Hsing Yu, Juan Carmona, Elana Fertig, Alexandr Kalinin, Brandon Signal, Benjamin Lengerich, Timothy Triche, and Simina Boca. Ten quick tips for deep learning in biology. *PLOS Computational Biology*, 18:e1009803, 03 2022. **DOI** 10.1371/journal.pcbi.1009803.

[4] Chaopeng Shen. A trans-disciplinary review of deep learning research for water resources scientists. *Water Resources Research*, 54, 12 2017. **DOI** 10.1029/2018WR022643.

[5] R. Dechter. Learning while searching in constraint-satisfaction problems. 1986.

[6] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 689–696, 2011.

[7] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022. **DOI** 10.1109/TNNLS.2021.3084827.

[8] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168(2):022022, feb 2019. **DOI** 10.1088/1742-6596/1168/2/022022. URL https://dx.doi.org/10.1088/1742-6596/1168/2/022022.

[9] Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/38db3aed920cf82ab059bfccbd02be6a-Paper.pdf.

[10] Lutz Prechelt. *Early Stopping - But When?*, pages 55–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. **DOI** 10.1007/3-540-49430-8$_3$. URL https://doi.org/10.1007/3-540-49430-8_3.

[11] Arnold S. Berger. *Embedded Systems Design: An Introduction to Processes, Tools, & Techniques*. CMP Books, 2002.

[12] Gerrit Muller. Opportunities and challenges in embedded systems. To be determined (TBD).

[13] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. Tensorflow lite micro: Embedded machine learning for tinyml systems. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 800–811, 2021. URL https://proceedings.mlsys.org/paper_files/paper/2021/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf.

[14] Naeem Ullah and Ali Javed. Face mask detection and masked facial recognition dataset (mdmfr dataset), 2022. URL https://doi.org/10.5281/zenodo.6408603.

[15] Martín Abadi and et al. Tensorflow: Large-scale machine learning on heterogeneous systems. *arXiv preprint arXiv:1603.04467*, 2015.

[16] François Chollet. Keras: The python deep learning library. *GitHub repository*, 2015. URL https://github.com/fchollet/keras.

[17] Martín Abadi and et al. Tensorflow lite: A lightweight library for mobile and edge devices. *arXiv preprint arXiv:1712.07950*, 2017.

[18] Arduino. Arduino nano 33. https://store.arduino.cc/usa/nano-33-iot.