Trabalho de Conclusão de Curso

Uma análise para sistemas embarcados de modelos de rede neural convolucional para classificação do porte de capacete de segurança.

Derek Nielsen Araújo Alves
dnaa@ic.ufal.br

Orientador:

Prof. Dr. Erick de Andrade Barboza

Maceió, junho de 2023

Derek Nielsen Araújo Alves

Uma análise para sistemas embarcados de modelos de rede neural convolucional para classificação do porte de capacete de segurança.

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, junho de 2023

.

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecária: Helena Cristina Pimentel do Vale CRB4 - 661

A474u Alves, Derek Nielsen Araújo.
Uma análise para sistemas embarcados de modelos de rede neural convolucional para classificação do porte de capacete de segurança / Derek Nielsen Araújo Alves . – 2023.
67 f. : il.

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

Bibliografia: f. 58-62.
Apêndice: f. 63-66.
Índice remecivo: f. 67

1. Sistemas embarcados. 2.TinyML. 3. Inteligência artificial. 4. Redes Neurais. 5. Aprendizado de máquina. I. Título.

CDU: 004.032.26

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. Tiago Figueiredo Vieira - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Prof. Me. Victor Diogho Heuer de Carvalho - Examinador
Campus do Sertão
Universidade Federal de Alagoas

Maceió, junho de 2023

Agradecimentos

De forma muito especial, gostaria de agradecer aos meus pais, Nelson Júnior e Doricelma, por ter dado todo suporte, incentivo e apoio durante estes cinco anos de universidade. Sem sombra de dúvidas não teria chegado até aqui sem eles e sem ter recebido a melhor educação possível no ambiente domiciliar, além do esforço para me garantir a melhor educação possível durante meu crescimento. Em memória dos meus avós paternos Nelson e Severina, por terem sido sempre presentes durante a minha infância e adolescência, e aos meus avós maternos Manoel e Dalimar, a quem não tive a oportunidade de conhecer, mas tenho a certeza que também seriam sempre presentes e me apoiariam nesta jornada. Agradeço também a toda a minha família, pelos momentos que dividimos juntos.

Ao professor Erick Barboza, por ter despertado a ideia para este trabalho durante a disciplina de aprendizagem de máquina em sistemas embarcados e por ter aceitado ser meu orientador, sendo bastante participativo, atencioso, solícito e acompanhado todo o progresso no desenvolvimento deste trabalho. Além de ter demostrado satisfação com os resultados obtidos a cada evolução no desenvolvimento. Ao professor Tiago Vieira, por ter dado a oportunidade de participar em projetos de pesquisa, ajudando a me enriquecer profissionalmente. Embora tenhamos nos conhecido melhor há pouco tempo, é uma pessoa a quem tenho bastante estima.

Aos demais professores que me deram oportunidades e me serviram de inspiração durante o curso. Em especial aos professores Bruno Nogueira e Rian Pinheiro, orientadores de um projeto de iniciação científica em que participei, possibilitando assim uma maior desenvoltura em pesquisa científica e publicação de artigos científicos.

Não poderia esquecer do agradecimento aos meus amigos e colegas da graduação, que sempre fizeram os períodos na universidade mais agradáveis, convivendo quase que diariamente. De forma mais que especial cabe o agradecimento aos amigos: Ruan, Darlysson, João Brito, Matheus Gêda, John Davi (*Cláudio*), Mateus Felismino, Lucas Massa, Yuri, Roger, Bruna, Luana, Leonardo, Ascânio, Caio e Vinícius. Agradeço também aos meus amigos de longa data: Halan, Elvis, Lucas Mendes, Gabriel e Luigui.

Por último mas não menos importante, gostaria de agradecer a coordenação do curso e ao corpo administrativo do Instituto de Computação e da Universidade Federal de Alagoas. Agradecimento em especial ao outrora coordenador de curso Prof. Dr. Thiago Cordeiro e ao técnico Marcelo de Gusmão por terem sempre sido atenciosos e prestativos com relação as demandas que surgiram durante o andamento do curso.

Resumo

Esta monografia aborda a criação de modelos de aprendizagem de máquina para sistemas embarcados, neste contexto propomos alguns modelos treinados para a classificação do correto uso de capacetes de segurança e fazemos uma análise de viabilidade desses modelos com relação ao seu tamanho, acurácia e perdas, de modo a apontar quais seriam os melhores modelos para se embarcar em algumas placas de desenvolvimento disponíveis. Tal análise de viabilidade foi realizada considerando modelos que utilizaram quatro-mil e oitocentas imagens. Os resultados experimentais obtidos mostram que é possível embarcar esses modelos em diversas placas de desenvolvimento, desde que atentemos as restrições de cada hardware, pouca memória, pouco processamento, baixo consumo de energia, e outros.

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 work addresses the creation of machine learning models for embedded systems. In this context, we propose some trained models for classifying the correct use of safety helmets and conduct a feasibility analysis of these models in terms of their size, accuracy and losses, in order to identify the best models to be deployed on various available development boards. Based on this feasibility analysis, our experimental results show that it is possible to deploy these models on different development boards, provided that we pay attention to the constraints of each hardware, such as limited memory, processing power, low energy consumption, and others. Additionally, we created a database containing approximately four thousand images, which we used to train our neural network models.

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

A Tecnologia tem um impacto crescente em nossa vida diária e causa diversas mudanças em nosso cotidiano. Essas mudanças estão presentes em nossos telefones, carros, eletrodomésticos e outros dispositivos que usamos para melhorar nosso bem-estar. A aprendizagem de máquina é uma das tecnologias mais transformadoras de nosso tempo, empregada por empresas, acadêmicos e várias outras comunidades. Ela está em constante evolução e revela novos casos de uso em diferentes ramos de atuação [2].

Com base nesta premissa e considerando a redução no tamanho dos modelos de aprendizagem profunda [3], torna-se possível executar tais modelos em dispositivos na borda (sistemas embarcados) para enfrentar diversos desafios. Uma das aplicações mais difundidas desses modelos é o reconhecimento de padrões sonoros, implementado em assistentes virtuais populares como *Alexa*, *Siri* e *Google Assistant*. De acordo com [3], alguns desses modelos podem ter apenas 14 kilobytes, permitindo sua execução em microcontroladores, apesar das limitações de hardware, como memória limitada, capacidade de processamento reduzida, baixo consumo de energia, entre outros.

No contexto dos assistentes virtuais, suas implementações envolvem a identificação de padrões em áudio. Contudo, identificação de padrões a partir de imagens é outro ramo em que modelos de aprendizagem de máquina são bastante utilizados. A aprendizagem de máquina está lado a lado com o ramo de processamento de imagem [4], que visa manipular e aprimorar as imagens utilizando técnicas computacionais para tal. A junção dessas duas técnicas é conhecida como como visão computacional [5], O objetivo da visão computacional é permitir que os computadores "vejam"e entendam os dados visuais de maneira semelhante à visão humana, e para isto, os algoritmos de visão envolvem uma combinação de técnicas de processamento de imagem e aprendizado de máquina.

Embora os modelos de classificação de imagens possam ser mais complexos do que os modelos de classificação de sons, ainda assim é possível embarcar tais modelos em hardware

limitados. Para tanto, é necessária uma análise de viabilidade, considerando os hardwares disponíveis e os requisitos do problema a ser resolvido.

Levando em conta os aspectos relacionados a segurança do trabalho, é de conhecimento geral que a indústria da construção civil é atormentada por situações de risco ocupacional e condições pobres de trabalho [6], sendo assim, viabilizar um produto que restrinja o acesso a um determinado ambiente caso o porte de equipamento de segurança não esteja sendo utilizado corretamente, ou ainda, que conscientize sobre o uso do equipamento, iria reduzir os riscos tanto para os funcionários quanto para a empresa, reduzindo o número de acidentes de trabalho e possíveis complicações trabalhistas.

Diante disso, nosso trabalho se concentra em identificar padrões em imagens, especificamente relacionados ao porte adequado de equipamentos de segurança no ambiente de trabalho (indústrias, construção civil, armazéns, etc.).

Nosso objetivo geral neste trabalho é avaliar os modelos de aprendizagem de máquina para reconhecimento de padrões em imagens e selecionar os melhores candidatos para embarcar eficientemente.

Os objetivos específicos deste trabalho são a geração de um banco de dados personalizado utilizando modelos de difusão latente [7, 8], a criação de modelos utilizando *TensorFlow* [9] e a análise de viabilidade desses modelos em alguns dispositivos pré selecionados, o que pode ser utilizado para o desenvolvimento de trabalhos futuros, bem como produtos.

Para isso, algumas etapas serão seguidas, como ter o conjunto de dados para treinar nossos modelos, tratar e pré-processar os dados, criar os modelos, avaliar seu desempenho e tamanho, e então decidir sobre os melhores candidatos.

O restante desta monografia está organizada da seguinte forma: No capítulo dois iremos abordar a fundação teórica que é a base do nosso trabalho, em seguida, no capítulo três iremos discutir a metodologia utilizada para obter nossos resultados, resultados esses que se seguem no capítulo quatro juntamente com a análise de viabilidade dos modelos gerados, e por fim, nosso último capítulo apresenta as nossas conclusões.

2

Fundamentação Teórica

ESTE capítulo irá abordar os principais fundamentos utilizados na elaboração deste trabalho, desde os conceitos de segurança no trabalho que envolvem a utilização correta do capacete de segurança até detalhes sobre as ferramentas utilizadas para a geração dos nossos modelos de classificação.

2.1 Uso de equipamentos de proteção individual

A indústria da construção é uma indústria muito perigosa na qual ocorrem lesões ocupacionais fatais e não fatais com mais frequência [6] devido à sua natureza única. Ela é caracterizada por mudanças contínuas, uso de muitos recursos diferentes, más condições de trabalho, falta de emprego estável e ambientes insalubres (por exemplo, ruído, vibração, poeira, manuseio de carga e exposição direta ao clima). Além disso, requer coordenação de diferentes empreiteiros, subempreiteiros e operações interdependentes, o que pode resultar em um aumento do risco de lesões. Essas lesões e doenças relativas ao trabalho não só afetam a segurança e a saúde dos trabalhadores, mas também afetam a economia, devido aos altos custos para tratar essas lesões e também o tempo ocioso desses trabalhadores enquanto se recuperam de acidentes, dessa maneira faz-se necessário reforçar as políticas e a fiscalização com relação ao uso de equipamentos individuais, não apenas no âmbito da construção civil, mas em qualquer ambiente de trabalho que necessite do uso de tais equipamentos.

Adiante, iremos demonstrar a importância de se utilizar o capacete de segurança como equipamento de proteção individual, e mostrar exemplos do uso correto e incorreto do capacete de segurança.

2.1.1 Uso do capacete de segurança

Os equipamentos de proteção individual (EPI) dos funcionários desempenham um papel importante no sistema de medidas preventivas para manter as condições de trabalho seguras e reduzir as doenças ocupacionais. Segundo Nikulin e Romanov [10] as estatísticas mostram que 10-15% de todas as mortes na Rússia são devidas à falta de EPI, uso indevido ou defeitos técnicos. O uso inadequado ou insuficiente de EPI é responsável por 30% de todas as doenças ocupacionais crônicas. Dessa forma, se faz importante o uso de todos os equipamentos de proteção individual corretamente, e isto inclui o uso correto do capacete de segurança.

Uma forma de melhorar o fornecimento e o gerenciamento do uso atual de EPIs é a instalação de dispositivos que verificam se o equipamento está sendo usado corretamente, neste quesito a criação de modelos de inteligência artificial que podem classificar o uso correto ou incorreto destes equipamentos é de vital importância para que tais dispositivos sejam criados. O nosso foco é monitorar o uso de proteção individual para a cabeça dos funcionários, ou seja, a utilização de capacetes de segurança, a seguir nós temos um exemplo do porte correto do capacete de segurança (figura 2.1) e do porte incorreto do capacete (figura 2.2).

Figura 2.1: Imagem gerada para nosso banco de dados, porte correto do capacete

Figura 2.2: Imagem gerada para nosso banco de dados, porte incorreto do capacete

2.2 IA generativa e a geração do banco de dados

Inteligência Artificial Generativa se refere a um tipo de sistema de inteligência artificial que é capaz de criar ou gerar novos dados, como imagens, texto, áudio e até mesmo vídeos, que se assemelham a exemplos do conjunto de dados em que foram treinados. Diferentemente dos sistemas de IA convencionais, que são projetados para realizar tarefas específicas com base em exemplos existentes, os modelos generativos têm a capacidade de produzir novos conteúdos de forma original. Eles são treinados usando técnicas avançadas, como redes neurais generativas adversariais (GANs) e modelos de linguagem com atenção, que aprendem as características e padrões dos dados de treinamento para gerar saídas criativas e realistas.

Um exemplo popular de IA generativa são as GANs [11], que consistem em duas partes principais: um gerador e um discriminador. O gerador gera amostras sintéticas com base em um ruído aleatório, enquanto o discriminador avalia se as amostras são reais ou sintéticas. Durante o treinamento, o gerador e o discriminador se enfrentam em um jogo de adversários, buscando melhorar suas habilidades. Isso resulta em um gerador capaz de produzir saídas cada vez mais autênticas.

A IA generativa tem aplicações em várias áreas, como criação de arte, geração de conteúdo criativo, síntese de voz e música, preenchimento de lacunas em imagens, criação de personagens virtuais e muito mais. Ela permite a criação de dados inéditos com base em padrões existentes, expandindo as capacidades criativas da inteligência artificial.

É importante notar que a IA generativa também pode levantar questões éticas e desafios

relacionados ao uso responsável, como a possibilidade de criação de conteúdo falso ou enganoso. Portanto, é fundamental que sejam implementados mecanismos de controle e supervisão adequados ao usar essa tecnologia.

Neste quesito, utilizamos os modelos de difusão latentes que podem ser vistos na subseção abaixo para gerar as imagens contidas em nosso banco de dados.

2.2.1 Modelos de difusão latentes

Mais especificamente, para gerar nosso banco de dados, utilizamos modelos de difusão latentes como proposto por [7], esses modelos nos possibilitam descrever a imagem desejada em forma de texto e a partir desta descrição são utilizados modelos complexos que geram a imagem solicitada. A seguir temos algumas imagens (figuras 2.3, 2.4, 2.5, 2.6) geradas utilizando esses modelos:

Figura 2.3: Imagem gerada utilizando o comando: "Volkswagen beetle 1970 with mount fuji in the background, photorealistic"

Figura 2.4: Imagem gerada utilizando o comando: "Cat sitting on a belarusian mat, photorealistic"

Figura 2.5: Imagem gerada utilizando o comando: "Darth vader ordering a burger and fries from the Death Star canteen"

Figura 2.6: Imagem gerada utilizando o comando: "iguazu falls, 4k"

2.3 Redes Neurais Convolucionais - CNNs

No campo do aprendizado profundo, uma CNN (rede neural convolucional) trata-se de uma classe de rede neural artificial, proposta por LeCun et al. Desde sua proposição, o uso dessas CNNs se mostraram muito eficientes para resolver problemas de classificação, portanto, neste trabalho iremos utilizar CNNs para classificar o uso correto ou não do capacete de segurança. A rede neural convolucional vem se tornando ultimamente uma das alternativas mais viáveis aos métodos de classificação mais tradicionais. Levando em conta que as CNNs são algoritmos supervisionados [13], pois são uma classe de rede neural artificial. A aprendizagem supervisionada envolve aprender uma associação entre um conjunto de variáveis de entrada X e uma variável de saída Y [14], e após isso, aplicar esta mesma associação para realizar inferências sobre dados nunca vistos antes pelo modelo. A aprendizagem supervisionada é a metodologia mais importante na área de aprendizado de máquina e também tem uma importância central no processamento de dados multimídia.

Um dos problemas para se treinar um algoritmo supervisionado parte da necessidade de se ter uma grande base de dados rotulados para que possamos criar essas regras de associação e assim extrair os *features* ou características desejadas, partindo desta necessidade por uma alta quantidade de dados, geramos nossos dados utilizando os modelos de difusão latentes citados anteriormente. Para extrairmos estes features utilizando uma CNN precisamos de três componentes básicos: A camada convolucional, a camada de pooling e a camada totalmente conectada. Qualquer arquitetura por mais básica que seja de uma CNN necessita desses três componentes.

É importante observar que a arquitetura e a estrutura exatas das CNNs podem mudar dependendo do problema em questão e das necessidades específicas do problema abordado. A seguir iremos dar uma sucinta descrição das camadas utilizadas em uma CNN.

2.3.1 Camada convolucional

Pode-se dizer que a camada convolucional é a principal camada de uma rede neural convolucional, esta camada é responsável por extrair as características dos dados de entrada. para extrair essas features são utilizados filtros de tamanho reduzido de modo que esses filtros percorrem realizando convoluções em todo o dado de entrada. Durante o treinamento do modelo esses filtros são ajustados de modo a melhorar a identificação das características contidas nos dados. No início do treinamento esses filtros começam a ser ajustados aos dados, e com o passar das épocas de treinamento os filtros vão se modificando e se ajustando melhor de modo a melhorar a precisão na identificação das features.

2.3.2 Camada de pooling

A camada de pooling é uma camada responsável pela operação de agrupamento, esta etapa é fundamental em redes neurais convolucionais para reduzir a dimensionalidade entre as camadas. O pooling combina um intervalo de valores em um número menor de valores, ou seja, reduz a dimensionalidade.

O processo de pooling tem o objetivo de representar um coletivo de informações em uma única informação valiosa, de modo a preservar características úteis e removendo características não úteis. Os operadores de pooling fornecem uma forma de invariância de transformação espacial, não apenas reduzindo a complexidade computacional das camadas superiores, mas também eliminando algumas conexões entre as camadas convolucionais. Essa camada diminui a resolução do mapa de recursos da camada anterior e cria um novo mapa de recursos com resolução menor.

Dessa maneira a camada de pooling tem dois propósitos principais: reduzir a quantidade de parâmetros ou pesos, reduzindo a complexidade computacional e controlar o overfitting, pois, um conjunto de informações é reduzido a uma única informação. Espera-se que um método de pooling ideal extraia apenas informações relevantes e descarte os detalhes irrelevantes. Existem algumas formas de pooling como *Maxpooling* (figura 2.7), *Averagepooling* (figura 2.8) e outros, como podemos observar em [1].

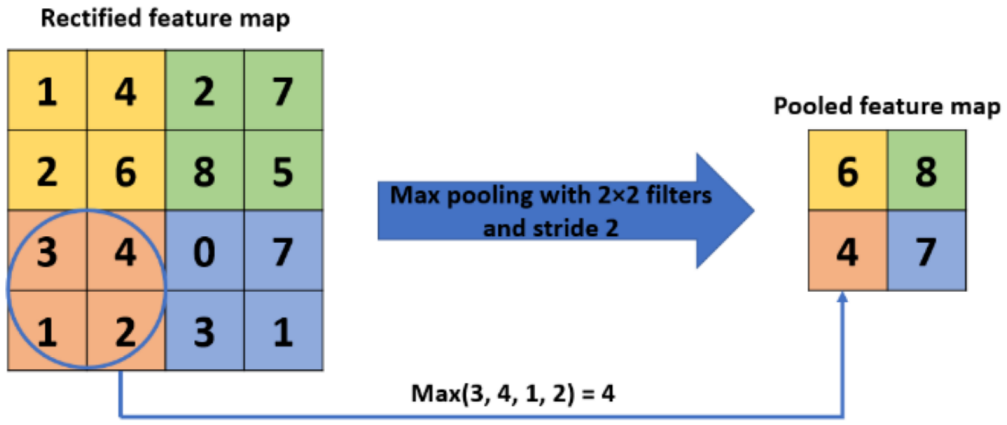

Figura 2.7: Max pooling, imagem retirada de: [1]

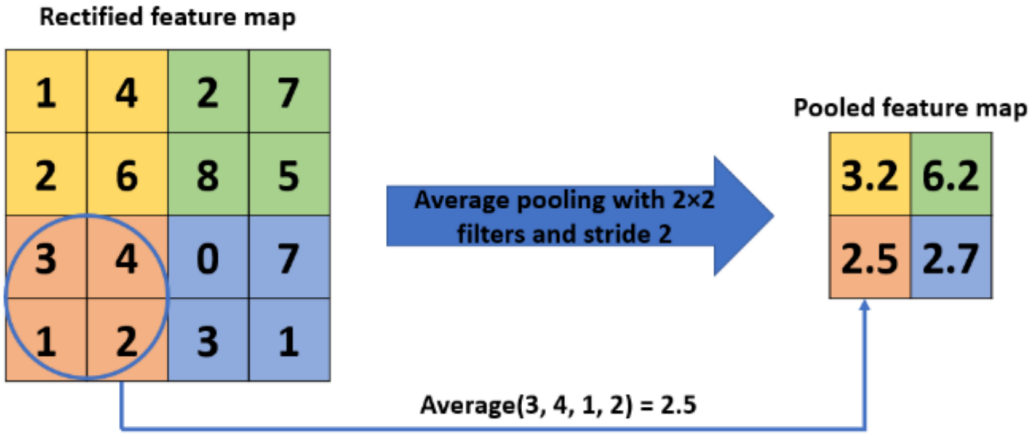

Figura 2.8: Average pooling, imagem retirada de: [1]

2.3.3 Camada totalmente conectada

No final da rede CNN, uma ou mais camadas totalmente conectadas são usadas para realizar a classificação ou regressão com base nos recursos extraídos. Essas camadas conectam todos os neurônios da camada anterior a todos os neurônios da camada de saída, independentemente da estrutura espacial dos recursos. Camadas totalmente conectadas têm a vantagem de serem capazes de capturar relacionamentos complexos entre feições extraídas por camadas convolucionais. No entanto, também pode ser propensa a sofrer *overfitting*, especialmente se a rede for muito profunda ou se houver muitos parâmetros na rede. Para mitigar a possibilidade de overfitting, utilizamos algumas técnicas como o *Dropout* e o *EarlyStopping*, que iremos apresentar a seguir.

2.4 Prevenindo overfitting

O *overfitting* (sobre-ajuste) é um problema fundamental no aprendizado de máquina supervisionado que impede que os modelos generalizem perfeitamente para o conjunto de dados não visto

durante o treinamento [15]. Ele pode ser causado por ruído, tamanho limitado do conjunto de treinamento e complexidade do classificador. Desta forma, algumas técnicas foram propostas para abordar essas causas e mitigar os efeitos do sobre-ajuste. Algumas dessas estratégias são:

- Early stopping (parada antecipada);
- Redução da rede com o intuito de filtrar o ruído nos dados de treinamento;
- Data augmentation (aumento nos dados)
- Regularização
  - Dropout regularization

Nas subseções adiante iremos abordar as técnicas de Dropout Regularization e Early Stopping que foram utilizadas para treinar nossos modelos de rede neural.

2.4.1 Dropout Regularization

Como já discutido, o overfitting pode se tornar um problema sério em redes neurais. dessa forma o dropout [16] é uma ferramenta que resolve esse problema. a idéia principal por trás do dropout é remover alguns neurônios aleatoriamente (e suas conexões) da rede neural durante cada etapa do treinamento, evitando assim o sobre-ajuste desses neurônios para uma característica mais marcante, porém que não seja geral. Esta técnica reduz muito o overfitting e fornece uma melhoria significativa em relação a outros métodos de regularização.

2.4.2 Early stopping

No early stopping [17, 18], podemos utilizar um conjunto de dados para validação de modo a detectar quando o overfitting começa durante o treinamento supervisionado de uma rede neural. A partir desta detecção, interrompemos o treinamento para evitar que este se adapte demais ao conjunto de treino e comece a perder poder de generalização para dados não vistos anteriormente. Os critérios para a parada antecipada podem ser definidos empiricamente levando em conta os dados gerados durante o treinamento.

2.5 TinyML

TinyML [3, 2] é um acrônimo para Tiny Machine Learning, que se refere à aplicação de técnicas de aprendizado de máquina a dispositivos com recursos limitados, como microcontroladores de baixa potência, sistemas embarcados e sensores de borda. O objetivo do TinyML é executar modelos de aprendizado de máquina diretamente no dispositivo que gera os dados, em vez de enviá-los para processamento em nuvem. Os principais objetivos e benefícios do TinyML são:

- Eficiência energética: a execução de modelos de aprendizado de máquina diretamente no dispositivo elimina a necessidade de transferir grandes quantidades de dados para a nuvem, economizando energia e prolongando a vida útil da bateria de dispositivos alimentados por bateria.
- Latência reduzida: ao executar modelos de aprendizado de máquina localmente, você pode tomar decisões em tempo real sem depender da latência causada pela comunicação com servidores remotos.
- Privacidade e segurança: armazenar dados e modelos de aprendizado de máquina em seu dispositivo ajuda a proteger sua privacidade, eliminando a necessidade de compartilhar dados confidenciais com serviços em nuvem. Além disso, as informações são mantidas localmente, o que pode aumentar a segurança.
- Baixo custo: equipamentos baratos e recursos limitados tornam a tecnologia mais acessível, permitindo que soluções de aprendizado de máquina sejam implementadas em uma ampla variedade de produtos e cenários.

Desta maneira, ao trazermos os recursos de aprendizado de máquina para dispositivos de ponta, permitimos que sejam utilizados em áreas como Internet das Coisas (IoT), saúde, automação residencial, manufatura inteligente e agricultura de precisão. Abrindo assim novas possibilidades para aquisição, processamento e análise de dados em tempo real em pequenos dispositivos distribuídos.

3

Metodologia

3.1 Banco de dados

Devido a falta de um banco de dados que dispusesse de imagens para classificação do uso de capacete de segurança com boa qualidade, decidimos criar um banco de dados. Para criamos o banco de dados, como já abordado no capítulo anterior, utilizamos uma ferramenta baseada em modelos de difusão latentes chamada de *Stable Diffusion*. O Stable Diffusion é uma espécie de rede neural generativa profunda desenvolvida pelo grupo CompVis[1] na Universidade de Munique, em conjunto com a startup Runway[2]. Tanto a implementação como os pesos do modelo foram lançados como código aberto, como o modelo é liberado sob uma licença permissiva, ele nos concede todos os direitos sobre as imagens geradas, desde que estas não sejam ilegais ou prejudiciais, isto nos permitiu criar o banco de dados sem as preocupações que viriam com o uso de imagens protegidas por direito autoral.

Utilizamos uma versão do *Stable Diffusion* [3] que roda na ferramenta Google Colaboratory, em um ambiente virtual que dispõe de 12.7GB de memória ram e 15GB de VRAM disponíveis em uma placa de vídeo Tesla T4. Este ambiente é suficiente para gerarmos lotes de imagens utilizando o Stable Diffusion.

Para gerarmos as imagens, utilizamos o seguinte texto abaixo nos campos de positivo e negativo fornecidos no Stable Diffusion:

- Positive
  - a person wearing a safety helmet inside a factory, photorealistic
- Negative

[1] https://github.com/CompVis/latent-diffusion
[2] https://runwayml.com
[3] https://colab.research.google.com/drive/1l1STQX0PRZlrWEnWNlUhdX7poFGZBP0d?usp=sharing

  - lowres, ((bad anatomy)), ((bad hands)), text, missing finger, extra digits, fewer digits, blurry, ((mutated hands and fingers)), (poorly drawn face), ((mutation)), ((deformed face)), (ugly), ((bad proportions)), ((extra limbs)), extra face, (double head), (extra head), ((extra feet)), monster, logo, cropped, worst quality, low quality, normal quality, jpeg, humpbacked, long body, long neck, ((jpeg artifacts)), ((out of frame)), (blur)

O comando positivo explicita o que queremos na imagem, já o comando negativo explicita o que não queremos na imagem. No comando negativo, as informações entre parêntesis tem mais ênfase do que as que não estão entre parêntesis, dessa forma, quando escrevemos ((bad anatomy)) queremos dizer que não desejamos que a imagem contenha má anatomia na geração das pessoas, e estamos dando maior destaque essa informação. Além dessas informações, definimos um passo de 50 para gerar as imagens, quanto maior o passo, mais tempo o modelo passa refinando as imagens, o que geralmente resulta em imagens com melhor qualidade, o banco de dados[4] pode ser encontrado no rodapé desta página.

3.2 Implementação dos modelos

Implementamos 12 modelos de rede neural convolucional (6 modelos coloridos e 6 modelos em escala de cinza). Para implementar estes modelos utilizamos o ambiente do Google Colaborator, executando *Python3*, possuindo também 12.7GB de RAM e 15GB de VRAM em uma GPU Tesla T4. A ferramenta principal para a criação dos modelos foi o tensorflow juntamente com o keras. Estas ferramentas são utilizadas desde a criação dos modelos até a conversão dos mesmos para os sistemas embarcados.

Para o carregamento das imagens e a divisão do banco em conjuntos de treino e validação foi utilizado a ferramenta do tensorflow keras (**tensorflow.keras.ImageDataGenerator** e seu método **flow from directory**)[5], nestas ferramentas especificamos uma série de parâmetros como:

- Diretório que contém as imagens;
- Resolução das imagens;
- Tamanho do batch;
- Modo de cor;
  - Colorido, escala de cinza.

[4] https://drive.google.com/file/d/1ZcpoGMpmDLp5kgHnyF3NaBmdTGMgGY9b/view?usp=sharing
[5] https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator#flow_from_directory

- Modo de classe;
  - Classificação binária, outras formas de classificação;
- Subset;
  - Treino, validação, teste.
- Tamanho do split entre os subsets.

O *ImageDataGenerator* é um objeto que pode ser instanciado de modo a configurar o carregamento das imagens utilizando o *flow from directory*, que efetivamente carrega os dados e já manipula as imagens para o formato desejado, isto é, redimensiona as imagens para a resolução configurada e converte para o tipo de dados configurado, de modo a deixar o banco de dados pronto para ser utilizado nos modelos. Para visualizarmos estas imagens, utilizamos o OpenCV[6] juntamente com o MatPlotLib[7], plotando assim essas amostras.

O MatPlotLib também é utilizado extensamente durante todo o nosso trabalho, para plotarmos as visualizações do histórico de treinamento, gráficos em barra das acurácias de validação e perdas de validação, matrizes de confusão, dentre outros.

Para criar os modelos, utilizamos a ferramenta **Sequential** do *tensorflow*, onde podemos especificar as camadas internas da rede neural de forma sequencial, então importamos a ferramenta **layers** que contém diversos tipos de camadas que podemos utilizar. As camadas utilizadas foram as camadas de: entrada, convolução 2D, maxpooling 2D, flatten, dropout e a última camada sendo densa.

- Camada de entrada (input)
  - Esta camada fica responsável por receber as entradas para o modelo, os parâmetros utilizados para instanciar esta camada foram: dimensões da imagem e tamanho do batch.
- Camada convolucional 2D (Conv2D)
  - Esta é a camada que realiza as convoluções, como já explanado no capítulo anterior. Ela recebe como parâmetros a quantidade de filtros, tamanho do kernel, função de ativação (ReLU), dentre outros.
- Camada de maxpooling 2D (MaxPooling2D)
  - O papel desta camada é realizar a redução de dimensionalidade entre as camadas convolucionais, os parâmetros necessários para que o pooling seja efetuado são: tamanho da janela (pool size), o passo da janela de pooling (stride) e a forma de padding.

[6] https://pypi.org/project/opencv-python/
[7] https://matplotlib.org

- Camada de achatamento (flatten)
  - O papel desta camada é achatar as dimensões das camadas anteriores em uma única dimensão, encaminhando assim para a saída do modelo. Esta camada não recebe parâmetros adicionais.
- Camada de dropout (Dropout)
  - A tarefa desta camada é remover alguns neurônios e suas conexões ao aleatório, de modo a prevenir o overfitting. Esta camada recebe como parâmetro um número entre 0 e 1 que representa a quantidade de neurônios que serão removidos, utilizamos 0.5 para efetuar um dropout em 50% dos neurônios da camada anterior.
- Camada densa (Dense)
  - O papel desta camada é de saída do modelo, esta camada irá realizar a classificação com base em todas as camadas anteriores, como estamos fazendo uma classificação binária, ela recebe como parâmetros o número de neurônios, que neste caso é 1, e a função de ativação, que neste caso é a função sigmoide.

O otimizador utilizado para o treinamento da rede foi o *Adam* [19]. Este otimizador é popular em redes neurais e combina os métodos *RMSprop* [20] e *Adagrad* [21]. Ele usa uma taxa de aprendizado adaptável para ajustar os pesos e vieses da rede neural e manter uma média móvel de gradiente de modo que a convergência do treinamento é melhorada. O otimizador recebe alguns parâmetros, como: a taxa de aprendizagem (utilizamos 0.0005), e os parâmetros Beta1 (0.9) e Beta2 (0.999), que representam a taxa de decaimento exponencial para o primeiro e segundo momentos, respectivamente.

Além disso, utilizamos o callback **EarlyStopping**, como já citado anteriormente, os parâmetros recebidos para instanciar o callback são: variável a ser monitorada (perda de validação), paciência (10 épocas) e o parâmetro afirmativo para retorno dos melhores pesos (True). Desta maneira, prevenimos o overfitting e retornamos o melhor peso treinado em nossos diversos modelos.

3.3 Conversão dos modelos

Para convertermos os modelos, utilizamos o TensorFlow Lite[8], que é uma ferramenta do Tensorflow desenvolvida para ser leve e eficiente, para que seja possível embarcar modelos em dispositivos que possuem recursos limitados. O TensorFlow Lite permite que os desenvolvedores implantem modelos de aprendizado de máquina em smartphones, tablets, vestíveis, dispositivos de Internet das Coisas (IoT) e outros dispositivos de borda.

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

Para a conversão dos modelos, tomamos duas abordagens:

- Poda
  - Esta técnica se refere a remoção seletiva de certas conexões de uma rede neural treinada de modo a melhorar a eficiência, reduzir a complexidade computacional e reduzir o tamanho dos modelos.
- Quantização (reduzir pesos de Float32 para Int8)
  - A quantização é uma técnica utilizada para transformar de um tipo de dados para outro, saindo de um tipo de dados maior, como no caso de Float32 para um tipo de dados menor, como Int8, reduzindo de 3 bytes a quantidade para representar os pesos da rede.

Então convertemos os modelos para esses dois formatos, onde faremos uma análise com relação ao espaço utilizado por cada um desses modelos e também verificaremos a viabilidade desses modelos em diversos microcontroladores.

3.4 Análise dos modelos

Para a análise dos modelos, utilizamos algumas ferramentas para visualização de dados como o matplotlib e o seaborn[9]. A princípio são visualizados os resultados durante o treinamento, e vemos que todos os modelos não sofreram *overfitting* devido as técnicas utilizadas para a prevençao deste problema. Em seguida, utilizamos o *tensorflow* para efetuar inferências de teste utilizando o banco de dados para teste, que contém 800 imagens, sendo 400 imagens da classe **Helmet** e 400 imagens da classe **NoHelmet**.

Após efetuarmos as inferências, podemos observar a acurácia no teste destes modelos e elencar os melhores candidatos para efetuarmos mais testes nos modelos convertidos para TensorFlow Lite.

A análise de todos os modelos convertidos com relação ao uso de memória é feita em comparação a alguns microcontroladores pré-selecionados, sendo estes:

- STM32L452
  - 160 kilobytes de RAM
- NRF52840
  - 256 kilobytes de RAM
- ESP32-S

[9] https://seaborn.pydata.org

  - 520 kilobytes de RAM

- NRF54H20

  - 1024 kilobytes de RAM

E a partir destes dados juntamente com a visualização dos mesmos em tabelas, podemos descartar mais alguns modelos, reduzindo assim nosso conjunto inicial para aqueles que consideramos ser os melhores candidatos para embarcar.

3.5 Análise final

Nesta análise final, utilizamos novamente o matplotlib e o seaborn, juntamente com o *Scikit learn*[10] para gerar matrizes de confusão e relatórios de classificação.

Levando em consideração todas as análises até o momento, reduzimos do conjunto inicial de modelos para aqueles que apresentam ser os melhores candidatos para embarcar, e portanto, para estes modelos fizemos um comparativo entre as acurácias do modelo convencional, do modelo convertido para TensorFlow Lite e do modelo quantizado (INT8) e convertido para TensorFlow Lite.

Então, apresentamos nossas considerações finais sobre esses modelos, como por exemplo: quais dispositivos acomodariam bem estes modelos.

[10] https://scikit-learn.org/stable/

4

Resultados e Discussões

NESTA seção veremos nossos principais resultados, que vão desde a criação de um banco de dados personalizado para o nosso problema, até a geração de vários modelos fazendo assim um comparativo entre esses modelos e possíveis candidatos de placas que serão utilizadas para embarcar os modelos criados.

4.1 Algumas definições

De modo a evitar a repetição de alguns termos, os modelos que se seguem a partir da seção 4.3 utilizam um conjunto de ferramentas iguais, portanto, iremos explanar abaixo:

Batch, o batch se trata da quantidade de imagens por lote que utilizamos para treinar nosso modelo, utilizamos 20 imagens por batch em todos os modelos, então, essa é uma informação para a dimensão de entrada dos nossos modelos, que ficará mais claro durante a exposição dos nossos modelos.

A resoluções utilizadas foram: 32x32, 64x64, 96x96, 128x128, 256x256 e 512x512. Além disto, utilizamos imagens coloridas (3 canais - RGB) e imagens em escala de cinza (1 canal).

Para o treinamento do modelo foram utilizadas como limite máximo 100 épocas, porém, utilizamos a ferramenta de *EarlyStopping* contida no [22] que basicamente interrompe o treinamento do modelo caso alguma condição seja atingida. Condição esta que foi definida monitorando a variável perda de validação (val_loss) a fim de minimizar esta variável em 10 épocas, caso o valor não diminua nesta quantidade de épocas, o treinamento é terminado, caso o valor melhore, serão dadas mais 10 épocas para avaliar se o modelo melhora. Utilizamos também a flag que restaura os melhores pesos (restore_best_weights = True) garantindo assim que obtenhamos o melhor modelo definido pelo EarlyStopping.

Utilizamos camadas convolucionais de duas dimensões (*Conv2D* juntamente com camadas de *MaxPooling2D* a fim de se reduzir a dimensionalidade a cada camada interna da rede neural. Após as camadas de convolução e maxpooling, temos uma camada *Flatten* que achata as

dimensões em uma única dimensão e interconecta a camada de saída com a anterior. E por último temos uma camada de *Dropout* antes da saída do modelo, esta camada é responsável por 'desligar' alguns neurônios aleatoriamente a fim de garantir que o modelo não fique muito especializado em uma característica, generalizando melhor, utilizamos um dropout em 50% dos neurônios nesta camada de dropout. As técnicas de earlystopping juntamente com o dropout visam evitar que o problema conhecido como *Overfitting* ou sobre-ajuste aconteça, este fenômeno acontece quando o nosso modelo se adapta muito ao conjunto de treino (aumento de precisão) e acaba perdendo precisão para o conjunto de validação.

4.2 Banco de dados

Como discutido nos capítulos anteriores, utilizamos modelos de difusão latente para gerar imagens e assim compor nosso banco de dados, uma pequena amostra do banco de dados pode ser vista abaixo, na figura 4.1.

O banco de dados é composto por:

- 4800 imagens em resolução 512x512 com três canais de cor
  - 2400 imagens para o uso correto do capacete de segurança
  - 2400 imagens não utilizando o capacete de segurança

Sendo estas imagens divididas da seguinte forma:

- 4000 imagens para treinamento e validação
  - 2000 imagens para o uso correto do capacete de segurança
  - 2000 imagens não utilizando o capacete de segurança
- 800 imagens para teste dos modelos
  - 400 imagens para o uso correto do capacete de segurança
  - 400 imagens não utilizando o capacete de segurança

Definido o banco de dados, para treinar cada um dos modelos precisamos readequar a resolução e os canais de cor do modelo, onde utilizamos o redimensionamento das imagens para adequar a resolução. Em caso de modelos em escala de cinza, além do redimensionamento nós efetuamos a conversão das imagens para escala de cinza.

Vale ressaltar que o número de imagens geradas foi bem maior que as quatro mil imagens que já citamos, tivemos que fazer um tratamento passando por cada uma das imagens descartando as imagens que não se adequavam aos nossos requisitos, restando assim as quatro-mil e oitocentas imagens que compõem o nosso conjunto de dados.

Figura 4.1: Amostra de imagens do banco de dados.

4.3 Modelo 512x512 colorido

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 512x512 colorido.

4.3.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, possui uma camada de entrada no formato (20, 512, 512, 3), onde 20 é o batch, 512x512x3 de resolução e imagem colorida.

As camadas mais internas que se seguem são camadas convolucionais 2D juntamente com camadas de *MaxPooling2D*, seguidas de uma camada de *flatten*, *Dropout* que efetua o dropout em 50% dos neurônios e por fim a última camada, que tem dimensão (20, 1), onde 20 é a quantidade de imagens no batch e 1 é o neurônio que irá definir se as imagens aferidas estão portando

corretamente ou não o capacete de segurança, a estrutura completa pode ser visualizada na figura 4.2 abaixo.

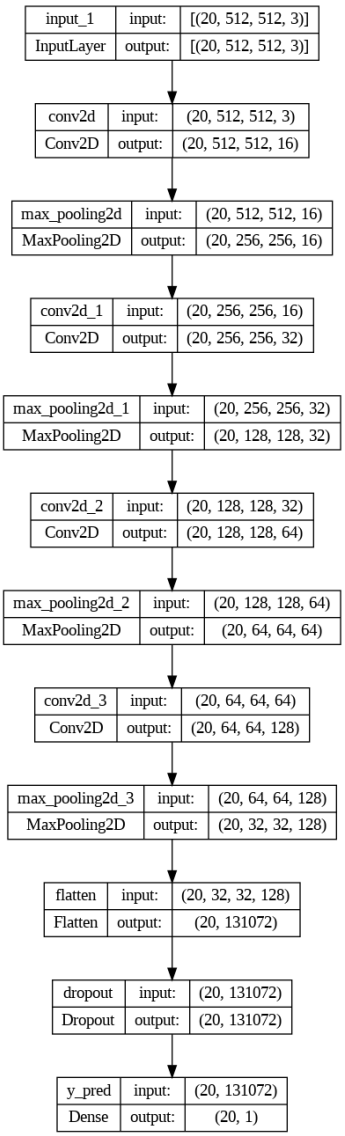

Figura 4.2: Arquitetura do modelo 512x512 colorido.

4.3.2 Entrada do Modelo

Para este modelo, utilizamos imagens com dimensão 512x512 com três canais (RGB), algumas dessas imagens e suas classes podem ser visualizadas abaixo, como ilustra a figura 4.3.

Figura 4.3: Amostra de imagens para o modelo com resolução 512x512 colorido.

4.3.3 Treinamento do Modelo

Definimos uma quantidade fixa de 100 épocas para o treinamento, porém utilizamos o *EarlyStopping* com parâmetros: 10 épocas, minimizando a perda de validação. Dessa maneira, o modelo 512x512 colorido teve a acurácia de validação em torno de 91.12% obtida na época 17 do treinamento, nesta época obtivemos a melhor perda de validação cujo valor foi de 0.2313 (figura 4.4).

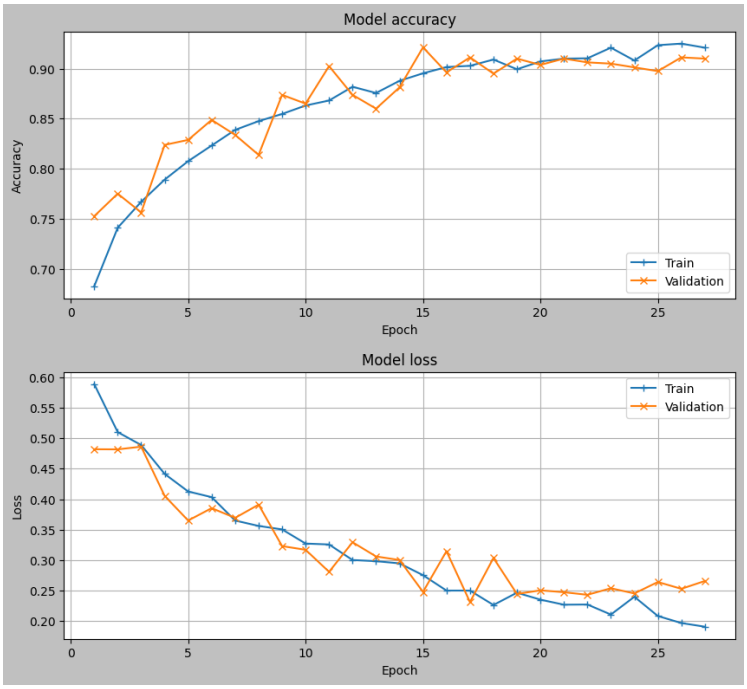

Figura 4.4: Evolução da acurácia e perda durante o treinamento do modelo 512x512 colorido

4.4 Modelo 512x512 escala de cinza

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 512x512 em escala de cinza.

4.4.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura da seção anterior, porém, como este modelo é em escala de cinza, temos apenas um canal de cor. As camadas mais internas se comportam da mesma maneira que no modelo anterior, com a diferença no número de neurônios por camada devido a quantidade de canais ser menor, como podemos ver na figura 4.5 abaixo.

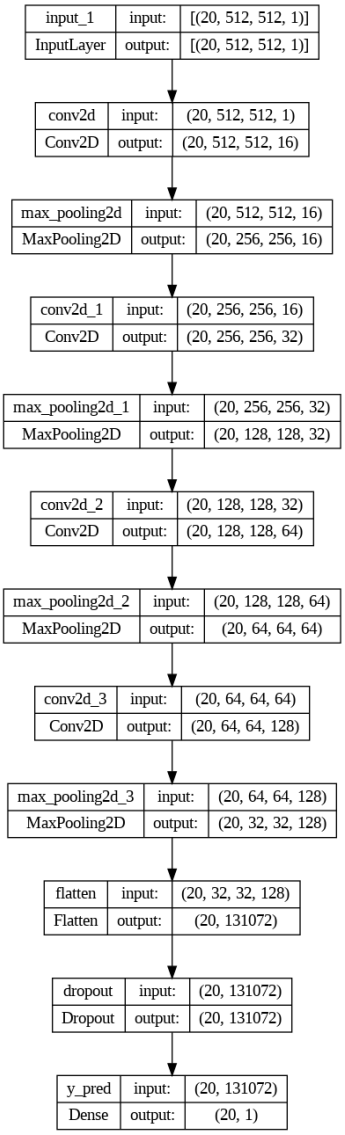

Figura 4.5: Arquitetura do modelo 512x512 em escala de cinza.

4.4.2 Entrada do Modelo

Para este modelo, as imagens são 512x512 em escala de cinza, como visto na figura 4.6.

Figura 4.6: Amostra de imagens para o modelo com resolução 512x512 em escala de cinza.

4.4.3 Treinamento do Modelo

Utilizando os mesmos parâmetros da seção anterior, este modelo teve a melhor acurácia de validação de 86.75% e a melhor perda de validação 0.3003 na época 39 (figura 4.7).

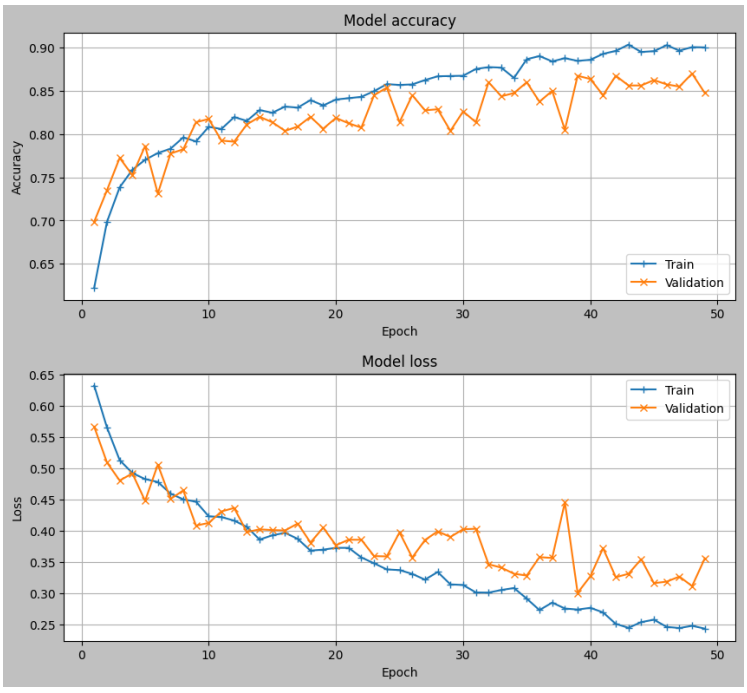

Figura 4.7: Evolução da acurácia e perda durante o treinamento do modelo 512x512 em escala de cinza

4.5 Modelo 256x256 colorido

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 256x256 colorido.

4.5.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura do modelo 512x512 colorido. As camadas mais internas se comportam da mesma maneira que no modelo anterior, com a diferença no número de neurônios por camada devido a resolução ser menor, como podemos ver na figura 4.8 abaixo.

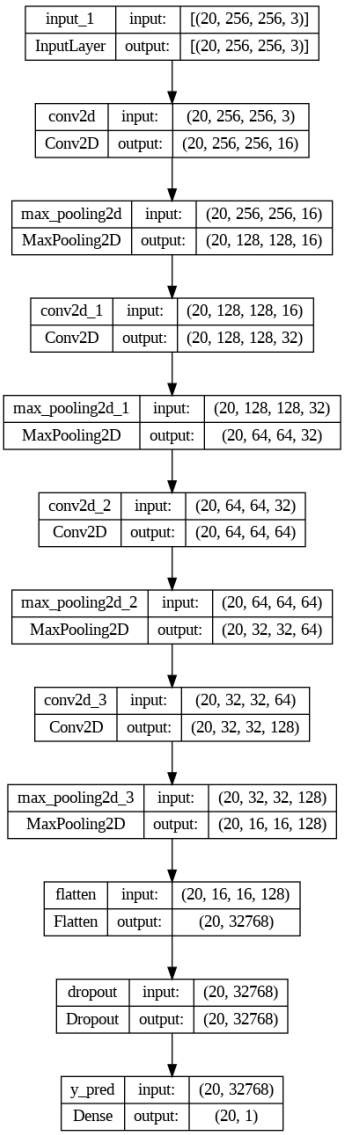

Figura 4.8: Arquitetura do modelo 256x256 colorido.

4.5.2 Entrada do Modelo

Para este modelo, as imagens são 256x256 coloridas, como visto na figura 4.9.

Figura 4.9: Amostra de imagens para o modelo com resolução 256x256 colorido.

4.5.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, este modelo teve a melhor acurácia de validação de 93.88% e a melhor perda de validação 0.1798 na época 32 (figura 4.10).

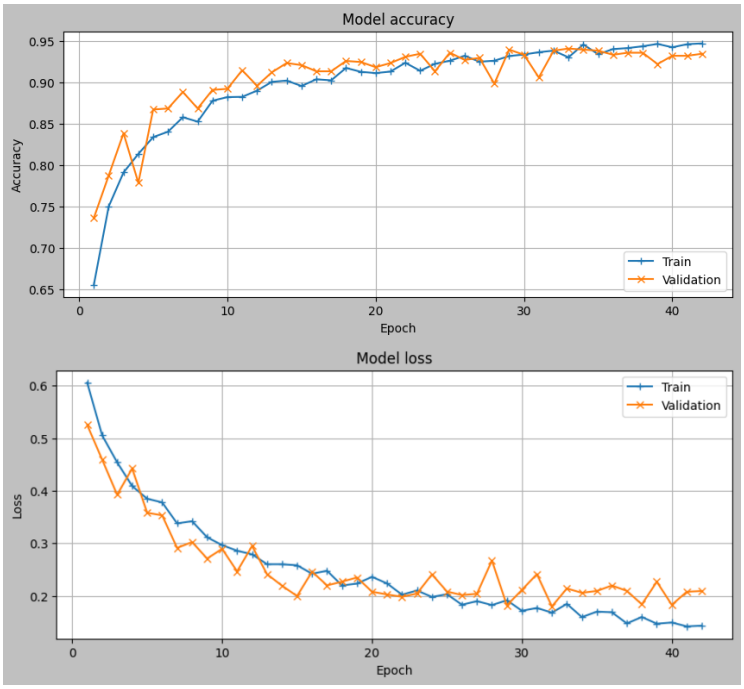

Figura 4.10: Evolução da acurácia e perda durante o treinamento do modelo 256x256 colorido

4.6 Modelo 256x256 em escala de cinza

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 256x256 em escala de cinza.

4.6.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura das seções anteriores, porém, com menor resolução e em escala cinza. As camadas mais internas se comportam da mesma maneira que nos modelos anteriores, com a quantidade menor de neurônios nas camadas, como podemos ver na figura 4.11 abaixo.

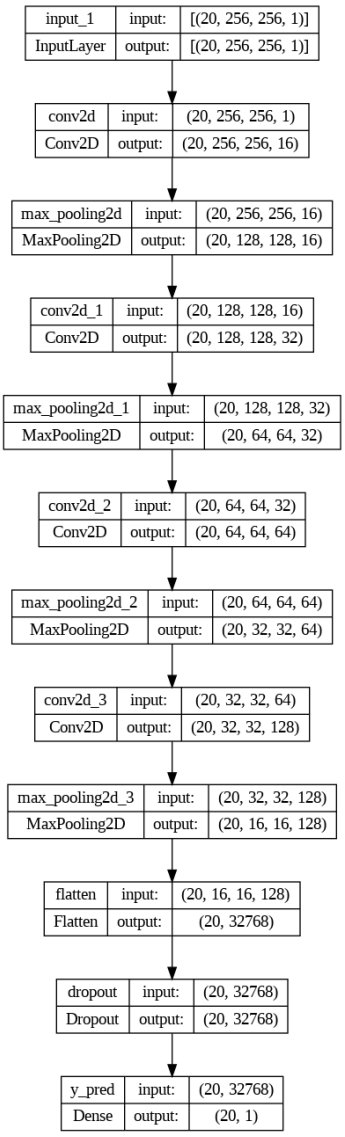

Figura 4.11: Arquitetura do modelo 256x256 em escala de cinza.

4.6.2 Entrada do Modelo

Para este modelo, as imagens são 256x256 em escala de cinza, como visto na figura 4.12.

Figura 4.12: Amostra de imagens para o modelo com resolução 256x256 em escala de cinza.

4.6.3 Treinamento do Modelo

Utilizando os mesmos parâmetros da seção anterior, este modelo teve a melhor acurácia de validação de 91.25% e a melhor perda de validação 0.2228 na época 66 (figura 4.13).

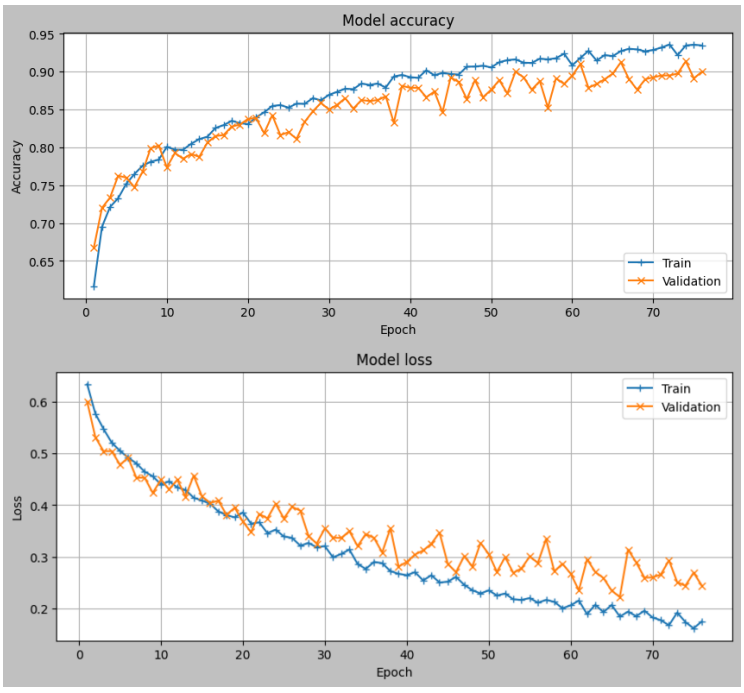

Figura 4.13: Evolução da acurácia e perda durante o treinamento do modelo 256x256 em escala de cinza

4.7 Modelo 128x128 colorido

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 128x128 colorido.

4.7.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura dos modelos anteriores. As camadas mais internas se comportam da mesma maneira, com a diferença no número de neurônios por camada devido a resolução ser menor, como podemos ver na figura 4.14 abaixo.

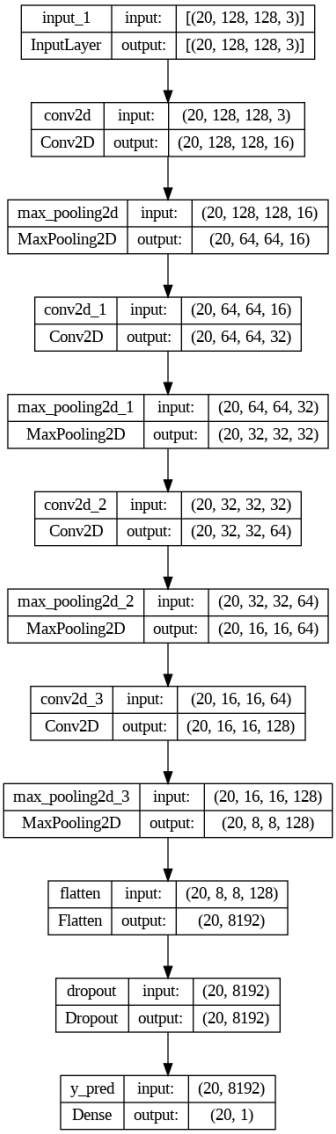

Figura 4.14: Arquitetura do modelo 128x128 colorido.

4.7.2 Entrada do Modelo

Para este modelo, as imagens são 128x128 coloridas, como visto na figura 4.15.

Figura 4.15: Amostra de imagens para o modelo com resolução 128x128 colorido.

4.7.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, este modelo teve a melhor acurácia de validação de 94.63% e a melhor perda de validação 0.1243 na época 52 (figura 4.16).

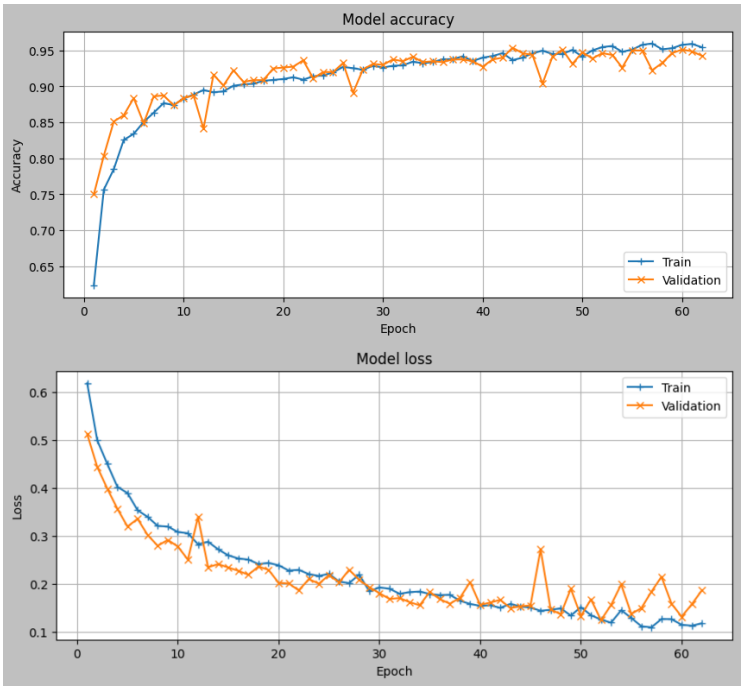

Figura 4.16: Evolução da acurácia e perda durante o treinamento do modelo 128x128 colorido

4.8 Modelo 128x128 em escala de cinza

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 128x128 em escala de cinza.

4.8.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura das seções anteriores, porém, com menor resolução e em escala cinza. As camadas mais internas se comportam da mesma maneira que nos modelos anteriores, com a quantidade menor de neurônios nas camadas, como podemos ver na figura 4.17 abaixo.

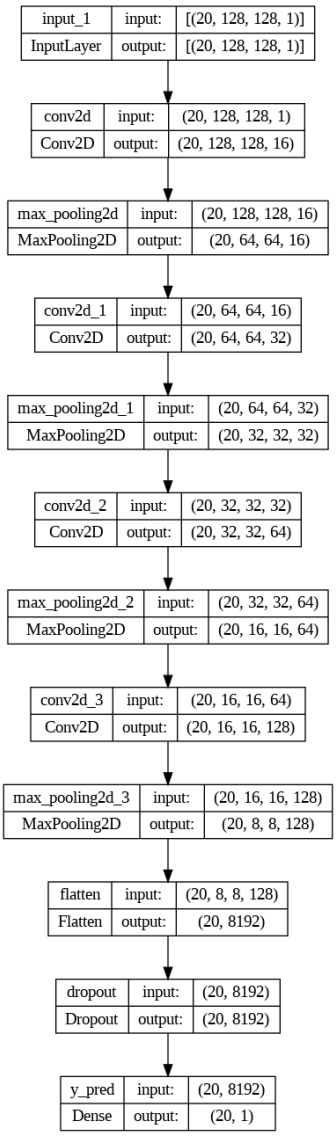

Figura 4.17: Arquitetura do modelo 128x128 em escala de cinza.

4.8.2 Entrada do Modelo

Para este modelo, as imagens são 128x128 em escala de cinza, como visto na figura 4.18.

Figura 4.18: Amostra de imagens para o modelo com resolução 128x128 em escala de cinza.

4.8.3 Treinamento do Modelo

Utilizando os mesmos parâmetros da seção anterior, este modelo teve a melhor acurácia de validação de 90.62% e a melhor perda de validação 0.2283 na época 51 (figura 4.19).

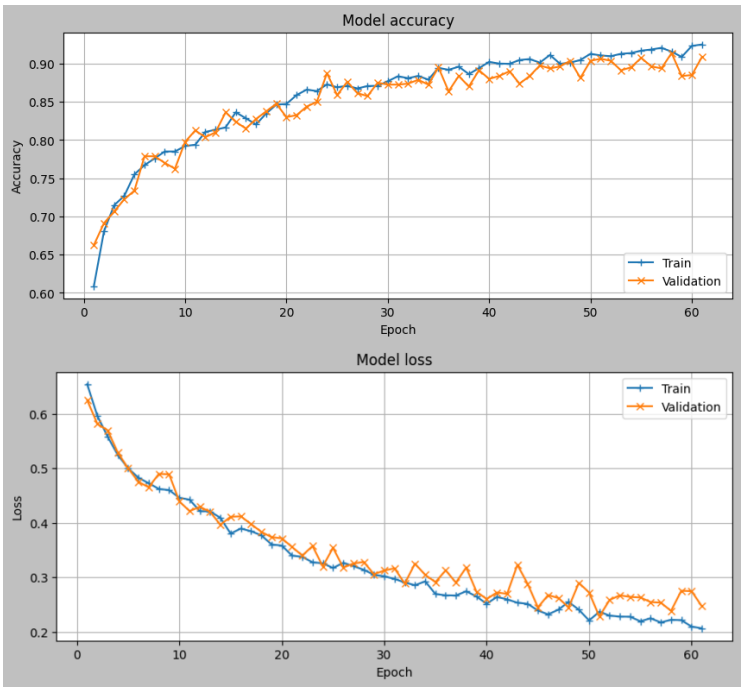

Figura 4.19: Evolução da acurácia e perda durante o treinamento do modelo 128x128 em escala de cinza

4.9 Modelo 96x96 colorido

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 96x96 colorido.

4.9.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura dos modelos anteriores. As camadas mais internas se comportam da mesma maneira, com a diferença no número de neurônios por camada devido a resolução ser menor, como podemos ver na figura 4.20 abaixo.

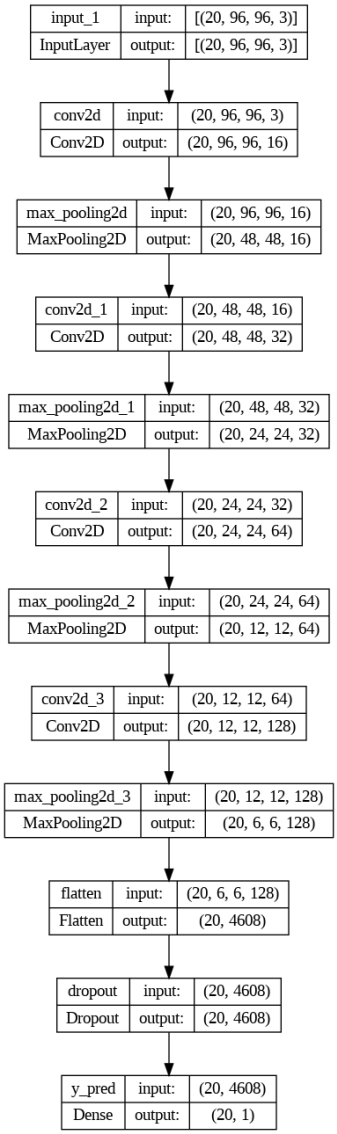

Figura 4.20: Arquitetura do modelo 96x96 colorido.

4.9.2 Entrada do Modelo

Para este modelo, as imagens são 96x96 coloridas, como visto na figura 4.21.

Figura 4.21: Amostra de imagens para o modelo com resolução 96x96 colorido.

4.9.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, este modelo teve a melhor acurácia de validação de 95.13% e a melhor perda de validação 0.1320 na época 46 (figura 4.22).

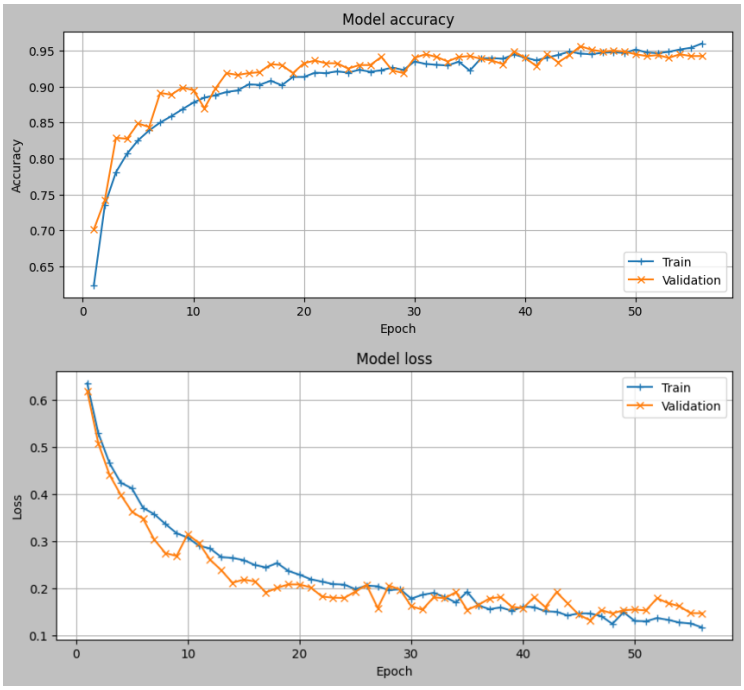

Figura 4.22: Evolução da acurácia e perda durante o treinamento do modelo 96x96 colorido

4.10 Modelo 96x96 em escala de cinza

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 96x96 em escala de cinza.

4.10.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura das seções anteriores, porém, com menor resolução e em escala cinza. As camadas mais internas se comportam da mesma maneira que nos modelos anteriores, com a quantidade menor de neurônios nas camadas, como podemos ver na figura 4.23 abaixo.

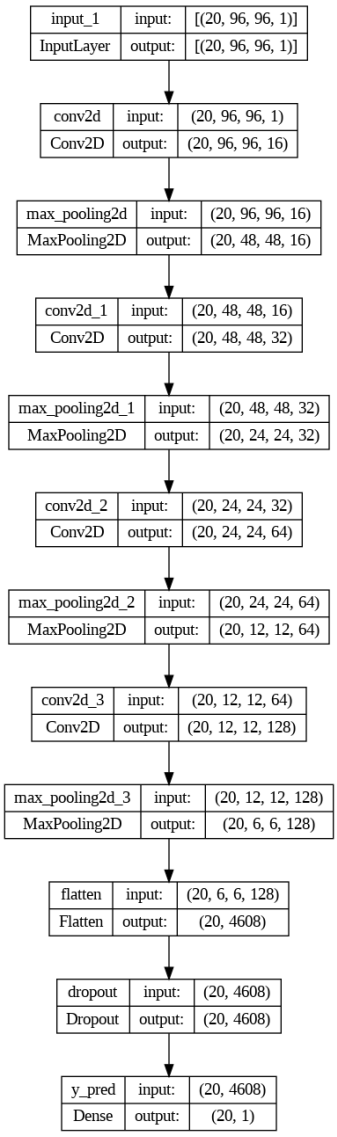

Figura 4.23: Arquitetura do modelo 96x96 em escala de cinza.

4.10.2 Entrada do Modelo

Para este modelo, as imagens são 96x96 em escala de cinza, como visto na figura 4.24.

Figura 4.24: Amostra de imagens para o modelo com resolução 96x96 em escala de cinza.

4.10.3 Treinamento do Modelo

Utilizando os mesmos parâmetros da seção anterior, este modelo teve a melhor acurácia de validação de 89.88% e a melhor perda de validação 0.2347 na época 51 (figura 4.25).

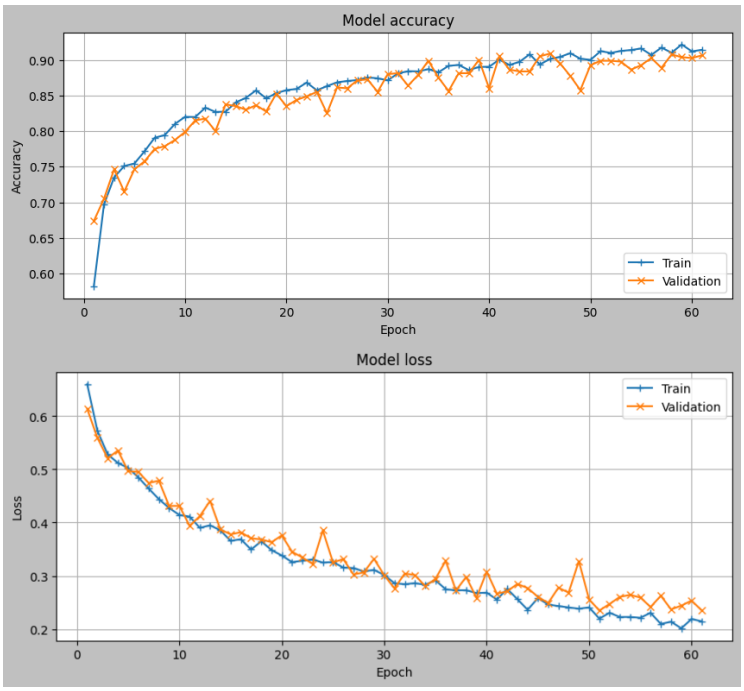

Figura 4.25: Evolução da acurácia e perda durante o treinamento do modelo 96x96 em escala de cinza

4.11 Modelo 64x64 colorido

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 64x64 colorido.

4.11.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura dos modelos anteriores. As camadas mais internas se comportam da mesma maneira, com a diferença no número de neurônios por camada devido a resolução ser menor, como podemos ver na figura 4.26 abaixo.

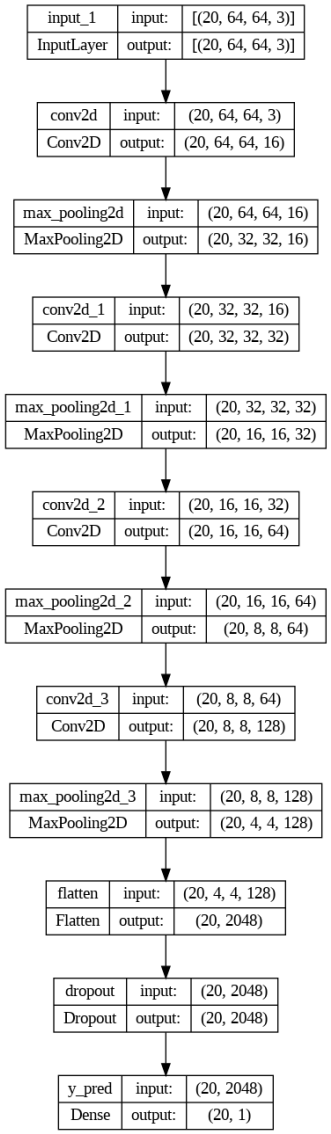

Figura 4.26: Arquitetura do modelo 64x64 colorido.

4.11.2 Entrada do Modelo

Para este modelo, as imagens são 64x64 coloridas, como visto na figura 4.27.

Figura 4.27: Amostra de imagens para o modelo com resolução 64x64 colorido.

4.11.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, este modelo teve a melhor acurácia de validação de 95.13% e a melhor perda de validação 0.1617 na época 27 (figura 4.28).

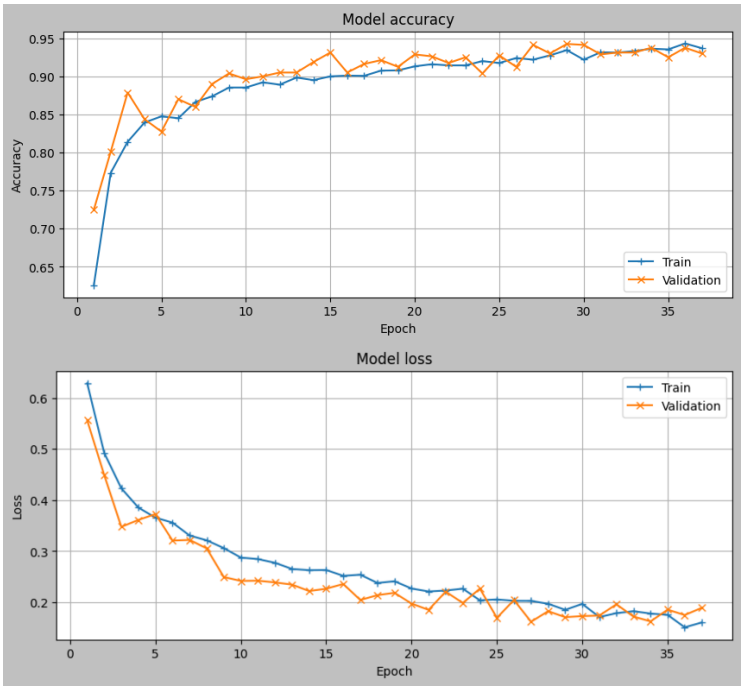

Figura 4.28: Evolução da acurácia e perda durante o treinamento do modelo 64x64 colorido

4.12 Modelo 64x64 em escala de cinza

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 64x64 em escala de cinza.

4.12.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura das seções anteriores, porém, com menor resolução e em escala cinza. As camadas mais internas se comportam da mesma maneira que nos modelos anteriores, com a quantidade menor de neurônios nas camadas, como podemos ver na figura 4.29 abaixo.

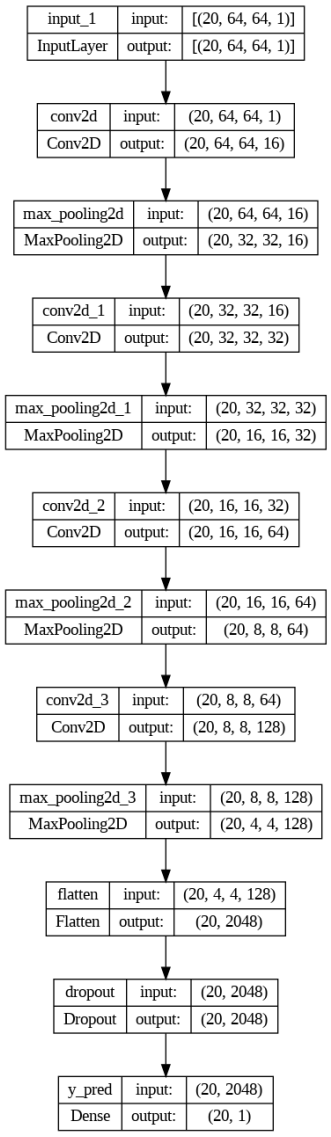

Figura 4.29: Arquitetura do modelo 64x64 em escala de cinza.

4.12.2 Entrada do Modelo

Para este modelo, as imagens são 64x64 em escala de cinza, como visto na figura 4.30.

Figura 4.30: Amostra de imagens para o modelo com resolução 64x64 em escala de cinza.

4.12.3 Treinamento do Modelo

Utilizando os mesmos parâmetros da seção anterior, este modelo teve a melhor acurácia de validação de 86.75% e a melhor perda de validação 0.3011 na época 39 (figura 4.31).

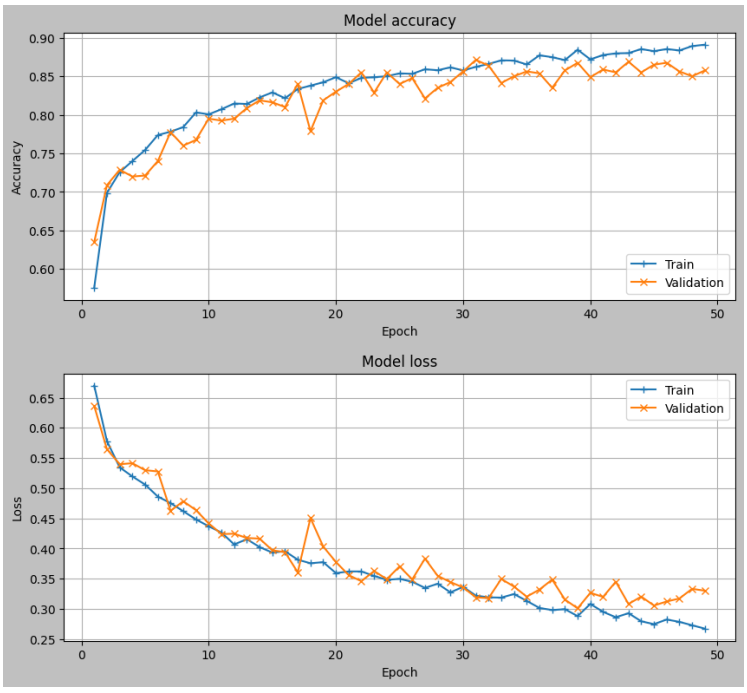

Figura 4.31: Evolução da acurácia e perda durante o treinamento do modelo 64x64 em escala de cinza

4.13 Modelo 32x32 colorido

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 32x32 colorido.

4.13.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura dos modelos anteriores. As camadas mais internas se comportam da mesma maneira, com a diferença no número de neurônios por camada devido a resolução ser menor, como podemos ver na figura 4.32 abaixo.

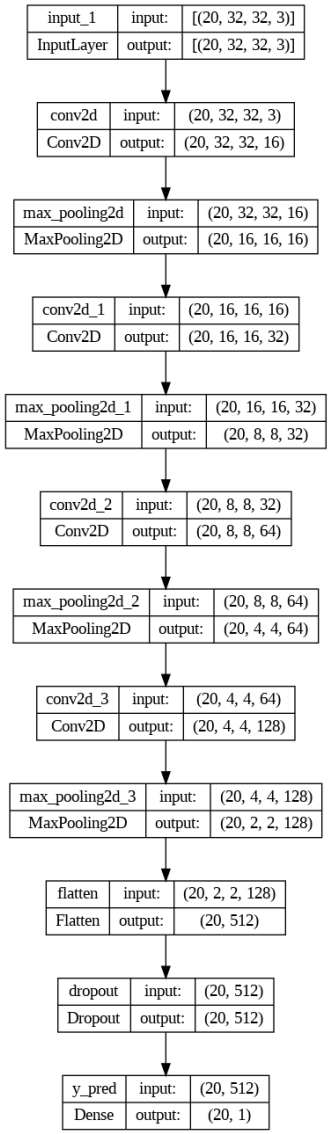

Figura 4.32: Arquitetura do modelo 32x32 colorido.

4.13.2 Entrada do Modelo

Para este modelo, as imagens são 32x32 coloridas, como visto na figura 4.33.

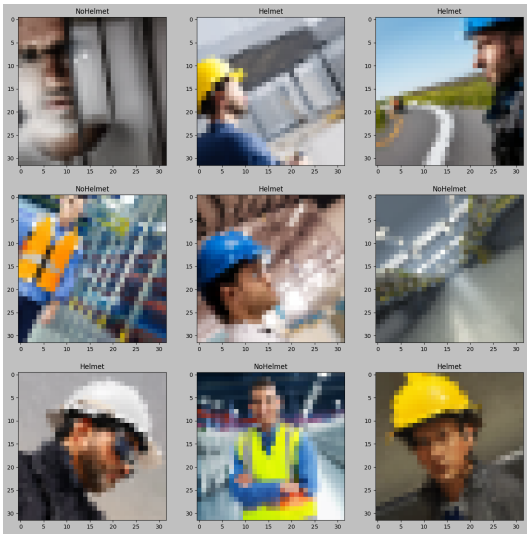

Figura 4.33: Amostra de imagens para o modelo com resolução 32x32 colorido.

4.13.3 Treinamento do Modelo

Utilizando os mesmos parâmetros das seções anteriores, este modelo teve a melhor acurácia de validação de 92.00% e a melhor perda de validação 0.2196 na época 39 (figura 4.34).

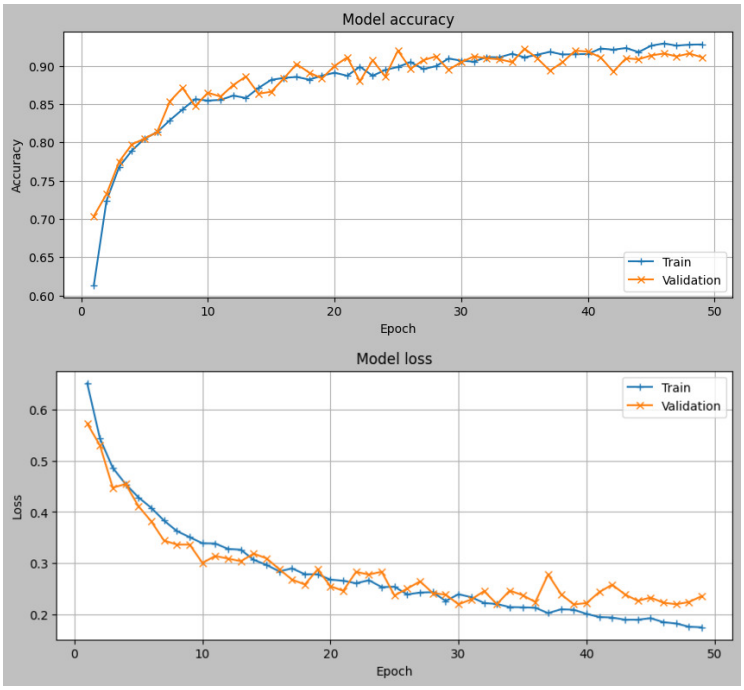

Figura 4.34: Evolução da acurácia e perda durante o treinamento do modelo 32x32 colorido

4.14 Modelo 32x32 em escala de cinza

Nesta seção iremos discutir a arquitetura do modelo, iremos visualizar as imagens de entrada e os resultados de treinamento para o modelo 32x32 em escala de cinza.

4.14.1 Arquitetura do Modelo

Com relação à estrutura da rede neural convolucional, ela é idêntica a arquitetura das seções anteriores, porém, com menor resolução e em escala cinza. As camadas mais internas se comportam da mesma maneira que nos modelos anteriores, com a quantidade menor de neurônios nas camadas, como podemos ver na figura 4.35 abaixo.

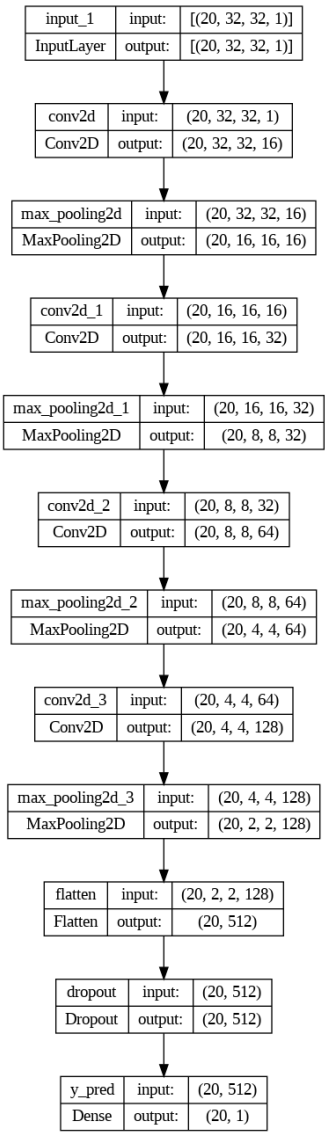

Figura 4.35: Arquitetura do modelo 32x32 em escala de cinza.

4.14.2 Entrada do Modelo

Para este modelo, as imagens são 32x32 em escala de cinza, como visto na figura 4.36.

Figura 4.36: Amostra de imagens para o modelo com resolução 32x32 em escala de cinza.

4.14.3 Treinamento do Modelo

Utilizando os mesmos parâmetros da seção anterior, este modelo teve a melhor acurácia de validação de 83.63% e a melhor perda de validação 0.3597 na época 44 (figura 4.37).

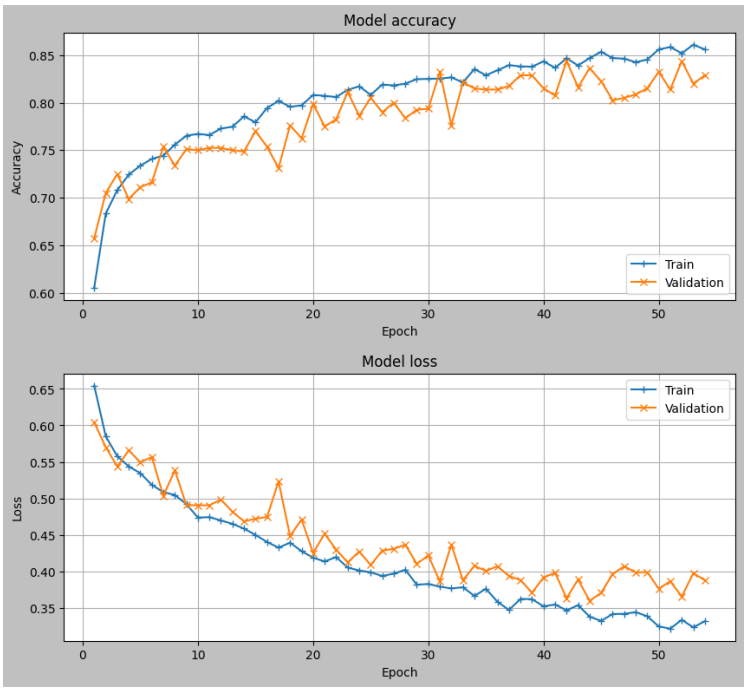

Figura 4.37: Evolução da acurácia e perda durante o treinamento do modelo 32x32 em escala de cinza

4.15 Comparativo entre os modelos levando em conta as acurácias de teste

Nesta seção reunimos os modelos elencados nas seções anteriores e realizamos o teste das mesmas em um conjunto de teste que possui 800 imagens jamais vistas pelos modelos de rede neural treinados, dessa forma podemos avaliar o desempenho dessas redes e termos uma primeira noção de quais modelos serão os mais indicados com relação as acurácias.

Vemos na figura 4.38 abaixo que o melhor modelo levando em conta a acurácia de teste é o modelo 64x64 colorido, porém, nos cabe uma análise mais profunda de todos esses modelos levando em consideração o tamanho desses modelos e a capacidade dos microcontroladores pré-selecionados.

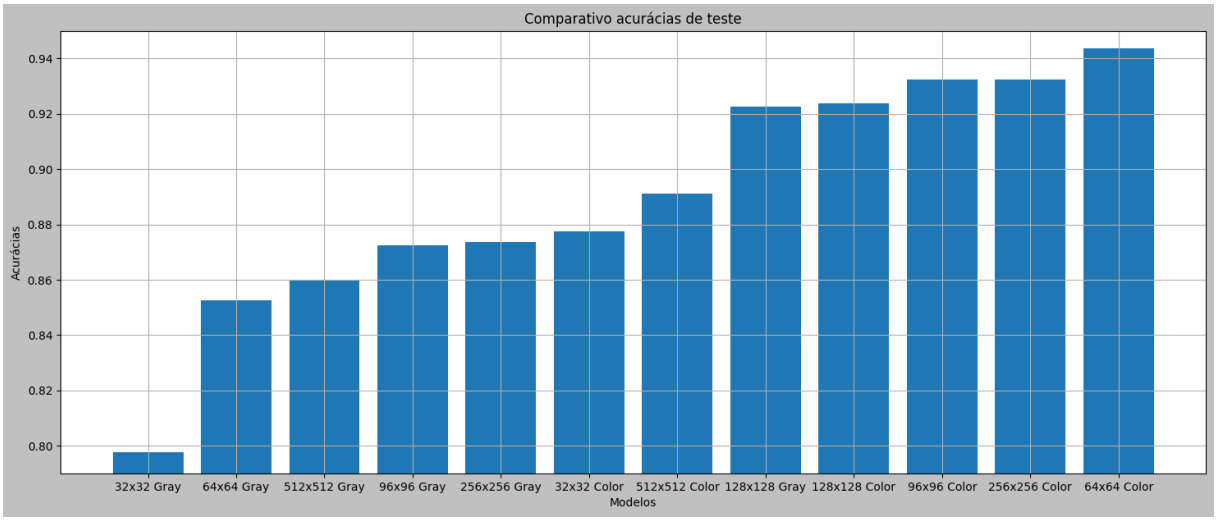

Figura 4.38: Gráfico comparativo entre as acurácias de teste dos modelos

4.16 Análise de tamanho dos modelos

A seguir, na tabela 4.1 temos um comparativo entre os modelos convencionais e convertidos, podemos ver que os modelos apenas convertidos e podados sofreram uma redução de tamanho que variam entre 3 e 7%, vemos também que os modelos quantizados sofreram uma redução de até 75.05% relacionados aos modelos convencionais, isto indica que os modelos quantizados, como já era de se esperar se darão melhor em relação as restrições de memória.

Além disso na mesma tabela podemos observar o tamanho das imagens levando em conta o tipo de dados utilizado no modelo, bem como o total de memória utilizado (tamanho do modelo + tamamho da image) em cada um dos cenários, observamos como esperado que o menor modelo seria o 32x32 em escala de cinza, e o maior modelo é o 512x512 com imagens coloridas.

Tabela 4.1: Tabela comparativa entre os tamanhos dos modelos

Resolução(pixels)(canais)	Modelo	Modelo(bytes)	Imagem(bytes)	Total(bytes)	Redução(%)
32x32 Gray	Convencional	423520	4096	427616	0.00%
	TFLite	394660	4096	398756	6.81%
	TFLite Quantizado	107144	1024	108168	74.70%
32x32 Color	Convencional	425600	12288	437888	0.00%
	TFLite	395812	12288	408100	7.00%
	TFLite Quantizado	108296	3072	111368	74.55%
64x64 Gray	Convencional	429664	16384	446048	0.00%
	TFLite	400804	16384	417188	6.72%
	TFLite Quantizado	107200	4096	111296	75.05%
64x64 Color	Convencional	431744	49152	480896	0.00%
	TFLite	401956	49152	451108	6.90%
	TFLite Quantizado	108352	12288	120640	74.90%
96x96 Gray	Convencional	439904	36864	476768	0.00%
	TFLite	411044	36864	447908	6.56%
	TFLite Quantizado	109760	9216	118976	75.05%
96x96 Color	Convencional	441984	110592	552576	0.00%
	TFLite	412196	110592	522788	6.74%
	TFLite Quantizado	110912	27648	138560	74.91%
128x128 Gray	Convencional	454240	65536	519776	0.00%
	TFLite	425380	65536	490916	6.35%
	TFLite Quantizado	113344	16384	129728	75.05%
128x128 Color	Convencional	456320	196608	652928	0.00%
	TFLite	426532	196608	623140	6.53%
	TFLite Quantizado	114496	49152	163648	74.91%
256x256 Gray	Convencional	552554	262144	814698	0.00%
	TFLite	523684	262144	785828	5.22%
	TFLite Quantizado	137920	65536	203456	75.04%
256x256 Color	Convencional	554624	786432	1341056	0.00%
	TFLite	524836	786432	1311268	5.37%
	TFLite Quantizado	139072	196608	335680	74.92%
512x512 Gray	Convencional	945760	1048576	1994336	0.00%
	TFLite	916900	1048576	1965476	3.05%
	TFLite Quantizado	236224	262144	498368	75.02%
512x512 Color	Convencional	947840	3145728	4093568	0.00%
	TFLite	918052	3145728	4063780	3.14%
	TFLite Quantizado	237376	786432	1023808	74.96%

4.16.1 Análise de tamanho e microcontroladores

Aqui, iremos fazer uma análise dos modelos e seus tamanhos com relação a limitação de memória dos microcontroladores, as quantidades de memória disponíveis são a seguinte:

- STM32L452 - 160 kilobytes de RAM
- NRF52840 - 256 kilobytes de RAM
- ESP32-S - 520 kilobytes de RAM
- NRF54H20 - 1024 kilobytes de RAM

Se observarmos a tabela 4.2, consideramos que é possível embarcar os modelos marcados pela cor verde, e que não é possível embarcar os modelos marcados como vermelho, a tabela

apresenta o tamanho total que uma imagem e o modelo ocupam na memória, mas devemos levar em consideração também o tamanho do código que precisa estar em memória ram. Dessa forma deixamos uma margem para que o firmware esteja presente.

Vemos que utilizando essa margem, o modelo 512x512 colorido não se encaixa em nenhum dos microcontroladores disponíveis, então o descartamos para futuras análises. Vemos que o modelo 512x512 em escala de cinza pode ser embarcado na NRF54H20, e por conseguinte todos os modelos menores que este. O próximo microcontrolador capaz de embarcar um modelo grande é a ESP32-S, ela possui memória o suficiente para o modelo 256x256 colorido, e por consequência, todos os modelos menores. Podemos embarcar no NRF52840 o modelo 128x128 colorido, seguido de todos os modelos menores. Já para o microcontrolador STM32L452 é possível embarcar o modelo 96x96 em escala de cinza e os modelos menores que este.

Tabela 4.2: Comparativo entre modelos e capacidade de memória ram das placas em bytes. verde: ram suficiente; vermelho: ram insuficiente

Resolução(pixels)(canais)	Modelo	STM32L452 MCU	NRF52840 MCU	ESP32-S MCU	NRF54H20 MCU
32x32 Gray	Convencional	427616	427616	427616	427616
	TFLite	398756	398756	398756	398756
	TFLite Quantizado	108168	108168	108168	108168
32x32 Color	Convencional	437888	437888	437888	437888
	TFLite	408100	408100	408100	408100
	TFLite Quantizado	111368	111368	111368	111368
64x64 Gray	Convencional	446048	446048	446048	446048
	TFLite	417188	417188	417188	417188
	TFLite Quantizado	111296	111296	111296	111296
64x64 Color	Convencional	480896	480896	480896	480896
	TFLite	451108	451108	451108	451108
	TFLite Quantizado	120640	120640	120640	120640
96x96 Gray	Convencional	476768	476768	476768	476768
	TFLite	447908	447908	447908	447908
	TFLite Quantizado	118976	118976	118976	118976
96x96 Color	Convencional	552576	552576	552576	552576
	TFLite	522788	522788	522788	522788
	TFLite Quantizado	138560	138560	138560	138560
128x128 Gray	Convencional	519776	519776	519776	519776
	TFLite	490916	490916	490916	490916
	TFLite Quantizado	129728	129728	129728	129728
128x128 Color	Convencional	652928	652928	652928	652928
	TFLite	623140	623140	623140	623140
	TFLite Quantizado	163648	163648	163648	163648
256x256 Gray	Convencional	814698	814698	814698	814698
	TFLite	785828	785828	785828	785828
	TFLite Quantizado	203456	203456	203456	203456
256x256 Color	Convencional	1341056	1341056	1341056	1341056
	TFLite	1311268	1311268	1311268	1311268
	TFLite Quantizado	335680	335680	335680	335680
512x512 Gray	Convencional	1994336	1994336	1994336	1994336
	TFLite	1965476	1965476	1965476	1965476
	TFLite Quantizado	498368	498368	498368	498368
512x512 Color	Convencional	4093568	4093568	4093568	4093568
	TFLite	4063780	4063780	4063780	4063780
	TFLite Quantizado	1023808	1023808	1023808	1023808

Vimos na seção anterior que o melhor modelo com relação á acurácia de teste foi o 64x64 colorido. Vale ressaltar que este modelo não é comportado pelo STM32L452, e para este microcontrolador iremos fazer uma outra análise com relação a um modelo menor.

4.17 Análise final

Nesta seção iremos efetuar uma análise final dos modelos obtidos, efetuando a medição de acurácia de teste dos modelos convertidos para TFLite e TFLite quantizado, a seguir iremos expor algumas matrizes de confusão referentes a todos os modelos, exceto o modelo 512x512 colorido que foi descartado anteriormente.

A seguir os resultados de teste para o modelo 32x32 colorido convertido (figura 4.39), obteve 91.5% de acurácia em sua versão TFLite quantizada.

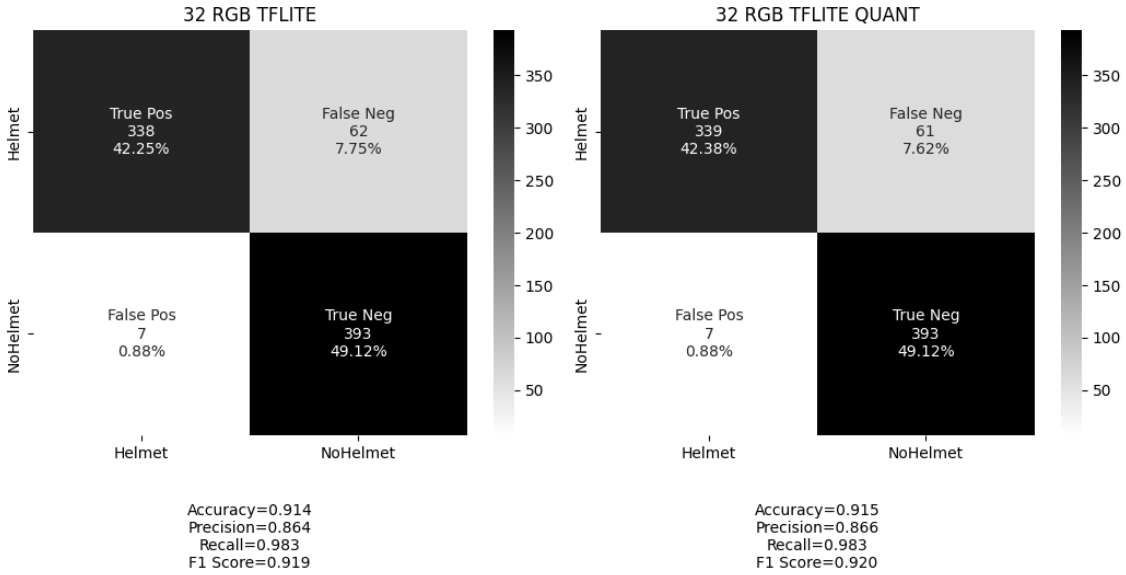

Figura 4.39: Matriz confusão 32x32 colorido convertido

Abaixo os resultados de teste para o modelo 32x32 em escala de cinza convertido (figura 4.40), obteve 81.1% de acurácia em sua versão TFLite quantizada.

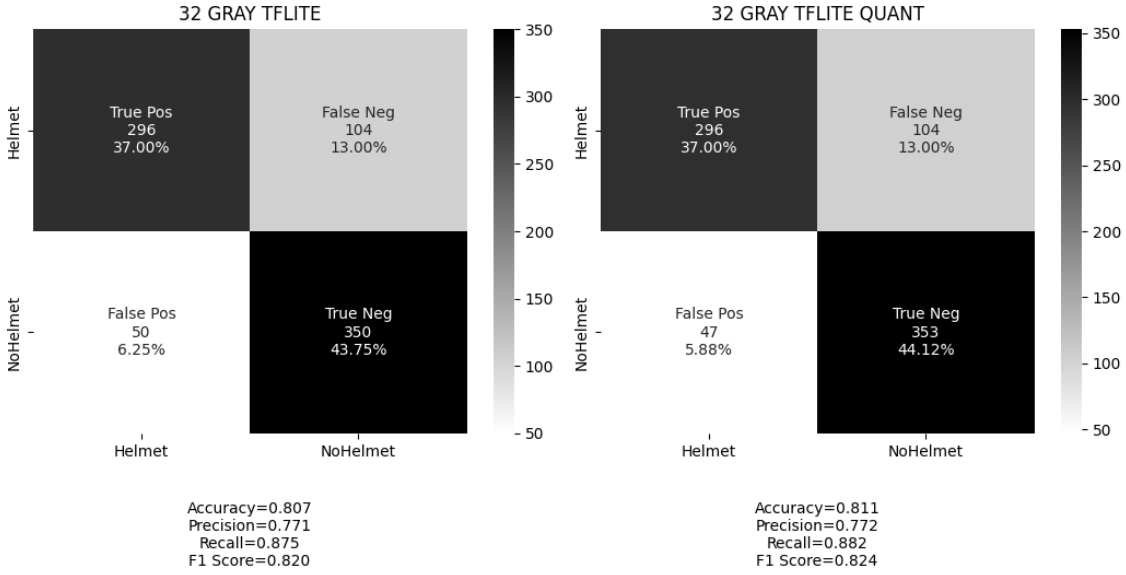

Figura 4.40: Matriz confusão 32x32 em escala de cinza convertido

Os resultados de teste para o modelo 64x64 colorido convertido (figura 4.41), obteve 94.5% de acurácia em sua versão TFLite quantizada.

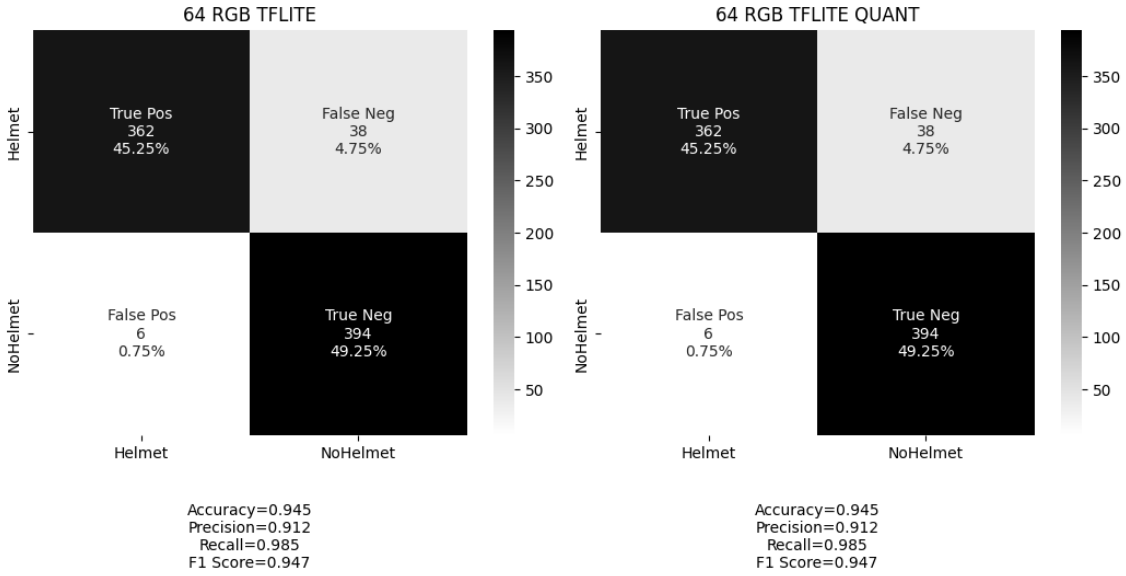

Figura 4.41: Matriz confusão 64x64 colorido convertido

Os resultados de teste para o modelo 64x64 em escala de cinza convertido (figura 4.42), obteve 85.6% de acurácia em sua versão TFLite quantizada.

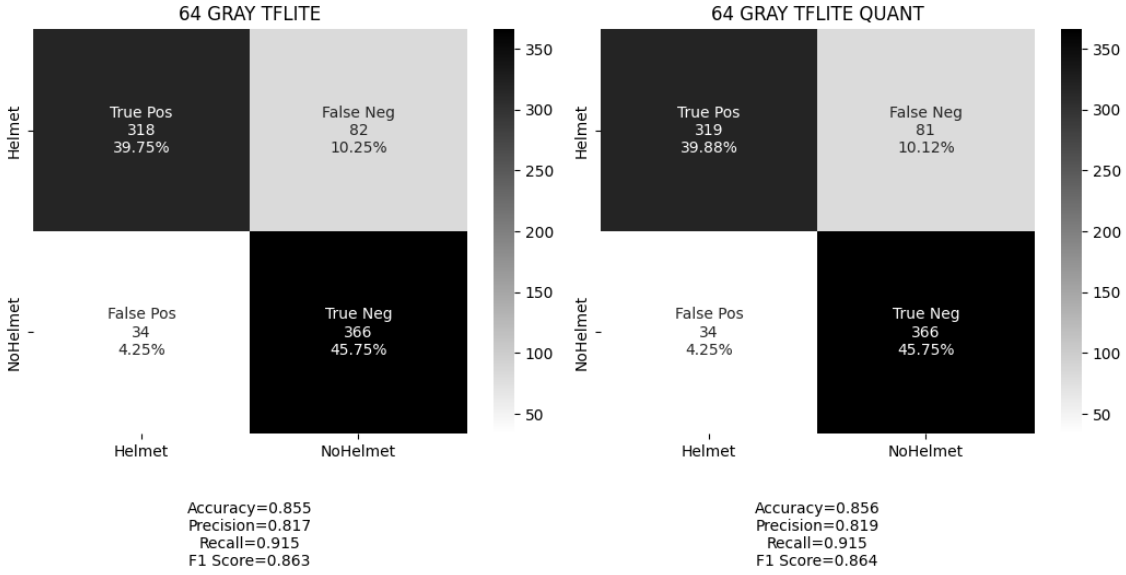

Figura 4.42: Matriz confusão 64x64 em escala de cinza convertido

Os resultados de teste para o modelo 96x96 colorido convertido (figura 4.43), obteve 94.8% de acurácia em sua versão TFLite quantizada.

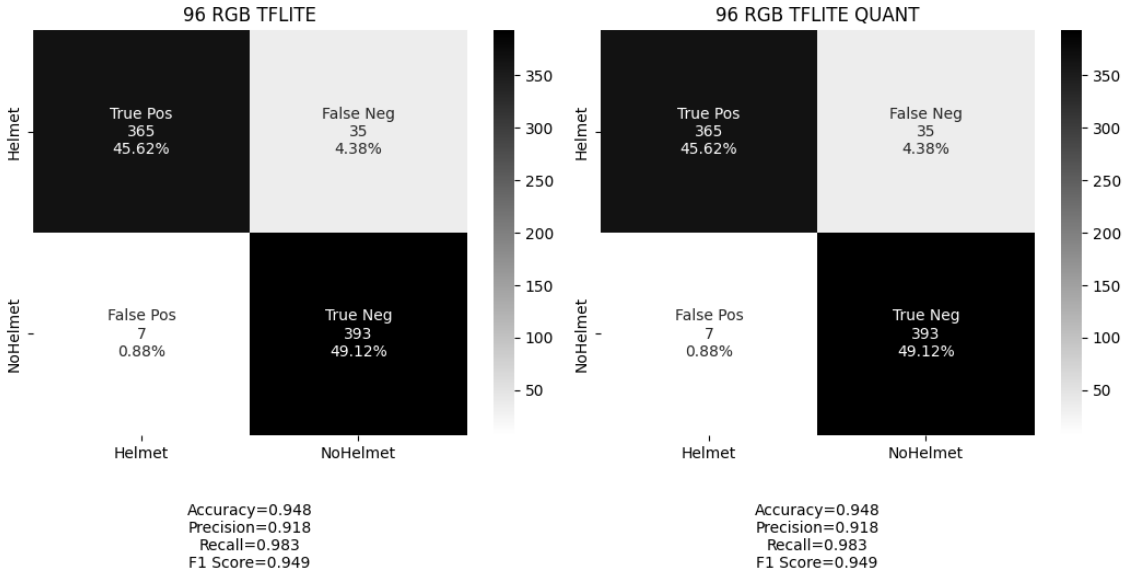

Figura 4.43: Matriz confusão 96x96 colorido convertido

Os resultados de teste para o modelo 96x96 em escala de cinza convertido (figura 4.44), obteve 89.1% de acurácia em sua versão TFLite quantizada.

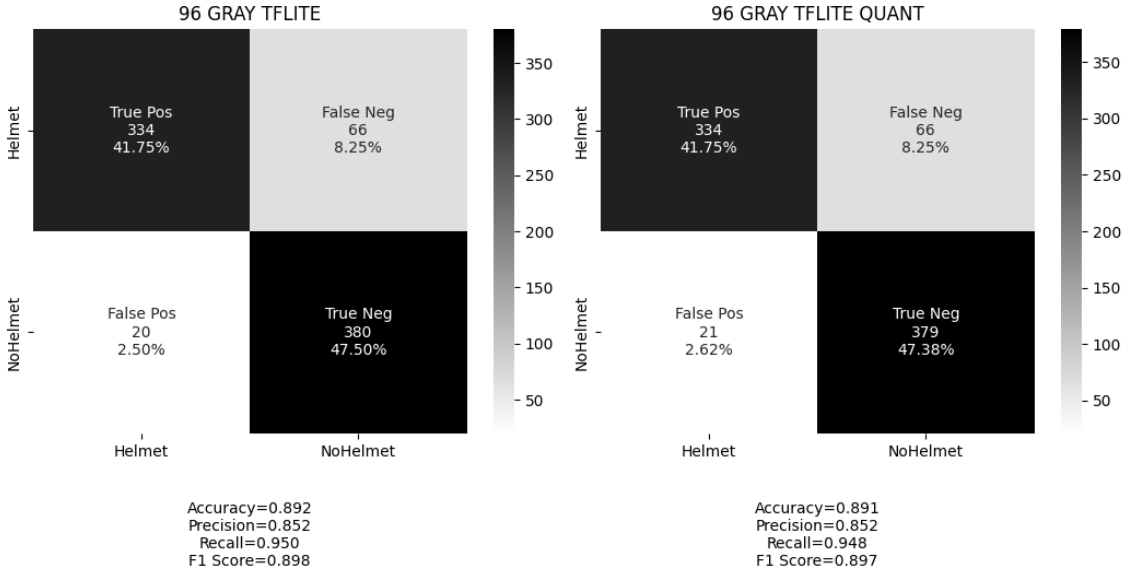

Figura 4.44: Matriz confusão 96x96 em escala de cinza convertido

Os resultados de teste para o modelo 128x128 colorido convertido (figura 4.45), obteve 92.9% de acurácia em sua versão TFLite quantizada.

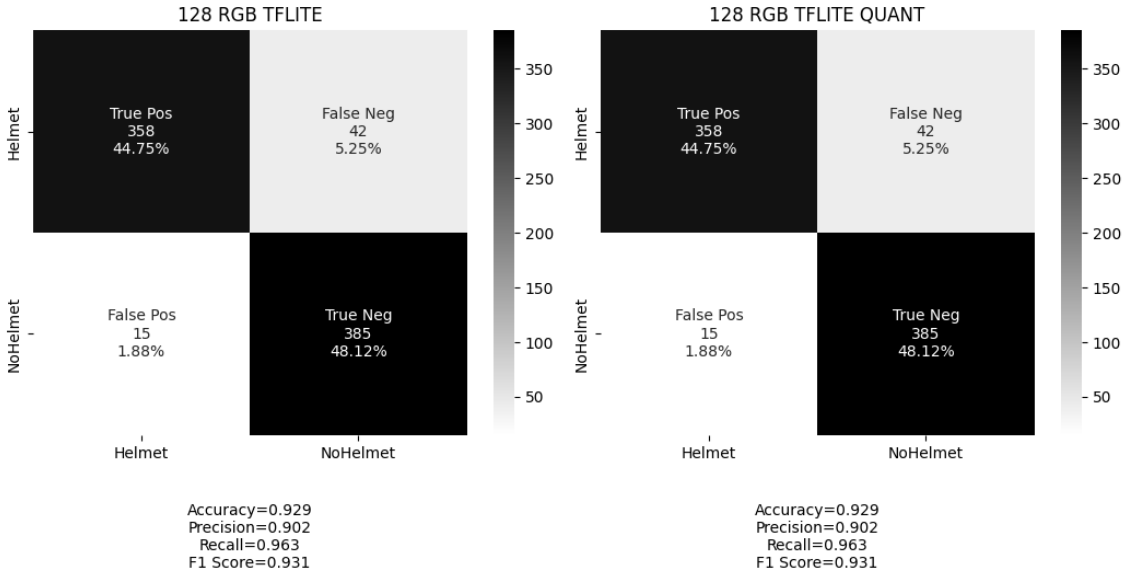

Figura 4.45: Matriz confusão 128x128 colorido convertido

Os resultados de teste para o modelo 128x128 em escala de cinza convertido (figura 4.46), obteve 91.6% de acurácia em sua versão TFLite quantizada.

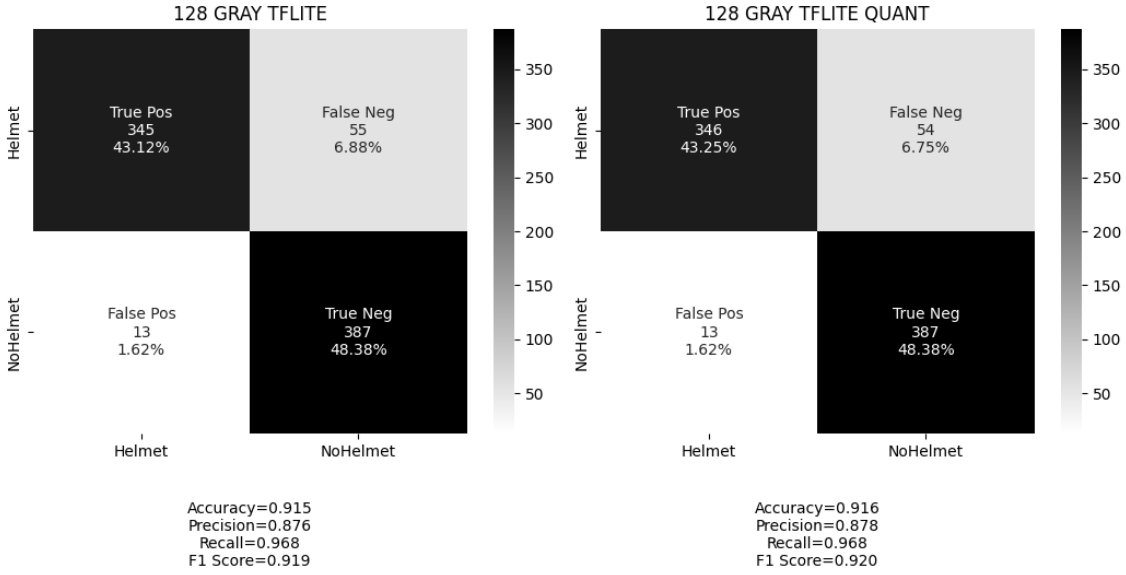

Figura 4.46: Matriz confusão 128x128 em escala de cinza convertido

Os resultados de teste para o modelo 256x256 colorido convertido (figura 4.47), obteve 93.6% de acurácia em sua versão TFLite quantizada.

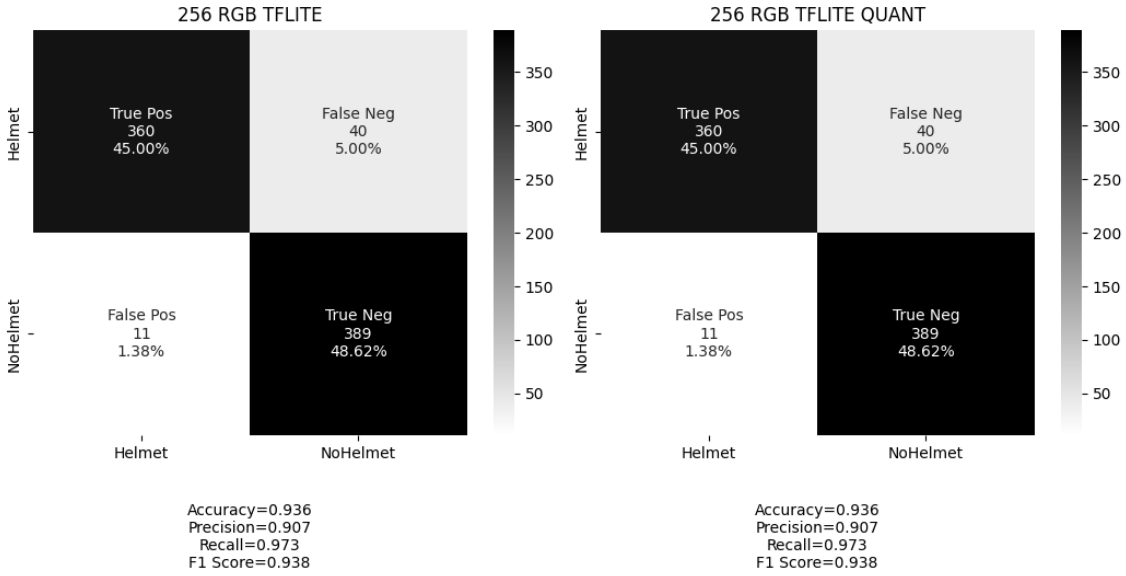

Figura 4.47: Matriz confusão 256x256 colorido convertido

Os resultados de teste para o modelo 256x256 em escala de cinza convertido (figura 4.48), obteve 89.7% de acurácia em sua versão TFLite quantizada.

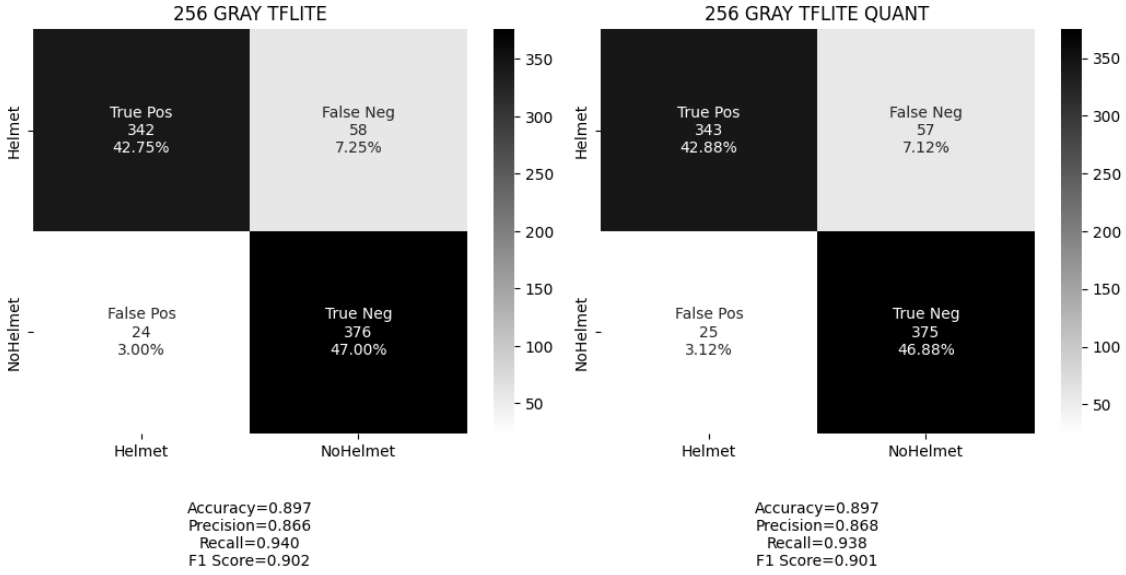

Figura 4.48: Matriz confusão 256x256 em escala de cinza convertido

Os resultados de teste para o modelo 512x512 em escala de cinza convertido (figura 4.49), obteve 86% de acurácia em sua versão TFLite quantizada.

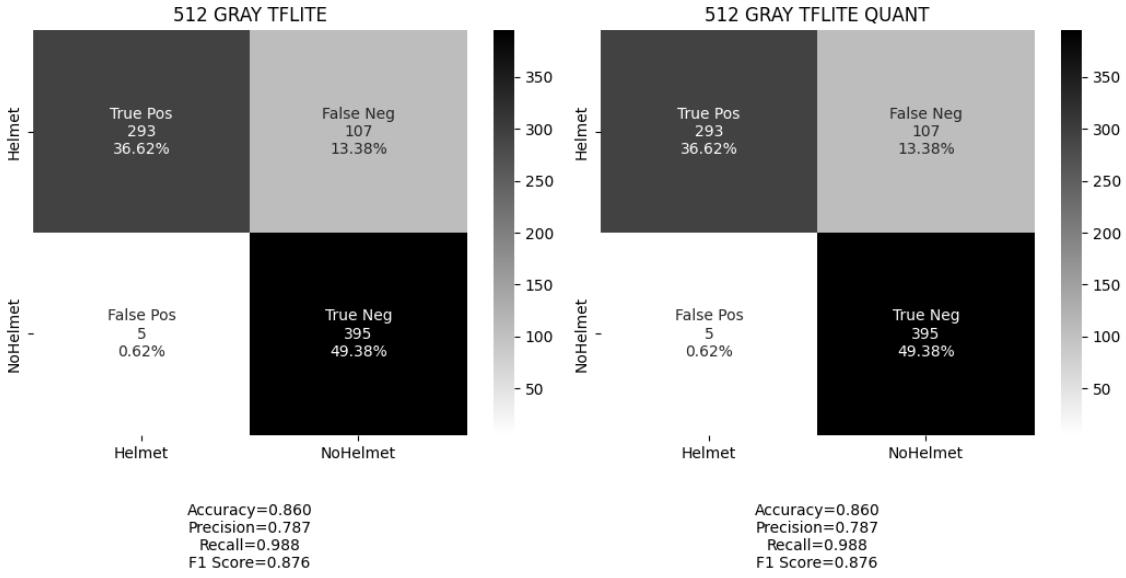

Figura 4.49: Matriz confusão 512x512 em escala de cinza convertido

Reunindo todas as informações das matrizes de confusão apresentadas acima em um único gráfico comparativo (figura 4.50) para os modelos quantizados, pois estes consumem menos memória e são ideais para embarcar, temos o seguinte resultado:

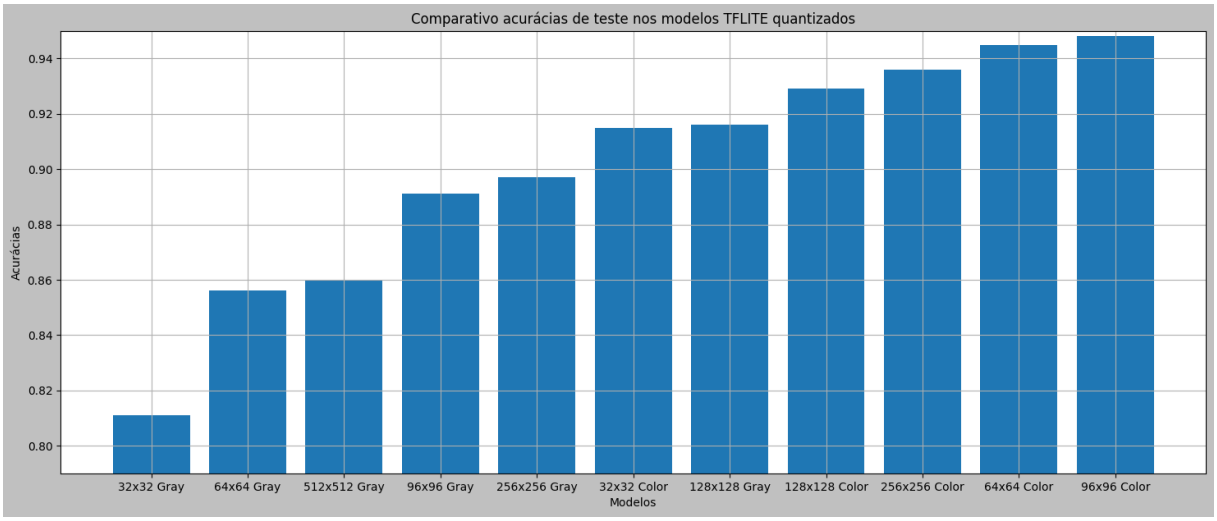

Figura 4.50: Matriz confusão 512x512 em escala de cinza convertido com quantização

Vemos que o modelo 96x96 colorido foi o que melhor se destacou, seguido de perto pelo modelo 64x64 colorido (figura 4.50). Ambos os modelos são uma excelente escolha (levando em conta a tabela 4.2 para embarcar nos microcontroladores NRF52840, ESP32-S e NRF54H20, porém, não é possível embarcar esses modelos no STM32L452 por conta de restrições de memória constantes na mesma tabela. Para tanto, iremos fazer uma análise específica para este microcontrolador.

O melhor modelo para embarcar no STM32L452 é o modelo 32x32 colorido, vimos pela figura 4.50 que ele é o melhor modelo com baixa dimensão, reforçado pela tabela 4.2 que explicita o tamanho do modelo.

Vimos que levando em consideração os modelos e cenários treinados, não faria sentido utilizar modelos maiores que 96x96 coloridos nesta problemática, estes modelos seriam mais pesados computacionalmente e segundo nosso conjunto de teste seriam menos precisos. Os modelos em escala de cinza ainda são úteis em situações onde o hardware só consegue capturar imagens em escala de cinza, sendo assim o modelo 128x128 se torna o melhor modelo em escala de cinza.

5

Conclusão

DURANTE este trabalho, abordamos a problemática de se utilizar modelos de aprendizagem de máquina em sistemas embarcados. Vimos os desafios que temos ao portar esses modelos para esses dispositivos, como pouca quantidade de memória, pouco poder de processamento, e outras restrições.

Fizemos isso por meio de uma análise de viabilidade relacionada a implementação de um modelo de reconhecimento de imagem que identifica o porte de capacete de segurança Vimos que o uso deste equipamento reduz drasticamente o número de lesões ou acidentes fatais no ambiente de trabalho, e que seria viável a criação de ferramentas que verifiquem o uso correto destes equipamentos, bem como oriente os funcionários a utilizá-los de forma eficiente.

Mais adiante, observamos as diversas técnicas e ferramentas que possibilitam a criação desses modelos, desde a criação do banco de dados utilizando a ferramenta Stable Diffusion, até a conversão destes modelos utilizando TensorFlow Lite a fim de se reduzir o tamanho dos mesmos para que seja possível embarcar em dispositivos com as limitações já citadas.

Por fim, nós apresentamos os resultados obtidos nesta monografia, que partem desde a criação do banco de dados para treinamento dos modelos, até a conversão de alguns modelos que julgamos ser candidatos mais relevantes para embarcar. Observamos que embora os modelos em escala de cinza propiciam um tamanho menor em comparação aos modelos coloridos, eles não apresentaram um resultado tão satisfatório em comparação aos modelos coloridos, levando em consideração a acurácia de teste obtida em todos os modelos.

O melhor modelo em escala de cinza (128x128, cujo tamanho é maior considerando alguns modelos coloridos) teve resultado semelhante ao menor dos modelos coloridos (32x32), desta forma, consideramos inviável implementar estes modelos em sistemas embarcados nesse contexto, a não ser que o hardware requisite uma implementação em escala de cinza.

Com relação aos modelos coloridos, elencamos os seguintes: 96x96 e 64x64 como melhores candidatos para os sistemas embarcados. Podemos utilizar estes em microcontroladores que possuam maior disponibilidade de memória ram (NRF52840, ESP32-S e NRF54H20), sendo

então o fator decisivo para a escolha destes modelos o tempo gasto para realizar a inferência sobre a imagem capturada.

Uma menção honrosa ao modelo 32x32, pois é ideal para hardware que possui pouca RAM, como é o caso do microcontrolador STM32L452, que possui 160 kilobytes de RAM, restando um espaço suficiente para a área de código que irá acompanhar o modelo.

No geral, vimos que é possível embarcar estes modelos em sistemas embarcados, atingindo assim o nosso objetivo proposto com o trabalho. Nossa monografia ilumina a possibilidade de novos trabalhos a serem executados posteriormente, como a implementação desses modelos em sistemas embarcados e a análise do tempo de inferência, bem como a proposição de algum produto que possa ser instalado no ambiente de trabalho com o intuito de reforçar no combate aos acidentes de trabalho. Além disso, pode influenciar diversos outros pesquisadores a trabalharem neste ramo e desta maneira propor novas soluções para novos problemas que possam surgir.

Referências bibliográficas

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

[2] Gian Marco Iodice. *TinyML Cookbook: Combine artificial intelligence and ultra-low-power embedded devices to make the world smarter*. Packt, 2022. ISBN 9781801814973. URL https://www.packtpub.com/product/tinyml-cookbook/9781801814973.

[3] Pete Warden and Daniel Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[4] Rafael C Gonzalez. *Digital image processing*. Pearson education india, 2009.

[5] Simon JD Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[6] Abel Pinto, Isabel L. Nunes, and Rita A. Ribeiro. Occupational risk assessment in construction industry – overview and reflection. *Safety Science*, 49(5):616–624, 2011. ISSN 0925-7535. **DOI** https://doi.org/10.1016/j.ssci.2011.01.003. URL https://www.sciencedirect.com/science/article/pii/S0925753511000051.

[7] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[8] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[9] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *Osdi*, volume 16, pages 265–283. Savannah, GA, USA, 2016.

[10] Andrei Nikulin and Anatoly Romanov. Control over the use of personal protective equipment by employees, head protection. *Ecology, Environment and Conservation*, 23 (1):384–389, 2017.

[11] Kunfeng Wang, Chao Gou, Yanjie Duan, Yilun Lin, Xinhu Zheng, and Fei-Yue Wang. Generative adversarial networks: introduction and outlook. *IEEE/CAA Journal of Automatica Sinica*, 4(4):588–598, 2017.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[13] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[14] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. *Machine learning techniques for multimedia: case studies on organization and retrieval*, pages 21–49, 2008.

[15] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[17] Rich Caruana, Steve Lawrence, and C Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 13, 2000.

[18] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[21] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[22] TensorFlow Developers. Tensorflow. *Zenodo*, 2022.