UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Arthur Borsonaro Uezu

Estudo e demonstração de modelos de redes neurais profundas para classificação de imagens em aplicações móveis

São José do Rio Preto - SP

2024

Arthur Borsonaro Uezu

Estudo e demonstração de modelos de redes neurais profundas para classificação de imagens em aplicações móveis

Monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Universidade Estadual Paulista "Júlio de Mesquita Filho" - (UNESP)

Instituto de Biociências, Letras e Ciências Exatas - (IBILCE)

Departamento de Ciências de Computação e Estatística

Bacharelado em Ciência da Computação

Orientador: Lucas Correia Ribas

São José do Rio Preto - SP

2024

U22e

Uezu, Arthur Borsonaro
Estudo e demonstração de modelos de redes neurais profundas para classificação de imagens em aplicações móveis / Arthur Borsonaro Uezu. -- São José do Rio Preto, 2024
59 p.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (UNESP), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Lucas Correia Ribas

1. Inteligência Artificial. 2. Aplicações Móveis. 3. Redes Neurais Profundas. 4. Classificação de Imagens. 5. Transfer Learning. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

Arthur Borsonaro Uezu

Estudo e demonstração de modelos de redes neurais profundas para classificação de imagens em aplicações móveis

Monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Banca Examinadora:

Prof. Dr. Lucas Correia Ribas
UNESP - Câmpus São José do Rio Preto
Orientador

Prof. Dr. Arnaldo Candido Junior
UNESP - Câmpus São José do Rio Preto

Prof. Dr. Rodrigo Capobianco Guido
UNESP - Câmpus São José do Rio Preto

São José do Rio Preto - SP
13/11/2024

Dedico esta monografia às políticas de permanência estudantil da UNESP, Campus São José do Rio Preto, que me permitiram ingressar e me manter durante o curso.

Agradecimentos

Agradeço aos meus pais, Angela e Jorge por confiarem em mim e me permitirem investir em meu futuro.

Agradeço aos meus padrinhos, Marineide e Pedro, por sempre me apoiarem.

Agradeço ao meu antigo bloco da moradia estudantil, o 2C, pelas amizades, histórias e momentos inesquecíveis.

Agradeço ao meu orientador, Lucas, pelo voto de confiança e dedicação ao seu trabalho.

"Concedei-me a serenidade para aceitar as coisas que não posso mudar, a coragem para mudar o que posso e a sabedoria para reconhecer a diferença."

(Epicteto)

Resumo

Esta monografia apresenta um estudo comparativo sobre o uso de redes neurais profundas para a classificação de imagens em dispositivos móveis, utilizando modelos otimizados para o contexto móvel como o MobileNet, InceptionV3 e ResNet. A pesquisa também abrange a implementação de uma plataforma móvel utilizando React Native e TensorFlow.js, na qual modelos pré-treinados são adaptados para operar em dispositivos com recursos limitados. Foi desenvolvido um aplicativo demo que permite a classificação de imagens em tempo real, destacando o equilíbrio entre precisão, consumo de recursos e desempenho. Além disso, são discutidos aspectos como o uso de Transfer Learning e os desafios inerentes à execução de redes neurais em dispositivos móveis além da comparação entre diferentes modelos quanto à sua eficiência e impacto no desempenho dos dispositivos. Ao final, é comprovada a eficiência e possibilidade de uso da técnica utilizada a partir dos resultados gerados e do desenvolvimento da plataforma demo.

Palavras-chave: Redes Neurais Profundas, Classificação de Imagens, Aplicações Móveis, Transfer Learning.

Abstract

This monograph presents a comparative study on the use of deep neural networks for image classification on mobile devices, utilizing models optimized for the mobile context, such as MobileNet, InceptionV3, and ResNet. The research also covers the implementation of a mobile platform using React Native and TensorFlow.js, where pre-trained models are adapted to operate on resource-constrained devices. A demo application was developed that allows real-time image classification, highlighting the balance between accuracy, resource consumption, and performance. Furthermore, aspects such as the use of Transfer Learning and the challenges inherent to running neural networks on mobile devices are discussed, as well as a comparison between different models regarding their efficiency and impact on device performance. In the end, the efficiency and feasibility of the technique used are proven through the generated results and the development of the demo platform.

Keywords: *Deep Neural Networks, Image Classification, Mobile Applications, Transfer Learning*

Sumário

Lista de ilustrações

Lista de abreviaturas e siglas

JS	*JavaScript*
1-NN	*1-Nearest Neighbor*
KNN	*K-Nearest Neighbor*
EQM	*Erro Quadrático Médio*
DMA	*Distância Média Absoluta*
RNA	*Redes Neural Artificial*
CNN	*Redes Neural Convolucional*
NLP	*Processamento de Linguagem Natural*
NPM	*Gerenciador de Pacotes do Node*
XML	*eXtensible Markup Language*
JXL	*JavaScript eXtensible Language*

1 Introdução

Segundo o relatório *The Mobile Economy 2024* da *Global System for Mobile Communications Association* (GSMA, 2024) no ano de 2023 existiam 5.4 bilhões de usuários únicos de dispositivos móveis, este número representa cerca de 68% da população mundial, com expectativa de atingir 74% da população até 2030. Nesse sentido, a área de visão computacional se torna relevante, uma vez que é possível realizar a captura e processamento de imagens com facilidade e baixo custo. Atualmente, vários aplicativos utilizam recursos de imagens no seu funcionamento como para biometria (aplicativos de banco para detecção facial), aplicativos de segurança domiciliar (interfones inteligentes), ferramentas de pesquisa na web (como o *Google Lens* e ferramentas de captura de imagens (como detecção de sorrisos). Além disso, houve uma explosão de aplicativos que tem como base a produção e consumo de imagens, tendo como exemplo aplicativos como o *TikTok*, um aplicativo baseado em conteúdo em vídeo feito por pessoas, para pessoas que, segundo relatório interno (TIKTOK, 2024) em 2023 atingiu 1 bilhão de usuários ativos mensais.

Observamos a popularidade de temas como Inteligência Artificial, Aprendizado de Máquina e Aprendizado Profundo na mídia e no mercado como mostra a Figura 1.1, porém, o seu uso em aplicações móveis abertas ao público não é recente, como em aplicativos de reconhecimento de voz desde 2014 (SHAO et al., 2019), correção de texto (MICROSOFT, 2024) e reconhecimento facial para marcação de fotos desde 2010 (PCMAG, 2010). Com o avanço da técnica, recursos para a utilização dessa tecnologia foram se tornando cada vez mais acessíveis, como a criação de bibliotecas e *frameworks* de desenvolvimento focados em inteligência artificial para aplicações móveis como *TensorFlow Lite* (TENSORFLOW, 2024), *PyTorch Mobile* (PYTORCH, 2024), *CoreML* (APPLE, 2024) e *ONNX* (ONNX, 2017).

Figura 1.1 – Popularidade da pesquisa no termo "Inteligência Artificial"entre 5 de maio de 2019 à 5 de maio de 2024

Fonte: https://trends.google.com.br/trends

A grande maioria das aplicações móveis que utilizam inteligência artificial de paradigma descritivo tem seus atributos e classes de saída bem definidos, servindo como um

software especialista, como foi citado nos exemplos anteriores, esses modelos servem para resolver um objetivo específico, como o reconhecimento de voz humana, reconhecimento de faces e digitais, porém, poucas delas permitem que seus usuários, de maneira ativa modificarem e reapropriarem um modelo já existente. Isso é possível através do método de *Transfer Learning*, onde podemos utilizar um vetor de características que é gerado por um modelo para modificar o espaço de atributos de saída, como por exemplo utilizar um modelo classificador de vozes para reconhecer outros tipos de sons, como de apitos, instrumentos ou uma voz específica (HOU et al., 2020).

Apesar do *Transfer Learning* ser uma técnica conhecida e muito utilizada, a aplicação dela no contexto de aplicação móvel apresenta dilemas que serão abordadas durante a dissertação da monografia, citando alguns exemplos (LIU et al., 2019):

1. Viés de busca (treinamento) em nuvem ou local e seu impacto na performance e utilização do aparelho.
2. Coleta e tamanho ideal do conjunto de dados.
3. Performance do modelo utilizando recursos móveis.
4. Modelo mais adequado ao contexto móvel, considerando fatores como utilização de memória física e tempo de processamento.

Todos esses fatores tem em comum sua preocupação com os recursos limitados do dispositivo e podem variar com o tipo da aplicação e seus requisitos, podemos utilizar como exemplo o reconhecimento de voz (SHAO et al., 2019) que utiliza um modelo que é executada de forma local, sem a necessidade de uma execução em nuvem, o que possibilita a utilização sem conexão com a Internet e é distribuído junto com o aparelho. O mesmo não é observado no reconhecimento de faces (PCMAG, 2010), onde o usuário carrega uma imagem na aplicação e um servidor realiza o reconhecimento e marcação do usuário reconhecido na foto por ser em base uma aplicação web e utilizar o modelo cliente-servidor e também pelos ganhos de performance dentro do aplicativo, que não tem de executar o modelo de maneira local (LIU et al., 2019).

1.1 Objetivos

Essa monografia se propõe a ser um estudo comparativo entre as atuais redes neurais convolucionais para dispositivos móveis, além de também promover um aplicativo *demo*, onde o mesmo será usado para realizar as comparações e servir como um caso de sucesso a ser seguido na implementação das mesmas.

Os principais aspectos a serem abordados são:

- Comparar as principais redes neurais convolucionais disponíveis para o uso, medindo sua acurácia, tempo de treinamento e seu impacto no dispositivo.
- Criar a plataforma a partir da ferramenta *React Native*, uma biblioteca de criação de aplicativos móveis.

Para que esses objetivos sejam alcançados, há a necessidade da criação de uma plataforma que permita de modo fácil a troca dos modelos utilizados e que tenha meios de medir descritores de performance dos mesmos para gerar estudos comparativos.

2 Fundamentação e Revisão da Literatura

Para a viabilização do projeto, há de se entender alguns conceitos como Inteligencia Artificial, Aprendizado de Máquina, modelos, *React Native*, *TensorFlow* e *TensorFlowJS*.

2.1 Inteligencia Artificial

Apesar da falta de consenso da comunidade científica em definir o que exatamente é Inteligência Artificial, autores clássicos da literatura como Russell e Norvig (2021) definem esse campo como o estudo de agentes que conseguem:

1. Pensar como humanos (imitar processos de pensamento humano).
2. Agir como humanos (replicar o comportamento humano).
3. Pensar racionalmente (simular a "mente" racional).
4. Agir racionalmente (tomar decisões para atingir metas com eficiência e precisão).

Inicialmente, essas ações eram realizadas por meio de sistemas especialistas, que a partir de uma pessoa especialista no assunto do domínio do problema a ser resolvido extraia uma série de regras e parâmetros para tomada de decisão do mesmo e a traduzia em código (FACELI et al., 2011).

Contudo, com o aumento da complexidade dos problemas a serem tratados e o fenômeno *Big Data*, tornou-se evidente a necessidade de sistemas autônomos que não dependessem da intervenção humana ou a dependência de especialistas (FACELI et al., 2011). A solução para este problema é o Aprendizado de Máquina.

2.2 Aprendizado de Máquina

Segundo Bishop (2006), Aprendizado de Máquina pode ser definida como um campo que envolve o desenvolvimento de algoritmos e modelos estatísticos que permitem aos computadores realizar tarefas específicas sem serem explicitamente programados para tal, mas sim a partir da observação de dados.

Para a tarefa de aprendizado, é necessário um conjunto de dados de entrada e uma função de indução, que consiga traduzir os atributos desses dados em um atributo de saída, conforme os dados são processados, o algoritmo de aprendizado calibra o modelo de solução (GÉRON, 2019).

O objetivo desse modelo pode ser separado em duas categorias, predição ou descrição. A predição utiliza os dados de treinamento para prever um rótulo ou valor com base nos atributos de entrada, normalmente seguindo um paradigma de algoritmo supervisionado, que guia o modelo a ser gerado já conhecendo a saída desejada para cada exemplo de entrada. Já a descrição tem a tarefa de descrever ou rotular um conjunto de dados, aproximando dados que parecem entre si e encontrando regras de associação e não necessita de um supervisor (GÉRON, 2019).

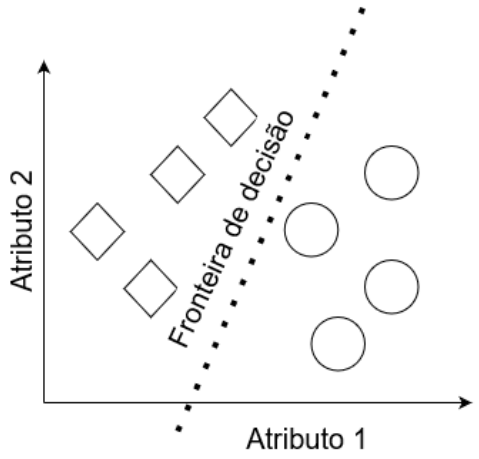
Esse processo de calibragem pode levar a fenômenos como *overfitting* ou *underfitting*, sendo o primeiro uma solução super-ajustada aos dados de treinamento e o segundo uma solução que é mal-ajustada ao que se propõe a fazer. Para garantir a qualidade do modelo, existem métricas e métodos que podem ser utilizados para assegurar uma boa precisão dos mesmos (FACELI et al., 2011).

2.2.1 Supervisionado

Os algoritmos de aprendizado supervisionado utilizam dados rotulados para relacionar dados de entrada com uma saída, que pode ser em formato de classe ou um valor numérico.

O aprendizado supervisionado contém, dentro de si, dois tipos de problema, um, se dá quando o conjunto das possibilidades de resposta é finita e nominal, sendo assim considerado um problema de **classificação**. O outro, é quando o conjunto de saída é infinito e ordenado, nesse caso, será um problema de **regressão** (GÉRON, 2019).

Figura 2.1 – Classificação

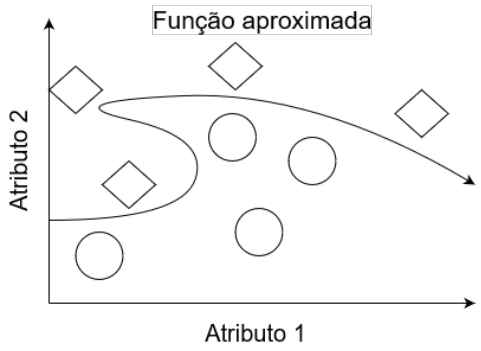


Fonte: Elaborado pelo autor, adaptado de (FACELI et al., 2011).

Classificação:

Algoritmos de classificação tem o objetivo de prever uma classe ou rótulo a partir de um conjunto de dados com classes ou rótulos conhecidos, conhecido como conjunto de

Figura 2.2 – Regressão



Fonte: Elaborado pelo autor, adaptado de (FACELI et al., 2011).

treinamento e, a partir disso, montar uma fronteira de decisão para que os dados de entrada sejam devidamente classificados.

Na Figura 2.1, é demonstrado a divisão feita pela fronteira de decisão entre duas classes diferentes, como por exemplo a identificação entre espécies diferentes de flores ou animais (RUSSELL; NORVIG, 2021).

Como exemplos de algoritmos simples de classificação, podemos citar:

1-Vizinho Mais Próximo (1-Nearest Neighbor, 1-NN)

Algoritmo simples que assume que dados parecidos ficarão mais próximos entre si, ele normalmente utiliza a função do cálculo da distância euclidiana para calcular a distância entre os dados rotulados (pertencentes a uma classe) que foram utilizados no treinamento e o dado de entrada, onde quando encontrado a menor distância entre eles, se assume que o dado de entrada pertença à mesma classe que o dado de treinamento encontrado.

Sua maior vantagem é a simplicidade de implantação, porém, esse algoritmo é muito afetado pela medida ou função de distância utilizada, onde a depender da escala de um dos atributos (seja muito grande, ou muito baixa), sua precisão é afetada, por isso, no pré-processamento é demandado a normalização de todos os dados.

K-Vizinhos Mais Próximos (K-Nearest Neighbor, K-NN)

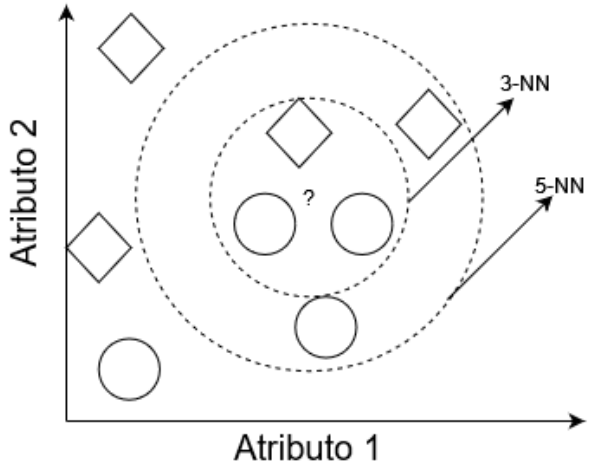
O K-NN é uma extensão do 1-NN, sua diferença é que, ao invés de utilizar o vizinho mais próximo, ele consulta os K vizinhos mais próximos que "votarão" na classe mais apropriada para aquele dado de entrada, a classe elegida será a classe de saída do algoritmo.

Na Figura 2.3, a interrogação sinaliza o dado de entrada e os diferentes formatos sinalizam classes, nesse caso, a votação em K=5 entre os círculos (3 votos) e os losangos (2 votos), a classe escolhida seria a dos círculos.

Regressão

Regressão utiliza estatística para quantificar uma saída a partir de uma variável de-

Figura 2.3 – K-NN



Fonte: Elaborado pelo autor.

pendente (resposta) com variáveis independentes (explicativas). (RUSSELL; NORVIG, 2021)

A regressão tenta atingir esse objetivo por tentar aproximar uma função de predição com base em dados de exemplo que se aproxime da função verdadeira em que esses dados se distribuem, como exemplo de alguns métodos simples de regressão, podemos citar:

Regressão linear

A regressão linear tenta predizer os valores dependentes ajustando uma linha reta aos valores independentes, tentando ao máximo se aproximar de uma função de reta real.

O modelo da regressão linear pode ser descrita como:

$$\min_{\beta} \left(\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right)$$

O objetivo da regressão linear é ajustar os coeficientes $\beta_0, \beta_1, \ldots, \beta_p$ de forma que a soma dos erros quadráticos seja minimizada, sendo y_i os valores observados, β_j são os coeficientes a serem ajustados, x_{ij} são os valores das variáveis explicativas.

Sua maior vantagem é a simples implementação e uso em muitas aplicações quantitativas, porém suas desvantagens incluem ser sensível a extremos (dados de treinamento com valores muito altos ou baixos em relação aos demais), e também assumir uma relação linear entre as variáveis.

Regressão de Ridge

A regressão de Ridge é uma extensão da regressão linear, mas que inclui uma regularização para evitar *overfitting*, um problema comum em situações onde há um número grande de variáveis explicativas ou há uma alta correlação entre elas, dessa forma, a regularização pode melhorar a generalização do modelo quando inserimos dados de teste.

O modelo da regressão de Ridge pode ser descrita como:

$$\min_{\beta} \left(\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right)$$

Onde o primeiro termo é o erro quadrático total, como na regressão linear, segundo termo, $\lambda \sum_{j=1}^{p} \beta_j^2$, é a penalização (regularização) sobre o tamanho dos coeficientes e λ é um parâmetro de regularização que controla o peso dessa penalização.

2.2.2 Não-supervisionado

O aprendizado não-supervisionado consiste em, a partir de propriedades intrínsecas aos dados de entrada, sem o auxílio de dados de exemplo, representar esses dados de maneira que seja relevante ao propósito da solução, seja identificando padrões e tendências ou facilitando o entendimento humano. Por esse motivo, nesses algoritmos, não existe um dado meta, isto é, um dado de saída (DECO; OBRADOVIC, 1999).

Dentro do aprendizado não-supervisionado, existem algumas tarefas descritivas que tem o objetivo de explorar ou descrever um conjunto de dados, essas, são divididas em 3 categorias (DECO; OBRADOVIC, 1999):

Sumarização: Com o objetivo de prover uma descrição simples e compacta dos dados de entrada, pode ser utilizado diversas ferramentas, de métodos estatísticos (média, mediana, desvio padrão) até softwares complexos de descrição de dados.

Associação: A associação tenta buscar um padrão entre os atributos do conjunto de dados de entrada, a fim de encontrar uma função ou relação causal entre os diferentes atributos. Pode-se citar dois algoritmos simples para a formação de regras de associação, *Apriori* e *Eclat*, que, de modo iterativo tentam por meio da formação de pares frequentes de dados encontrar um padrão.

Agrupamento: Já o agrupamento tenta encontrar similaridade entre os dados para a formação de grupos e a descrição entre eles. Um famoso algoritmo de agrupamento é o *K-Means*, onde ele forma *K* grupos a partir da calibração do ponto médio dos dados de cada grupo.

2.2.3 Estratégias de Validação

Para garantir a qualidade da solução induzida pelos algoritmos de inteligência artificial, foram criados diferentes métodos de avaliação, esses, podem ser divididos em avaliação de modelos de predição ou descrição (DAVIS; KULICK, 2005):

2.2.3.1 Avaliação de modelos preditivos

Em algoritmos supervisionados, a avaliação é feita a partir da análise de desempenho ao tentar classificar ou predizer valores que não foram utilizados no treinamento do mesmo.

Taxa de erro

A taxa de erro é a proporção de classificações incorretas, onde usualmente um acerto corresponde ao valor zero, e um erro corresponde ao valor um (função de custo 0-1), essa verificação pode ser expressa pela função (SPIEGEL; SCHILLER; SRINIVASAN, 2004):

$$\mathit{err}(f) = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq f(x_i))$$

Onde n é o número de objetos a serem classificados, f o classificador, a classe conhecida por y_i e a classe predita por $f(x_i)$.

O valor de $\mathit{err}(f)$ varia entre 0 e 1, onde quanto mais próximo à 0, menor a taxa de erros. A taxa de acerto pode ser obtida subtraindo 1 da taxa de erro ($ac(f) = 1 - \mathit{err}(f)$) (SPIEGEL; SCHILLER; SRINIVASAN, 2004).

Matriz de confusão

A matriz de confusão é uma maneira de representar matematicamente acertos e erros de classificação, onde linhas representam as classes reais e colunas as classes preditas pelo classificador, por esse motivo, caso existam k classes, a matriz será de dimensão kxk (FACELI et al., 2011).

Dentro de uma célula M_{ij} da matriz, é contido o número de vezes que a classe real i foi classificada como a classe j, assim a diagonal da matriz contém todos os acertos e as demais células os erros cometidos pelo classificador, dessa forma, podemos observar em quais classes o classificador apresenta maior dificuldade (FACELI et al., 2011).

Erro quadrático médio

Utilizado em problemas de regressão, o objetivo do método de avaliação do erro quadrático médio é verificar a distância entre o valor predito pela hipótese e o valor real, sendo essa distância equivalente ao valor do erro (KAY, 1993).

O valor do erro quadrático médio pode ser descrita como:

$$\text{EQM } (f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

f: Classificador. n: Número de objetos. y_i: Ponto real. x_i: Ponto predito.

Quanto menor o valor do EQM, melhor o desempenho da hipótese gerada pelo algoritmo de regressão (KAY, 1993).

Distância média absoluta

Com o mesmo princípio do Erro Quadrático Médio, seu objetivo é utilizar a distância absoluta como medida de erro entre valor predito e valor real (SPIEGEL; SCHILLER; SRINIVASAN, 2004).

$$\text{DMA } (f) = \frac{1}{n}\sum_{i=1}^{n} |y_i - f(x_i)|$$

f: Classificador. n: Número de objetos. y_i: Ponto real. x_i: Ponto predito.

O valor resultante que varia de 0 à 1 significa a quantidade média de erros (SPIEGEL; SCHILLER; SRINIVASAN, 2004).

Avaliação de modelos descritivos

A avaliação de modelos descritivos é uma tarefa complexa, pois, diferente dos modelos de aprendizado supervisionado, os resultados não são rotulados com uma resposta esperada, e também em muitos casos não há uma única solução para o problema dado (DECO; OBRADOVIC, 1999).

Nesse sentido, a avaliação se dá por meio da validação da estrutura gerada pelo modelo, para evitar coincidências e garantir que o resultado gerado seja consistentemente de qualidade e não apenas um "golpe de sorte".

Essa avaliação da qualidade estrutural é feita por meio de índices estatísticos que quantificam a qualidade de um aspecto de um agrupamento. Esses índices são utilizados por critérios de validação, que indicam como essas medidas serão utilizadas para validar a qualidade do agrupamento (DECO; OBRADOVIC, 1999).

2.3 Redes Neurais Artificiais

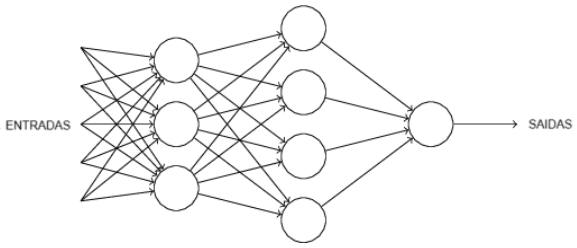
Redes Neurais Artificiais (RNAs) são modelos computacionais inspirados no funcionamento do cérebro humano. Projetados para reconhecer padrões complexos e aprender a partir de dados, elas são compostas por unidades interconectadas, chamadas neurônios artificiais (NIELSEN, 2015).

Um neurônio artificial é fundamental no processamento de uma rede neural artificial. Inspirado no neurônio biológico, ele recebe entradas numéricas e, a partir de um peso que controla a influência de cada valor de entrada, aplica uma função de ativação e gera uma saída.

O aprendizado em redes neurais se dá a partir do processo de ajuste dos pesos e viés de indução (treinamento) para que a rede de neurônios seja capaz de realizar tarefas específicas, como classificação ou previsão.

O treinamento de uma rede neural artificial pode ser dado a partir de tarefas de aprendizado supervisionadas ou não-supervisionadas, seguindo o critério do problema a ser resolvido. (RUSSELL; NORVIG, 2021)

Figura 2.4 – Rede neural simples

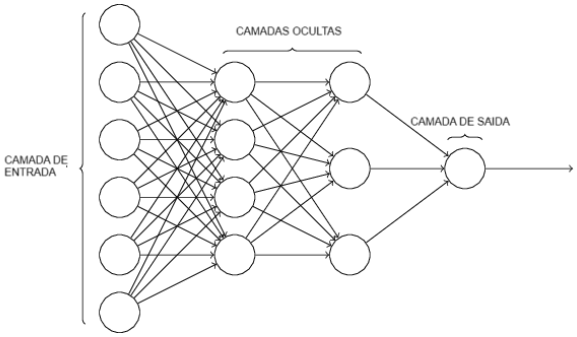


Fonte: Adaptado de (NIELSEN, 2015)

2.3.1 Redes Neurais Profundas

Uma Rede Neural Profunda é um tipo de rede neural artificial composta por múltiplas camadas de neurônios artificiais interconectados. A característica distintiva das redes neurais profundas é a presença de várias camadas ocultas entre a camada de entrada e a camada de saída. Essas camadas ocultas permitem que a rede aprenda representações hierárquicas de dados complexos, capturando características abstratas e não lineares dos dados de entrada (NIELSEN, 2015). As redes neurais profundas permitiram que diversas tarefas complexas fossem

Figura 2.5 – Rede neural profunda.



Fonte: Adaptado de (NIELSEN, 2015).

automatizadas, como o reconhecimento e classificação de imagens e objetos, processamento e geração de textos em linguagem natural, melhora na autonomia robótica e reconhecimento de voz.

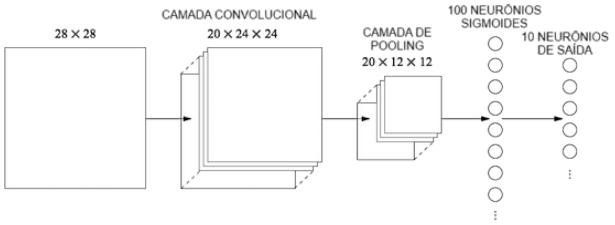
2.3.1.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais (CNNs) são arquiteturas projetadas especificamente para processar dados que têm uma estrutura de matriz, como imagens, vídeos, ou até dados

volumétricos. Elas foram inspiradas no processo visual dos animais, onde diferentes neurônios respondem a estímulos visuais distintos, como bordas, padrões e cores, criando uma hierarquia de abstração dos dados. As CNNs são amplamente utilizadas em tarefas de visão computacional, como reconhecimento de objetos, detecção de faces, e segmentação de imagens, devido à sua habilidade de capturar características espaciais e relações locais dos dados (NIELSEN, 2015).

Uma das principais características das CNNs é o uso de camadas convolucionais que são capazes de explorar a dependência espacial dos dados ao aplicar filtros (kernels) sobre pequenas regiões das imagens de entrada. Essas redes são eficientes na captura de padrões locais sem a necessidade de processamento de prévia extração de características. Cada camada subsequente em uma CNN tende a aprender características de nível mais alto, como formas complexas, objetos, ou até a semântica de uma imagem, conforme os dados passam por camadas sucessivas de convoluções e pooling. Essa estrutura pode ser observada na Figura 2.6.

Figura 2.6 – Rede neural convolucional.



Fonte: (NIELSEN, 2015)

As CNNs são divididas em três tipos principais de camadas que, combinadas, permitem a extração e interpretação das características dos dados:

Camadas de Convolução

As camadas de convolução são a base das CNNs e desempenham o papel de extrair características locais dos dados. Um filtro de tamanho fixo percorre a imagem de entrada e, em cada posição, realiza uma multiplicação e soma entre os valores da imagem e os valores do filtro, gerando um mapa de características. Cada filtro é treinado para reconhecer diferentes padrões, como bordas, texturas, ou partes de objetos .

Além disso, as convoluções podem ser realizadas com diferentes *stride* (passo) e *padding* (preenchimento) para ajustar o tamanho da saída em relação à entrada. Um passo maior resulta em um mapeamento de características menor e mais condensado, enquanto o preenchimento permite que as bordas da imagem sejam consideradas na convolução (RUSSELL; NORVIG, 2021).

Camadas de Pooling (Agrupamento)

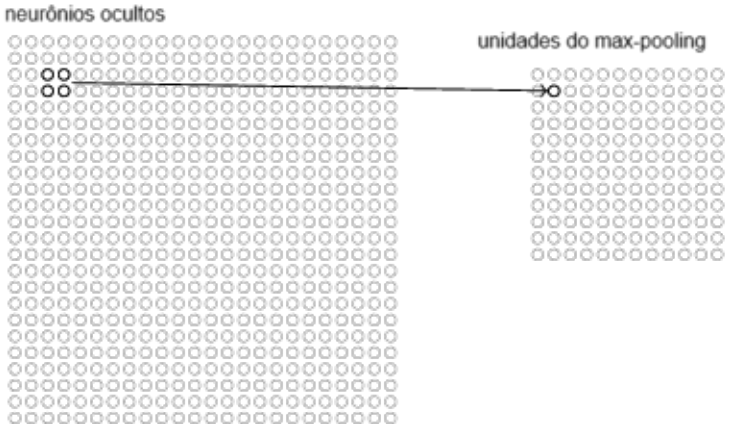
As camadas de *pooling* são usadas para reduzir a dimensionalidade espacial dos mapas de características gerados pelas camadas de convolução, mantendo as informações mais importantes. Essa redução diminui o custo computacional e a complexidade do modelo, além de mitigar o risco de *overfitting*.

O *pooling* pode ser realizado de várias formas, sendo as mais comuns:

- *Max pooling*: onde o valor máximo de cada região é selecionado.
- *Average pooling*: onde a média dos valores dentro de uma região é calculada.

Por exemplo, em um *max pooling* com um filtro 2x2 e *stride* de 2, apenas o maior valor em cada região 2x2 da entrada é mantido, e em um *average pooling* o a média dos valores de cada região 2x2 é mantida.

Figura 2.7 – Max-pooling.



Fonte: (NIELSEN, 2015)

Camadas Totalmente Conectadas

Após as camadas de convolução e agrupamento, a CNN utiliza camadas totalmente conectadas (*fully connected layers*) para combinar as características extraídas e realizar a classificação ou regressão. Nessas camadas, cada neurônio está conectado a todos os neurônios da camada anterior, assim como nas redes neurais tradicionais (RUSSELL; NORVIG, 2021).

O objetivo das camadas totalmente conectadas é integrar todas as características aprendidas nas camadas anteriores para tomar a decisão final, seja na classificação de imagens ou em outra tarefa.

Funções de Ativação

Em cada uma dessas camadas, as CNNs utilizam funções de ativação não lineares, como a *ReLU (Rectified Linear Unit)*, que aplica a função $f(x) = max(0, x)$. Isso permite

que a rede capture não linearidades nas características aprendidas, tornando-a capaz de lidar com dados complexos, como imagens.

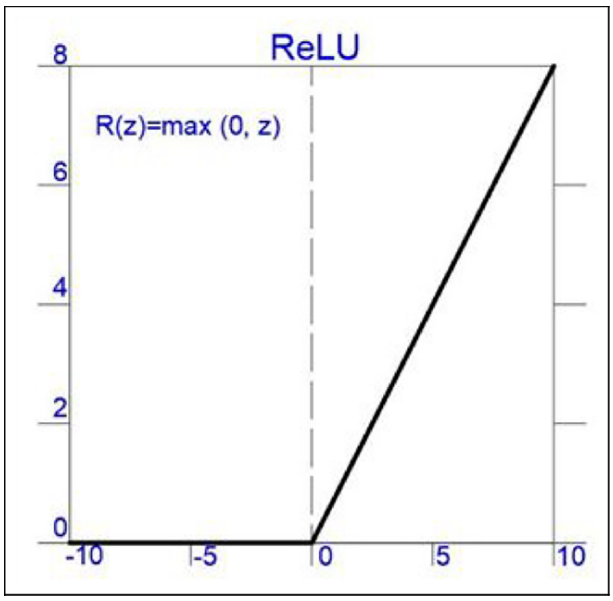


Figura 2.8 – Função de ativação $ReLU$.

Fonte: (ARAFIN; BILLAH; ISSA, 2023).

Outra função de ativação muito utilizada é a *Softmax*, descrita pela função:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{para } i = 1, 2, \ldots, K$$

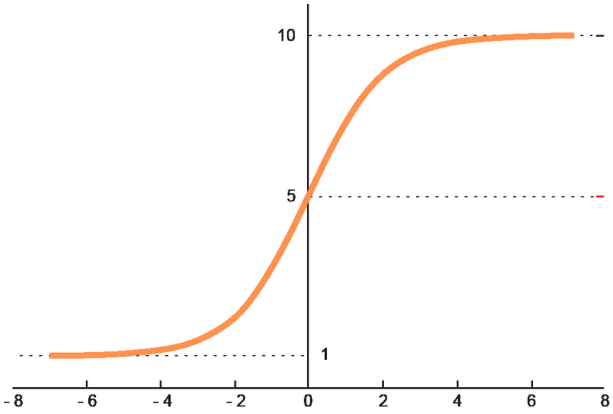
Onde z_i representa o valor da i-ésima saída e K é o número total de classes. A *Softmax* converte a saída de uma CNN em um vetor de probabilidades, onde cada valor representa a probabilidade de a entrada pertencer a uma determinada classe. A classe com a maior probabilidade é então selecionada como a previsão final do modelo.

2.3.1.2 Vision Transformers

Os *Vision Transformers* (ViTs) foram propostos como uma alternativa às redes neurais convolucionais (CNNs) para resolver tarefas de visão computacional, como classificação de imagens, segmentação e detecção de objetos. Ao contrário das CNNs, que utilizam convoluções para capturar informações espaciais em imagens, os ViTs adotam a arquitetura *Transformer*, que foi inicialmente desenvolvida para lidar com tarefas de processamento de linguagem natural (NLP). A arquitetura *Transformer* se destaca por seu mecanismo de auto-atenção que, aplicado ao domínio visual, permite capturar relações globais entre diferentes partes de uma imagem, independentemente de sua distância espacial (DOSOVITSKIY et al., 2021).

A principal ideia por trás dos *Vision Transformers* é tratar uma imagem da mesma maneira que uma sequência de *tokens* no contexto de NLP. Em vez de processar a imagem

Figura 2.9 – Função de ativação *Softmax*



Fonte: (ES-SABERY et al., 2021)

como um todo ou utilizar convoluções, a imagem é dividida em pequenos blocos chamados *patches*. Cada *patch* é então tratado como um *"token"* e processado por um modelo *Transformer*, que aplica o mecanismo de atenção para aprender as dependências entre esses *tokens (patches)*.

A arquitetura geral dos *Vision Transformers* é composta por três componentes principais:

1. **Divisão em Patches:** Ao invés de passar a imagem inteira como entrada, a imagem é dividida em pequenos *patches* (blocos retangulares). Cada *patch* é processado de forma independente e tratado como uma sequência de entrada para o *Transformer*, semelhante aos *tokens* em NLP. O tamanho dos *patches* é definido de acordo com as necessidades da tarefa e do modelo. Normalmente, uma imagem de dimensão 224×224 pode ser dividida em *patches* de 16×16, resultando em uma sequência de 196 *patches*. Cada patch é linearizado e transformado em um vetor de *embeddings* através de uma projeção linear. A fórmula para a projeção dos *patches* pode ser descrita como:

$$z_0^i = x^i \cdot E + e_{\text{pos}}^i$$

   Onde z_0^i é a representação do i-ésimo *patch*, x^i é o vetor de entrada linearizado do i-ésimo *patch*, E é a matriz de *embeddings* aprendida, e e_{pos}^i é o *embedding* posicional que codifica a localização do *patch* na imagem.

   A adição de *embeddings* posicionais é fundamental, pois, diferentemente das CNNs que capturam relações espaciais intrínsecas, os *Transformers* não têm conhecimento embutido da posição dos *patches*. Assim, o *embedding* posicional ajuda a manter a estrutura espacial da imagem.

2. **Camada Transformers:** A partir da sequência de *patches*, a camada *Transformer* é aplicada. Ela é composta por múltiplos blocos de atenção, onde o componente prin-

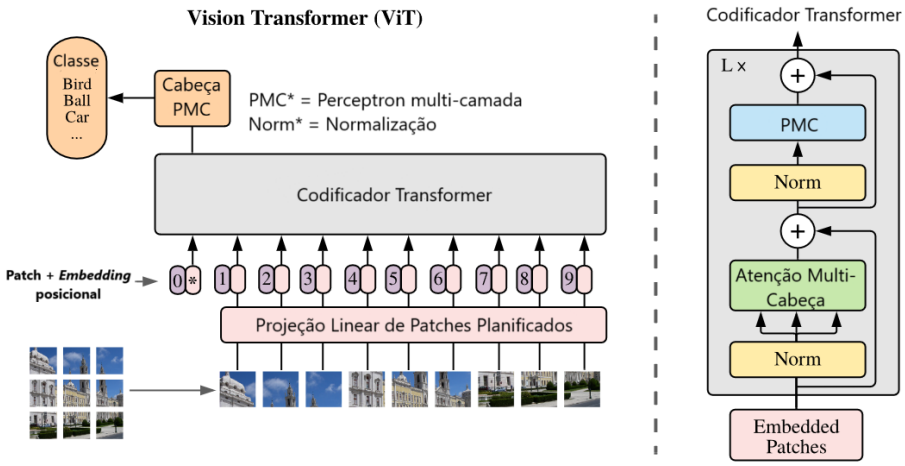
cipal é o mecanismo de auto-atenção multi-cabeça. Esse mecanismo permite que a rede aprenda as relações entre diferentes *patches*, considerando as interações globais da imagem. Ao contrário das convoluções, que são locais, a auto-atenção permite que o modelo entenda como cada parte da imagem (*patch*) se relaciona com todas as outras, independentemente de sua distância espacial (DOSOVITSKIY et al., 2021).

O mecanismo de atenção calcula uma pontuação de atenção para cada par de *patches*, permitindo que a rede "preste mais atenção"a certas regiões da imagem que são mais relevantes para a tarefa em questão. A fórmula da atenção é dada por:

$$\text{Atenção}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Onde Q (query), K (key), e V (value) são projeções lineares dos *embeddings* dos *patches*, e d_k é a dimensionalidade dos *key vectors*. A atenção multi-cabeça aplica esse processo de atenção várias vezes em paralelo, permitindo que a rede foque em diferentes partes da imagem simultaneamente (DOSOVITSKIY et al., 2021).

Figura 2.10 – Arquitetura *Vision Transformer*



Fonte: Adaptado de (DOSOVITSKIY et al., 2021)

3. **Camada de Classificação:** Após várias camadas de atenção e *feed-forward* dentro do bloco *Transformer*, a saída resultante para cada *patch* é um vetor de características refinadas. Para realizar a classificação, a abordagem comum é utilizar um token de classificação (*class token*), que é adicionado no início da sequência de *patches*. Esse *token* acumula informações globais da imagem ao longo das camadas do *Transformer*, e sua saída final é passada por uma camada totalmente conectada ou *Softmax* (PEARCE; BRINTRUP; ZHU, 2021) para gerar as predições de classe.

   A fórmula para a classificação pode ser descrita como:

$$y = \text{softmax}(W \cdot z_{\text{class}})$$

Onde z_{class} é o vetor resultante do *token* de classificação, e W são os pesos aprendidos da camada de saída.

2.4 Desenvolvimento Mobile

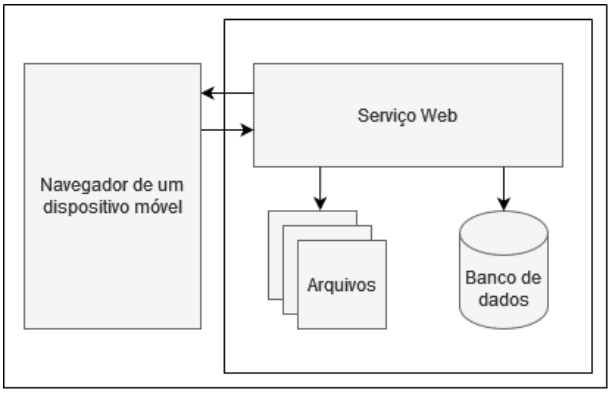
Um dos maiores desafios enfrentados no âmbito do desenvolvimento mobile é como entregar um aplicativo versátil, confiável e eficiente, que é suportado em múltiplas plataformas de uma maneira rápida visto a alta competitividade no mercado de soluções móveis (RAJ; TOLETY, 2012).

Para isso, foi criado uma gama de modos de implementação e *frameworks* que suportam a criação de diferentes tipos de aplicativos, desses modos, podemos destacar três abordagens principais:

2.4.1 Abordagem Web

Na abordagem web, o intuito é a aplicação ser acessada através de um navegador, que através de um servidor web, serve a interface gráfica e os dados necessários para o funcionamento da aplicação, essas características garantem uma portabilidade em múltiplas plataformas como Android, iOS e outros sistemas operacionais.

Figura 2.11 – Aplicação Web.



Fonte: Elaborado pelo autor, adaptado de (RAJ; TOLETY, 2012)

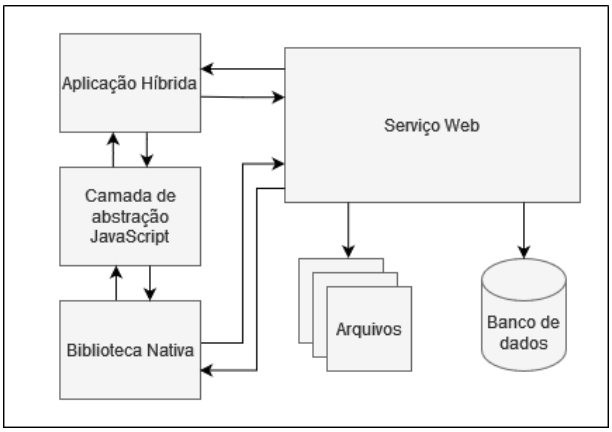
Normalmente, nessa abordagem o processamento e comunicação com o banco de dados é feita remotamente dentro do servidor, e não é necessário instalar a aplicação na memória física do dispositivo, isso também possibilita atualizações na aplicação sem a necessidade de descarregar a nova versão para o aparelho.

Apesar das vantagens, pela baixa performance oferecida por navegadores, a aplicação acaba dependendo de uma conexão constante com a Internet e requer um esforço considerável vinda do servidor, que vai ser responsável pela maior parte do processamento necessário para o funcionamento do app (RAJ; TOLETY, 2012).

2.4.2 Abordagem Híbrida

A abordagem híbrida utiliza o conceito de *Web-View*, que é uma aplicação com camada de abstração JavaScript que possui um navegador embutido no mesmo, utilizando toda a estrutura cliente-servidor da abordagem web, porém mantendo ainda funcionalidades e acesso à bibliotecas nativas do aparelho móvel.

Figura 2.12 – Aplicação Híbrida.



Fonte: Elaborado pelo autor, adaptado de (RAJ; TOLETY, 2012)

Essa abordagem permite que o projeto seja distribuído em lojas de aplicativo e também disponível via navegador de uma maneira mais limitada devido a falta de acesso às bibliotecas nativas em um *browser*.

Porém, aplicações híbridas ainda ficam a desejar no quesito de performance, já que a aplicação é executada com base em um navegador da web (RAJ; TOLETY, 2012).

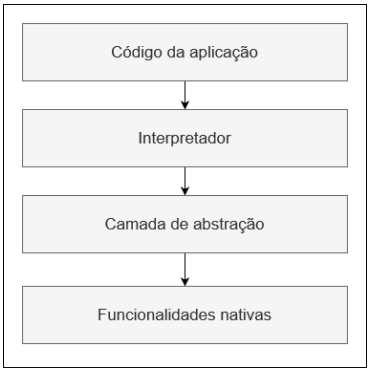
2.4.3 Abordagem Nativa

2.4.3.1 Nativo Interpretado

Uma abordagem interpretada significa que, na arquitetura da aplicação há um interpretador na aplicação que executa em tempo real o código da aplicação e traduz, por meio de uma camada de abstração as interações entre aplicação e funcionalidades nativas.

Devido a natureza de sua estrutura, a operação de tradução pode levar à queda de performance da aplicação, porém, essa abordagem possui um código extremamente portável para outras plataformas.

Figura 2.13 – Nativo Interpretado.

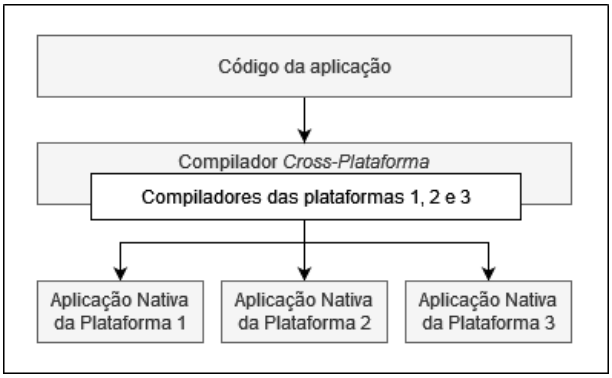


Fonte: Elaborado pelo autor, adaptado de (RAJ; TOLETY, 2012)

2.4.3.2 Nativo Compilado

No caso de uma aplicação nativa e compilada, diferente da interpretada, a tradução para diferentes sistemas ocorre na criação do executável da aplicação, onde o mesmo código passa por diferentes compiladores que resultam em uma aplicação nativa, onde a comunicação entre aplicação e funcionalidades do dispositivo ocorre de maneira direta.

Figura 2.14 – Nativo compilado.



Fonte: Elaborado pelo autor, adaptado de (RAJ; TOLETY, 2012)

É evidente que, diferente das outras abordagens, a característica de execução puramente nativa tem uma grande vantagem no quesito de compatibilidade e velocidade de execução.

2.5 React Native

O *React Native* (NATIVE, 2024), é uma biblioteca de desenvolvimento compilado nativo, desenvolvido pela empresa *Meta* e lançado em 2015, com o objetivo de utilizar a estrutura já criada da biblioteca para a *web ReactJS* e também se aproveitar da abundância de bibliotecas e pacotes disponibilizados pelo *Node Package Manager* (NPM), gerenciador de

pacotes do *NodeJS*, que é um interpretador que possibilita a execução de código *JavaScript* fora de um navegador.

2.5.1 JXL

Utilizando uma linguagem chamada *JXL* que possibilita renderizar dinamicamente a interface gráfica combinando lógica de programação em código *JavaScript* e a definição de elementos com uma linguagem de marcação parecida com *XML*, ela possibilita o uso de classes e funções para definição de componentes.

Exemplo de código em JXL:

```
import React from 'react';
import {Text, View} from 'react-native';

const HelloWorldApp = () => {
  return (
    <View
      style={{
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
      }}>
      <Text>Hello, world!</Text>
    </View>
  );
};
export default HelloWorldApp;
```

2.5.2 Componentes

Aplicativos em *React* são compostos de componentes, um componente é um pedaço de interface que contém sua própria lógica e aparência. Componentes podem ser compostos de outros componentes e podem ter diferentes tamanhos e objetivos, dessa forma, é possível construir uma aplicação com código altamente reutilizável, pois um componente pode ser utilizado em diversos trechos do código, mas contém um único arquivo de código fonte.

2.5.2.1 Componentes principais e personalizados

Há uma série de componentes considerados principais, pois eles são o menor bloco de construção disponível e estão diretamente conectados com os recursos nativos na compilação,

com isso, se tornam fundamentais para toda a estrutura da aplicação, você pode encontrar uma lista completa desses componentes no site oficial do React (NATIVE, 2024).

Componentes personalizados utilizam componentes principais e/ou outros componentes personalizados para criar novas partes de interface, além de conterem sua própria lógica programática, esses componentes formam a base lógica da aplicação, definindo comportamentos e funcionalidades do sistema a ser desenvolvido.

2.6 TensorFlow

TensorFlow (TENSORFLOW.JS, 2024) é um *framework* de código aberto focado em aprendizado de máquina, lançado em 2015 pela empresa *Google*, visando facilitar a criação, treinamento e exportação de redes neurais em múltiplas plataformas baseado na tecnologia proprietária de *Tensor Processing Unit* ou *Neural Processing Unit*, um *hardware* acelerador de execução de modelos de inteligência artificial.

Atualmente o *TensorFlow* possui um ecossistema de desenvolvimento e análise, provendo não só o *framework* básico escrito em Python, mas também ferramentas de análise e monitoramento, além diferentes modelos e casos de estudo.

Dentro desse ecossistema, foi criado uma biblioteca em *JavaScript*, o *TensorFlowJS*, que exporta as funcionalidades da biblioteca principal para o domínio da *web*, isso possibilita a interação entre *TensorFlow* e bibliotecas de desenvolvimento JS, como o *ReactJS* e *React Native*. A biblioteca também é portada para outras linguagens como C++, C#, Julia, R e Scala.

2.7 Trabalhos Relacionados

O avanço tecnológico em relação a redes neurais profundas impulsionou o desenvolvimento de diversas soluções para a classificação de imagens em dispositivos móveis. Esta seção tem o objetivo de revisar trabalhos relevantes na área, destacando os principais modelos e suas vantagens, desvantagens além de evidenciar o estado da arte atual.

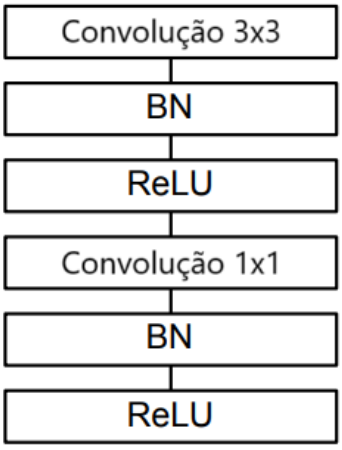
2.7.1 Principais modelos para dispositivos móveis

MobileNet: O *MobileNet* é uma das arquiteturas mais populares desenvolvidas especificamente para dispositivos móveis e sistemas embarcados. Proposto por Howard et al. (2017), o *MobileNet* utiliza convoluções separáveis em profundidade para reduzir o número de parâmetros e computações, mantendo um bom desempenho em termos de precisão.

Convoluções separáveis em profundidade:

MobileNet

Figura 2.15 – Estrutura da *MobileNet*.



Fonte: Adaptado de (HOWARD et al., 2017)

As *Depthwise Separable Convolutions* são uma variação das convoluções convencionais usadas para reduzir a complexidade computacional das redes neurais convolucionais, mantendo boa precisão. A ideia central é dividir a operação de convolução em duas etapas: *Depthwise Convolution* e *Pointwise Convolution*. (CHOLLET, 2017).

- *Depthwise Convolution*: Neste estágio, aplica-se um único filtro 2D a cada canal de entrada de forma independente. Se a imagem de entrada tiver 3 canais (como as imagens RGB), a *Depthwise Convolution* aplicará três filtros diferentes, um para cada canal, gerando uma saída com a mesma profundidade da entrada.
- *Pointwise Convolution*: Uma convolução 1x1 é aplicada para combinar os resultados dos canais anteriores. Ela atua sobre cada pixel da imagem e combina os canais de entrada, gerando uma nova saída com uma profundidade específica com o objetivo de introduzir uma interação entre os canais.

Normalização em lote (Batch Normalization):

A *Batch Normalization* (BN) normaliza as saídas de uma camada para que tenham uma distribuição com média 0 e desvio padrão 1. Posteriormente, dois parâmetros são aprendidos: um fator de escala e um deslocamento, que permitem que a rede recupere a flexibilidade perdida com a normalização, se necessário.

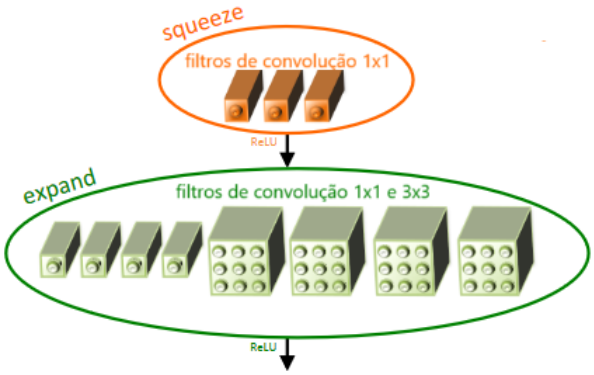
- Vantagens: Alta eficiência computacional e baixo consumo de energia além de ter um menor tamanho na memória física, facilitando a implementação em dispositivos com recursos limitados.

- Desvantagens: Apesar de ser eficiente, observa-se uma queda na precisão em comparação com modelos maiores e mais complexos.

SqueezeNet: Proposto por Iandola et al. (2016), o *SqueezeNet* foca em reduzir o número de parâmetros através do uso de um tipo de camada chamada *"fire"*, que combina convoluções de tamanho 1x1 e 3x3.

Camada fire:

Figura 2.16 – Estrutura da camada *fire*



Fonte: Adaptado de (IANDOLA et al., 2016)

Squeeze: Aplica filtros de convolução de tamanho 1px x 1px.

Expand: Aplica filtros de convolução 1px x 1px e 3px x 3px.

A estrutura da *SqueezeNet* é composta por uma camada de convolução seguida por 8 camadas *fire* e uma camada de convolução final, aplicando então o *Average Pooling* e uma camada com função de ativação *Softmax*.

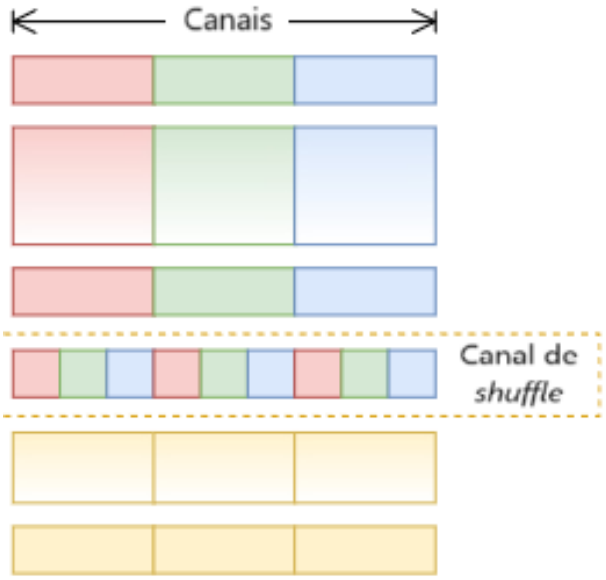
- Vantagens: Modelo extremamente leve, com uma alta taxa de compressão. Boa precisão relativa considerando seu tamanho.
- Desvantagens: A leveza leva a uma menor precisão em comparação com modelos maiores, embora seja adequado para aplicações que priorizam a eficiência.

ShuffleNet: Proposto por Zhang et al. (2017), o *ShuffleNet* utiliza convoluções agrupadas (*channels)* e uma operação de *shuffle* para melhorar a eficiência computacional.

Convoluções agrupadas: Divide os canais de entrada em vários grupos (*channels*) e realiza a convolução separadamente em cada grupo. Os canais dentro de um grupo interagem apenas entre si, e não com os canais de outros grupos.

Camada de embaralhamento (Channel Shuffle): A ideia principal é "embaralhar"os canais após as convoluções agrupadas, reorganizando os canais de maneira que eles possam se redistribuir em novos grupos na próxima camada de convolução.

Figura 2.17 – *Operação de shuffle.*



Fonte: Adaptado de (ZHANG et al., 2017)

- Vantagens: Alta eficiência computacional apresentando um bom equilíbrio entre precisão e complexidade.
- Desvantagens: A implementação complexa devido à operação de *shuffle*.

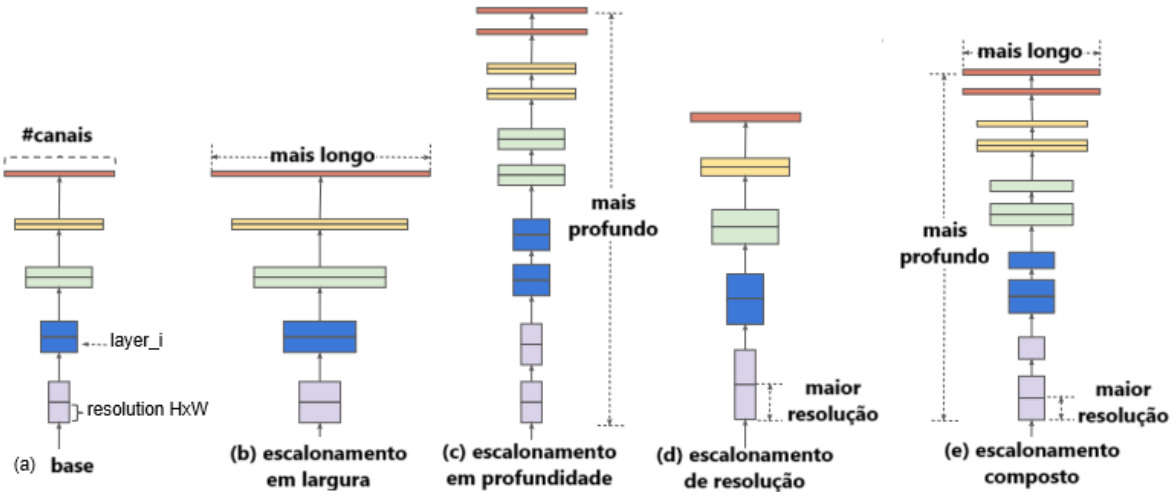
EfficientNet: Proposto por Tan e Le (2020), o *EfficientNet* utiliza um método de escalonamento composto que tenta ajustar propriedades da rede como a profundidade, largura e resolução de maneira a otimizar a eficiência do modelo.

As três dimensões são escaladas da seguinte maneira:

- Profundidade: Refere-se ao número de camadas ou a quantidade de convoluções sucessivas na rede. Aumentar a profundidade permite que a rede capture características mais complexas, aprendendo mais detalhes.
- Largura: Refere-se ao número de canais em cada camada. Redes mais largas podem capturar mais características em cada estágio.
- Resolução de Entrada: Refere-se ao tamanho das imagens de entrada. Aumentar a resolução permite que a rede capture mais detalhes espaciais nas imagens.

- Vantagens: Excelente precisão e eficiência computacional. Flexível e escalável para diferentes exigências de recursos.
- Desvantagens: É mais complexo de treinar e implementar em comparação com modelos mais simples.

Figura 2.18 – Escalonamento composto.



Fonte: (TAN; LE, 2020).

Inception: A arquitetura *Inception* (SZEGEDY et al., 2014), é conhecida por seu uso eficiente de convoluções em vários tamanhos dentro da mesma camada, permitindo capturar informações em diferentes escalas.

O principal componente dessa arquitetura é o *Inception Module*, que foi projetado para permitir que a rede explore diferentes tamanhos de filtros e resoluções simultaneamente. O *Inception Module* contém convoluções de diferentes tamanhos de filtro (1 x 1, 3 x 3 e 5 x 5) e uma operação de *pooling*. A combinação dessas operações permite que a rede capture informações locais (com filtros menores) e informações globais (com filtros maiores), além de realizar a compressão dos dados via *pooling*.

- Vantagens: Flexível na captura de características em diferentes escalas. Bom desempenho em tarefas de classificação de imagens, frequentemente alcançando altas precisões em *benchmarks*.
- Desvantagens: Estrutura relativamente complexa, o que dificulta a implementação em dispositivos com recursos limitados. Consumo de memória e tempo de inferência podem ser mais elevados em comparação com arquiteturas mais leves.

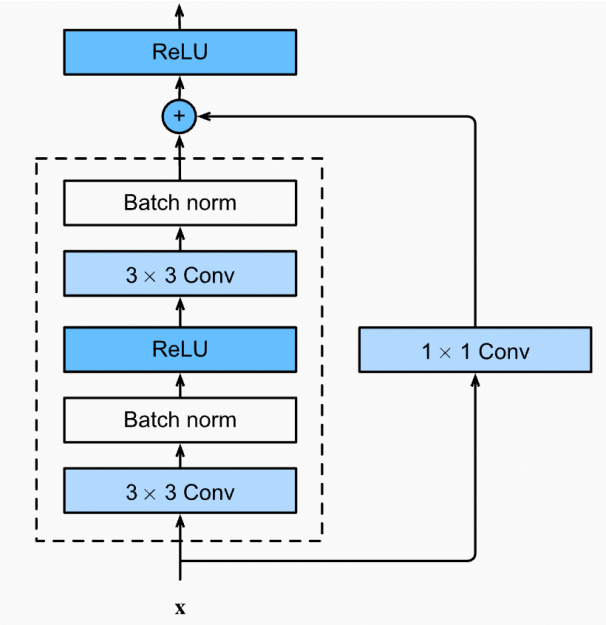
ResNet:

A *Residual Network (ResNet)* (HE et al., 2015) é conhecida por introduzir conexões de atalho (*skip connections*) que permitem a construção de redes extremamente profundas sem sofrer de problemas de degradação.

Skip connections: Essas conexões pulam uma ou mais camadas da rede e adicionam a entrada de uma camada diretamente à saída de uma camada posterior. (ZHANG et al.,

2021)

Figura 2.19 – *Skip connection.*



Fonte: Retirada de (ZHANG et al., 2021)

- Vantagens: Excelente desempenho em profundidade, permitindo a construção de modelos muito profundos sem perder precisão. Facilita o treinamento de redes profundas, mitigando problemas de gradientes desaparecendo.
- Desvantagens: Complexidade e tamanho da rede podem ser desafiadores para dispositivos móveis. Latência e uso de memória tendem a ser mais altos em comparação com redes mais leves.

2.7.2 Estado da Arte

O campo de classificação de imagens em dispositivos móveis tem registrado avanços notáveis, tanto no desenvolvimento de *hardware* especializado quanto na otimização de algoritmos de redes neurais. Esses progressos permitem a execução eficiente de redes neurais profundas em dispositivos com restrições de recursos, como *smartphones* e *tablets*. A seguir, são discutidos alguns dos principais pontos da evolução do estado da arte no tema.

Modelos Otimizados: O desenvolvimento de redes neurais profundas otimizadas para dispositivos móveis é um dos principais marcos dessa evolução. Redes como o *MobileNetV3* e o *EfficientNet-Lite* foram projetadas para maximizar a eficiência computacional, mantendo a precisão elevada. O *MobileNetV3* aprimorou seus predecessores através do uso de convoluções separáveis em profundidade e módulos de *Squeeze-and-Excitation* (SE), otimizando o *trade-off* entre precisão e consumo de recursos em aplicações móveis (HOWARD

et al., 2019). Já o *EfficientNet-Lite*, baseado no método de *Compound Scaling*, ajusta a profundidade, a largura e a resolução da rede de forma balanceada, resultando em modelos extremamente eficientes em termos de desempenho e consumo de energia (TAN; LE, 2019).

Aceleração de Hardware: O avanço no hardware de dispositivos móveis tem sido essencial para suportar modelos de *deep learning* mais complexos. O desenvolvimento de unidades de processamento neural (NPUs), como o *Neural Engine* da Apple e o Hexagon DSP da Qualcomm, permitiu a execução de modelos sofisticados, como o EfficientNet-Lite e o MobileNetV3, diretamente em dispositivos móveis, com menor impacto no consumo de bateria e maior velocidade de inferência (LEE et al., 2021). Esses aceleradores de hardware desempenham um papel crucial na viabilização de redes neurais profundas em tempo real em ambientes com recursos limitados.

Técnicas de Compressão de Modelos: Diversas abordagens têm sido desenvolvidas para tornar os modelos de deep learning mais compactos e eficientes. A quantização de modelos é uma das técnicas mais usadas, que reduz a precisão dos pesos e ativações (por exemplo, de 32 bits para 8 bits), diminuindo a necessidade de memória e acelerando a inferência sem comprometer significativamente a precisão (JACOB et al., 2018). Além disso, outras técnicas como pruning (podagem), que remove pesos irrelevantes, e distilação de conhecimento (knowledge distillation), que transfere o conhecimento de um modelo maior para um menor, são amplamente aplicadas para otimizar redes neurais para dispositivos móveis (HINTON; VINYALS; DEAN, 2015).

Ferramentas e Frameworks: A proliferação de ferramentas e frameworks tem facilitado o desenvolvimento e a execução de redes neurais profundas em dispositivos móveis. TensorFlow Lite e PyTorch Mobile são exemplos de frameworks que permitem a implementação de redes otimizadas, oferecendo suporte a quantização e outras técnicas de compressão. Eles também facilitam a execução de modelos em tempo real e com baixo consumo de recursos (ABADI et al., 2016). Esses frameworks são amplamente adotados na indústria devido à sua flexibilidade e capacidade de otimização para diferentes tipos de dispositivos móveis.

Treinamento Local em Dispositivos Móveis: Com o avanço de frameworks como o TensorFlow.js, tornou-se possível realizar tanto inferência quanto treinamento local em dispositivos móveis. Embora o treinamento em dispositivos móveis ainda seja uma área em desenvolvimento, estudos mostram que é viável realizar treinamento com eficiência energética em tarefas que exigem aprendizado contínuo ou personalização (GOROSHIN et al., 2020). Essa capacidade oferece novas possibilidades para a adaptação de modelos a dados específicos do usuário, sem a necessidade de conectividade constante com servidores remotos.

2.7.3 Conclusão

A evolução das redes neurais para classificação de imagens em dispositivos móveis foca em balancear precisão e eficiência computacional. Modelos como o MobileNetV3 e o EfficientNet-Lite representam avanços significativos, otimizando tanto o desempenho quanto o consumo de recursos. O uso crescente de técnicas de compressão de modelos e a adoção de hardware especializado têm permitido que redes mais complexas sejam executadas diretamente em dispositivos móveis com alta eficiência. A combinação dessas inovações tecnológicas e algorítmicas indica um futuro promissor para a classificação de imagens em plataformas móveis, com aplicações cada vez mais robustas e de baixo custo computacional.

3 Metodologia

3.1 Introdução

Neste capítulo, apresentamos a metodologia adotada, as ferramentas e plataformas utilizadas para o desenvolvimento de uma plataforma móvel para classificação de imagens utilizando diferentes modelos de redes neurais. A plataforma será desenvolvida com base em *React Native* para a construção da interface gráfica e *TensorFlowJS* para as operações de aprendizado de máquina, permitindo realizar a transferência de aprendizado em dispositivos móveis.

3.2 Descrição do Problema

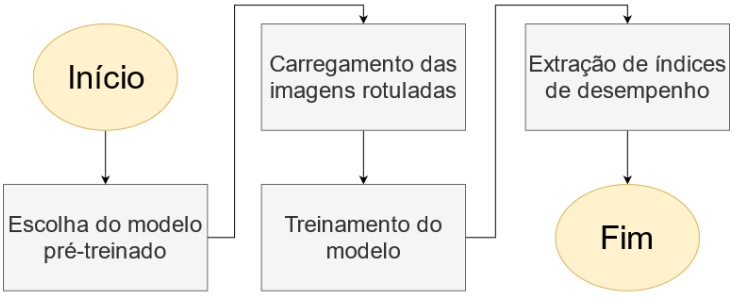
A classificação de imagens em dispositivos móveis possui limitações de recursos computacionais e de energia. Esta monografia visa desenvolver uma plataforma para adaptar modelos pré-treinados à plataforma móvel, realizando uma análise comparativa de precisão e desempenho entre os diferentes modelos.

3.3 Arquitetura da Solução

A arquitetura da solução tem objetivo de permitir a troca de diversos modelos pré-treinados para extração de um vetor características, a utilização de uma base de dados de imagens para o treinamento do modelo e extração de indicadores de desempenho de cada modelo e por fim a comparação será feita de modo manual ao extrair e catalogar os diferentes resultados obtidos.

Para isto, a plataforma segue o fluxo de execução da Figura 3.1.

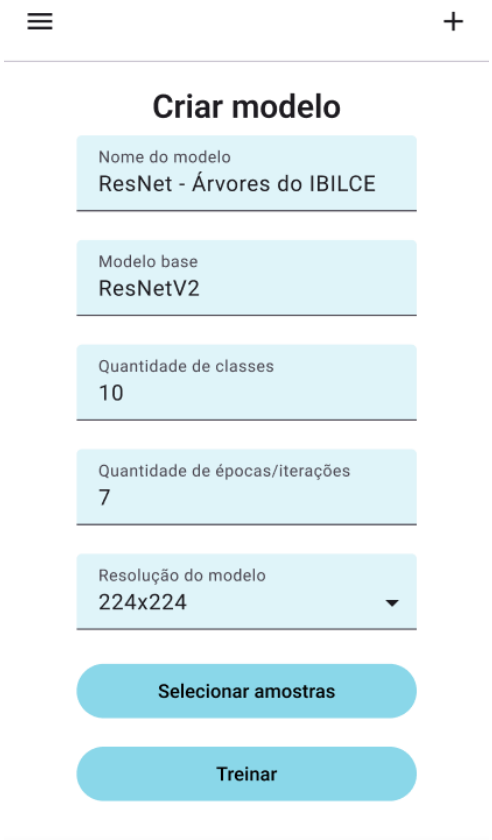
Figura 3.1 – Fluxo de utilização da plataforma.



Fonte: Elaborado pelo autor.

Escolha do modelo pré-treinado: A escolha do modelo é feita pela interface gráfica, apenas os três abordados no projeto estarão disponíveis, pois o tipo de modelo precisa ser especifico para o *TensorFlowJS* em modo de extração de características (*Feature Vector Extractor*) e atualmente o *website* oficial disponibilizado pelo *TensorFlow* Kaggle (2024) disponibiliza apenas algumas opções que cumprem estes requisitos. O design escolhido para a tela pode ser visto na Figura 3.2.

Figura 3.2 – Interface de criação de novos modelos.



Fonte: Elaborado pelo autor.

Carregamento das imagens rotuladas: A escolha das imagens pode ser feita através do carregamento de imagens do dispositivo e adicionando em classes definidas pelo usuário.

Treinamento do modelo: Assim que as imagens forem carregadas e os parâmetros de treinamento definidos, o treinamento ocorre e disponibiliza o novo modelo na tela de menu, como na Figura 3.3.

Extração de índices de desempenho: A partir daí, ao abrir o modelo criado, o usuário pode submeter imagens para a classificação, onde ele pode verificar a confiança do modelo naquela predição e o tempo que levou para realizar a classificação. O design escolhido para a tela pode ser visto na Figura 3.4.

Figura 3.3 – Interface do menu da plataforma.

Fonte: Elaborado pelo autor.

3.4 Modelos

Para este estudo, foram selecionados alguns dos modelos de redes neurais mais utilizados e eficientes para classificação de imagens em dispositivos móveis, todos extraídos do *website* Kaggle (2024).:

3.4.1 MobileNet

O MobileNet é um modelo de rede neural convolucional eficiente, especialmente projetado para dispositivos móveis e embarcados. Utiliza separações de convolução em profundidade para reduzir o número de parâmetros e operações, mantendo uma precisão razoável. Este modelo é altamente modular, permitindo ajustes no número de operações e no tamanho do modelo para equilibrar entre a precisão e a latência, conforme as restrições de hardware.

3.4.2 InceptionV3

O InceptionV3, parte da família Inception de modelos, é conhecido por sua eficiência e desempenho de alta precisão. Ele utiliza uma arquitetura de rede convolucional profunda

Figura 3.4 – Interface de classificação do aplicativo.

Fonte: Elaborado pelo autor.

com camadas Inception, que combinam várias convoluções de diferentes tamanhos em uma única camada, permitindo uma extração de características mais rica e detalhada.

3.4.3 ResNet

O ResNet é um modelo de rede neural profunda que utiliza blocos residuais para permitir o treinamento de redes muito profundas. Esta arquitetura ajuda a mitigar o problema da dissipação do gradiente e facilita o treinamento de modelos mais profundos e complexos, resultando em um desempenho robusto para a classificação de imagens.

3.5 Base de Dados de Imagens

3.5.1 CIFAR-10

Para o treinamento e avaliação dos modelos, utilizamos a base de dados *CIFAR-10* (KRIZHEVSKY, 2009), que ao total contém 60.000 imagens coloridas divididas em 10 classes com 6.000 imagens por classe. Cada classe contém 5.000 imagens de treinamento e 1.000 imagens de teste. Esta base de dados é amplamente utilizada na pesquisa de aprendizado de

máquina e oferece um bom equilíbrio entre complexidade e diversidade de dados para avaliação de modelos.

Figura 3.5 – CIFAR-10

3.6 Treinamento

Os treinamentos dos modelos foram realizados utilizando *transfer learning*, onde os modelos pré-treinados utilizando grandes bases de dados como o *ImageNet* (DENG et al., 2009) foram ajustados para a classificação das imagens da base de imagens *CIFAR-10* (KRIZHEVSKY, 2009). Este processo envolve a utilização das primeiras camadas da rede neural, que são responsáveis pela extração do vetor de características e a troca das camadas finais, que são responsáveis pela classificação.

O treinamento foi realizado diretamente no dispositivo móvel utilizando *TensorFlowJS*, permitindo avaliar a eficiência do processamento. O uso do *TensorFlowJS* facilita a integração com a plataforma *React Native* pela linguagem base de ambos ser JavaScript.

O *website* Kaggle (2024) disponibiliza os modelos pré-treinados em modo de extrator de características, com as duas últimas camadas de classificação já removidas, por isso, só precisamos nos preocupar em criar novamente a camada removida.

Parâmetros aplicados a todos os extratores de características (*MobileNet, InceptionV3, ResNet*) para a coleta de resultados:

- Iterações (cada iteração tem um número diferente de amostras de treino): 8.
- Quantidade de amostras de treino por iteração: 10, 20, 30, 50, 80, 130, 210 e 340.
- Número de execuções por iteração: 5.
- Quantidade de amostrar de teste por iteração: 250 para todas as iterações.
- Base de dados: CIFAR-10.
- Penúltima camada de classificação adicionada: 128 neurônios com função de ativação *ReLU*.

- Nova camada de classificação: 10 neurônios (um por classe) com função de ativação *Softmax*.
- Quantidade de épocas para treinamento: 10.
- Tamanho das imagens: 224px x 224px.

3.7 Métricas de Avaliação

Para avaliar o desempenho dos modelos, foram coletadas as seguintes métricas utilizando quantidades diferentes de imagens para treino distribuídas sob as 10 classes da CIFAR-10, e utilizando um total de 250 imagens de teste em todas as execuções:

3.7.1 Consumo de bateria e tempo decorrido

A partir de funções móveis nativas disponibilizadas pelo *React Native*, podemos coletar a porcentagem de bateria utilizada e o tempo decorrido em milissegundos durante os processos de treinamento e classificação.

3.7.2 Acurácia

A acurácia é a proporção de previsões corretas em relação ao total de previsões realizadas. É uma métrica simples e intuitiva para medir o desempenho geral de um modelo de classificação (DEVELOPERS, 2024).

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN}$$

Sendo, TP verdadeiro positivo, TN, verdadeiro negativo, FP falsos positivos e FN falsos negativos.

3.7.3 Recall

Medida da proporção de verdadeiros positivos em relação ao total de elementos que deveriam ter sido classificados como positivos (DEVELOPERS, 2024).

$$\text{Recall} = \frac{TP}{TP + FN}$$

3.7.4 F1-Score

Média harmônica entre precisão e recall, fornecendo uma única métrica que equilibra ambos os aspectos (DEVELOPERS, 2024).

$$\text{F1} = \frac{2 * \text{Precisão} * \text{Recall}}{\text{Precisão} + \text{Recall}}$$

3.7.5 Precisão

Precisão é a proporção de todas as classificações positivas do modelo que são realmente positivas (DEVELOPERS, 2024).

$$\text{Precisão} = \frac{TP}{TP + FP}$$

3.7.6 Heurísticas de Nielsen

As Heurísticas de Usabilidade de Nielsen são um conjunto de princípios usados no design de interfaces e interação humano-computador. Esses princípios servem como guias para identificar e resolver problemas de usabilidade em interfaces. As heurísticas são diretrizes que ajudam a garantir que um produto digital seja eficiente, intuitivo e fácil de usar (NIELSEN, 1995).

1. **Visibilidade do status do sistema:** O sistema deve sempre informar ao usuário o que está acontecendo, com retorno apropriado em tempo razoável.
2. **Compatibilidade entre o sistema e o mundo real:** O sistema deve falar a linguagem do usuário, usando termos, conceitos e convenções familiares ao seu público.
3. **Controle e liberdade do usuário:** Os usuários devem ter a opção de desfazer ou refazer suas ações facilmente.
4. **Consistência e padrões:** O design deve ser consistente em toda a interface.
5. **Prevenção de erros:** A melhor maneira de lidar com erros é preveni-los.
6. **Reconhecimento em vez de memorização:** A interface deve minimizar a carga de memória do usuário. Objetos, ações e opções devem ser visíveis, ou seja, o usuário não deve precisar lembrar de informações de uma parte do diálogo para outra.
7. **Flexibilidade e eficiência de uso:** O design deve atender tanto usuários iniciantes quanto avançados.
8. **Estética e design minimalista:** A interface não deve conter informações irrelevantes ou raramente necessárias.
9. **Ajuda aos usuários para reconhecer, diagnosticar e corrigir erros:** As mensagens de erro devem ser expressas em linguagem simples, indicar claramente o problema e sugerir uma solução construtiva.

10. **Ajuda e documentação:** Mesmo que o sistema seja fácil de usar, é importante fornecer ajuda e documentação acessível.

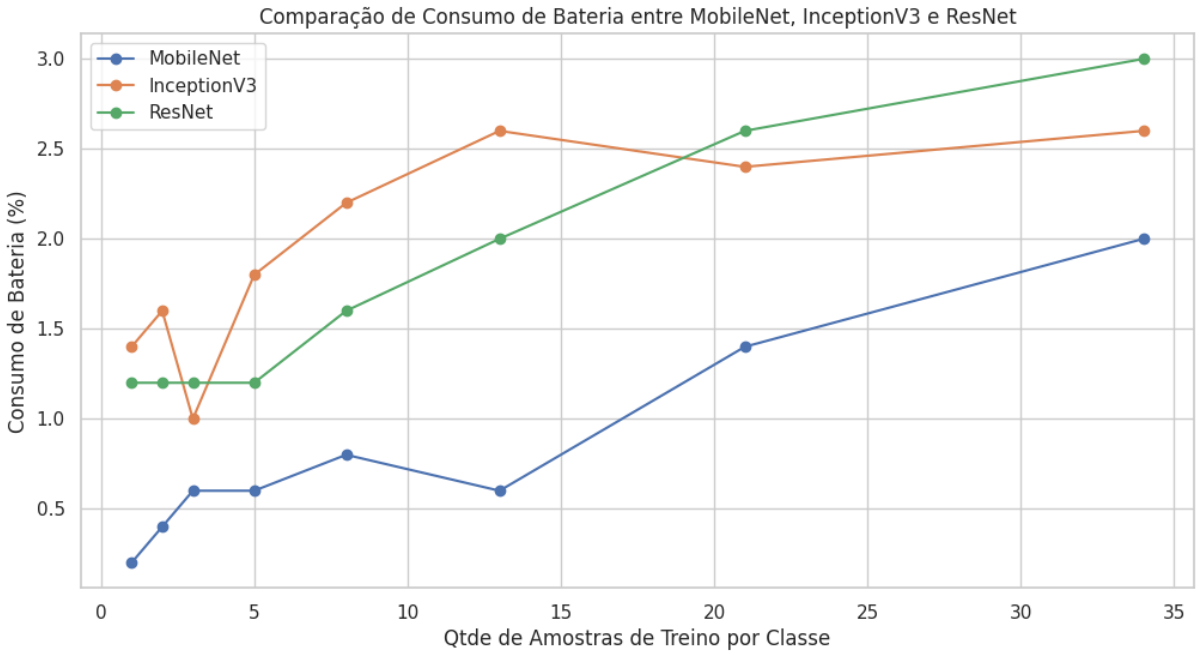
4 Experimentos e resultados

Para a análise dos resultados, vamos considerar alguns aspectos-chave baseados nos experimentos e nas métricas reportadas:

4.1 Consumo de Bateria e Tempo de Treinamento

Além da precisão, o consumo de bateria e o tempo de treinamento são métricas críticas em aplicações móveis. A Figura 4.1 apresenta o consumo de bateria dos modelos, evidenciando o *MobileNet* como o mais eficiente energeticamente. Já a Figura 4.2 ilustra os tempos de treinamento, destacando novamente o *MobileNet* como o mais rápido, enquanto *Inception* e *ResNet* apresentaram maior consumo de recursos.

Figura 4.1 – Consumo de bateria entre modelos, utilizando *Transfer Learning* para CIFAR-10.

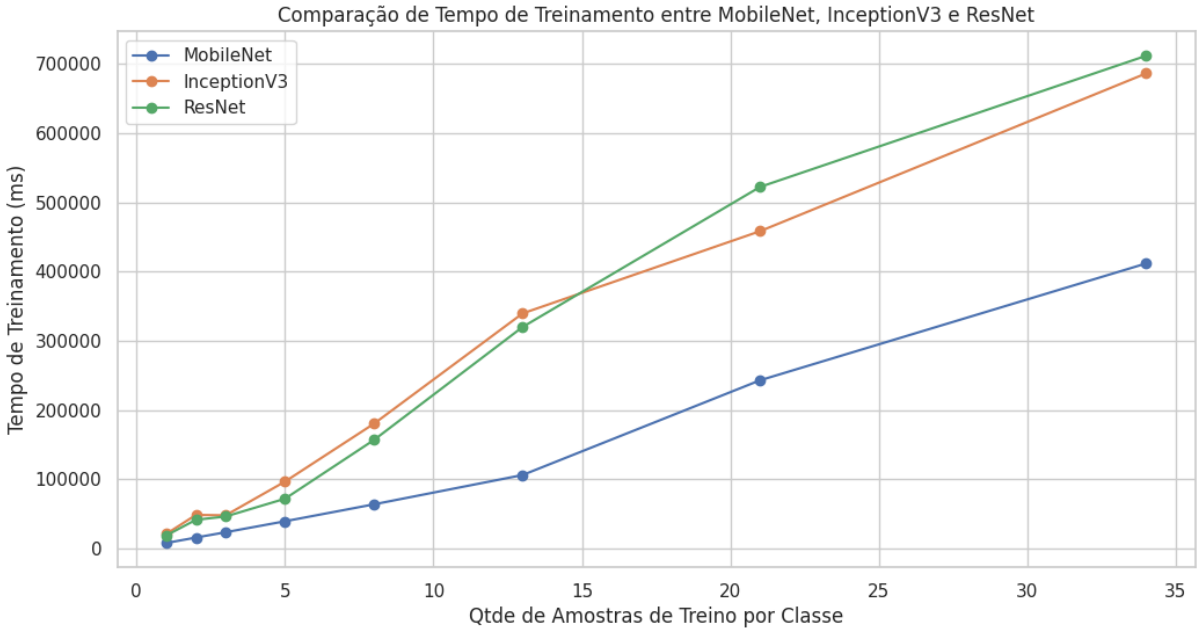


Fonte: Elaborado pelo autor.

4.2 Precisão

Comparando os três modelos implementados, observa-se que a *Inception* obteve consistentemente uma precisão mais elevada em relação aos demais, conforme ilustrado na Figura 4.3. No entanto, o *MobileNet* apresentou uma vantagem significativa no quesito eficiência

Figura 4.2 – Tempo de treinamento entre modelos, utilizando *Transfer Learning* para CIFAR-10.



Fonte: Elaborado pelo autor.

computacional, atingindo valores satisfatórios com menor tempo de treino e menor consumo de energia. A *ResNet*, por sua vez, demonstrou desempenho intermediário em termos de precisão, mas com um custo computacional mais elevado.

4.3 Recall e F1-Score

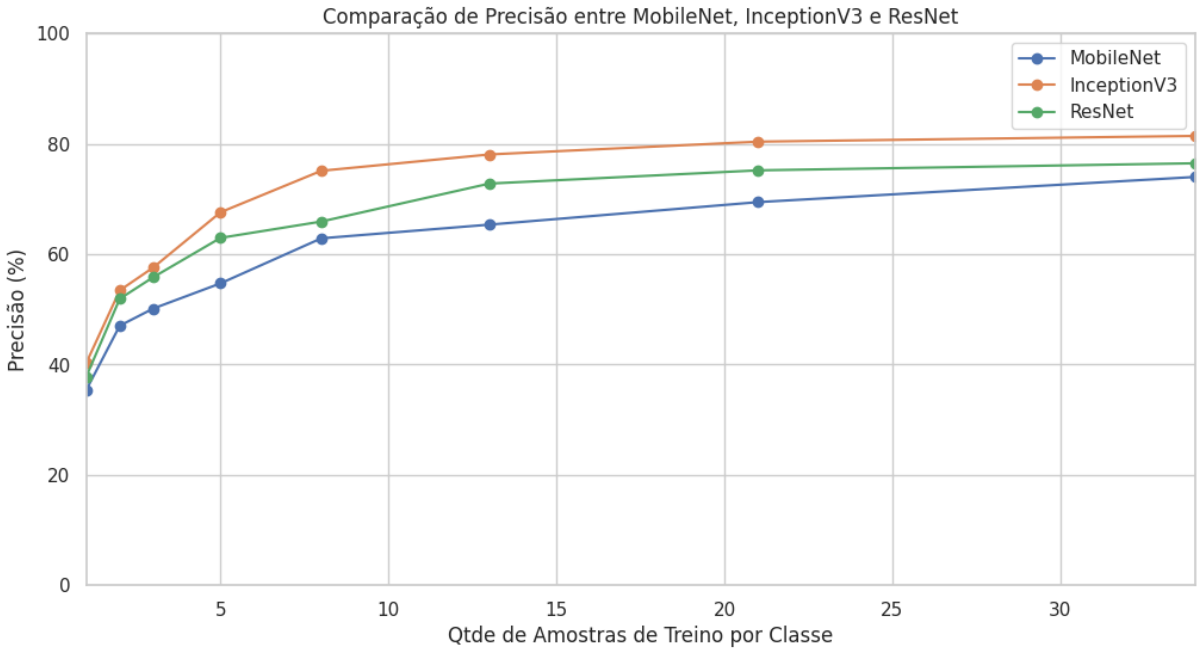
O *recall* mede a capacidade do modelo de identificar corretamente os exemplos positivos de uma classe, sendo uma métrica essencial para aplicações onde falsos negativos têm grande impacto. Como mostrado na Figura 4.4, o *MobileNet* apresentou valores moderados de *recall*, enquanto *Inception* e *ResNet* apresentaram desempenho ligeiramente superior. Contudo, o *recall* isolado não reflete o equilíbrio geral do modelo.

Para compensar os trade-offs entre precisão e *recall*, utilizamos o F1-Score, conforme ilustrado na Figura 4.5. Essa métrica harmônica mostrou que o *MobileNet* oferece um desempenho mais equilibrado, ideal para aplicações móveis que exigem eficiência energética sem comprometer severamente a performance.

4.4 Heurísticas de Nielsen

Visibilidade do status do sistema

Figura 4.3 – Precisão entre modelos utilizando *Transfer Learning* no CIFAR-10.



Fonte: Elaborado pelo autor.

- Meus Modelos (Figura 3.3): A listagem dos modelos treinados é clara e fornece fácil navegação. A visibilidade é garantida através da nomenclatura dos modelos.
- Criar Modelo (Figura 3.2): A interface apresenta de forma clara o progresso, com campos como "Modelo base", "Quantidade de classes", "Épocas", e a resolução do modelo. Contudo, seria interessante incluir uma barra de progresso durante o treinamento para melhor acompanhamento.
- Testar Modelo 3.4: A interface informa claramente o modelo utilizado (ResNet - Árvores do IBILCE), a classe prevista (Ipê - 90% de certeza), e o tempo de classificação (0,97 ms). O *feedback* imediato sobre o resultado é positivo, garantindo que o usuário saiba o estado atual da operação.

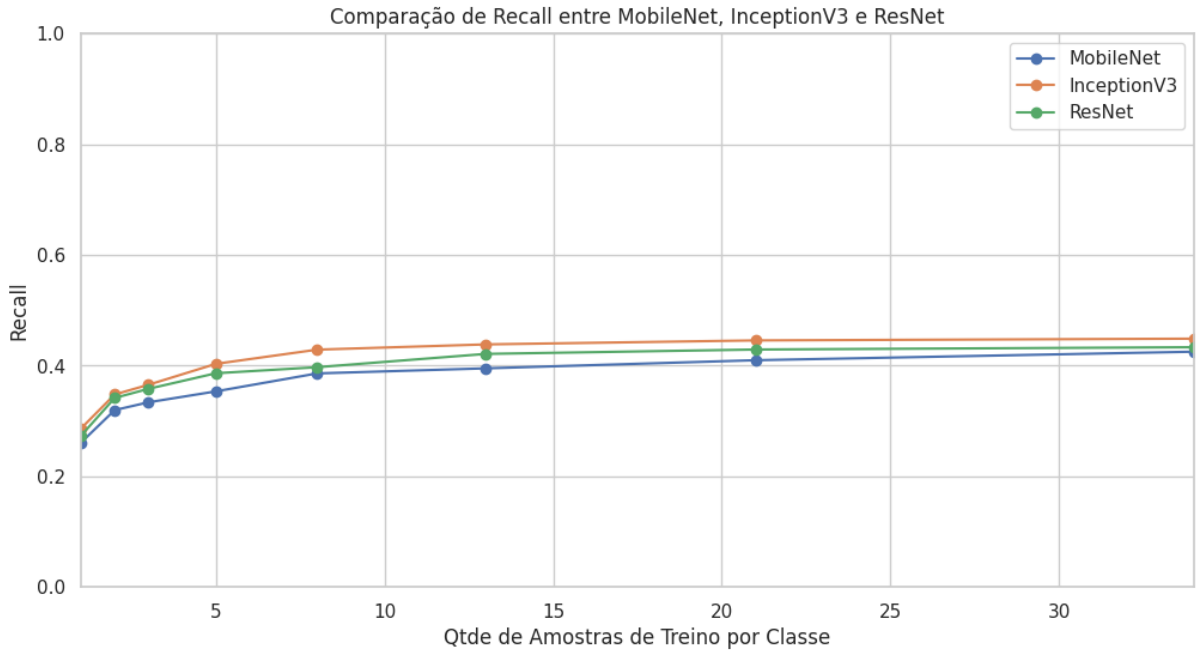
Compatibilidade entre o sistema e o mundo real

As interfaces utilizam termos familiares, como "Classe", "Modelo base"e "Selecionar imagem". Porém, no contexto do usuário geral, o termo "classe"pode não ser imediatamente compreensível. Explicações mais acessíveis para usuários iniciantes, ou incluir descrições explicando o que cada termo significa é algo a ser implementado no futuro.

Controle e liberdade do usuário

- Meus Modelos (Figura 3.3): A lista dos modelos oferece boas opções, mas faltam opções de gerenciamento, como editar, excluir ou visualizar o desempenho detalhado de cada

Figura 4.4 – *Recall* entre modelos, utilizando *Transfer Learning* para CIFAR-10.



Fonte: Elaborado pelo autor.

modelo.

- Criar Modelo (Figura 3.2): O usuário pode facilmente ajustar os parâmetros do modelo antes de treinar, o que é positivo. No entanto, após o treinamento começar, ainda não é possível que o usuário interrompa o processo se necessário.

- Testar Modelo (Figura 3.4): A interface oferece um botão "Selecionar nova imagem", fornecendo ao usuário controle para realizar novas tentativas.

Consistência e padrões

As interfaces são consistentes no design e estilo, com botões padronizados e cores repetitivas, utilizando a biblioteca Google (2014) para manter a consistência.

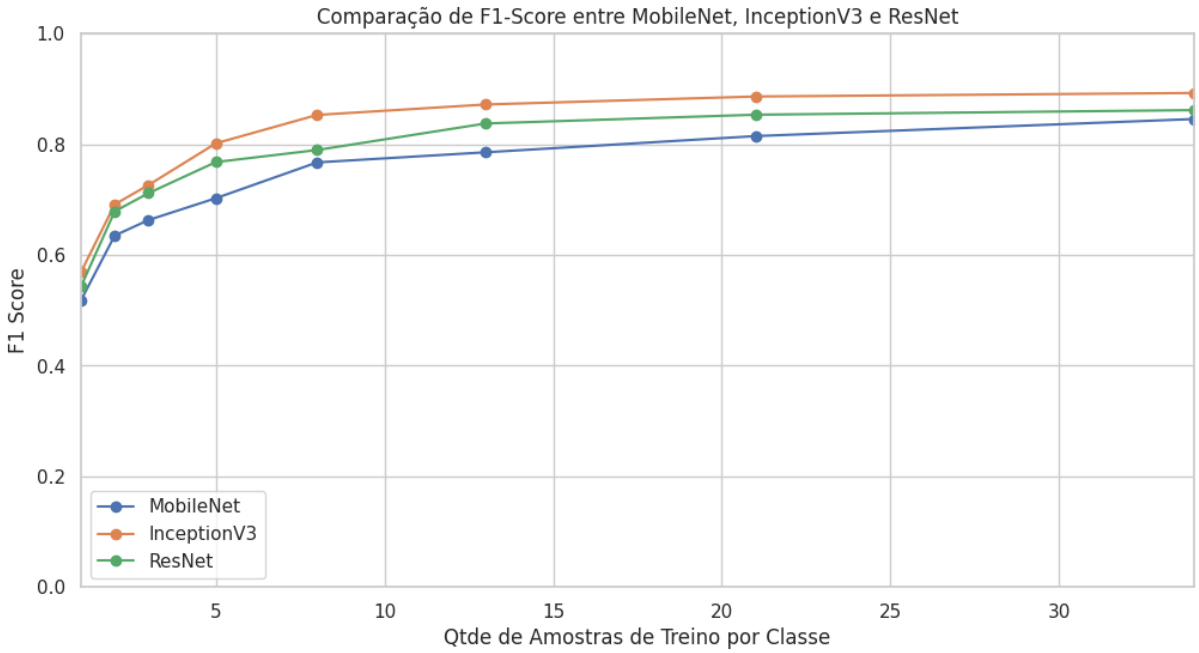
Prevenção de erros

Não há muitos elementos interativos complexos nas interfaces, o que reduz o risco de erros. Porém, campos como a quantidade de classes ou iterações ainda precisam de limites predefinidos para evitar entradas inválidas, ou avisos caso o programa atinja os limites de recurso do dispositivo no treino.

Reconhecimento em vez de memorização

Meus Modelos (Figura 3.3): A interface facilita a navegação entre os modelos, com seus nomes visíveis, o que evita a necessidade de memorização. Testar Modelo (Figura 3.4) e Criar Modelo (Figura 3.2): Não há necessidade de o usuário memorizar muitos detalhes. As

Figura 4.5 – F1-Score entre modelos, utilizando *Transfer Learning* para CIFAR-10.



Fonte: Elaborado pelo autor.

opções são claramente visíveis e há um botão "Selecionar amostras"que facilita o acesso ao conjunto de dados.

Flexibilidade e eficiência de uso

As interfaces parecem atender bem tanto usuários iniciantes quanto avançados. Contudo, ainda faltam maneiras para facilitar a importação/exportação de modelos, que iria aumentar a eficiência para usuários avançados.

Estética e design minimalista

As interfaces mantêm um design minimalista, com o foco apenas nos elementos necessários.

Ajuda aos usuários para reconhecer, diagnosticar e corrigir erros

Não há mensagens de erro customizadas nas interfaces. Apesar de ser importante garantir que ao cometer um erro (como inserir um número incorreto de classes) o sistema informe o usuário de maneira clara e o ajude a corrigi-lo, a plataforma foi criada para ser uma demonstração e não conta com tal robustez.

Ajuda e documentação

As interfaces não fornecem instruções explícitas. Um tutorial ou informações sobre o que cada campo significa (como "Quantidade de classes"ou "Resolução do modelo") é necessária.

4.5 Considerações Finais

Os resultados obtidos reforçam que a escolha do modelo ideal para aplicações móveis depende de um *trade-off* entre precisão e eficiência de recursos. O *MobileNet* se destaca por sua leveza e adequação a dispositivos com recursos limitados, enquanto a *ResNet* e principalmente a rede *Inception* oferece uma melhor performance de classificação, embora ao custo de maior uso de bateria e tempo de processamento. Dependendo da aplicação, cada um desses modelos pode ser mais apropriado conforme as necessidades específicas de balanceamento entre consumo energético, tempo de resposta e precisão.

Sobre a interface desenvolvida, a aplicação foca apenas na funcionalidade principal e ainda não se preocupa com a robustez de tratamento de erros, também espera-se que o usuário tenha um conhecimento mínimo sobre Redes Neurais para identificar o que parâmetros como "épocas", o que é um modelo pré-treinado e o impacto da resolução no modelo final.

5 Conclusão

Esta monografia apresentou um estudo comparativo sobre o uso de redes neurais profundas, especificamente os modelos *MobileNet, InceptionV3* e *ResNet*, para classificação de imagens em dispositivos móveis. O objetivo foi entender e comparar como essas arquiteturas otimizadas para contextos de baixa capacidade computacional, poderiam equilibrar eficiência, consumo de recursos e precisão em aplicações móveis.

O desenvolvimento de uma plataforma baseada em *React Native* e *TensorFlow.js* possibilitou a implementação de um aplicativo demo capaz de realizar classificações em tempo real, demonstrando que é possível adaptar modelos pré-treinados para funcionar eficientemente em dispositivos móveis. Esse estudo mostrou que, embora modelos leves como o *MobileNet* ofereçam uma excelente relação entre consumo de bateria e precisão, modelos mais robustos como o *InceptionV3* podem ser mais adequados para situações em que a alta precisão é necessária e os recursos do dispositivo permitem tal complexidade.

A técnica de *Transfer Learning* se mostrou muito eficiente para reduzir significativamente o tempo de treinamento e aumentar a precisão, obtendo resultados consideravelmente bons em um curto período de tempo.

Porém, o desenvolvimento móvel se demonstrou um desafio considerável por conta dos limites de *hardware* e rotinas de desempenho de *software*, alguns dos desafios enfrentados no desenvolvimento envolvem limitações para o desenvolvimento como o número máximo de arquivos abertos simultaneamente que agia como um gargalo para o fluxo de imagens providas pelo banco de dados, o sistema operacional fechando o aplicativo ao atingir o máximo de uso de *GPU* e a falta de compatibilidade de muitas funções da biblioteca *TensorFlow.js* com o *React Native*.

Os resultados demonstraram que a escolha do modelo ideal para classificação de imagens em dispositivos móveis depende do contexto de aplicação. *MobileNet* foi mais eficaz em relação ao consumo de bateria e velocidade de treinamento segundo os gráficos 4.1 e 4.2, enquanto os gráficos 4.3 e 4.5 apresentaram os modelos *InceptionV3* e *ResNet* como uma melhor solução em cenários onde a precisão e robustez do modelo eram mais importantes do que a eficiência energética e tempo de processamento.

Por fim, esta pesquisa evidenciou a viabilidade de executar redes neurais profundas em dispositivos móveis, contribuindo para o avanço da inteligência artificial nesse contexto e abrindo caminho para futuras pesquisas e inovações na área de visão computacional em plataformas móveis.

Referências

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

APPLE. *CoreML*. 2024. Disponível em: https://developer.apple.com/machine-learning/core-ml/.

ARAFIN, P.; BILLAH, M.; ISSA, A. Deep learning-based concrete defects classification and detection using semantic segmentation. *Structural Health Monitoring*, v. 23, 05 2023.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. New York: Springer, 2006. ISBN 9780387310732.

CHOLLET, F. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017. Disponível em: https://arxiv.org/abs/1610.02357.

DAVIS, C. P. P. K.; KULICK, J. *Evaluating the Effectiveness of Artificial Intelligence Systems in Intelligence Analysis*. Santa Monica, CA, 2005. Acesso em 23 de novembro de 2024. Disponível em: https://www.rand.org/pubs/technical$_reports/TR129.html$.

DECO, G.; OBRADOVIC, D. (Ed.). *Unsupervised Learning: Foundations of Neural Computation*. Cambridge, MA: MIT Press, 1999. ISBN 9780262041707.

DENG, J. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. [S.l.: s.n.], 2009.

DEVELOPERS, G. *Accuracy, Precision, and Recall*. 2024. Acessado em: 21 de novembro de 2024. Disponível em: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall?hl=pt-br.

DOSOVITSKIY, A. et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2021.

ES-SABERY, F. et al. Sentence-level classification using parallel fuzzy deep learning classifier. *IEEE Access*, PP, p. 1–1, 01 2021.

FACELI, K. et al. *Inteligência artificial: uma abordagem de aprendizado de máquina*. [S.l.]: LTC, 2011.

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. ed. Sebastopol, CA: O'Reilly Media, 2019. ISBN 9781492032649.

GOOGLE. *Material Design*. 2014. Accessed: 2024-10-13. Disponível em: https://material.io/design.

GOROSHIN, R. et al. Training deep networks on mobile hardware. In: . [S.l.: s.n.], 2020.

GSMA. *The Mobile Economy 2024*. 2024. Disponível em: https://www.gsma.com/solutions-and-impact/connectivity-for-good/mobile-economy/wp-content/uploads/2024/02/260224-The-Mobile-Economy-2024.pdf.

HE, K. et al. *Deep Residual Learning for Image Recognition*. 2015. Disponível em: https://arxiv.org/abs/1512.03385.

HINTON, G.; VINYALS, O.; DEAN, J. Distilling the knowledge in a neural network. In: . [S.l.: s.n.], 2015.

HOU, Y. et al. *Transfer Learning for Improving Singing-voice Detection in Polyphonic Instrumental Music*. 2020.

HOWARD, A. et al. Searching for mobilenetv3. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2019.

HOWARD, A. G. et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. Disponível em: https://arxiv.org/abs/1704.04861.

IANDOLA, F. N. et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size*. 2016. Disponível em: https://arxiv.org/abs/1602.07360.

JACOB, B. et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2018.

KAGGLE. 2024. Accessed: 2024-10-05. Disponível em: https://www.kaggle.com.

KAY, S. M. *Fundamentals of statistical signal processing: estimation theory*. USA: Prentice-Hall, Inc., 1993. ISBN 0133457117.

KRIZHEVSKY, A. *Learning Multiple Layers of Features from Tiny Images*. [S.l.], 2009. Disponível em: https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf.

LEE, H. et al. Device-edge co-inference with split deep neural networks for latency and energy reduction. *IEEE Internet of Things Journal*, 2021.

LIU, J. et al. Performance analysis and characterization of training deep learning models on mobile device. In: *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. [S.l.: s.n.], 2019. p. 506–515.

MICROSOFT. *Baixe o teclado inteligente SwiftKey e faça muito mais*. 2024. Disponível em: https://www.microsoft.com/pt-br/swiftkey.

NATIVE, R. *React Native Official Website*. 2024. Accessed: 2024-07-03. Disponível em: https://reactnative.dev/.

NIELSEN, J. *10 Usability Heuristics for User Interface Design*. 1995. Accessed: 2024-10-13. Disponível em: https://www.nngroup.com/articles/ten-usability-heuristics/.

NIELSEN, M. *Neural Networks and Deep Learning - Chapter 1*. 2015. Disponível em: http://neuralnetworksanddeeplearning.com/chap1.html.

ONNX. *Open Neural Network Exchange*. 2017. Disponível em: https://github.com/onnx/onnx.

PCMAG. Facebook adds facial recognition to photo tagging. 2010. Disponível em: https://www.pcmag.com/archive/facebook-adds-facial-recognition-to-photo-tagging-258166.

PEARCE, T.; BRINTRUP, A.; ZHU, J. *Understanding Softmax Confidence and Uncertainty*. 2021. Disponível em: https://arxiv.org/abs/2106.04972.

PYTORCH. *About PyTorch Foundation*. 2024. Disponível em: https://pytorch.org/foundation.

RAJ, C. R.; TOLETY, S. B. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: *2012 Annual IEEE India Conference (INDICON)*. [S.l.: s.n.], 2012. p. 625–629.

RUSSELL, S. J.; NORVIG, P. *Inteligência Artificial: Uma Abordagem Moderna*. [S.l.]: Pearson Brasil, 2021.

SHAO, C.-Y. et al. Hey siri: An on-device dnn-powered voice assistant. *arXiv preprint arXiv:1902.02156*, 2019.

SPIEGEL, M. R.; SCHILLER, J. J.; SRINIVASAN, R. A. *Probabilidade e Estatística*. São Paulo: McGraw-Hill, 2004. (Coleção Schaum). ISBN 9788586804991.

SZEGEDY, C. et al. *Going Deeper with Convolutions*. 2014. Disponível em: https://arxiv.org/abs/1409.4842.

TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. *Proceedings of the 36th International Conference on Machine Learning*, 2019.

TAN, M.; LE, Q. V. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. Disponível em: https://arxiv.org/abs/1905.11946.

TENSORFLOW. *TensorFlow*. 2024. Disponível em: https://www.tensorflow.org/.

TENSORFLOW.JS. *TensorFlow.js: Machine Learning for JavaScript*. 2024. Accessed: 2024-07-03. Disponível em: https://www.tensorflow.org/js.

TIKTOK. *TikTok: What's Next - 2024 Trend Report*. 2024. Disponível em: https://newsroom.tiktok.com/en-us/tiktok-whats-next-2024-trend-report.

ZHANG, A. et al. *ResNet (Residual Networks)*. Dive into Deep Learning, 2021. Accessed: 2024-10-05. Disponível em: https://d2l.ai/chapter$_{c}onvolutional - modern/resnet.html$.

ZHANG, X. et al. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. 2017. Disponível em: https://arxiv.org/abs/1707.01083.