Capítulo 4

Deep Learning na Prática: Classificando Imagens Faciais com Redes Neurais Convolucionais

Caio Pereira, Rodrigo Borges, José Rodrigues, Rodrigo Veras e Kelson Aires

Abstract

The advancement of Deep Learning techniques has transformed the way we analyze images, especially in the healthcare field, where computer vision holds great potential for assisting in diagnostics. This short course presents the fundamentals of image classification with Deep Learning, demonstrated in practice through an application involving facial images in the health domain. Participants will learn how to plan, conduct, and evaluate the model development process, from data preprocessing and augmentation to result analysis, with a focus on methodological rigor and reproducibility. Techniques involving pre-trained networks, support tools, and best evaluation practices will be discussed, preparing students and professionals for solid projects in Computer Vision.

Resumo

O avanço das técnicas de Deep Learning tem transformado o modo como analisamos imagens, especialmente na área da saúde, onde a visão computacional tem potencial para auxiliar em diagnósticos. Este minicurso apresenta os fundamentos de classificação de imagens com Deep Learning, mostrados na prática com uma aplicação em imagens faciais relacionadas à área da Saúde. Os participantes aprenderão a planejar, conduzir e avaliar o processo de desenvolvimento de um modelo, desde o pré-processamento e aumento de dados até a análise de resultados, com foco no rigor metodológico e na reprodutibilidade. Serão discutidas técnicas de uso de redes pré-treinadas, ferramentas de apoio e boas práticas de avaliação, preparando estudantes e profissionais para projetos sólidos em Visão Computacional.

4.1. Introdução

Nos últimos anos, a área de Visão Computacional tem sido significativamente influenciada pelo avanço das técnicas de *Deep Learning* (Aprendizado Profundo). O uso de

Redes Neurais Convolucionais (CNNs) tem possibilitado progressos importantes em tarefas como identificação de objetos, reconhecimento facial e interpretação de expressões, aproximando o desempenho computacional ao nível humano em diversos contextos [Goodfellow et al. 2016].

A análise de imagens faciais, em particular, emergiu como um dos campos mais promissores para aplicações de *Deep Learning* na área da saúde. As faces humanas contêm inúmeras informações que vão além da identidade: podem revelar emoções, estados cognitivos e, potencialmente, indicadores sutis de condições neurológicas e psiquiátricas [Alves Rodrigues et al. 2023].

As CNNs, diferentemente dos métodos tradicionais que dependiam de extração manual de características, possuem a capacidade de aprender automaticamente as características mais relevantes diretamente das imagens. Essa propriedade é particularmente valiosa quando lidamos com características faciais sutis que podem não ser facilmente percebidas pelo olho humano ou descritas por métodos convencionais [Pereira et al. 2024].

4.1.1. Representação Digital de Imagens

Para compreender como as CNNs processam imagens, é fundamental entender como as imagens são representadas digitalmente. Uma imagem digital pode ser definida como a representação visual de um objeto através de uma função bidimensional f(x,y), onde x e y são coordenadas espaciais e o valor de f em qualquer ponto (x,y) corresponde à intensidade ou nível de cinza naquele ponto. Na figura 4.1, temos um exemplo da representação digital de uma imagem [Gonzalez and Woods 2002].

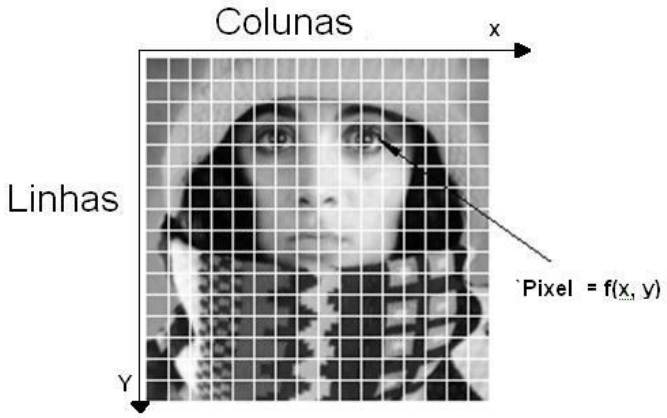

Figura 4.1. Representação digital de uma imagem. Fonte: [Claro et al. 2020]

Na prática, uma imagem digital é representada por uma matriz de elementos discretos chamados *pixels*. Para imagens em escala de cinza, cada *pixel* é representado por um único valor numérico que indica sua intensidade, geralmente variando de 0 (preto) a 255 (branco). Já em imagens coloridas, cada *pixel* é tipicamente representado por três valores correspondentes às intensidades dos canais vermelho (R), verde (G) e azul (B), o chamado modelo RGB. A Figura 4.2 ilustra essa representação matricial de uma imagem digital [Marques Filho and Neto 1999].

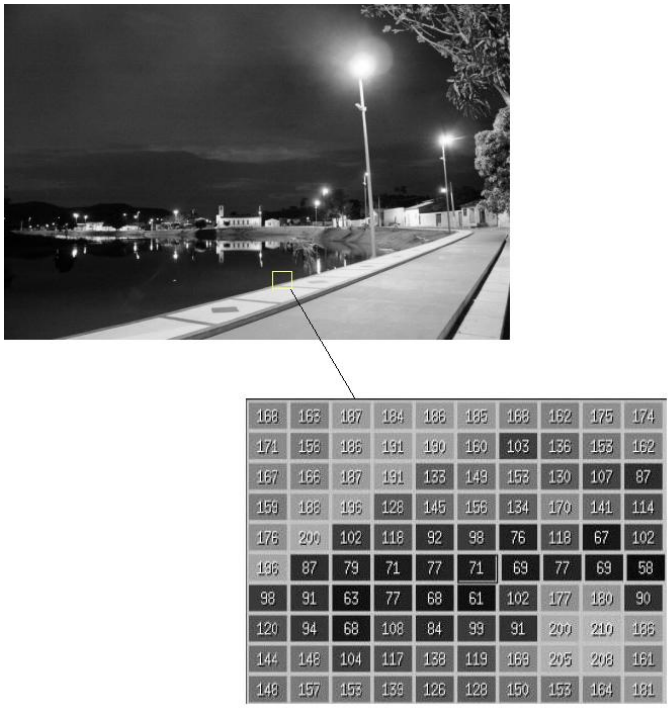

Figura 4.2. Representação matricial de uma imagem digital em escala de cinza com valores de intensidade para cada pixel. Fonte: [Claro et al. 2020]

4.1.2. Aprendizado de Máquina, Redes Neurais Artificiais e *Deep Learning*

O Aprendizado de Máquina (*Machine Learning*) é uma subárea da Inteligência Artificial que desenvolve algoritmos capazes de aprender padrões a partir de dados, sem necessidade de programação explícita. Estes algoritmos utilizam princípios de indução para generalizar a partir de exemplos, criando hipóteses aplicáveis a novos dados [Lorena et al. 2021].

No contexto de classificação de imagens, o Aprendizado Supervisionado é particularmente relevante: o algoritmo aprende a mapear entradas (imagens) para saídas (classes ou rótulos) a partir de um conjunto de dados rotulados. Por exemplo, em um sistema de classificação facial para Transtorno do Espectro Autista (TEA), o modelo seria treinado com imagens faciais rotuladas como 'com TEA' ou 'sem TEA' [Alves Rodrigues et al. 2023].

As Redes Neurais Artificiais (RNAs) são modelos matemáticos inspirados na estrutura e funcionamento do cérebro humano. Uma RNA consiste em unidades de processamento interconectadas (neurônios artificiais) organizadas em camadas. Cada conexão entre neurônios possui um peso associado que é ajustado durante o processo de aprendizagem [Haykin 2007]. A Figura 4.3 ilustra um exemplo de uma rede neural artificial, destacando sua estrutura típica composta por três tipos principais de camadas:

- Camada de entrada: Recebe os dados (no caso de imagens, os valores dos pixels);
- Camadas intermediárias ou ocultas: Realizam transformações não-lineares nos dados;

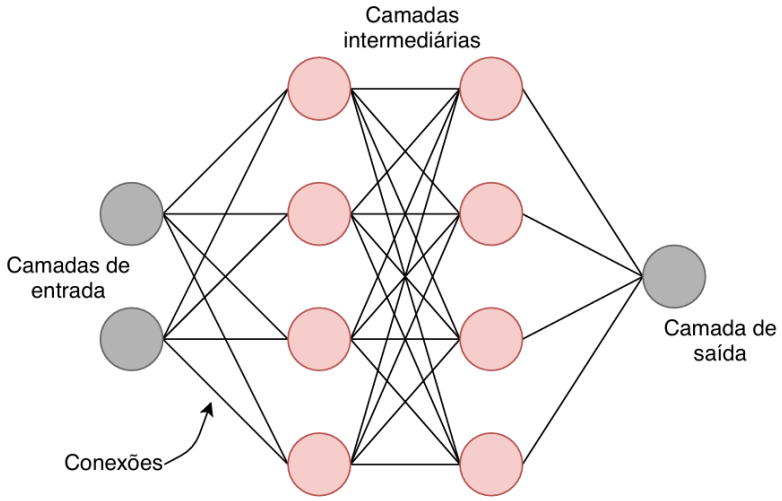

Figura 4.3. Exemplo de uma rede neural artificial. Fonte: [Vogado et al. 2019]

- Camada de saída: Produz a classificação final.

O processo de aprendizagem em RNAs envolve o ajuste gradual dos pesos das conexões para minimizar a diferença entre as previsões da rede e os valores reais. Isso é feito através de algoritmos como o *backpropagation*, que propaga o erro da camada de saída para as camadas anteriores [Rezende 2003].

O *Deep Learning* (Aprendizado Profundo) representa uma evolução das RNAs tradicionais, caracterizado pelo uso de múltiplas camadas intermediárias – daí o termo "profundo"– que permitem a extração automática de características em diferentes níveis de abstração [Goodfellow et al. 2016].

4.1.3. Aplicações em Imagens Faciais

A análise de imagens faciais através de *Deep Learning* tem encontrado aplicações diversas e impactantes em múltiplos campos, com destaque especial para a área da saúde. Esses sistemas têm demonstrado capacidade de detectar padrões sutis que podem estar associados a condições neurológicas, genéticas e psiquiátricas.

No contexto específico do Transtorno do Espectro Autista (TEA), pesquisas recentes têm explorado o potencial das CNNs para identificar biomarcadores faciais que possam auxiliar no diagnóstico precoce. O TEA é um transtorno do neurodesenvolvimento caracterizado por dificuldades na comunicação social e padrões restritos e repetitivos de comportamento, interesses ou atividades [American Psychiatric Association 2013].

O diagnóstico precoce do TEA é crucial para intervenções terapêuticas oportunas, que podem significativamente melhorar o prognóstico. No entanto, o diagnóstico tradicional baseia-se principalmente em observações comportamentais, podendo ser tardio e subjetivo. A identificação de biomarcadores objetivos, como alterações sutis na morfologia facial, representa uma abordagem promissora para complementar os métodos diagnósticos existentes [Alves Rodrigues et al. 2023].

Outras aplicações relevantes de *Deep Learning* em imagens faciais na área da saúde incluem a identificação de síndromes genéticas, por meio da detecção de características faciais associadas a condições como síndrome de Down, Williams e DiGeorge. Também é possível realizar a análise de expressões faciais para avaliar respostas emocionais em transtornos como depressão e esquizofrenia. Além disso, essas técnicas permitem estimar a idade biológica, identificando sinais de envelhecimento precoce relacionados a condições metabólicas, e avaliar a simetria facial, detectando assimetrias sutis que podem ser indicativas de problemas neurológicos.

É importante destacar que sistemas baseados em *Deep Learning* para análise facial em contextos médicos devem ser considerados ferramentas complementares e não substitutivas ao julgamento clínico. Nos próximos capítulos, exploraremos em detalhes a teoria e a prática de desenvolvimento de CNNs para classificação de imagens faciais, com foco em aplicações relacionadas à saúde, particularmente na detecção de características associadas ao TEA.

4.2. Redes Neurais Convolucionais

As Redes Neurais Convolucionais (CNNs) representam uma classe especializada de redes neurais particularmente eficaz para processamento e análise de imagens. Diferentemente das redes neurais tradicionais, as CNNs são projetadas para explorar a estrutura espacial presente em dados como imagens, aproveitando propriedades como localidade espacial e invariância à translação [LeCun et al. 1998].

O conceito de CNN foi inicialmente apresentado por Yann LeCun e Fukushima na década de 90, mas só ganhou ampla popularidade no século XXI com o aumento do poder computacional e a disponibilidade de grandes conjuntos de dados [Fukushima 1988, LeCun et al. 1998]. Hoje, as CNNs constituem o estado da arte em praticamente todas as tarefas de visão computacional, desde classificação e detecção de objetos até segmentação semântica e reconhecimento facial.

A principal vantagem das CNNs sobre outras abordagens é sua capacidade de aprender automaticamente hierarquias de características. Nas primeiras camadas, a rede aprende características básicas como bordas e texturas; em camadas intermediárias, padrões mais complexos como olhos, narizes e bocas; e nas camadas finais, representações abstratas que permitem a classificação. Este aprendizado hierárquico é particularmente valioso para análise facial, onde diferentes níveis de características são relevantes [Goodfellow et al. 2016].

4.2.1. Componentes

As CNNs são compostas por diversos tipos de camadas, cada uma com funções específicas no processo de extração de características e classificação. Vamos explorar os principais componentes:

Camadas Convolucionais: A operação de convolução é o coração das CNNs. Nesta operação, filtros (ou *kernels*) deslizam sobre a imagem de entrada, realizando multiplicações elemento a elemento e somas, produzindo mapas de características (*feature maps*) que destacam padrões específicos na imagem [Goodfellow et al. 2016]. Em aplicações de

análise facial, as camadas convolucionais são cruciais para extrair características morfológicas e texturais. Filtros nas primeiras camadas detectam bordas e texturas fundamentais, enquanto filtros em camadas mais profundas capturam estruturas faciais complexas [Alves Rodrigues et al. 2023]. A Figura 4.4 ilustra um exemplo de operação de convolução, onde um filtro 3×3 é aplicado sobre a imagem de entrada, gerando um mapa de características.

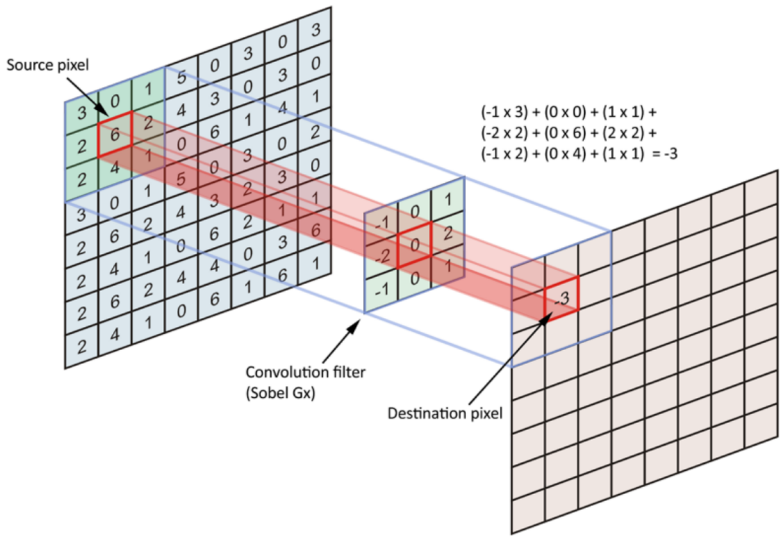

Figura 4.4. Ilustração da operação de convolução aplicada a uma imagem de entrada utilizando um filtro Sobel (Gx). O filtro 3×3 realiza multiplicações elemento a elemento com uma região da imagem, e os resultados são somados para gerar um valor no mapa de características. Esse processo permite a detecção de bordas horizontais, destacando padrões estruturais importantes na imagem. Fonte: [Hachilif et al. 2019]

Camadas de *Pooling*: Após as camadas convolucionais, geralmente são aplicadas camadas de *pooling*, que reduzem a dimensionalidade espacial dos mapas de características. Esta redução diminui o custo computacional e ajuda a rede a se tornar invariante a pequenas translações e distorções [Scherer et al. 2010]. Os dois tipos mais comuns de *pooling* são o *Max Pooling* e o *Average Pooling*. O *Max Pooling*, por exemplo, seleciona o valor máximo em cada região do mapa de características, preservando as informações mais proeminentes, como ilustrado na Figura 4.5. Nela, uma janela de 2x2 percorre a matriz de entrada e extrai o maior valor de cada quadrante para formar uma nova matriz de saída com dimensões reduzidas. Já o *Average Pooling* calcula a média dos valores em cada região, o que resulta em uma suavização das características. Na prática, o *Max Pooling* é mais utilizado, pois tende a preservar informações discriminativas importantes enquanto descarta detalhes menos relevantes [Boureau et al. 2010].

Camadas de Ativação: As funções de ativação introduzem não-linearidades no modelo, permitindo que a rede aprenda representações complexas. Sem estas não-linearidades, a rede se comportaria como uma única transformação linear, limitando severamente sua capacidade de aprendizado [Glorot et al. 2011]. As principais funções de ativação incluem:

- ReLU (*Rectified Linear Unit*): A mais popular devido à sua simplicidade e eficácia.

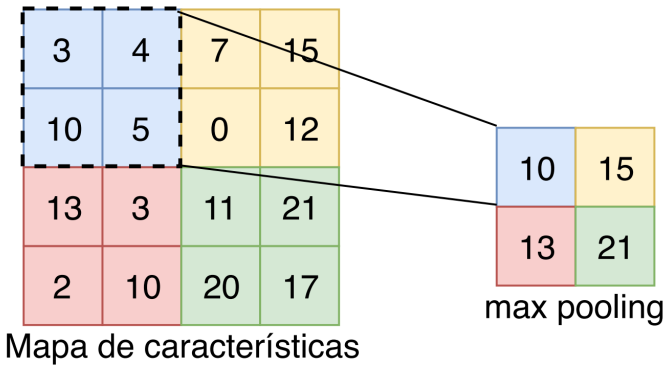

Figura 4.5. Operação de *Max Pooling* com uma janela 2×2. Fonte: [Vogado et al. 2019]

Substitui valores negativos por zero, evitando o problema do desaparecimento do gradiente em redes profundas;

- Sigmoid: Mapeia os valores para o intervalo [0,1], útil para a camada de saída em problemas de classificação binária;
- Softmax: Usada na camada de saída para problemas de classificação multiclasse, convertendo os valores em probabilidades que somam 1;
- LeakyReLU: Uma variação da ReLU que permite um pequeno gradiente negativo, evitando "neurônios mortos".

A escolha da função de ativação pode impactar significativamente o desempenho do modelo. Na prática, ReLU é frequentemente a escolha padrão para camadas intermediárias, enquanto Softmax é utilizada na camada de saída para classificação multiclasse [Nair and Hinton 2010].

Camadas *Flatten* e *Fully Connected*: Após várias camadas convolucionais e de *pooling*, os mapas de características são convertidos em um vetor unidimensional através da camada *Flatten*. Este vetor é então processado por uma ou mais camadas totalmente conectadas (*Fully Connected* ou *Dense*), onde cada neurônio está conectado a todos os neurônios da camada anterior.

As camadas *Fully Connected* realizam o raciocínio de alto nível, combinando todas as características extraídas pelas camadas convolucionais para realizar a classificação final. A última camada *Fully Connected* geralmente contém um número de neurônios igual ao número de classes, com uma função de ativação Softmax para gerar probabilidades de classificação [Goodfellow et al. 2016].

Em modelos para classificação do TEA, por exemplo, a camada final teria dois neurônios (com TEA/sem TEA), ou mais neurônios se estiver classificando diferentes subtipos ou níveis de severidade [Alves Rodrigues et al. 2023].

Estas camadas contêm a maior parte dos parâmetros da rede e, portanto, são mais propensas ao overfitting. Técnicas de regularização como Dropout são frequentemente aplicadas nestas camadas para melhorar a generalização do modelo [Srivastava et al. 2014].

4.2.2. Hiperparâmetros

Os hiperparâmetros são variáveis que controlam o processo de treinamento e a arquitetura da CNN. Diferentemente dos parâmetros (pesos e vieses) que são aprendidos durante o treinamento, os hiperparâmetros são definidos antes do início do treinamento e permanecem fixos durante todo o processo. A seleção adequada de hiperparâmetros é crucial para o desempenho do modelo [Bergstra and Bengio 2012].

Número e Tamanho das Camadas: A profundidade da rede (número de camadas) e o tamanho de cada camada (número de filtros ou neurônios) determinam a capacidade de representação do modelo. Redes mais profundas podem aprender representações mais complexas, mas também requerem mais dados e são mais propensas ao *overfitting* [Szegedy et al. 2015].

Estudos empíricos mostram que, em geral, é melhor usar redes mais profundas com camadas menores do que redes rasas com camadas maiores, pois isso permite aprender hierarquias de características mais ricas [He et al. 2016]. No entanto, a escolha ideal depende da complexidade do problema e da quantidade de dados disponíveis.

Taxa de Aprendizado: A taxa de aprendizado controla o tamanho dos passos durante a otimização. Uma taxa muito pequena resulta em convergência lenta, enquanto uma taxa muito grande pode impedir a convergência ou levar a mínimos locais subótimos [Ruder 2016].

Existem várias estratégias para definir a taxa de aprendizado. Uma delas é utilizar um valor fixo ao longo de todo o treinamento, como 0.001. Outra abordagem é a redução programada, na qual a taxa é diminuída em intervalos predefinidos ou quando o desempenho do modelo estagna. Também existem métodos de redução adaptativa, que ajustam automaticamente a taxa com base no desempenho, como o `ReduceLROnPlateau` no Keras. Além disso, programações cíclicas, como as propostas por [Smith 2017], fazem a taxa variar ciclicamente entre valores mínimo e máximo.

A escolha da taxa de aprendizado é frequentemente considerada o hiperparâmetro mais crítico em *Deep Learning*. Por isso, uma prática recomendada é realizar testes preliminares com diferentes valores, como 1×10^{-2}, 1×10^{-3}, 1×10^{-4}, para identificar o intervalo mais promissor para o problema específico [Bengio 2012].

Número de Épocas e Tamanho do *Batch*: Uma época representa uma passagem completa pelo conjunto de dados de treinamento. O número ideal de épocas depende da complexidade do problema e do tamanho do conjunto de dados. O treinamento deve continuar até que o modelo convirja, mas deve ser interrompido antes que ocorra *overfitting* [Goodfellow et al. 2016].

O tamanho do *batch* determina quantos exemplos são processados antes de atualizar os parâmetros do modelo. *Batches* maiores proporcionam estimativas de gradiente mais estáveis, mas requerem mais memória e podem levar a soluções com menor capacidade de generalização. *Batches* menores introduzem mais ruído no treinamento, o que pode ajudar a escapar de mínimos locais, mas pode tornar a convergência mais difícil [Keskar et al. 2017].

Em aplicações com conjuntos de dados limitados, como frequentemente ocorre em

estudos clínicos de TEA, técnicas como validação cruzada e *early stopping* são essenciais para determinar o número ideal de épocas e evitar *overfitting* [Alves Rodrigues et al. 2023].

Funções de Perda: A função de perda (*loss function*) quantifica a diferença entre as previsões do modelo e os valores reais, orientando o processo de otimização. A escolha da função de perda depende do tipo de problema [Janocha and Czarnecki 2017]. Para problemas de classificação, as funções mais comuns incluem:

- *Binary Cross-Entropy*: Para classificação binária (ex: com TEA/sem TEA).
- *Categorical Cross-Entropy*: Para classificação multiclasse (ex: diferentes subtipos de TEA).
- *Focal Loss*: Uma modificação da cross-entropy que atribui mais peso a exemplos difíceis, útil para conjuntos de dados desbalanceados [Lin et al. 2017].

Otimizadores: Os otimizadores controlam como os parâmetros do modelo são atualizados com base nos gradientes calculados. Cada otimizador tem características distintas que afetam a velocidade de convergência e a qualidade da solução final [Ruder 2016].

Na prática, Adam é frequentemente a escolha inicial devido ao seu bom desempenho geral e baixa necessidade de ajuste. No entanto, SGD com *momentum* às vezes atinge melhor generalização em tarefas de classificação de imagens [Wilson et al. 2017].

4.2.3. Arquiteturas

A evolução das arquiteturas de CNNs tem sido impulsionada principalmente pela competição *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), que desafiou pesquisadores a desenvolver modelos cada vez mais precisos para classificação e detecção de objetos em larga escala [Russakovsky et al. 2015].

Estas arquiteturas representam marcos importantes no desenvolvimento das CNNs e são frequentemente utilizadas como base para transferência de aprendizado em problemas específicos, como a classificação de imagens faciais para detecção de TEA.

DenseNet: Proposta por Huang et al. (2017), a *Densely Connected Convolutional Network* (DenseNet) estendeu a ideia de conexões de atalho, conectando cada camada a todas as camadas subsequentes dentro de um bloco denso. Essa arquitetura apresenta blocos densos nos quais cada camada recebe inputs de todas as camadas anteriores, permitindo a reutilização de características ao longo da rede. Além disso, *DenseNet* utiliza menos parâmetros, o que a torna menos propensa ao *overfitting*, e possibilita uma propagação eficiente do gradiente.

MobileNet: Desenvolvida por Howard et al. (2017), a *MobileNet* foi projetada especificamente para aplicações móveis e embarcadas com restrições de recursos computacionais. Essa arquitetura utiliza convoluções separáveis em profundidade (*depthwise separable convolutions*) e inclui hiperparâmetros que permitem controlar o tamanho e a latência do modelo. Comparada a arquiteturas similares, a *MobileNet* possui significativamente menos parâmetros e operações, garantindo bom desempenho para aplicações em tempo real.

A eficiência da *MobileNet* a torna particularmente útil para implementações em dispositivos móveis, potencialmente permitindo ferramentas de triagem de TEA acessíveis em regiões com recursos limitados [Alves Rodrigues et al. 2023].

Em aplicações de classificação facial para detecção de TEA, a escolha da arquitetura depende de vários fatores, incluindo tamanho do conjunto de dados, recursos computacionais disponíveis e requisitos de desempenho. Estudos recentes têm demonstrado bons resultados com arquiteturas como *DenseNet* e *MobileNet*, especialmente quando combinadas com técnicas de transferência de aprendizado [Alves Rodrigues et al. 2023, Pereira et al. 2024].

4.3. Construção de uma CNN

Após compreender os componentes e arquiteturas das CNNs, vamos explorar como construir uma rede convolucional básica utilizando Keras com TensorFlow. Para fins didáticos, desenvolveremos uma CNN genérica que segue a estrutura clássica introduzida por [LeCun et al. 1998] e refinada em trabalhos posteriores.

Nossa CNN será projetada para classificação binária de imagens faciais com dimensões 64×64×3, possuindo uma estrutura sequencial de camadas convolucionais, *pooling*, e camadas densas. Essa arquitetura é suficientemente complexa para demonstrar os princípios fundamentais, mas simples o bastante para facilitar a compreensão.

O Keras simplifica consideravelmente a implementação através de sua API intuitiva. O código 4.1 ilustra a construção da nossa CNN.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
   Flatten, Dense, Dropout, Activation, BatchNormalization

# Definicao do modelo
model = Sequential()

# Blocos convolucionais
model.add(Conv2D(32, kernel_size=(3, 3), padding='same',
   input_shape=(64, 64, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Classificacao
model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
```

```
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

# Compilacao
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Listing 4.1. Definição e compilação de uma CNN moderna em Keras para classificação binária

A primeira camada convolucional utiliza 32 filtros 3×3 para extrair características básicas da imagem de entrada. A seguir, a camada *BatchNormalization* normaliza as ativações, o que estabiliza e acelera o processo de treinamento. A função de ativação ReLU introduz não-linearidade, e a camada *MaxPooling2D* reduz as dimensões espaciais pela metade. O segundo bloco convolucional segue o mesmo padrão, mas com 64 filtros, aumentando a capacidade de representação enquanto as dimensões espaciais continuam diminuindo.

Após extrair características através das camadas convolucionais, a operação *Flatten* transforma o mapa de características tridimensional em um vetor unidimensional. Este vetor alimenta uma camada densa com 128 neurônios, que também é seguida por *BatchNormalization* e ReLU. A camada de *Dropout* que desativa aleatoriamente 50% dos neurônios durante o treinamento para prevenir *overfitting*. A camada final utiliza ativação sigmoid para produzir a probabilidade da classe positiva.

Na etapa de compilação, definimos o otimizador *Adam* e a função de perda *binary_crossentropy*, que é a escolha padrão e matematicamente adequada para problemas de classificação binária. A acurácia foi selecionada como a métrica para monitorar o desempenho do modelo durante o treinamento e a avaliação.

Ao projetar CNNs para problemas específicos como a detecção de TEA em imagens faciais, algumas considerações importantes incluem a resolução das imagens, a profundidade da rede, e técnicas de regularização. Imagens faciais geralmente beneficiam-se de resoluções maiores (128×128 ou 224×224) para preservar características sutis. A arquitetura pode ser ajustada adicionando mais camadas convolucionais para aumentar a capacidade de representação, ou incorporando técnicas como *Batch Normalization*, já incluída em nosso exemplo para garantir um treinamento mais estável e eficiente.

4.4. Transferência da Aprendizado

A transferência de aprendizado representa uma das técnicas mais importantes no arsenal do *deep learning* moderno, especialmente quando lidamos com conjuntos de dados limitados, como frequentemente ocorre em aplicações médicas. Esta abordagem permite aproveitar o conhecimento adquirido por uma rede neural em uma tarefa para melhorar seu desempenho em outra tarefa relacionada, mesmo quando os domínios são diferentes.

[Pan and Yang 2010] definem transferência de aprendizado como a situação onde o que foi aprendido em um cenário é explorado para melhorar a generalização em outro cenário.

No contexto da classificação de imagens faciais para detecção de TEA, a transferência de aprendizado é particularmente valiosa. Treinar uma CNN do zero geralmente requer dezenas de milhares de imagens rotuladas, um volume raramente disponível em estudos clínicos. Além disso, o treinamento completo demanda recursos computacionais significativos e tempo considerável. A transferência de aprendizado oferece uma alternativa eficiente, permitindo que modelos pré-treinados em grandes conjuntos de dados como ImageNet (com mais de um milhão de imagens) sejam adaptados para tarefas específicas com muito menos dados [Tajbakhsh et al. 2016].

Existem duas abordagens principais para a transferência de aprendizado em CNNs: extração de características e ajuste fino (*fine-tuning*). Cada abordagem tem seus méritos e é adequada para diferentes cenários, dependendo da similaridade entre as tarefas de origem e destino, e da quantidade de dados disponíveis para a nova tarefa.

4.4.1. Extração de Características

A extração de características é uma abordagem direta de transferência de aprendizado que aproveita a capacidade das CNNs de aprender representações hierárquicas dos dados. Nas primeiras camadas, as CNNs aprendem características genéricas como bordas e texturas; nas camadas intermediárias, padrões mais complexos; e nas camadas finais, representações específicas para a tarefa original [Yosinski et al. 2014]. A ideia fundamental é utilizar uma rede pré-treinada como um extrator fixo de características, removendo as camadas de classificação final e mantendo o restante da rede congelado (pesos não treináveis).

Nesta abordagem, as imagens são processadas pela CNN pré-treinada até uma camada intermediária específica, gerando vetores de características que são então utilizados para treinar um classificador simples como SVM (*Support Vector Machine*) ou até mesmo uma rede neural mais simples. Esta técnica é particularmente eficaz quando os dados disponíveis são limitados e a tarefa alvo é significativamente diferente da tarefa original.

Para implementar a extração de características usando Keras, podemos utilizar um modelo pré-treinado como a *VGG16* ou *ResNet50*, removendo as camadas de classificação e congelando os pesos das camadas convolucionais. Vemos um exemplo no código 4.2.

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model

# Carregar modelo pre-treinado sem as camadas de classificacao
base_model = VGG16(weights='imagenet', include_top=False,
   input_shape=(224, 224, 3))

# Congelar todas as camadas (tornar nao treinaveis)
for layer in base_model.layers:
    layer.trainable = False

# Funcao para extrair caracteristicas de um conjunto de imagens
```

```
def extract_features(images):
    features = base_model.predict(images)
    # Reshape para vetor unidimensional
    features = features.reshape(features.shape[0], -1)
    return features
```

Listing 4.2. Extração de características com VGG16 pré-treinada

4.4.2. Ajuste Fino

O ajuste fino (*fine-tuning*) representa uma abordagem mais avançada de transferência de aprendizado, onde algumas ou todas as camadas da rede pré-treinada são retreinadas com uma taxa de aprendizado reduzida. Esta técnica permite que o modelo se adapte mais especificamente à nova tarefa, mantendo o conhecimento geral adquirido durante o pré-treinamento.

O processo geralmente começa com um modelo pré-treinado. Em seguida, substitui-se sua camada de saída por uma nova camada adaptada ao problema específico, e a rede é treinada com uma taxa de aprendizado menor. Normalmente, congela-se a maior parte das camadas da rede e treina-se inicialmente apenas as camadas adicionadas. Posteriormente, mais camadas podem ser "descongeladas" progressivamente para um refinamento adicional. Uma implementação básica de ajuste fino com Keras pode ser vista no código 4.3.

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense,
   GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Carregar modelo base pre-treinado
base_model = ResNet50(weights='imagenet', include_top=False,
   input_shape=(224, 224, 3))

# Adicionar novas camadas de classificacao
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

# Modelo final
model = Model(inputs=base_model.input, outputs=predictions)

# Congelar camadas do modelo base
for layer in base_model.layers:
    layer.trainable = False

# Primeira fase: treinar apenas as camadas adicionadas
model.compile(optimizer='adam', loss='binary_crossentropy',
   metrics=['accuracy'])
```

```
model.fit(train_data, train_labels, epochs=5, validation_data=(
    val_data, val_labels))

# Segunda fase: ajuste fino (fine-tuning)
# Descongelar algumas camadas do modelo base
for layer in base_model.layers[-10:]:
    layer.trainable = True

# Recompilar com taxa de aprendizado menor
model.compile(optimizer=Adam(learning_rate=1e-5), loss='
    binary_crossentropy', metrics=['accuracy'])
model.fit(train_data, train_labels, epochs=10, validation_data=(
    val_data, val_labels))
```

Listing 4.3. Modelo com saída sigmoid e perda binária

O ajuste fino geralmente produz modelos com melhor desempenho do que a simples extração de características, especialmente quando o conjunto de dados de destino tem tamanho moderado a grande. [Yosinski et al. 2014] demonstraram que mesmo quando as tarefas são aparentemente distantes, o ajuste fino pode melhorar significativamente o desempenho em comparação com o treinamento do zero.

Na prática, a estratégia de ajuste fino deve considerar diversos fatores. O número de camadas a descongelar depende do tamanho do conjunto de dados disponível e da similaridade entre as tarefas. Com conjuntos de dados pequenos, descongelar muitas camadas pode levar ao *overfitting*. A taxa de aprendizado deve ser significativamente menor (geralmente 10 a 100 vezes) que a utilizada para treinamento do zero, para evitar destruir o conhecimento previamente adquirido.

Para problemas de análise facial como a detecção de TEA, o ajuste fino tem demonstrado resultados promissores. [Alves Rodrigues et al. 2023] aplicaram esta técnica com a arquitetura DenseNet para classificar imagens faciais de indivíduos com e sem TEA, alcançando acurácias superiores a 90%. Os autores observaram que o ajuste fino de apenas as camadas finais produziu os melhores resultados, porque as características genéricas aprendidas nas camadas iniciais da rede já eram adequadas para capturar estruturas faciais relevantes.

4.5. Estudo de Caso

Para demonstrar a aplicação prática dos conceitos apresentados, desenvolvemos um estudo de caso focado na detecção do Transtorno do Espectro Autista (TEA) através de imagens faciais utilizando redes neurais convolucionais. Este caso ilustra como as técnicas de *deep learning* podem ser aplicadas em problemas reais da área da saúde, apresentando tanto as potencialidades quanto os desafios inerentes a essas aplicações.

4.5.1. Problema

O Transtorno do Espectro Autista é um distúrbio do neurodesenvolvimento caracterizado por dificuldades na comunicação social e interpessoal, retardo de linguagem, comportamento repetitivo e disfunção sensorial. Estima-se que uma em cada 44 crianças nos Esta-

dos Unidos seja acometida por esse transtorno, com números similares sendo observados globalmente [Centers for Disease Control and Prevention 2023].

A intervenção precoce desempenha papel crucial no desenvolvimento de indivíduos com TEA devido à plasticidade cerebral característica dos primeiros anos de vida. Os ganhos decorrentes da intervenção precoce podem reduzir consideravelmente os gastos familiares no tratamento das crianças com TEA, bem como os custos dos sistemas de saúde pública. Além disso, a intervenção precoce é de grande valor para reduzir os impactos do TEA sobre as famílias e a sociedade [Alves Rodrigues et al. 2023].

4.5.2. Base de Dados

Para este estudo de caso, utilizamos o "*Autism Image Data*"disponibilizado na plataforma Kaggle, que representa o único conjunto de dados publicamente disponível com imagens faciais bidimensionais estáticas de indivíduos com TEA. Este *dataset* foi criado especificamente para pesquisas em detecção automática de TEA utilizando técnicas de visão computacional.

O conjunto de dados é composto por 2.938 imagens faciais no formato JPEG com dimensões padronizadas de 224×224x3 *pixels*. As imagens foram coletadas através de buscas sistemáticas na web e processadas para focar nas características faciais, com remoção de fundos e recorte centrado no rosto. A distribuição do *dataset* é equilibrada entre as classes e organizada em conjuntos de treinamento (2.538 imagens, 86%), validação (200 imagens, 7%) e teste (200 imagens, 7%), com proporção igual entre indivíduos com TEA e desenvolvimento típico em cada divisão. Na figura 4.6, observamos exemplos das imagens contidas nesse *dataset*.

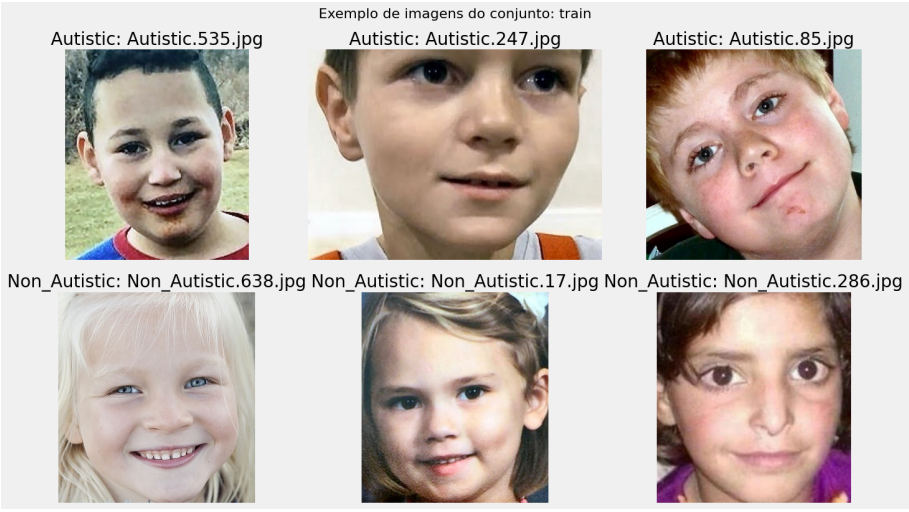

Figura 4.6. Exemplos de imagens do conjunto de treino, com amostras de indivíduos com (linha superior) e sem TEA (linha inferior). Fonte: https://www.kaggle.com/datasets/cihan063/autism-image-data

O pré-processamento incluiu normalização dos valores dos *pixels* para o intervalo [0, 1] e verificação da qualidade das imagens. Todas as imagens são coloridas e possuem qualidade adequada para preservar características faciais relevantes. É importante reconhecer as limitações inerentes a este conjunto de dados, incluindo variabilidade em

qualidade, iluminação e pose devido à coleta web, além da ausência de controle rigoroso sobre verificação independente dos diagnósticos. Essas limitações são típicas de estudos exploratórios e devem ser consideradas na interpretação dos resultados.

Apesar das limitações, este dataset tem sido amplamente utilizado na literatura científica como *benchmark* para avaliação de métodos de detecção automática de TEA, permitindo comparações consistentes entre diferentes abordagens metodológicas.

4.5.3. Aumento de Dados

O aumento de dados (*data augmentation*) representa uma técnica fundamental para melhorar a generalização de modelos de *deep learning*, especialmente quando trabalhamos com conjuntos de dados limitados. Esta técnica gera novos exemplos de treinamento a partir dos dados existentes através de transformações que preservam a classe da imagem, efetivamente aumentando o tamanho do conjunto de dados de forma artificial [Mumuni and Mumuni 2022].

Em análise de imagens faciais para detecção de TEA, o aumento de dados requer cuidado especial para evitar transformações que possam alterar características morfológicas importantes. Transformações como rotações extremas ou distorções perspectivas podem modificar relações espaciais cruciais entre características faciais, potencialmente removendo ou criando artificialmente os biomarcadores que estamos tentando detectar.

Para este estudo de caso, como mostra a figura 4.7, aplicamos apenas o espelhamento horizontal (*horizontal flip*), uma transformação que preserva todas as características morfológicas importantes enquanto dobra efetivamente o tamanho do conjunto de dados. Esta escolha conservadora é justificada pela natureza específica do problema, onde características faciais sutis podem ser fundamentais para a classificação.

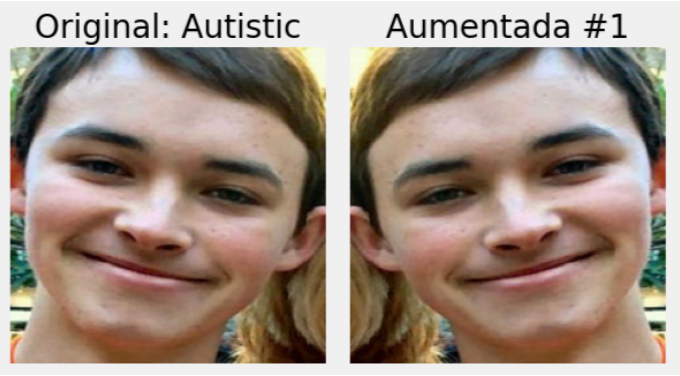

Figura 4.7. Exemplo de aumento de dados aplicado a uma imagem do *dataset*. À esquerda, a imagem original; à direita, a versão aumentada por espelhamento horizontal.

Testes preliminares com outras transformações como rotação, *zoom* e brilho revelaram que essas modificações tendiam a degradar o desempenho do modelo, provavelmente porque alteravam características faciais relevantes para a detecção de TEA. Este resultado reforça a importância de considerar a natureza específica do problema ao escolher estratégias de aumento de dados.

A técnica de aumento de dados não apenas aumenta a quantidade de dados dis-

poníveis, mas também melhora a capacidade de generalização do modelo ao expô-lo a variações naturais que podem ocorrer em aplicações reais. No contexto da detecção de TEA, onde eventualmente o modelo pode ser aplicado a imagens capturadas em condições variadas através de dispositivos móveis, essa robustez adicional é particularmente valiosa.

4.5.4. Implementação

A implementação prática do modelo para detecção de TEA utilizou técnicas de transferência de aprendizado com múltiplas arquiteturas CNN pré-treinadas. Foram avaliadas cinco arquiteturas: *MobileNet*, *MobileNetV2*, *DenseNet201*, *VGG19* e *ResNet50V2*, todas inicializadas com pesos pré-treinados na *ImageNet* e adaptadas para o problema específico de classificação binária de TEA, como mostra o código 4.4.

```
from tensorflow.keras.applications import DenseNet201, MobileNet
   , VGG19, ResNet50V2
from tensorflow.keras.layers import Dense,
   GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

def create_model(base_architecture='DenseNet201', input_shape
   =(224, 224, 3)):
    # Selecao da arquitetura base
    if base_architecture == 'DenseNet201':
        base_model = DenseNet201(weights='imagenet', include_top
           =False, input_shape=input_shape)
    elif base_architecture == 'MobileNet':
        base_model = MobileNet(weights='imagenet', include_top=
           False, input_shape=input_shape)
    elif base_architecture == 'VGG19':
        base_model = VGG19(weights='imagenet', include_top=False
           , input_shape=input_shape)
    elif base_architecture == 'ResNet50V2':
        base_model = ResNet50V2(weights='imagenet', include_top=
           False, input_shape=input_shape)

    # Congelar camadas do modelo base
    for layer in base_model.layers:
        layer.trainable = False

    # Adicionar camadas de classificacao personalizadas
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.4)(x)
    predictions = Dense(1, activation='sigmoid')(x)  #
       Classificacao binaria
```

```
    # Modelo final
    model = Model(inputs=base_model.input, outputs=predictions)

    # Compilacao para classificacao binaria
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=['accuracy', 'precision', 'recall']
    )

    return model, base_model
```

Listing 4.4. Implementação do modelo para detecção de TEA usando transferência de aprendizado com arquiteturas pré-treinadas na ImageNet, adaptadas para classificação binária.

Para classificação binária, utilizamos uma única saída com ativação *sigmoid*, que mapeia os valores para o intervalo [0,1] representando a probabilidade de pertencer à classe TEA. Esta abordagem é mais direta e eficiente para problemas de duas classes, utilizando a função de perda *binary_crossentropy* que é otimizada especificamente para este cenário.

No código 4.5, observa-se que o carregamento dos dados foi adaptado para o formato binário, onde as classes são representadas como 0 (desenvolvimento típico) e 1 (TEA):

```
from tensorflow.keras.preprocessing.image import
   ImageDataGenerator

# Configuracao dos geradores para classificacao binaria
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Carregamento dos dados com class_mode='binary'
train_generator = train_datagen.flow_from_directory(
    'autism_dataset/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',  # Modo binario
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    'autism_dataset/train',
    target_size=(224, 224),
    batch_size=32,
```

```
    class_mode='binary',
    subset='validation'
)

test_generator = test_datagen.flow_from_directory(
    'autism_dataset/test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
```

Listing 4.5. Preparação dos geradores de dados para treino, validação e teste com classificação binária

O treinamento foi conduzido em duas fases seguindo as melhores práticas de transferência de aprendizado como mostra o código 4.6. Na primeira fase, apenas as camadas de classificação adicionadas foram treinadas por 20 épocas, mantendo todas as camadas do modelo base congeladas.

```
# Primeira fase treinamento das camadas superiores
model, base_model = create_model('DenseNet201')

history_phase1 = model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=5,
           restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(patience=3, factor
           =0.5)
    ]
)

# Segunda fase fine tuning opcional
# Descongelar algumas camadas superiores do modelo base
for layer in base_model.layers[-10:]:
    layer.trainable = True

# Recompilar com taxa de aprendizado menor
model.compile(
    optimizer=Adam(learning_rate=0.0001),  # Taxa 10x menor
    loss='binary_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

history_phase2 = model.fit(
    train_generator,
    epochs=10,
```

```
    validation_data=validation_generator,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=3,
           restore_best_weights=True)
    ]
)
```

Listing 4.6. Treinamento e Fine tuning com DenseNet201

Durante os experimentos, foram implementadas técnicas de regularização e monitoramento para prevenir *overfitting*. O *Early Stopping* interrompe o treinamento quando a perda de validação para de melhorar, enquanto o `ReduceLROnPlateau` ajusta automaticamente a taxa de aprendizado quando o modelo atinge um platô de desempenho.

Os resultados experimentais demonstraram que as arquiteturas *DenseNet201* e *MobileNet* obtiveram os melhores desempenhos. A *DenseNet201* alcançou acurácia média de 90,7% com desvio padrão de 1,64%, chegando a 93,5% nos melhores casos. A *MobileNet*, embora com acurácia máxima ligeiramente inferior (92,5%), apresentou maior estabilidade com desvio padrão de apenas 0,68%, indicando comportamento mais consistente entre diferentes execuções.

Para avaliar o desempenho final do modelo e analisar detalhadamente seus resultados, implementamos uma rotina de avaliação abrangente que calcula métricas de desempenho, gera a matriz de confusão e analisa a distribuição das probabilidades preditas. O código 4.7 demonstra essa análise.

```
# Avaliacao do modelo no conjunto de teste
test_results = model.evaluate(test_generator)
print(f"Acuracia no teste {test_results[1]:.3f}")
print(f"Precisao no teste {test_results[2]:.3f}")
print(f"Recall no teste {test_results[3]:.3f}")

# Predicoes para analise detalhada
predictions = model.predict(test_generator)
# Para classificacao binaria aplicamos threshold de 05
predicted_classes = (predictions > 0.5).astype(int).flatten()
true_classes = test_generator.classes

# Matriz de confusao
from sklearn.metrics import confusion_matrix,
   classification_report
cm = confusion_matrix(true_classes, predicted_classes)
print("Matriz de Confusao")
print(cm)
print("\nRelatorio de Classificacao")
print(classification_report(true_classes, predicted_classes,
                            target_names=['Desenvolvimento Tipico'
                               , 'TEA']))

# Analise das probabilidades preditas
```

```
print(f"\nEstatisticas das predicoes")
print(f"Probabilidade media para classe TEA {predictions.mean()
    :.3f}")
print(f"Desvio padrao {predictions.std():.3f}")
print(f"Predicoes com alta confianca maior que 08 ou menor que
    02 {((predictions > 0.8) | (predictions < 0.2)).sum()}/{len(
    predictions)}")
```

Listing 4.7. Avaliação do modelo no conjunto de teste

4.5.5. Explicabilidade com GradCAM

A interpretabilidade de modelos de *deep learning* é crucial em aplicações médicas, onde a confiança e compreensão das decisões do modelo são tão importantes quanto sua precisão. Para este fim, implementamos o *Gradient-weighted Class Activation Mapping* (GradCAM), uma técnica que permite visualizar quais regiões da imagem são mais importantes para as decisões do modelo [Selvaraju et al. 2017].

O GradCAM funciona calculando os gradientes da classe prevista em relação aos mapas de características de uma camada convolucional específica. Estes gradientes são então utilizados como pesos para combinar os mapas de características, produzindo um mapa de calor que destaca as regiões mais influentes para a classificação. O código 4.8 apresenta uma implementação simplificada dessa técnica, incluindo a geração e visualização do mapa de calor sobre a imagem de entrada.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
import cv2

def grad_cam(model, img_array, layer_name, class_idx):
    grad_model = Model([model.inputs], [model.get_layer(
        layer_name).output, model.output])

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[:, class_idx]

    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    heatmap = tf.reduce_sum(tf.multiply(pooled_grads,
        conv_outputs[0]), axis=-1)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(
        heatmap)

    return heatmap.numpy()

def visualize_gradcam(model, image, true_label, predicted_label,
    class_names):
```

```
    for layer in reversed(model.layers):
        if 'conv' in layer.name.lower():
            last_conv_layer = layer.name
            break
    else:
        base_model = model.layers[0]
        for layer in reversed(base_model.layers):
            if 'conv' in layer.name.lower():
                last_conv_layer = layer.name
                break

    img_array = np.expand_dims(image, axis=0)
    heatmap = grad_cam(model, img_array, last_conv_layer,
       predicted_label)

    heatmap_resized = cv2.resize(heatmap, (image.shape[1], image
       .shape[0]))
    heatmap_colored = cv2.applyColorMap(np.uint8(255 *
       heatmap_resized), cv2.COLORMAP_JET)
    superimposed = cv2.addWeighted(np.uint8(255 * image), 0.6,
       heatmap_colored, 0.4, 0)

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))
    axes[0].imshow(image)
    axes[0].set_title(f'Original\nVerdadeiro {class_names[
       true_label]}')
    axes[1].imshow(heatmap_resized, cmap='jet')
    axes[1].set_title(f'Mapa de Calor\nPredito {class_names[
       predicted_label]}')
    axes[2].imshow(superimposed)
    axes[2].set_title('Sobreposicao')

    for ax in axes:
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

Listing 4.8. Implementação simplificada do GradCAM

A técnica GradCAM fornece uma visualização das regiões da imagem que influenciam as decisões do modelo, mas seus resultados devem ser interpretados com cautela. As áreas destacadas não necessariamente correspondem a biomarcadores clínicos reais do TEA. Essa interpretabilidade permite verificar possíveis vieses, como atenção a elementos irrelevantes, e avaliar se o modelo está focando em regiões anatomicamente plausíveis, além de facilitar a comunicação com profissionais da saúde ao tornar o processo de decisão mais transparente.

Neste estudo, o GradCAM foi utilizado para explorar o comportamento de uma CNN na detecção do TEA por meio de imagens faciais, demonstrando desempenho pro-

missor. No entanto, destaca-se que esses modelos devem atuar como apoio ao julgamento clínico, e não como substitutos. A implementação prática ainda requer atenção a fatores como viés nos dados, capacidade de generalização e validação clínica rigorosa.

Referências

[Alves Rodrigues et al. 2023] Alves Rodrigues, J. N., Teixeira Aires, K. R., Branco Soares, A. C., Machado, V. P., and de Melo Souza Veras, R. (2023). Classification of facial images using deep learning models to support asd identification. In *2023 XLIX Latin American Computer Conference (CLEI)*, pages 1–9. IEEE.

[American Psychiatric Association 2013] American Psychiatric Association (2013). *Diagnostic and statistical manual of mental disorders (DSM-5)*. American Psychiatric Pub.

[Bengio 2012] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

[Bergstra and Bengio 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305.

[Boureau et al. 2010] Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118.

[Centers for Disease Control and Prevention 2023] Centers for Disease Control and Prevention (2023). Data and statistics on autism spectrum disorder. Accessed: 2025-05-25.

[Claro et al. 2020] Claro, M. L., Vogado, L. H. S., Santos, J. D., and Veras, R. M. S. (2020). Utilização de técnicas de data augmentation em imagens: Teoria e prática. In Teles, A. S., Calçada, D. B., and Veras, N. L., editors, *Livro de Minicursos - VIII Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI 2020)*, chapter 3, pages 47–71. Sociedade Brasileira de Computação, Porto Alegre.

[Fukushima 1988] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130.

[Glorot et al. 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings.

[Gonzalez and Woods 2002] Gonzalez, R. C. and Woods, R. E. (2002). *Digital Image Processing*. Prentice Hall.

[Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Hachilif et al. 2019] Hachilif, R., Baghdadi, R., and Benhamida, F. (2019). *Graduation Thesis: Implementing and Optimizing Neural Networks using Tiramisu*. Phd thesis, Unknown Institution.

[Haykin 2007] Haykin, S. (2007). *Redes neurais: princípios e prática*. Bookman Editora.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778.

[Janocha and Czarnecki 2017] Janocha, K. and Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

[Keskar et al. 2017] Keskar, N. S., Nocedal, J., Tang, D., Mudigere, D., and Smelyanskiy, M. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

[LeCun et al. 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lin et al. 2017] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pages 2980–2988.

[Lorena et al. 2021] Lorena, A., Faceli, K., Almeida, T., de Carvalho, A., and Gama, J. (2021). *Inteligência Artificial: uma abordagem de Aprendizado de Máquina*. LTC, 2 edition.

[Marques Filho and Neto 1999] Marques Filho, O. and Neto, H. V. (1999). *Processamento digital de imagens*. Brasport.

[Mumuni and Mumuni 2022] Mumuni, A. and Mumuni, F. (2022). Data augmentation: A comprehensive survey of modern approaches. *Array*, 16:100258.

[Nair and Hinton 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Pan and Yang 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[Pereira et al. 2024] Pereira, C. B. A. A., Barros, P., Rodrigues, J. N., Araújo, P., Borges, R., Almeida, K., and Veras, R. (2024). Identificação de parkinson em imagens faciais usando modelos de deep learning pré-treinados. In *Anais da XII Escola Regional de Computação do Ceará, Maranhão e Piauí*, pages 169–178. SBC.

[Rezende 2003] Rezende, S. O. (2003). *Sistemas inteligentes: fundamentos e aplicações*. Editora Manole Ltda.

[Ruder 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[Russakovsky et al. 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[Scherer et al. 2010] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, volume 6354 of *Lecture Notes in Computer Science*, pages 92–101. Springer.

[Selvaraju et al. 2017] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626. IEEE.

[Smith 2017] Smith, L. N. (2017). Cyclical learning rates for training neural networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472.

[Srivastava et al. 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[Szegedy et al. 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1–9.

[Tajbakhsh et al. 2016] Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., and Liang, J. (2016). Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312.

[Vogado et al. 2019] Vogado, L., Claro, M., Santos, J., and Veras, R. (2019). Deep learning em imagens: aplicando cnns com keras e tensorflow. In *Minicursos ERCAS-PI e ENUCOMPI 2019*, chapter 6, pages 65–96. Sociedade Brasileira de Computação, Porto Alegre.

[Wilson et al. 2017] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4151–4161.

[Yosinski et al. 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27.