Relatório Técnico: Implementação e Análise do Algoritmo Rede Neural Convolucional

Ícaro José Batista de Oliveira

Implementação
de uma rede convolucional
para o desafio de Ciência de dados

02/12/2024

1 INTRODUÇÃO

Reconhecer formas e figuras é uma característica intrínseca dos animais; essa capacidade de identificar - e, consequentemente, aprender o significado de cada imagem - com base nos padrões de uma figura ocorre de maneira natural. O processo envolve diversos órgãos e nervos e, como é conhecido pela biologia e medicina, funciona da seguinte forma: inicia-se na camada de entrada (o olho), onde a informação é convertida em sinais elétricos e, por fim, enviada ao cérebro. No cérebro, o córtex visual processa essa informação e a compara com situações já conhecidas e familiares, classificando o dado de "entrada". Como em diversos campos da ciência, a natureza serve como fonte de inspiração para descobertas e avanços científicos; com o reconhecimento de imagens (e sinais), não foi diferente. O processo de aprendizado das Redes Neurais Convolucionais (e o próprio modelo), utilizadas para tarefas de reconhecimento de sinais, é mais uma das descobertas humanas baseadas em princípios e funcionamentos naturais. Assim, as Redes Neurais Convolucionais representam mais uma inovação tecnológica, mostrando como a biologia do ser vivo continua a ser uma fonte de inpiração para avanços científicos.

O projeto atual, vem então, fazer uso desses modelos para classificar as imagens das faces de acordo com o sexo biológico (masculino ou feminino). O projeto então, fará uso de diversas faces masculinas e femininas para treinamento, validação e teste do modelo.

O Dataset é composto de 188 entradas (isso é, 188 linhas), e a distribuição de imagens por sexo é a seguinte:

Table 1: Descrição do Dataset por gênero

Sexo	**Quantidade**
Masculino	134
Feminino	52

1.1 Sobre Redes Neurais Convolucionais

As Redes Neurais Convolucionais (CNNs) são fundamentais para a resolução de problemas de visão computacional. Sua arquitetura começa com a camada de entrada, onde imagens ou conjuntos de pixels são organizados em uma matriz tridimensional, contendo altura, largura e número de canais de cor. A seguir, a Camada Convolucional extrai características da imagem, como bordas, linhas e formas, essenciais para o aprendizado do modelo. Após isso, os dados passam para a Camada de Pooling, que reduz a dimensionalidade, o que diminui a complexidade computacional e ajuda a prevenir overfitting. Finalmente, as camadas Dropout e Flatten preparam os dados para as decisões finais do modelo. As CNNs são importantes por sua capacidade de detectar e aprender padrões locais nas imagens, com base nos filtros (ou Kernels), na geração de mapa de característica e passagem de uma camada para outra. Esse modo de operar as torna eficazes em tarefas como reconhecimento de objetos, detecção e segmentação de imagens, além de melhorar a eficiência computacional ao reduzir a redundância e evitar ajustes excessivos aos dados de treinamento.

2 METODOLOGIA

2.1 Preparação dos dados

No primeiro momento, foi realizado o estudo do Dataset, que, na verdade, consistia em um álbum com diversas fotos de rostos masculinos e femininos. Como ponto de partida, foram criadas duas listas. A primeira era composta pelos nomes das imagens, ou seja, uma representação das imagens. A segunda,

com o mesmo tamanho, continha as anotações dos labels de cada foto. Ou seja, há uma interseção baseada na posição das duas listas: por exemplo, a posição 0 de ambas as listas corresponderia à imagem e ao seu respectivo label.

Após isso, foi realizado o redimensionamento para 250 x 200 pixels e a normalização dos valores RGB para a faixa de 0 a 1, com o objetivo de facilitar o processo de convergência e aprendizado do modelo.

Por fim, foi realizada uma análise de como os rostos estavam distribuídos, conforme mostrado na Tabela 1. Havia quase o triplo de rostos masculinos em relação aos femininos. Para contornar possíveis problemas decorrentes do modelo dar prioridade à classe com mais dados, foi empregado um processo de balanceamento. Esse processo consistia na realização de over-sampling na classe menos representada. Com esse processo de over-sampling, o número total de amostras aumentou de 188 para 268.

Em seguida, foi feita a divisão do conjunto de 268 entradas em conjuntos de treinamento, validação e teste, com uma proporção de 50%, 30% e 20%, respectivamente.

Por fim, os conjuntos ficaram com a seguinte forma e quantidade:

Table 2: Descrição do Dataset por gênero

Tipo	**Descrição**
Treinamento	(134, 250, 200, 3) (134,)
Validação	(54, 250, 200, 3) (54,)
Teste	(80, 250, 200, 3) (80,)

2.2 Implementação do Algoritmo

Em relação ao modelo, foi escolhido um com 5 camadas, e a modelagem dele foi feita de uma maneira sequencial. Primeiro, a camada de entrada. Essa camada não define somente como são os dados de entrada (250x200x3), ela já é operacional, visto que uma operação de convolução ocorre logo de início. Os parâmetros usados nela estabelecem a quantidade de filtros igual a 32, sendo esses filtros de tamanho 3x3, e o stride (movimento do kernel) é píxel a píxel. Para o padding, usou-se o parâmetro same, que adiciona zeros ao redor das bordas da entrada na operação convolucional. Isso garante que o filtro 3x3 passe por todos os pixels, incluindo as bordas, e que as dimensões espaciais (largura e altura) da saída sejam mantidas inalteradas. No entanto, em operações subsequentes, como o MaxPooling, o tamanho da saída é reduzido dependendo do tamanho da janela de pooling e do stride, mesmo com o uso de padding. A função de ativação utilizada é a ReLU (Rectified Linear Unit), para todas as camadas convolucionais. O uso dessa função é, como já explicado, para introduzir não-linearidade e fazer com que os dados aprendam padrões complexos, além de realizar uma filtragem dos dados que são passados de uma camada convolucional para outra. Por exemplo, se em alguma célula dos mapas de características existisse um valor negativo, ele seria transformado em 0, evitando que valores não relevantes fossem passados adiante.

Para as outras camadas convolucionais, a arquitetura é a mesma. O tamanho do filtro é 3x3, o stride é 1, o padding é same, e a função de ativação é ReLU. A diferença principal está no fato de que os filtros são variados. Na segunda camada convolucional, o filtro tem tamanho 64, enquanto na terceira camada o filtro passa a ter tamanho 128. A escolha desses filtros foi baseada em escolhas empíricas, ou seja, testes entre diferentes filtros. O fato de, nas primeiras camadas, o foco do aprendizado ser em características mais simples foi um dos fatores para o uso do filtro 32 e 64 nas duas primeiras camadas. Na última camada, que já carrega um maior nível de abstração, foi aplicado um filtro de 128.

Interessante notar que entre as camadas 1-2, 2-3 e 3-4, existem camadas de pooling, que visam reduzir a dimensionalidade dos mapas de características. Essas camadas também utilizam o parâmetro padding='same', assim como na camada convolucional, para que o pooling funcione corretamente na camada de saída. O tamanho da janela de pooling é 2x2, e é aplicado o MaxPooling, ou seja, dentro da janela de 4 células, o valor máximo será selecionado.

Antes da camada densa (última camada), há as camadas de Dropout e Flatten. A primeira é utilizada para introduzir regularização e evitar que o modelo memorize ou "sobreajuste" os dados. Ela funciona desligando, a cada interação, durante o treinamento, uma quantidade especificada de neurônios. Dessa forma, nenhum neurônio fica "encarregado" demais de memorizar os dados, e o

aprendizado é distribuído entre os neurônios. A taxa de dropout utilizada no modelo foi de 0.5, ou seja, metade dos neurônios são desligados aleatoriamente.

A camada Flatten, como já dito anteriormente, é usada para transformar a saída da última camada convolucional em uma entrada 1D para a camada densa. Por fim, a camada densa utiliza a função de ativação "relu" na camada intermediária e a sigmoid para gerar a saída. Como o problema exige a classificação binária, a saída é única e pode assumir o valor 0 (Masculino) ou 1 (Feminino).

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 250, 200, 32)	896
max_pooling2d (MaxPooling2D)	(None, 125, 100, 32)	0
conv2d_1 (Conv2D)	(None, 125, 100, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 63, 50, 64)	0
conv2d_2 (Conv2D)	(None, 63, 50, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 32, 25, 128)	0
dropout (Dropout)	(None, 32, 25, 128)	0
flatten (Flatten)	(None, 102400)	0
dense (Dense)	(None, 256)	26,214,656
dropout_1 (Dropout)	(None, 256)	0
preds (Dense)	(None, 1)	257

Figure 1: Arquitetura da solução.

2.3 Validação

Estrutura formada, é necessário compilar o modelo no Keras. Primeiro, foi estabelecido um otimizador, sendo o escolhido o "Adam", com uma taxa de aprendizado inicial de 0.001. O otimizador Adam apresenta o termo de momentum e uma adaptação da taxa de aprendizado, o que acelera o treinamento do modelo.

Para o treinamento, foi utilizado um número máximo de épocas de 15. O tamanho do batch foi definido como 16, o que significa que os pesos seriam atualizados a cada 16 dados (imagens) passados para o modelo. Com isso, o modelo realiza aproximadamente 8 iterações por época (134 dados de teste divididos por 16). A escolha do tamanho do batch foi feita de forma empírica. Observou-se que, para batches maiores, o modelo priorizava certas classes mais do que outras, possivelmente devido ao desequilíbrio inicial das classes no dataset ou a padrões específicos observados nos dados.

Foi utilizada uma validação simples: a cada época, os dados de teste que o modelo não havia visto eram apresentados, com as mesmas métricas de avaliação e a mesma função de perda. Esse processo foi de grande importância, visto que os ajustes no modelo foram baseados tanto na perda e acurácia do treinamento quanto na perda e acurácia da validação. A escolha da quantidade de época, foi também baseada em análises, onde a partir da 15º época, a perda da validação estava começando a aumentar em valor. Adicionado a isso, e visto que a acurácia do treinamento estava aumentando e a do conjunto de validação se mantendo a mesma, bem como o cálculo do F1-score e do AUC-ROC não estavam aumentando, estabeleceu 15 como o número máximo, visto que esse comportamento tinha tendência a ser "overfitting".

Foram feitos diversos outros testes com números de batchs, épocas, Kernel e outros parâmetros diferente, os parâmetros finais foram os melhores escolhidos a mão. Não foram usados nenhum métodos

de otimização de HyperParâmetros, sendo os parâmetros encontrados resultados de análises e testes empirícos.

3 RESULTADOS e DISCUSSÃO

As métricas de avaliação utilizadas foram:

- **Acurácia:** Mede a proporção de previsões corretas em relação ao total de previsões feitas. A acurácia no conjunto de teste foi de 92,59%, o que indica um bom desempenho na classificação das amostras do conjunto de teste.
- **F1-Score:** Combina a precisão e recall, sendo especialmente útil em problemas com classes desbalanceadas. O F1-Score para a classe 0 foi de 0.93, enquanto para a classe 1 foi de 0.92, com um F1-Score médio ponderado de 0.93. Esses valores indicam que o modelo tem um bom equilíbrio entre precisão e recall, sendo eficiente na classificação de ambas as classes.
- **Área sob a Curva ROC (AUC-ROC):** Mede a capacidade do modelo de distinguir entre as classes. O AUC-ROC obtido foi de 0.98, o que sugere que o modelo tem uma excelente capacidade de discriminação entre as classes, com uma alta taxa de acerto.

Foi plotado uma Matriz de Confusão. O gráfico a seguir relaciona os valores reais com os classificados pelo modelo, mostrando que apesar da presença de falsos-positivos e falsos-negativos (2 para homem e 2 para mulher) o modelo ainda conseguiu classificar 50 faces corretamente.

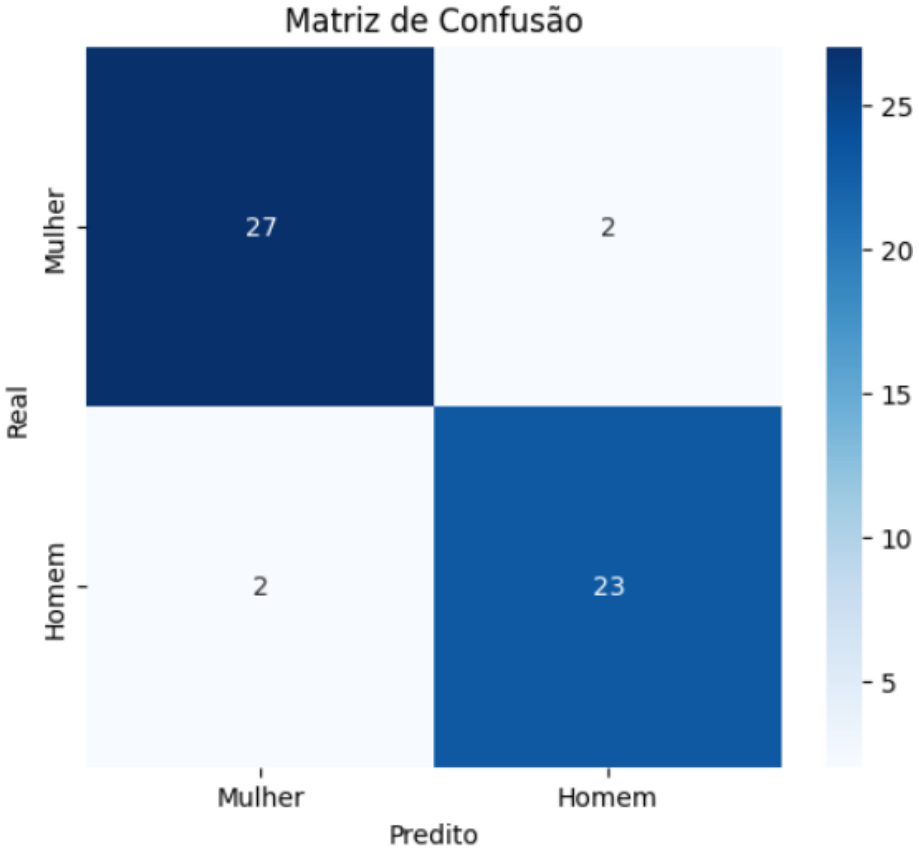

Figure 2: Matriz de Confusão.

Foi calculado um relatório de classificação com diversas métricas, entre elas o F1-score, como mostrado pela Tabela 3, a seguir.

Essa Tabela 3 mostra que houve uma boa precisão de acerto tanto na classe 0 (Masculino) quanto na classe 1 (Feminino), com F1-Score de 0.93 para a classe 0 e 0.92 para a classe 1. Esses valores indicam que o modelo tem uma boa capacidade de não favorecer nenhuma das duas classes (não há desequilíbrio), além de demonstrar uma boa precisão e recall, ou seja, a capacidade de identificar corretamente as classes.

Classe	**Precisão**	**Recall**	**F1-Score**	**Suporte**
0	0.93	0.93	0.93	29
1	0.92	0.92	0.92	25
Média Macro	0.93	0.93	0.93	54
Média Ponderada	0.93	0.93	0.93	54

Table 3: Relatório de Classificação - F1-Score

A falta de balanceamento dos dados, como descrita inicialmente nesse relatório, foi então mitigada pelos métodos de balanceamento de classes, como apontado pelo cálculo de F1-score.

Por fim, analisando as imagens que foram classificadas erroneamente, não houve um motivo central em relação ao erro. Os motivos foram variados. Por exemplo, após a normalização de uma imagem, é possível perceber 'borrões', como se outra imagem estivesse por cima. Essa mesma imagem, entretanto, no conjunto das imagens, já apresentava baixa qualidade, além da linha negra na parte superior dela.

Já outra imagem parece ser menos nítida do que as outras (desde antes da normalização), o que pode ter contribuído para o erro também. Por fim, as outras duas imagens não apresentaram problemas aparentes e pode ter sido apenas um erro do modelo, que julgou que as formas e características daquela face pertenciam a outra classe.

Para melhorar o desempenho do modelo, testar outras arquiteturas com mais camadas (e com mais filtros nessas camadas) pode ser uma abordagem interessante. A variação do tamanho do filtro, principalmente em camadas mais profundas, também é válida, para captura de informações diferentes, visto que foi usado somente filtros 3x3. Testes com diferentes taxas de dropout (sempre 0.5 neste projeto) seria outra coisa a se tentar.

4 CONCLUSÃO

Foi possível entender como é o processo de anotação de dados e como relacionar a estrutura de dados com o "label" das imagens que esses labels representam. Processos de normalização, redimensionamento de imagens e balanceamento do dataset foram outras tarefas aprendidas nesse projeto. Por fim, a montagem de uma rede convolucional, construindo as camadas do modelo sequencial, proporcionou diversos testes diferentes e o entendimento de como os parâmetros configuráveis (padding, kernel e outros) influenciavam o modelo. Durante o treinamento, a criação de callbacks para servir como métrica para o treinamento, bem como o uso de batches, foi outro desenvolvimento interessante.

Finalmente, para melhoria do projeto, a aplicação de métodos sistemáticos de tuning seria uma boa ideia para a escolha dos hiperparâmetros (taxa de aprendizado, número de épocas, tamanho do batch, etc.) e para a escolha de uma arquitetura que se encaixasse melhor com os dados e com o que foi proposto. Além disso, o aumento de dados (data augmentation) pode ser aplicado no projeto para proporcionar maior diversidade de dados.

Referências

[1] TensorFlow Team, *Convolutional Neural Networks Tutorial*, https://www.tensorflow.org/tutorials/images/cnn?hl=pt-br, Acessado em: dez. 2024.

[2] M. Mishra, *Convolutional Neural Networks Explained*, Towards Data Science, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, Acessado em: dez. 2024.

[3] LIMA, Gabriel. *Lidando com desbalanceamento de dados*. Alura, 2020. Disponível em: https://www.alura.com.br/artigos/lidando-com-desbalanceamento-dados?srsltid=AfmBOoqhkHw8wJMfm_H4Fbb01F9bg3xzteVr7fpHAcC7A4IVCIOhG5gI. Acessado em: dez. 2024.

[4] FILHO, Mário. *Precisão, Recall e F1-Score em Machine Learning*. Disponível em: https://mariofilho.com/precisao-recall-e-f1-score-em-machine-learning/. Acessado em: dez. 2024.

[5] DATA CAMP. *AUC: Área sob a Curva*. Disponível em: https://www.datacamp.com/pt/tutorial/auc. Acessado em: dez. 2024.