Estudo de redes residuais para classificação de atividades esportivas

Lucas de Jesus Batista Gonçalves

***Resumo*—Este trabalho explora a modelagem de um modelo de deep learning utilizando redes neurais profundas residuais para a classificação de imagens de esportes, abrangendo 100 classes distintas. As redes residuais, introduzidas por He et al. (2015), permitem a construção de modelos profundos e eficazes, mitigando problemas de degradação do gradiente. A tarefa é desafiadora devido à alta variabilidade das imagens, como diferentes ângulos de câmera e condições de iluminação, porém os resultados demonstram a eficácia do modelo e oferecem insights valiosos para futuras aplicações na classificação de imagens esportivas.**

***Palavras chave*—Classificação, Esportes, ResNet, Inception, Deep Learning, Degradação do gradiente.**

***Abstract*—This work explores the modeling of a deep learning model using residual neural networks for the classification of sports images, covering 100 distinct classes. Residual networks, introduced by He et al. (2015), enable the construction of deep and effective models, mitigating gradient degradation problems. The task is challenging due to the high variability of the images, such as different camera angles and lighting conditions, but the results demonstrate the effectiveness of the model and provide valuable insights for future applications in sports image classification.**

***Index Terms*—Classification, Sports, ResNet, Inception, Deep Learning, Gradient vanishing**

I. Introdução

O avanço da tecnologia e a crescente disponibilidade de grandes volumes de dados têm impulsionado significativamente o desenvolvimento de modelos de aprendizado profundo, especialmente na área de visão computacional. Uma aplicação notável dessa tecnologia é a classificação de imagens, tarefa essencial em diversas áreas, como segurança, saúde e entretenimento. Para este trabalho, estudou-se sobre a classificação de imagens representando diferentes tipos de esportes, valendo-se de redes convolucionais inspiradas em blocos residuais (ResNet) e blocos do tipo Inception (GoogleLeNet).

II. Metodologia

Fundamentalmente, o desenvolvimento deste trabalho se deu em torno de redes neurais residuais (ResNet) [1], bem como indo além e introduzindo blocos Inception (GoogleLeNet) [2]. A escolha da arquitetura foi baseada em um processo de experimentação inicial, considerando a eficácia e a eficiência computacional das abordagens.

Lucas de Jesus é participante do programa de pós graduação em engenharia Elétrica e de Computação da UFG - Universidade Federal de Goiás,e-mail: lucasdejesus@discente.ufg.br .

A implementação e o treinamento dos modelos foram realizados utilizando o framework PyTorch [3], devido à sua flexibilidade e suporte para operações de deep learning. Além do PyTorch, foram empregadas bibliotecas auxiliares como Torchvision, Numpy e Matplotlib para a leitura e manipulação de imagens. O objetivo principal era implementar os algoritmos de aprendizado profundo de maneira direta e eficiente, sem recorrer a ferramentas de alto nível que pudessem abstrair os detalhes da implementação.

Os testes foram conduzidos em um notebooks Jupyter (podem ser encontrados em [4]), em um computador com as seguintes configurações:

- **GPU**: NVIDIA RTX 2060 Super
- **CPU**: Intel i3 de 10ª geração
- **Memória RAM**: 16 GB

III. O conjunto de dados

Os dados utilizados para o treinamento do modelo de aprendizado de máquina desenvolvido consiste em 100 classes de esportes e é dividido em três partes - treino, validação e teste:

- **Treino:** Composto de cerca de 13500 imagens.
- **Validação:** Composto de cerca de 500 imagens.
- **Teste:** Composto de 500 imagens.

De modo a extrair o máximo dos dados disponíveis, um processo de aumentação de dados foi utilizado, incluindo operações como corte aleatório e redimensionamento, espelhamento horizontal, ajuste aleatório de brilho, contraste e saturação, rotação aleatória de até 20 graus e normalização dos valores de pixel para as imagens.

IV. Modelo 1: Blocos Residuais com *Bottleneck*

A implementação deste modelo seguiu a abordagem descrita por He et al. (2016)[1], composta por quatro blocos residuais terminando em uma camada totalmente conectada. Para otimizar o uso de recursos computacionais, optou-se por utilizar a arquitetura com *bottleneck*, conforme recomendado na literatura. Esta abordagem permite reduzir a dimensionalidade das camadas internas, diminuindo o número de parâmetros e, consequentemente, o tempo de treinamento e a memória necessária, sem comprometer o desempenho do modelo.

Os blocos com *bottleneck* são compostos por três camadas: uma camada de convolução 1x1 que reduz a dimensionalidade, seguida por uma camada de convolução 3x3 que processa a informação, e outra camada de convolução 1x1 que restaura a dimensionalidade original (Fig. 1). Essa estrutura é eficiente porque a camada intermediária de convolução 3x3 opera em

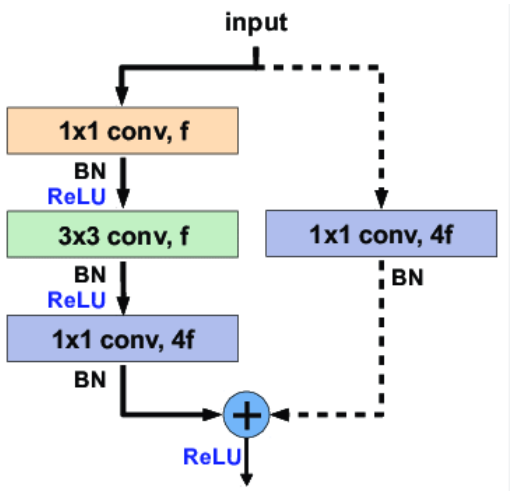

Figura 1: Bloco bottleneck

Tabela I: Estrutura do Modelo 1

Camada	Tamanho do Kernel	Saída
Entrada	-	3 @ 224×224
Conv0	7×7	64 @ 112×112
MaxPool	3×3, stride 2	64 @ 56×56
Bloco 1 (1 vezes)		
Conv1_1	1×1	64 @ 56×56
Conv1_2	3×3	64 @ 56×56
Conv1_3	1×1	256 @ 56×56
Bloco 2 (2 vezes)		
Conv2_1	1×1	128 @ 28×28
Conv2_2	3×3	128 @ 28×28
Conv2_3	1×1	512 @ 28×28
Bloco 3 (3 vezes)		
Conv3_1	1×1	256 @ 14×14
Conv3_2	3×3	256 @ 14×14
Conv3_3	1×1	1024 @ 14×14
Bloco 4 (1 vezes)		
Conv4_1	1×1	512 @ 7×7
Conv4_2	3×3	512 @ 7×7
Conv4_3	1×1	2048 @ 7×7
AvgPool	7×7	2048 @ 1×1
FC	-	100 (classes)

uma dimensão reduzida, economizando cálculos. Cada bloco residual inclui uma conexão de atalho que soma a entrada do bloco à sua saída, facilitando a propagação do gradiente e permitindo a construção de redes mais profundas e eficazes. Em adição também valeu-se do uso de técnicas de regularização por meio de *Batch Normalization* e *Weight Decay* por laço L2.

A. Treinamento

Durante o processo de treinamento, foram implementadas etapas cruciais para garantir a eficácia e estabilidade do modelo de aprendizado de máquina. Inicialmente, adotou-se um procedimento de aumento de dados e balanceamento de classes nos conjuntos de dados utilizados. Esse processo foi fundamental para fornecer uma variedade representativa de exemplos de treinamento e evitar o viés em direção às classes majoritárias.

Devido à versatilidade em algorítimos de aprendizagem de máquina, utilizou-se da função de entropia cruzada como perda, uma vez que fornece uma medida objetiva da discrepância entre as previsões do modelo e os rótulos verdadeiros.

O tamanho do lote (batch) utilizado durante o treinamento foi determinado experimentalmente, com base na capacidade de memória da GPU, garantindo uma utilização eficiente dos recursos computacionais disponíveis e um fluxo de treinamento suave.

Como otimizador, optou-se pelo AdamW, uma variante do algoritmo Adam, disponibilizado pelo PyTorch. Esta escolha se baseou em sua eficiência comprovada e em sua capacidade de mitigar problemas de convergência numérica, aspecto crítico para treinamento eficaz de modelos de deep learning. Embora tenham sido realizados experimentos com outros otimizadores, como Adam, SGD e rmsprop, o AdamW se destacou como a escolha mais estável e eficiente para este contexto.

Para a inicialização dos pesos, inicialmente valeu-se da inicialização Xavier [5], a qual foi posteriormente substituida para a inicialização *He* ou *Kaiming*, a qual possui melhores resultados no uso das funções de ativação retificadoras, tais como ReLU [6][7].

Finalmente, como escalonador de taxa de aprendizado, o escalonador de ciclo único (OneCycleLR) demonstrou ser a melhor escolha uma vez que permite a ocorrência do fenômeno conhecido como super convergência (citar), o qual permite um treinamento mais rápido e eficiente, resultando em modelos com melhor generalização e desempenho superior. Essa abordagem foi crucial para alcançar resultados significativos em um período de tempo reduzido. Temos então:

- **Batch Size**: 128
- **Escalonador de LR**: OneCycleLR
- **Otimizador**: AdamW
- **Taxa de Aprendizado Inicial**: 0.01
- **Função de Loss**: CrossEntropyLoss
- **Weight decay**: Regularização L2 = 0.08
- **Weight initialization**: He/Kaiming
- **Épocas**:60

B. Resultados

Com base no trabalho desenvolvido, obteu-se as seguintes métricas para o modelo treinado:

Tabela II: Loss e Acurácia do Modelo 1 (melhores valores)

Conjunto	Loss	Acurácia
Treinamento	0.0037	0.9995
Validação	0.2906	0.9240
Teste	-	0.9300

A análise das curvas sugere que a instabilidade causada pela subida da taxa de aprendizagem feita pelo escalonador, apesar de subir temporariamente o valor das perdas em torno da época 30, fez com que o modelo possivelmente saisse de um ponto de mínimo local, permitindo uma acurácia de acima de 90%.

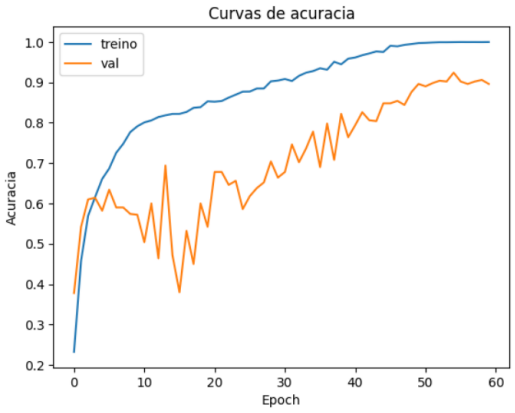

Figura 2: Curva de acurácia - Modelo 1

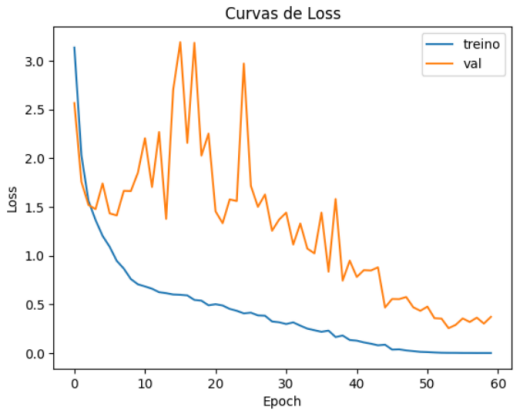

Figura 3: Curva de perdas - Modelo 1

Figura 4: Exemplos de predições - Modelo 1

Esta hipótese é reforçada pelo fato de que em experimentos anteriores, com outros mecanismos de escalonagem, os valores obtidos para a acurácia se limitavam à entorno de 75%.

V. Modelo 2: Adição de bloco GoogleLeNet

Em adição ao modelo envolvendo blocos residuais, também experimentou-se com a inclusão de blocos Inception, introduzidos no modelo GoogleLeNet [2]:

O modelo em especial faz seu uso no lugar do primeiro bloco bloco residual, conforme a Tabela III. Para o restante

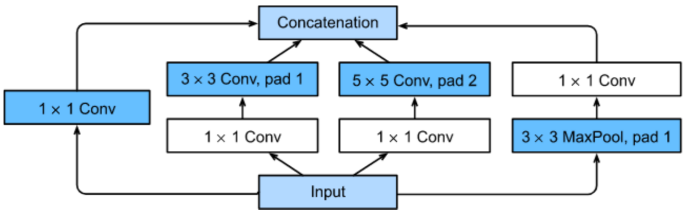

Figura 5: Bloco Inception - GoogleLeNet

Tabela III: Estrutura do Modelo 2

Camada	Tamanho do Kernel	Saída
Entrada	-	3 @ 224 × 224
Conv0	7 × 7	64 @ 112 × 112
MaxPool	3 × 3, stride 2	64 @ 56 × 56
Bloco 1 (Inception)		
convb1_1	1 × 1	64 @ 56 × 56
convb2_1	1 × 1	96 @ 56 × 56
convb2_2	3 × 3	128 @ 56 × 56
convb3_1	1 × 1	16 @ 56 × 56
convb3_2	5 × 5	32 @ 56 × 56
MaxPool	3 × 3	64 @ 56 × 56
convb4_1	1 × 1	32 @ 56 × 56
Bloco 2 (2 vezes)		
Conv2_1	1 × 1	128 @ 28 × 28
Conv2_2	3 × 3	128 @ 28 × 28
Conv2_3	1 × 1	512 @ 28 × 28
Bloco 3 (3 vezes)		
Conv3_1	1 × 1	256 @ 14 × 14
Conv3_2	3 × 3	256 @ 14 × 14
Conv3_3	1 × 1	1024 @ 14 × 14
Bloco 4 (1 vezes)		
Conv4_1	1 × 1	512 @ 7 × 7
Conv4_2	3 × 3	512 @ 7 × 7
Conv4_3	1 × 1	2048 @ 7 × 7
AvgPool	7 × 7	2048 @ 1 × 1
FC	-	100 (classes)

do processo de treino, mantêve-se os mesmos parâmetros empregados para o treino do modelo 1

A. Resultados

Com base no trabalho desenvolvido, obteu-se as seguintes métricas para o modelo treinado:

Tabela IV: Loss e Acurácia do Modelo 2 (melhores valores)

Conjunto	Loss	Acurácia
Treinamento	0.0024	0.9996
Validação	0.3107	0.944
Teste	-	0.9300

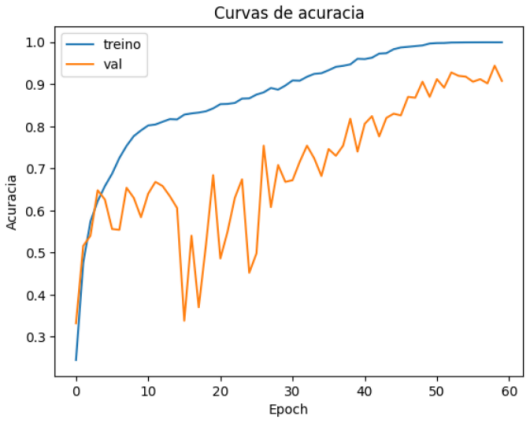

Figura 6: Curva de acurácia - Modelo 2

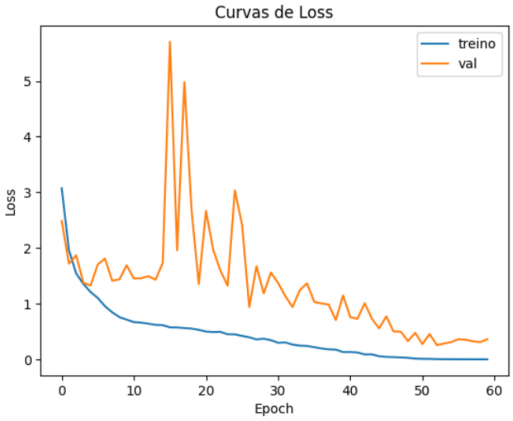

Figura 7: Curva de perdas - Modelo 2

Figura 8: Exemplos de predições - Modelo 2

Pelas análises das curvas do modelo, é possível inferir que ambos os modelos possuem desempenho similar. No entanto, a introdução do bloco Inception reduziu significativamente os picos espúrios e as oscilações nas curvas de acurácia e perdas (Figs. 6 e 7), resultando em um treinamento ligeiramente mais estável e bem comportado.

VI. Considerações finais

Através deste trabalho, foi possível perceber que a tarefa de realizar o treinamento de um modelo de deep learning utilizando redes neurais profundas residuais e atingir uma acurácia acima de 90% para os dados de validação não é simples, dado o dataset apresentado. No entanto, a pesquisa e a implementação de diversas estratégias amplamente utilizadas permitiram atingir os objetivos propostos, demonstrando a eficácia dos métodos empregados. Apesar dos desafios, os resultados obtidos indicam a validade das abordagens utilizadas. Além disso, com ajustes finos nos parâmetros e mais tempo de treinamento, é plausível alcançar resultados ainda melhores, reforçando o potencial dos modelos residuais e dos blocos Inception para a classificação de imagens esportivas.

Referências

[1] K. He, X. Zhang, S. Ren, and J. Sun. (2015) *Deep Residual Learning for Image Recognition*. [Online] Disponível em: https://arxiv.org/abs/1512.03385 . Acesso: 21 mai. 2024.

[2] C. Szeged, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. (2014) *Going Deeper with Convolutions*. [Online] Disponível em: https://arxiv.org/abs/1409.4842 . Acesso: 21 mai. 2024.

[3] The PyTorch Foundation. [Online] Disponível em: https://pytorch.org/ . Acesso: 21 mai. 2024.

[4] *Códigos e recursos*. [Online] Disponível em: https://github.com/Mwar22/RNP-Trabalho1. Acesso: 23 mai. 2024.

[5] X. Glorot and Y. Bengio. (2010) *Understanding the difficulty of training deep feedforward neural networks*. [Online] Disponível em: https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf . Acesso: 21 mai. 2024.

[6] J. Brownlee. *Weight Initialization for Deep Learning Neural Networks*. [Online] Disponível em: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/ . Acesso: 21 mai. 2024.

[7] K. He, X. Zhang, S. Ren, and J. Sun. (2015) *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. [Online] Disponível em: https://arxiv.org/abs/1502.01852 . Acesso: 21 mai. 2024.

Lucas de Jesus Batista Gonçalves Bacharel em Engenharia de Computação na Universidade Federal de Goiás. Jovem tranquilo e dedicado com a centelha da curiosidade. Possui áreas como a engenharia e sistemas computacionais como baricentro de interesse.