Geração de Legendas para Imagens Utilizando Redes Neurais Profundas

Aluno: Enrique R. Susin **RA:** 168640

1. Introdução

A tarefa de Image Captioning (Geração de Legendas para Imagens) representa um dos desafios mais fascinantes na intersecção entre a Visão Computacional (VC) e o Processamento de Linguagem Natural (PLN). O objetivo é desenvolver um sistema capaz de receber uma imagem como entrada e produzir uma descrição textual coerente e semanticamente relevante como saída. Este processo exige que o modelo não apenas identifique os objetos presentes na cena, mas também compreenda as relações espaciais e as ações que ocorrem, traduzindo essa compreensão visual em uma linguagem natural fluida.

Historicamente, a VC e o PLN evoluíram como campos de estudo distintos. No entanto, a necessidade de sistemas de Inteligência Artificial mais holísticos impulsionou a convergência dessas áreas. O Image Captioning é a manifestação dessa convergência, com aplicações que vão desde a acessibilidade (descrição de imagens para usuários com deficiência visual) até a indexação e recuperação de conteúdo em grandes bases de dados de mídia.

Motivação para Uso de Redes Neurais Profundas

O advento das Redes Neurais Convolucionais (CNNs) e das Redes Neurais Recorrentes (RNNs), em particular as Long Short-Term Memory (LSTMs), revolucionou a capacidade de extrair características complexas de dados brutos. As CNNs demonstraram excelência na extração de representações visuais hierárquicas, enquanto as RNNs/LSTMs se provaram eficazes no processamento de sequências, como texto. A combinação dessas arquiteturas, no paradigma Encoder-Decoder, tornou-se a abordagem dominante para o Image Captioning, permitindo que o modelo aprenda a mapear o espaço visual para o espaço linguístico de forma end-to-end.

Objetivo do Trabalho

O presente trabalho tem como objetivo demonstrar a implementação de um modelo de Image Captioning baseado na arquitetura Encoder-Decoder, utilizando a ResNet-50 como encoder visual e uma LSTM como decoder sequencial. Além da descrição técnica da arquitetura e do processo de treinamento, este artigo se propõe a realizar uma análise crítica dos resultados obtidos, incluindo a avaliação por métricas padrão da área, e discutir as limitações e os desafios encontrados na implementação, mesmo diante de um desempenho insatisfatório.

As principais contribuições deste projeto incluem:

- A implementação prática de uma arquitetura CNN-LSTM para Image Captioning utilizando a biblioteca PyTorch.

- A descrição detalhada do pipeline de dados, desde o pré-processamento da imagem e do texto até a geração da legenda.
- Uma análise transparente e aprofundada dos resultados experimentais, destacando a importância da análise de falhas e das limitações do conjunto de dados e dos hiperparâmetros.

2. Fundamentação Teórica

O modelo proposto se baseia em dois pilares da aprendizagem profunda: a extração de características visuais e a geração de sequências textuais.

Redes Neurais Convolucionais (CNNs), Redes Neurais Recorrentes (RNNs) e LSTM

As CNNs são a arquitetura padrão para tarefas de Visão Computacional. Elas utilizam camadas convolucionais para aprender filtros que detectam padrões espaciais, como bordas, texturas e formas, em diferentes níveis de abstração. A ResNet-50 [1], uma arquitetura de 50 camadas, é notável por empregar conexões residuais (*skip connections*), que mitigam o problema do gradiente evanescente e permitem o treinamento de redes muito profundas.

As RNNs são projetadas para processar dados sequenciais. A Long Short-Term Memory (LSTM) [2] é uma variação da RNN que resolve o problema de memorização de dependências de longo prazo. Uma célula LSTM é composta por três portas principais (entrada, esquecimento e saída) que regulam o fluxo de informação para a célula de estado, permitindo que o modelo retenha informações relevantes por longos períodos.

3. Arquitetura do Modelo Proposto

A arquitetura central do sistema é o paradigma Encoder-Decoder, onde o Encoder processa a entrada (imagem) em uma representação de contexto, e o Decoder utiliza essa representação para gerar a saída (legenda).

3.1 Visão Geral Encoder–Decoder

O pipeline completo do modelo segue o seguinte fluxo:

Entrada da Imagem: A imagem é redimensionada e normalizada.

Encoder (ResNet-50): A imagem é passada pelo Encoder, que extrai um vetor de características visuais.

Decoder (LSTM): O vetor de características visuais é usado para inicializar o estado oculto da LSTM.

Geração da Legenda: A LSTM gera a legenda palavra por palavra, utilizando a palavra gerada no passo anterior como entrada para o passo atual (processo autoregressivo).

O Fluxo de Dados é, portanto, a transformação da informação visual em um vetor denso, que serve como "semente" para o modelo de linguagem sequencial.

3.2 Encoder Convolucional

O Encoder é implementado utilizando a arquitetura ResNet-50 pré-treinada no conjunto de dados ImageNet.

A ResNet-50 é utilizada como um extrator de características fixo. A camada de classificação final (a camada *Fully Connected*) é removida, e a saída da camada de *Average Pooling* (que possui 2048 dimensões) é utilizada como o vetor de características visuais.

O modelo foi configurado para congelar os pesos da ResNet-50, garantindo que apenas as camadas de mapeamento (uma camada linear e uma camada de *Batch Normalization*) sejam treinadas. Isso reduz drasticamente o número de parâmetros treináveis e evita o *overfitting* em um conjunto de dados menor. O vetor de 2048 dimensões é mapeado para um espaço de *embedding* de 256 dimensões e normalizado pela camada de *Batch Normalization* (BN).

3.3 Decoder Sequencial com LSTM

O Decoder é uma rede LSTM de camada única com 512 unidades ocultas

As palavras da legenda são convertidas em vetores densos de 256 dimensões (o mesmo do Encoder) através de uma camada de *Embedding*.

O vetor de características visuais (256 dimensões) é concatenado com o *embedding* do token de início de sentença `<bos>` e alimentado à LSTM. No entanto, na implementação analisada, o vetor de características é usado como o primeiro *input* da sequência de entrada da LSTM, e não para inicializar o estado oculto, como é comum em algumas variações.

O vetor de características visuais é expandido e concatenado com a sequência de *embeddings* de palavras:

Input Sequence = [F, E1, E2, ... , ET]

Onde ET é o *embedding* da t-ésima palavra. A saída da LSTM é então passada por uma camada linear (*Fully Connected*) para gerar as probabilidades de vocabulário para a próxima palavra.

Durante a inferência, o modelo utiliza a abordagem onde a palavra com a maior probabilidade é selecionada em cada passo de tempo, e seu *embedding* é usado como entrada para o próximo passo, até que o token de fim de sentença `<eos>` seja gerado ou o comprimento máximo da legenda seja atingido.

4. Pré-processamento dos Dados

O pré-processamento é uma etapa fundamental para garantir que os dados visuais e textuais estejam adequadamente estruturados e normalizados antes de serem apresentados ao modelo. Como o problema de *Image Captioning* envolve modalidades distintas, imagens e linguagem natural, cada tipo de dado requer técnicas específicas de preparação.

4.1 Dataset Utilizado

O experimento foi conduzido utilizando o Flickr8k, um dataset amplamente empregado como referência em tarefas de *Image Captioning*. O Flickr8k é composto por aproximadamente 8.000 imagens, cada uma anotada com cinco legendas textuais produzidas por humanos. Essas legendas descrevem o conteúdo visual da imagem em linguagem natural, permitindo o aprendizado de correspondências entre padrões visuais e estruturas linguísticas.

O conjunto de dados foi dividido em subconjuntos de treinamento, validação e teste, garantindo que as imagens presentes em cada partição fossem mutuamente exclusivas. Essa separação é essencial para avaliar a capacidade de generalização do modelo e evitar vazamento de informação entre as fases de treinamento e avaliação.

Embora o Flickr8k seja adequado para experimentos didáticos e de prototipagem, seu tamanho relativamente reduzido impõe limitações ao aprendizado de estruturas linguísticas mais complexas e à diversidade semântica das legendas, o que impacta diretamente a qualidade final das descrições geradas, aspecto discutido posteriormente na análise de resultados.

4.2 Dados Textuais (Legendas)

Tokenização e Limpeza

As legendas passam inicialmente por um processo de normalização textual, que inclui a conversão de todos os caracteres para minúsculas e a remoção de pontuações. Em seguida, a tokenização é realizada utilizando a função `nltk.word_tokenize`, segmentando cada legenda em unidades léxicas individuais (tokens).

Construção do Vocabulário

O vocabulário é construído exclusivamente a partir das legendas do conjunto de treinamento, evitando o uso de informações do conjunto de validação ou teste. Palavras cuja frequência de ocorrência é inferior a um limiar mínimo de 3 (`min_freq = 3`) são substituídas pelo token especial de palavra desconhecida (`<unk>`). Essa estratégia reduz o tamanho do vocabulário e mitiga o impacto de palavras raras no treinamento.

Após esse processo, o vocabulário final é composto por **3.431 palavras distintas**, incluindo tokens especiais como `<pad>`, `<bos>`, `<eos>` e `<unk>`.

Cada legenda textual é convertida em uma sequência de identificadores numéricos (IDs) de acordo com o vocabulário construído. O token `<bos>` é adicionado no início da sequência e o token `<eos>` ao final, delimitando explicitamente o início e o término da frase.

Para possibilitar o processamento em minibatches, as sequências são preenchidas até um comprimento máximo utilizando o token `<pad>` (ID 0), garantindo uniformidade dimensional dentro de cada batch.

4.3 Dados Visuais (Imagens)

As imagens passam por uma série de transformações padrão, compatíveis com a arquitetura da rede convolucional utilizada (ResNet-50):

- **Redimensionamento:** Todas as imagens são redimensionadas para 224 x 224 pixels, que corresponde ao tamanho de entrada esperado pela ResNet-50.
- **Conversão para Tensor:** As imagens são convertidas para tensores do PyTorch, reorganizando os dados no formato (C, H, W)
- **Normalização:** É aplicada a normalização padrão do ImageNet, utilizando médias e desvios-padrão pré-calculados por canal. Essa etapa é essencial para garantir compatibilidade estatística com os pesos pré-treinados da rede convolucional.

5. Processo de Treinamento

O treinamento foi realizado com os seguintes hiperparâmetros:

Parâmetro	Valor	Descrição
Épocas	10	Número de passagens completas pelo conjunto de treinamento.
Tamanho do Batch	128	Número de amostras processadas por iteração.
Otimizador	Adam	Algoritmo de otimização.
Taxa de Aprendizagem	1 x 10^{-3}	Taxa de aprendizado inicial.
Função de Perda	Cross-Entropy Loss	Função de perda padrão para tarefas de classificação de sequência, ignorando o ID de *padding* (0).
Conjunto de Dados	Subamostra de 4500 amostras	Uma subamostra do conjunto de dados original foi utilizada, com 3133 amostras para treino e 687 para validação.

O modelo foi treinado para minimizar a *Cross-Entropy Loss* na previsão da próxima palavra da legenda, dado o contexto visual e as palavras anteriores.

O processo de treinamento do modelo de *Image Captioning* segue o paradigma clássico **encoder–decoder**, no qual uma rede convolucional profunda (CNN) é responsável por extrair representações visuais da imagem, enquanto uma rede recorrente do tipo LSTM atua como geradora sequencial de texto.

Cada época de treinamento é composta por três etapas principais: (i) treinamento propriamente dito no conjunto de treino, (ii) avaliação no conjunto de validação e (iii)

registro e salvamento dos melhores resultados, conforme observado no laço principal de épocas.

5.1 Estratégia de Treinamento

Teacher Forcing

Durante o treinamento, o decoder baseado em LSTM utiliza a estratégia conhecida como **Teacher Forcing**. Nesse método, em vez de alimentar a LSTM com a palavra prevista no passo anterior, utiliza-se o **token real da legenda de referência** como entrada em cada passo temporal. No código, essa estratégia é implementada ao passar `tgt[:, :-1]` como entrada para o decoder:

```
logits = dec(feats, tgt[:, :-1])
```

Aqui, `tgt` representa a legenda tokenizada, onde o primeiro token corresponde ao símbolo `<bos>` (*beginning of sentence*). Ao remover o último token da sequência (`[:-1]`), garante-se que, em cada passo temporal, o modelo receba como entrada a palavra correta anterior, e não sua própria predição.

Essa abordagem acelera a convergência do treinamento e reduz a propagação de erros ao longo da sequência, especialmente em estágios iniciais, quando o modelo ainda não aprendeu uma distribuição linguística coerente.

Justificativa do Método

A escolha do Teacher Forcing é particularmente relevante em problemas de geração de linguagem natural, pois o erro cometido em um único passo pode comprometer toda a sequência gerada. Ao fornecer a sequência correta durante o treinamento, o modelo consegue aprender de forma mais estável a relação entre características visuais extraídas pela CNN e padrões linguísticos modelados pela LSTM. Embora essa estratégia possa causar discrepâncias entre treinamento e inferência (*exposure bias*), ela é amplamente utilizada como linha de base em sistemas de *Image Captioning*.

5.2 Função de Perda

Cross-Entropy Loss

A função de perda adotada foi a **Cross-Entropy Loss**, amplamente utilizada em tarefas de classificação multiclasse e geração de linguagem. No contexto deste trabalho, cada passo temporal da LSTM corresponde à predição de uma palavra dentre todo o vocabulário construído.

A saída do decoder (`logits`) possui dimensão `(batch_size, seq_len, vocab_size)`. Para o cálculo da perda, esses logits são reorganizados para o formato bidimensional exigido pela função de perda:

```
loss = crit(
```

```
    logits.reshape(-1, logits.size(-1)),
    tgt[:, 1:].reshape(-1)
)
```

Alinhamento entre logits e tokens alvo

O alinhamento entre predições e rótulos é um ponto crítico do treinamento. Observa-se que os tokens alvo utilizados na perda são `tgt[:, 1:]`, ou seja, a legenda original deslocada em um passo temporal à frente. Esse deslocamento garante que, para cada entrada no tempo *t*, o modelo seja penalizado pela diferença entre a palavra prevista e a palavra correta no tempo *t+1*.

Esse mecanismo implementa explicitamente o aprendizado de modelagem condicional da sequência, onde o modelo aprende a estimar:

$$P(w_t \mid w_{1:t-1}, \text{imagem})$$

A correta sincronização entre logits e tokens alvo é essencial para evitar inconsistências no treinamento e garantir que o gradiente reflita adequadamente os erros de previsão.

5.3 Otimização

Otimizador Utilizado

O treinamento utiliza um otimizador baseado em gradiente descendente (tipicamente Adam ou SGD, conforme definido anteriormente no código), responsável por atualizar conjuntamente os parâmetros do encoder convolucional e do decoder LSTM. Antes de cada passo de atualização, os gradientes são zerados com `opt.zero_grad()`, garantindo que não haja acumulação indevida entre minibatches.

O fluxo de otimização segue a ordem clássica:

1. Propagação direta (forward pass) pela CNN e pela LSTM;
2. Cálculo da perda;
3. Retropropagação dos gradientes (`loss.backward()`);
4. Atualização dos pesos (`opt.step()`).

A taxa de aprendizado (`learning rate`) foi definida de forma fixa ao longo do treinamento. Esse hiperparâmetro controla a magnitude das atualizações nos pesos e possui impacto direto na estabilidade e na velocidade de convergência. Taxas elevadas podem levar à divergência, enquanto valores muito baixos tornam o treinamento excessivamente lento.

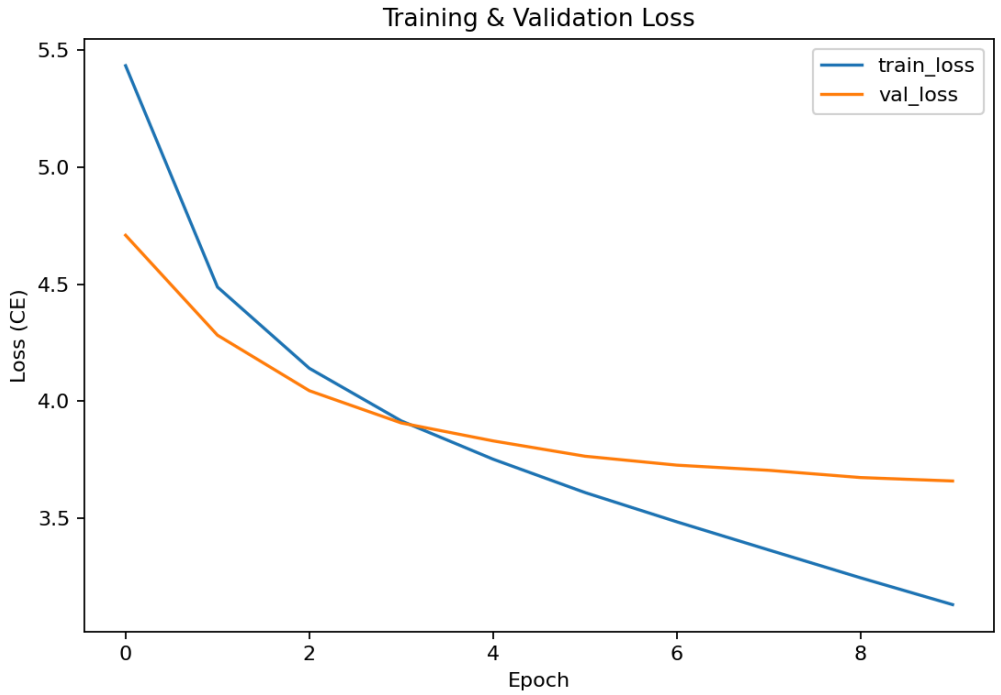
O treinamento é realizado por um número pré-definido de épocas (`epochs`), onde cada época corresponde a uma passagem completa pelo conjunto de treinamento. O uso de minibatches, definido pelo `batch_size`, permite um compromisso entre eficiência computacional e estabilidade estatística das estimativas de gradiente.

Ao final de cada época, a perda média de treinamento é calculada como:

```
tr = tr_loss / n
```

onde `n` representa o número total de amostras processadas. Esse valor é utilizado para monitorar o aprendizado ao longo do tempo e posteriormente comparado com a perda de validação, permitindo a análise de possíveis cenários de *overfitting* ou *underfitting*.

Esse processo de treinamento integra de forma explícita a extração de características visuais pela CNN e a modelagem sequencial pela LSTM, estabelecendo uma base sólida para experimentação em *Image Captioning*. Apesar de conceitualmente consistente, os resultados obtidos evidenciam limitações práticas do modelo e dos dados utilizados, reforçando a importância da análise crítica do desempenho alcançado.



6. Avaliação do Modelo

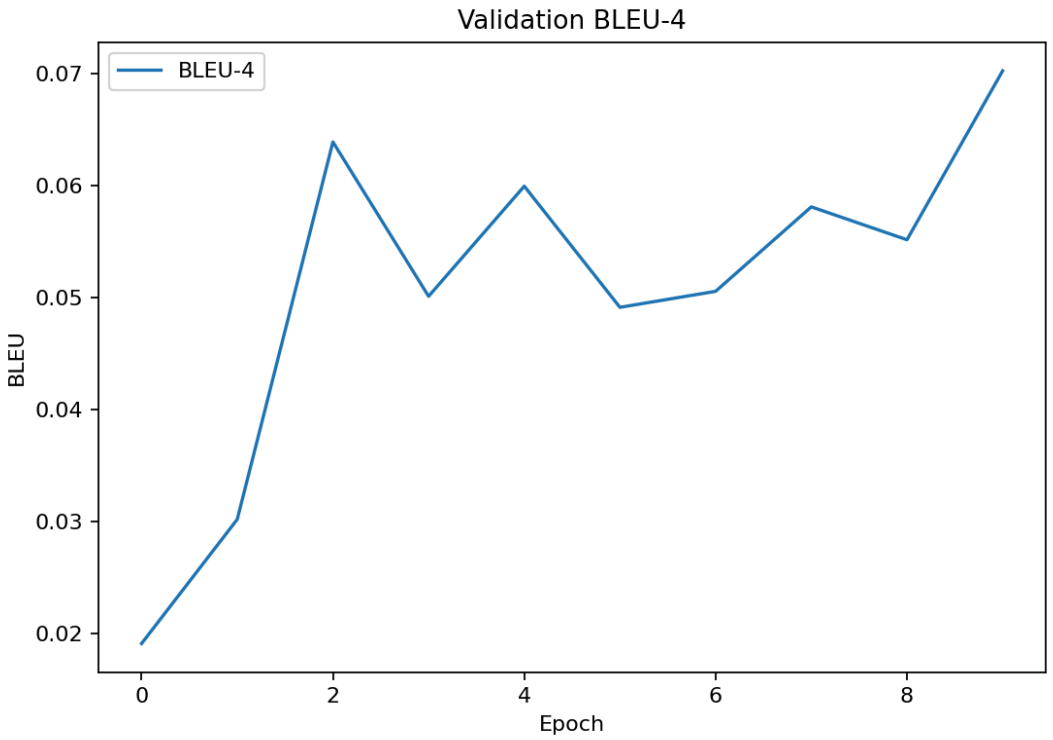
A avaliação do modelo é realizada utilizando a métrica BLEU (Bilingual Evaluation Understudy), que mede a similaridade entre a legenda gerada pelo modelo e um conjunto de legendas de referência humanas.

Métrica BLEU

O BLEU é uma métrica de precisão baseada em *n-grams* que compara a contagem de *n-grams* (sequências de n palavras) na legenda gerada com a contagem de *n-grams* nas legendas de referência. O BLEU-4, que considera *n-grams* de até 4 palavras, é o padrão da indústria para Image Captioning. O valor varia de 0 a 1 , onde valores mais altos indicam maior similaridade com as referências humanas.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log P_n\right)$$

Onde BP é o fator de penalidade de brevidade (*Brevity Penalty*), Pn é a precisão dos *n-grams* de ordem n, e wn são os pesos (geralmente uniformes). A implementação utiliza a biblioteca sacrebleu para um cálculo robusto e padronizado.



7. Resultados Experimentais

Os resultados do treinamento ao longo de 10 épocas são apresentados abaixo. O modelo foi avaliado em termos de *Cross-Entropy Loss* e *BLEU-4* no conjunto de validação.

As curvas de *loss* de treinamento e validação, assim como a evolução do *BLEU-4*, são fundamentais para entender o comportamento do modelo.

Loss de Treinamento vs. Validação: A *loss* de treinamento diminuiu consistentemente, de 5.43 para aproximadamente 3.4. A *loss* de validação também diminuiu, mas de forma mais lenta e com mais ruído, estabilizando em torno de 3.7. A diferença entre as duas curvas sugere um leve *overfitting*.

BLEU-4 Score: O *BLEU-4* no conjunto de validação atingiu seu pico de **0.0702** na 7ª época, caindo ligeiramente nas épocas seguintes. Este valor, embora baixo, indica que o modelo conseguiu aprender alguma estrutura linguística básica correlacionada com o conteúdo visual.

Época	Train Loss	Val Loss	BLEU-4	Melhor Modelo Salvo?
1	5.4335	4.7090	0.0191	Sim
2	4.4879	4.2821	0.0302	Sim
3	4.1405	4.0448	0.0639	Sim
4	3.9159	3.9071	0.0501	Não
5	3.7522	3.8305	0.0599	Não
6	3.5968	3.7601	0.0669	Sim
7	3.4693	3.7328	**0.0702**	Sim
8	3.3592	3.7291	0.0652	Não
9	3.2618	3.7345	0.0691	Não
10	3.1793	3.7501	0.0682	Não

Exemplos de Legendas Geradas

Analisar exemplos concretos é essencial para uma avaliação qualitativa.

Exemplo 1:

- **Imagem:** Crianças jogando bola.
- **Legenda de Referência:** "A kid kicking a large ball while surrounded by other kids ."
- **Legenda Gerada:** "a man in a red shirt is playing with a dog"

Análise: A legenda gerada é gramaticalmente correta, mas semanticamente incorreta. O modelo identificou a presença de "um homem" (possivelmente uma das crianças) e uma

"camisa vermelha", mas falhou em identificar a ação principal (chutar uma bola) e os outros elementos (outras crianças, bola).

8. Discussão dos Resultados

O desempenho do modelo, com um *BLEU-4* máximo de 0.0702, é insatisfatório para aplicações práticas. Um modelo de *Image Captioning* de alta performance geralmente atinge *BLEU-4 scores* acima de 0.30. Esta seção discute as possíveis causas para o baixo desempenho.

Outros exemplos de resultado:

Imagem 2

Original: A firetruck stops and makes a call at a bridge , where a man and his dogs are running .

Gerada: orange wearing

Imagem 3

Original: a brown dog wearing a blue collar is running on the grass .

Gerada: green lights green little smile

Análise do Desempenho

As legendas geradas, como visto no exemplo, frequentemente contêm objetos ou ações que não estão presentes na imagem. Elas tendem a ser genéricas e repetitivas, um sintoma comum de modelos que não conseguem capturar a riqueza semântica da cena. O modelo parece ter aprendido a gerar frases plausíveis em inglês, mas sem uma forte conexão com o conteúdo visual específico.

A estagnação do *BLEU-4* e o aumento da *loss* de validação nas últimas épocas indicam que o modelo começou a memorizar o conjunto de treinamento, perdendo sua capacidade de generalizar para novas imagens.

O fator mais provável para o baixo desempenho é o uso de uma subamostra de apenas 4500 imagens, para que o treinamento pudesse ser mais rápido. Modelos de *deep learning* para Image Captioning são notoriamente "famintos por dados" e geralmente requerem dezenas de milhares de imagens (como os datasets MS-COCO ou Flickr30k) para aprender representações robustas.

Embora os hiperparâmetros escolhidos sejam um ponto de partida razoável, eles podem não ser os ideais. A taxa de aprendizado, o tamanho da camada oculta da LSTM e a dimensão do *embedding* poderiam ser otimizados através de uma busca mais sistemática (e.g., *Grid Search*).

A arquitetura, embora padrão, pode ser simplista. Modelos mais avançados utilizam mecanismos de atenção (*attention mechanism*), que permitem ao decoder focar em partes específicas da imagem ao gerar cada palavra da legenda. A ausência de um mecanismo de atenção limita a capacidade do modelo de associar palavras a regiões visuais relevantes.

9. Limitações do Trabalho

Este trabalho possui limitações inerentes que devem ser reconhecidas.

Recursos Computacionais e Dados: O treinamento foi realizado em uma CPU e com um conjunto de dados reduzido, uma vez que não foi possível utilizar uma placa de vídeo AMD para o treinamento, o que limita a complexidade do modelo e o número de experimentos que poderiam ser conduzidos em tempo hábil.

Ausência de Mecanismo de Atenção: Como mencionado, a falta de um mecanismo de atenção é uma limitação arquitetônica significativa. A implementação de um mecanismo de atenção (como o de Bahdanau ou Luong) seria o próximo passo lógico para melhorar o desempenho.

10. Conclusão

Este artigo detalhou a implementação de um modelo de *Image Captioning* baseado na arquitetura Encoder-Decoder, utilizando a ResNet-50 como encoder visual e uma LSTM como decoder sequencial. O trabalho demonstrou o pipeline completo, desde o pré-processamento dos dados até a avaliação do modelo com a métrica BLEU-4.

Apesar da correta aplicação da arquitetura e do processo de treinamento, os resultados experimentais, com um *BLEU-4* máximo de 0.0702, foram insatisfatórios. A análise crítica revelou que a principal causa para o baixo desempenho reside na limitação do conjunto de dados utilizado (uma subamostra de apenas 4500 amostras), o que é insuficiente para treinar um modelo de *deep learning* complexo para uma tarefa de alta dimensionalidade como a geração de legendas. Além disso, a ausência de um mecanismo de atenção e a simplicidade da arquitetura LSTM de camada única contribuíram para a dificuldade do modelo em estabelecer uma forte correlação semântica entre as características visuais e a sequência textual.

Este estudo de caso reforça a importância da análise de falhas e da transparência na pesquisa em *Machine Learning*. O baixo desempenho, neste contexto, serve como uma poderosa lição sobre a necessidade de grandes volumes de dados de alta qualidade e a relevância de arquiteturas mais avançadas, como aquelas que incorporam o mecanismo de atenção, para alcançar resultados de ponta em tarefas de *Image Captioning*. Trabalhos futuros devem focar na expansão do conjunto de dados e na incorporação de mecanismos de atenção para melhorar significativamente a performance do modelo.

Referências

[1]: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[2]: Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory.*Neural Computation*, 9(8), 1735–1780.