PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Sistema De Busca Reversa De Imagens Utilizando Aprendizado Por Contraste

Lucca Buffara de Almeida

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Ciência da Computação

Rio de Janeiro, junho de 2022

Pontifícia Universidade Católica
do Rio de Janeiro

Lucca Buffara de Almeida

Sistema De Busca Reversa De Imagens Utilizando Aprendizado Por Contraste

Projeto Final de Graduação

Relatório de Projeto Final, apresentado ao programa Ciência da Computação da PUC-Rio como requisito parcial para a obtenção do titulo de Bacharel em Ciência da Computação.

Orientador : Felipe Jordão Pinheiro de Andrade
Coorientador: Prof. Marcelo Gattass

Rio de Janeiro
junho de 2022

Resumo

Almeida, Lucca Buffara de; Andrade, Felipe Jordão Pinheiro de; Gattass, Marcelo. **Sistema De Busca Reversa De Imagens Utilizando Aprendizado Por Contraste**. Rio de Janeiro, 2022. p. Relatório Final de Projeto Final II – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este projeto final apresenta uma metodologia de aprendizado por contraste para realizar a comparação de similaridade entre imagens. Este projeto utiliza redes siamesas com triplet loss e apresenta o resultado de treinamentos realizados na base CIFAR-10. Além de apresentar a metodologia e os resultados do treinamento, este projeto apresenta uma arquitetura web para servir o algoritmo desenvolvido como um sistema de busca por imagens no ambiente de computação em nuvem AWS.

Palavras-chave

Aprendizados por contraste; Redes siamesas; Triplet Loss; AWS; Busca por imagens;.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

IA – Inteligência Artificial

kNN – *k-nearest neighbors* (K-ésimo vizinhos mais próximos)

ML – Machine Learning (Aprendizado de Máquina)

URL – *Uniform Resource Locator* (Localizador Uniforme de Recursos)

API – *Application Programming Interface* (Interface de Programação de Aplicação)

HTML – *HyperText Markup Language* (Linguagem de Marcação de Hipertexto)

CSS – *Cascading Style Sheets* (Folha de Estilo em Cascatas)

HTTP – *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)

JSON – JavaScript Object Notation (Notação de Objetos JavaScript)

1
Introdução

Nos últimos anos, a disponibilidade de grandes conjuntos de dados, combinada com crescimento de capacidade computacional e o aprimoramento de algoritmos de IA, levou a um aumento incomparável de interesse no campo de aprendizado profundo (Pugliese et al., 2021). Os avanços de algoritmos de aprendizado profundo permitiu que novas aplicações fossem desenvolvidas pela indústria e que novas soluções fossem introduzidas a problemas antigos.

Um dos campos de pesquisa mais afetados com o avanço destes algoritmos foi a área de visão computacional (Kühl et al., 2020). Esta área de pesquisa foi revolucionada quando redes profundas começaram a ser utilizadas para reduzir significativamente erros em tarefas de análise de imagens, como por exemplo classificação. Um exemplo conhecido foi quando, Krizhevsky et al. (2012), utilizaram redes convolucionais para reduzir o erro no desafio de classificação de imagens, ImageNet (Russakovsky et al., 2015) em mais de 10 pontos percentuais. Em 2011 o erro top-5 neste desafio era de 26%, em 2012 com a introdução da rede convolucional AlexNet, este erro foi reduzido para 15.3%. Avanços nessa área também foram aplicados para desenvolver algoritmos que muitas vezes ultrapassam a performance humana. Alguns exemplos de aplicações em que estes algoritmos se mostraram melhores que humanos incluem: reconhecimento facial (Schroff et al., 2015), classificação de câncer de pele (Esteva et al., 2017) e reconhecimento de voz (Xiong et al., 2018).

Diversos sistemas que requerem análises sobre imagens vem adotando redes convolucionais como o ponto principal de seus algoritmos. Um tipo de sistema que esse projeto irá abordar são sistemas de busca reversa de imagens.

Este projeto utiliza o método de aprendizado por contraste para realizar a comparação de similaridade entre imagens. Além do desenvolvimento desta metodologia, este trabalho propõe uma arquitetura web para servir um sistema que utiliza essa metodologia.

2
Situação atual

Busca reversa de imagens é a tarefa de buscar por imagens em uma base, dada uma imagem como parâmetro de busca. Além de ser um campo de muito interesse do mundo acadêmico, busca reversa por imagens é amplamente utilizado na indústria. Sistemas de busca como Google, Yandex e Bing oferecem a funcionalidade de realizar buscas utilizando uma imagem ao invés de linguagem natural. Existem também diversas empresas que têm como foco principal o desenvolvimento desses sistemas. Por exemplo, o produto principal da empresa Lykdat, é um sistema de busca por imagens para artigos de vestuário.

Na literatura, a tarefa de buscar por imagens similares em uma base é conhecida por *image retrieval*. Após a popularização de redes convolucionais profundas, progressos surpreendentes vem acontecendo na forma em que podemos gerar representações de imagens. Antes da utilização de redes profundas, algoritmos como SIFT (Lowe, 2004) e BoW (Sivic e Zisserman, 2003) dominavam a forma de gerar representações úteis de imagens para diferentes tarefas. Ao longo dos anos esses algoritmos foram sendo substituídos por características extraídas por redes profundas, que são capazes de aprender representações de imagens diretamente dos dados. Essa representação gerada por modelos profundos e comumente chamada de *embedding*.

Diversos outros campos de visão computacional tem objetivos semelhantes a *image retrieval*, um desses campos é o de reconhecimento facial. Assim como em busca reversa de imagens, sistemas de reconhecimento facial recebem uma imagem de um rosto humano e buscam rostos similares em uma base. Uma abordagem que tem sido utilizada com sucesso para reconhecimento facial é o treinamento de redes siamesas por aprendizado por contraste (Wu et al., 2017), (Salomon et al., 2020).

Para este projeto foi decidido utilizar esta técnica de aprendizado por contraste para treinar uma rede siamesa capaz de medir a similaridade entre imagens na base de dados CIFAR-10, uma base amplamente utilizada na literatura. Para a arquitetura das redes siamesas foi decidido utilizar as redes residuais por também serem amplamente estudadas. Após o treinamento deste modelo foi implementado um sistema de busca reversa de imagens que utiliza o modelo siamês.

3
Objetivo

Este projeto tem como objetivo o treinamento de um modelo de rede neural siamesa para desenvolver um sistema de busca reversa de imagens sobre a base de dados CIFAR-10. O sistema ao receber uma imagem de uma das classes do CIFAR-10, retorna as 10 imagens que o algoritmo desenvolvido julgar as mais semelhantes. Deve ser desenvolvida uma interface web para interagir com o sistema.

Para realizar o treinamento da rede siamesa foi decidido utilizar a função de custo triplet loss, uma técnica de aprendizado por contraste. Após ter o modelo treinado foi utilizado o ambiente computacional, AWS (*Amazon Web Services*) para desenvolver um sistema que utiliza o modelo e realiza buscas por imagens sobre a base CIFAR-10. O sistema desenvolvido oferece uma interface web para realizar as buscas.

Para alcançar este objetivo foram realizadas as seguintes tarefas:

- Estudar sobre aprendizado por contraste, modelos siameses e triplet loss.
- Realizar treinamentos do modelo siamês na base CIFAR-10 a fim de escolher o melhor modelo para ser utilizado em um sistema de busca reversa de imagens.
- Estudar sobre tecnologias em nuvem.
- Implementar um sistema que realiza busca reversa de imagens sobre o CIFAR-10.

4
Treinamento das redes siamesas

Esta seção irá apresentar os diferentes estudos realizados e os resultados obtidos nos treinamentos realizados.

4.1
Estudos conceituais

Para implementar e treinar o algoritmo de similaridade que será utilizado no sistema foi necessário estudar sobre os tópicos que serão apresentados nesta seção. Além de estudar sobre as tecnicas necessárias para a implementacao do modelo foi necessario estudar como utilizar o modelo para a comparação entre imagens.

4.1.1
Aprendizado por contraste

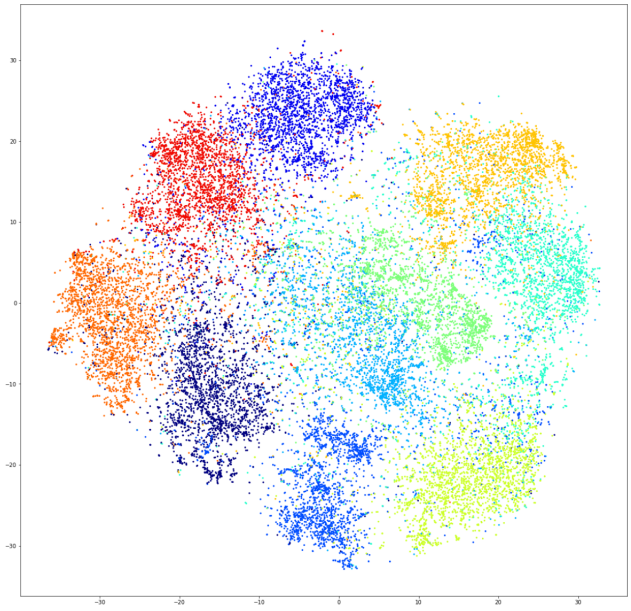

Figura 4.1: Exemplo de visualização de embeddings 2D CIFAR-10 (Cook, 2017)

Para implementar o algoritmo de similaridade deste projeto, foi escolhido utilizar aprendizado por contraste para guiar o treinamento do modelo desenvolvido.

Aprendizado por contraste é uma técnica de IA cujo objetivo é aprender um espaço vetorial em que amostras similares estejam próximas uns dos outros e amostras dissimilares estejam distantes. Aprendizado por contraste pode ser utilizado tanto em aplicações de aprendizado supervisionado quanto não supervisionado e atualmente é uma das técnicas mais poderosas em aplicações auto-supervisionadas (Weng, 2021).

Na prática em aprendizado por contraste treinamos um modelo de IA para realizar um mapeamento de um certo dado X a um espaço vetorial $\mathbb{R}^d$, com o objetivo de que dados da mesma classe estejam próximos uns dos outros neste espaço.

Para conseguir guiar os modelos a gerar esse mapeamento, diversas funções de custo foram desenvolvidas ao longo dos anos. Uma das primeiras funções de custo introduzidas foi a contrastive loss, Hadsell et al. (2006). Esta função recebe como entrada apenas duas amostras (x_i, x_j) e minimiza a distância quando as amostras são da mesma classe e maximiza quando são de classes diferentes.

Versões mais modernas de funções de custo para aprendizado por contraste tendem a incluir múltiplos pares positivos e negativos. A função de custo que foi utilizada para esse projeto, a triplet loss, recebe um par de amostras da mesma classe e uma amostra de uma outra classe para seu cálculo. A triplet loss será apresentada em detalhe mais à frente neste capítulo.

A Figura 4.1 ilustra um exemplo de um modelo treinado na base de dados CIFAR-10 cujas imagens foram mapeadas para um espaço de duas dimensões utilizando aprendizado por contraste. Cada cor representa uma das 10 classes presentes nesta base. Pode se observar que clusters são formados para cada classe da base, já que durante o treinamento vetores da mesma classe foram sendo aproximados.

4.1.2 Redes neurais convolucionais siamesas

Desde sua primeira demonstração prática para reconhecimento de dígitos manuscritos por LeCun et al. (1998), redes neurais convolucionais tem se tornado a mais importante classe de redes neurais no campo da inteligência artificial. Diferente de algoritmos tradicionais de visão computacional em que características visuais de imagens devem ser determinadas manualmente, redes convolucionais conseguem otimizar filtros para a extração dessas características de uma forma automática. Desde sua proliferação essas redes têm sido utilizadas para praticamente qualquer tarefa em que é necessário realizar alguma análise sobre características de imagens.

Uma arquitetura popular para aprendizado por contraste são redes neurais convolucionais siamesas primeiramente utilizadas na tarefa de verificação de assinaturas, Bromley et al. (1993). No campo de aprendizado profundo redes siamesas são uma classe de redes neurais que contém uma ou mais sub-redes idênticas. As redes são ditas idênticas pois elas têm a mesma arquitetura e compartilham seus parâmetros e pesos. As atualizações dos pesos durante o treinamento ocorre para todas as sub-redes. Redes siamesas recebem um tensor de entrada e geram um embedding como sua saída. Para realizar o treinamento dessas redes podemos usar uma função de custo como triplet loss ou contrastive loss para guiar o treinamento da rede para gerar vetores de saída que estejam mais próximos uns dos outros para a mesma classe e mais distantes para classes diferentes. Diferente de redes treinadas para classificação, o objetivo do treinamento dessas redes é gerar uma função de similaridade que calcula o quão similar às duas entradas são. Na prática, após seu treinamento podemos utilizar apenas uma das sub-redes para gerar nosso vetor de saída e realizar o cálculo de similaridade baseado na distância de dois vetores de saída. Redes siamesas são utilizadas em diversas aplicações como reconhecimento facial, identificação de pessoas, reconhecimento de caligrafia e qualquer outra tarefa em que é necessário ter uma função para medir similaridade.

4.1.3 Redes residuais

Para o desenvolvimento do modelo siamês foi decidido utilizar as redes residuais como o backbone do modelo. O backbone do modelo nada mais é que a arquitetura do extrator de características da rede neural. Para este projeto foi decidido utilizar as redes residuais já que elas foram desenvolvidas para tentar diminuir o problema de degradação encontrado durante o treinamento de redes muito profundas.

Como foi observado por Srivastava et al. (2015) treinamentos de redes muito profundas acabam apresentando um problema de degradação. Com o aumento de camadas na arquitetura a precisão da rede neural satura e rapidamente diminui.

A Figura 4.2 ilustra este fenômeno de degradação. Utilizando uma rede com 56 camadas foi obtido um erro maior do que ao utilizar uma rede com 20 camadas, tanto no conjunto de treinamento quanto no conjunto de testes.

Para tentar solucionar o problema de degradação em redes mais profundas, He et al. (2016), introduziram a arquitetura de redes residuais (ResNet). Redes residuais possuem o que são chamados de blocos residuais. Em redes neurais tradicionais, cada camada está diretamente conectada à próxima. Em

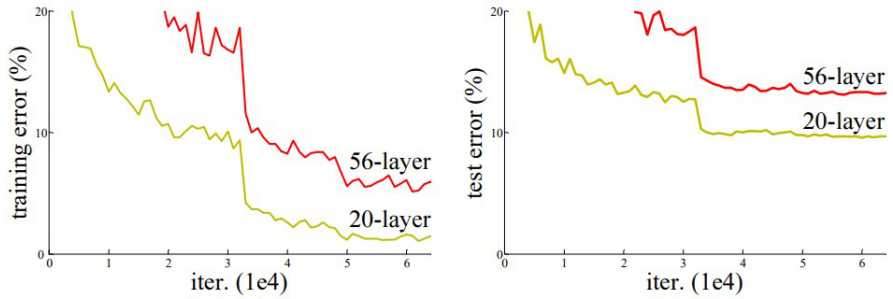

Figura 4.2: Erro de treinamento e teste para classificação na base CIFAR-10. He et al. (2016)

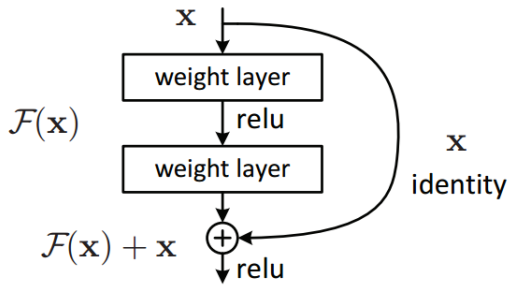

Figura 4.3: Bloco residual. He et al. (2016)

uma rede com blocos residuais, cada camada está conectada a próxima e também diretamente a camadas duas ou três posições à frente.

Para entender a decisão por trás da criação de um bloco residual podemos considerar: que tendo uma rede com n camadas, que produz um certo erro para um problema e uma outra rede com m camadas, sendo $m > n$. Considerando que a segunda rede tenha suas primeiras n camadas iguais a primeira rede treinada, intuitivamente esperamos que esta rede performe pelo menos tão bem quanto a com menos camadas. Se a rede com n camadas gera o melhor erro possível para a solução, as camadas restantes $m-n$ devem apenas repassar esses valores adiante, aprendendo uma função de identidade. Porém, experimentos realizados mostram que aprender esse mapeamento não é algo trivial.

Em vez de esperar que as camadas da rede aprendam um certo mapeamento $H(x)$, os autores explicitamente guiam a rede a aprender um mapeamento residual, $F(x) = H(x) - x$. O mapeamento original então fica reformulado para: $H(x) := F(x) + x$. De acordo com a hipótese dos autores, seria mais fácil otimizar este mapeamento residual que o original. Em casos extremos em que um mapeamento de identidade é ótimo para a solução, a rede precisaria apenas aprender a zerar o resíduo $F(x)$. Experimentos realizados pelos autores mostram que de fato a utilização destes blocos residuais permitem que modelos mais profundos sejam treinados sem o problema de degradação. Realizando

o empilhamento destes blocos residuais os autores introduziram a família de redes ResNets. Redes residuais são atualmente uma das mais populares arquiteturas para redes profundas (Khan et al., 2020).

Para este projeto os modelos siameses utilizaram como backbone as ResNets com 18, 34 e 50 camadas.

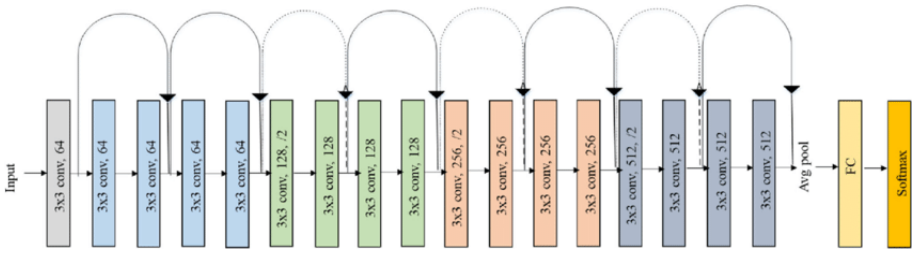

Figura 4.4: Arquitetura ResNet18. Ramzan et al. (2019)

4.1.4 Triplet loss

Para realizar o treinamento do modelo siamês para esse projeto foi escolhido utilizar a função de custo triplet loss. Esta função ganhou popularidade depois de ter sido introduzida por Schroff et al. (2015) foi utilizada para atingir o estado da arte da época na tarefa de reconhecimento facial. Em paralelo com o que este projeto busca alcançar, a tarefa de reconhecimento facial também procura encontrar a similaridade entre imagens, porém no caso de faces humanas. Após sua divulgação ela foi amplamente utilizada em outras tarefas cujo objetivo seja encontrar similaridades entre dados, Kha Vu (2021). A triplet loss é dada pela seguinte fórmula:

$$\sum_{i=1}^{N}[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha]$$

- Os sobrescritos a, p e n correspondem às imagens âncora, positiva e negativa respectivamente.
- $f(x)$ recebe uma imagem de entrada e retorna um embedding.
- O último termo α é uma margem entre pares positivos e negativos. Este valor não é otimizado durante o treinamento e deve ser configurado no inicio do treinamento.

Durante o treinamento do modelo temos como objetivo minimizar a fórmula acima. Dada uma imagem âncora a, uma outra imagem da mesma classe p e uma imagem de outra classe n, buscamos minimizar a distância euclidiana entre a e p em quanto maximizamos a distância euclidiana entre a e

n. Assim, treinamos o algoritmo para gerar embeddings de imagens da mesma classe mais próximos uns dos outros e de classes diferentes mais distantes. Tendo o modelo treinado podemos utilizar a distância euclidiana entre os embeddings gerados como uma métrica similaridade.

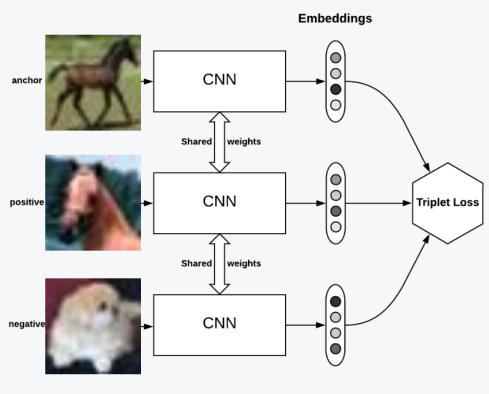

Figura 4.5: Redes siamesas com triplet loss

4.1.5 Vizinhos mais proximos

Depois de ter um modelo siamês treinado para gerar um mapeamento de uma imagem para um espaço vetorial, é necessário um algoritmo que retorne os embeddings mais próximos uns dos outros. Para a implementação deste projeto foi utilizado o algoritmo K-ésimo vizinhos mais próximos (kNN).

K-ésimo vizinho mais próximo, primeiramente desenvolvido por Fix e Hodges (1989), é um algoritmo de aprendizado não paramétrico amplamente utilizado para classificação e regressão. O algoritmo recebe uma entrada e retorna os k elementos mais próximos presentes na base de treinamento. Para classificação após se ter os k elementos mais próximos é realizada uma votação entre eles para classificar a entrada. O kNN não é um algoritmo de aprendizado ou seja precisamos apenas criar uma instância do algoritmo passando a base em que se deseja calcular os vizinhos mais próximos. Todas as entradas serão utilizadas para inferência.

Para este projeto, cada imagem presente no conjunto de testes da base de imagens será primeiramente processada por um modelo siamês, para se obter o embedding daquela imagem. Após esta etapa os embeddings extraídos serão utilizados para instanciar o algoritmo de kNN. O kNN inicializado então será utilizado para achar os 10 embeddings mais proximos ao embedding da imagem recebida pelo sistema. A distância euclidiana será utilizada como métrica de proximidade para o kNN.

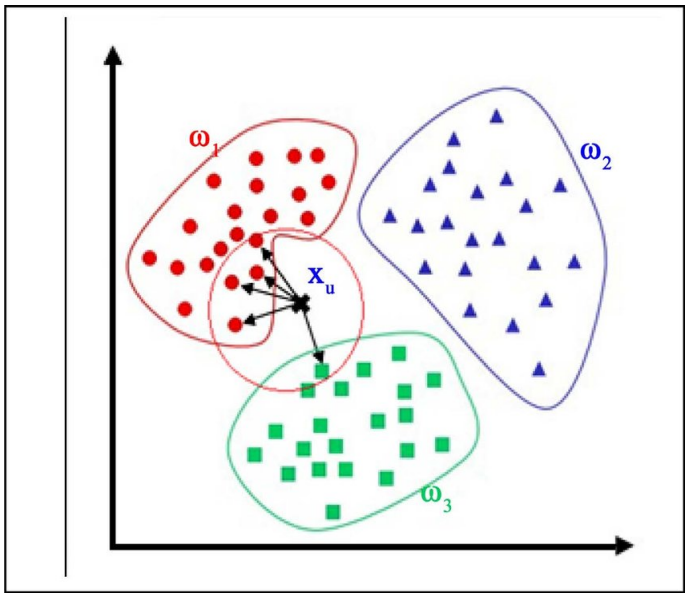

Figura 4.6: kNN. Zhang (2017)

4.2 Treinamento

Nesta seção será apresentado a base de dados utilizada para o treinamento, a metodologia de treinamento dos modelos siameses, os diferentes parâmetros que foram variados e os resultados obtidos.

4.2.1 CIFAR-10

Para o desenvolvimento do projeto foi utilizada a base de dados CIFAR-10 (Krizhevsky et al., 2009) como a base de treinamento do modelo siamês e do sistema. O CIFAR-10 é uma base bem estabelecida no campo de pesquisa de redes neurais profundas. A base é composta de 60.000 imagens. 50.000 imagens fazem parte de seu conjunto de treinamento e 10.000 do conjunto de testes. A base contém 10 classes distintas sendo elas: avião, automóvel, pássaro, gato, veado, cachorro, sapo, cavalo, barco e caminhão. Todas as imagens tem um tamanho pequeno de 32x32 pixels. Foi escolhida desenvolver o sistema utilizando essa base de dados pois ela é amplamente utilizada em pesquisa e suas pequenas imagens permitiram que diversas iterações de treinamento/experimentos fossem realizadas.

4.2.2 Metodologia de treinamento

Utilizando o framework de aprendizado de máquina, PyTorch foi implementado um script de treinamento do modelo siamês.

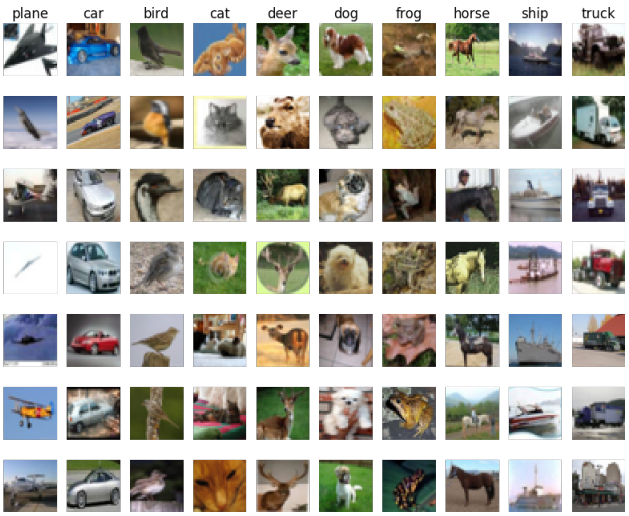

Figura 4.7: Classes na base CIFAR-10

Foram feitos diversos treinamentos utilizando a base de dados CIFAR-10 para estudar o melhor modelo para a solução. Como métrica de performance default do modelo foi escolhido utilizar a precisão Top-1, ou seja utilizando o conjunto de testes iremos medir a proporção de casos de testes em que o vetor mais próximo seja da mesma classe. Como a base CIFAR-10 contém imagens pequenas de apenas 32x32 pixels, foi possível iterar por diferentes configurações do modelo. Cada modelo treinado foi inicializado com pesos de redes pré-treinadas no dataset ImageNet (Russakovsky et al., 2015). Pesquisas mostram que inicializar modelos profundos com pesos pré-treinados em datasets grandes tendem a acelerar a convergência do treinamento, Khan et al. (2020). O único pré-processamento realizado sobre as imagens foi a normalização dos valores dos pixels das imagens, utilizando o valor de 0,5 para a média e desvio padrão. Em todos os experimentos o tamanho do batch ficou fixo em 128 imagens, por limitações de memória. Todos os treinamentos foram realizados em 100 épocas utilizando uma taxa de aprendizado inicial de 0,1 sendo multiplicada por 0,1 a cada 10 épocas. Para o otimizador foi escolhido utilizar o algoritmo do gradiente descendente estocástico com momento 0,9 e decaimento de 5e-4. Durante os experimentos também foi variada a dimensão do embedding de saída do modelo. Foram utilizadas as dimensões 64, 128 e 512. Todos os treinamentos foram realizados utilizando uma GPU RTX 2080 Ti em uma máquina local.

4.2.3 Resultados obtidos

Backbone	**Dimensão do embedding**	α **= 0.1**	α **= 0.3**	α **= 0.5**
ResNet18	64	86.49%	**86.84%**	85.68%
	128	87.09%	86.19%	**86.25%**
	512	**87.47%**	86.61%	83.42%
ResNet34	64	86.64%	86.51%	85.1%
	128	87.31%	86.08%	**85.76%**
	512	**87.49%**	**86.97%**	46.88%
ResNet50	64	85.99%	**86.13%**	79.35%
	128	86.74%	85.6%	**82.13%**
	512	**87.18%**	85.53%	44.93%

Tabela 4.1: Resultados top-1 obtidos nos treinamentos das redes siamesas no CIFAR-10.

4.2.4 Discussão sobre os resultados

Pelos resultados dos treinamentos fica evidente que o parâmetro de margem α da triplet loss tem a maior influência sobre o treinamento do modelo. Ao utilizar o valor 0,5 os treinamentos acabam apresentando instabilidades principalmente ao utilizar uma dimensao maior do embedding de saida.

Uma outra observação retirada do treinamento dos modelos foi que mesmo aumentando a profundidade não se teve um ganho de precisão no sistema. Isso pode ser atribuído ao tamanho pequeno de 32x32 pixels das imagens do CIFAR-10. Ao passar pelas camadas da rede residual a imagem diminui de tamanho a cada operação convolucional. Como a imagem é muito pequena, apenas as camadas iniciais estão de fato aprendendo características úteis das imagens.

O tamanho do espaço vetorial onde é realizado o mapeamento também não teve influência significativa nos resultados obtidos.

5
Projeto e implementação do sistema

Este capítulo apresenta os diferentes estudos tecnológicos realizados para a implementação do sistema e o que foi desenvolvido.

5.1
Requisitos do sistema

Para desenvolver um sistema de busca por imagens sobre a base CIFAR-10, os seguintes requisitos devem ser cumpridos:

- O sistema deve ser inicializado com as imagens do CIFAR-10 e os embeddings correspondentes de cada imagem.
- O sistema deve utilizar um dos modelos siameses desenvolvidos neste projeto para realizar a extração de embeddings.
- Ao receber uma imagem JPEG, de uma das 10 classes presentes no CIFAR-10 como entrada, o sistema deve extrair o embedding dessa imagem e buscar as 10 imagens que possuem embeddings mais próximos ao da imagem recebida.
- Deve existir uma interface web para poder visualizar os resultados do sistema.
- O sistema deve ser implementado no ambiente AWS.

5.2
Estudos sobre tecnológias necessarias

A seguir serão apresentadas as tecnologias que foram estudadas para realizar a implementação do sistema no ambiente de nuvem AWS.

5.2.1
Armazenamento em nuvem

O primeiro estudo sobre quais serviços em nuvem serão necessários para implementar a solução, foi em respeito a armazenação das imagens do CIFAR-10 que serão visualizadas pelo usuário. Como o escopo do projeto foi limitado para rodar no ambiente de nuvem da AWS, apenas este provedor foi analisado.

Para armazenar as imagens que serão servidas foi decidido utilizar o serviço S3. O S3 permite armazenamento de objetos em nuvem e também nos permite criar URLs únicas e públicas para cada objeto salvo na nuvem. Poderiamos armazenar as imagens em um bancos de dados e retorná-las nas requisições, porém isso poderia implicar em um aumento considerável de dados trafegados pelo back-end. Uma solução mais simples, que foi utilizada no desenvolvimento do sistema foi trafegar apenas as URLs das imagens na comunicação entre o front-end e o back-end. Desse jeito o próprio front-end fica responsável por fazer as requisições ao S3 e carregá-las.

5.2.2 Servidores dedicados e sistemas serverless

Para servir a API que será responsável por receber, executar o modelo, buscar e retornar as imagens mais semelhantes, foram estudadas duas alternativas de serviços em nuvem para a implementação.

A primeira alternativa estudada foi o AWS Lambda. O AWS Lambda oferece uma solução para criar aplicações seguindo o modelo de desenvolvimento serverless. Aplicações serverless diferentes de outros modelos de computação em nuvem não requerem que o desenvolvedor se preocupe com manutenção e administração de servidores para mantê-los rodando sem interrupção. O próprio provedor de nuvem, que no caso do projeto é a Amazon, fica responsável por escalar e providenciar recursos de acordo com a demanda do código que será executado. Para o desenvolvedor, é necessário apenas construir um container utilizando a ferramenta Docker, contendo todas as bibliotecas e ferramentas necessárias para executar seu código. Após a criação do container é possível armazená-lo em um repositório que é consultado para a inicialização da aplicação. Aplicações serverless abstraem todas as tarefas rotineiras de provisionamento e escala de infraestrutura. Uma outra vantagem em utilizar uma arquitetura serverless é que só será cobrado do usuário do serviço quando a aplicação é executada, sendo cobrado de acordo com o tempo de execução e uso de processamento computacional utilizado pela aplicação. Com a arquitetura serverless as aplicações só são inicializadas quando um evento aciona a execução do código. Com a AWS se a aplicação não receber uma requisição em cinco minutos a aplicação é automaticamente desligada. Ao receber um evento que sinaliza que o código deverá ser executado, o serviço cria uma instância do container e o mantém ligado para processar as requisições. Uma limitação dessa abordagem é o tempo necessário para ligar a aplicação depois dos cinco minutos sem sua utilização. Outra limitação imposta pelo Lambda é que ficamos limitados a executar nosso código em CPU.

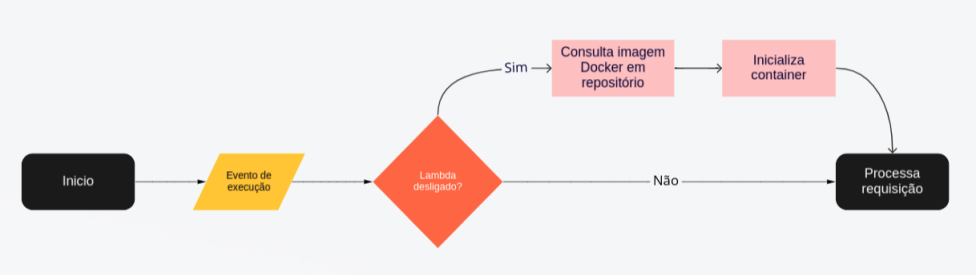

Figura 5.1: Inicialização serviço Lambda

O cold-start é o tempo necessário para instanciar o container caso a aplicação esteja desligada. Como a aplicação proposta irá conter diversos componentes pesados como bibliotecas e um modelo de rede neural profunda, ela pode ter um cold-start muito elevado podendo ser prejudicial para a aplicação.

Uma segunda abordagem possível para a implementação da API seria utilizar o serviço de servidor dedicado. Com um servidor dedicado seria necessário realizar todas as tarefas rotineiras como manutenção, provisionamento e escala de máquinas que rodam na nuvem. Uma vantagem em utilizar um servidor dedicado seria que não haveria tempos de cold-start e a aplicação sempre estaria rodando sendo que a inicialização iria ocorrer apenas quando a máquina fosse instanciada. Para servir a aplicação em um servidor dedicado foi também escolhido criar um container contendo todas as ferramentas e bibliotecas necessárias. Outra desvantagem em utilizar servidores dedicados é o custo para utilizar este serviço. Como a aplicação estará sempre ligada estaremos utilizando recursos mesmo sem a utilização da aplicação. O custo também depende da instância da máquina escolhida, o quanto de memória alocamos para ela e se realizaremos a inferência em GPU ou CPU.

Foi decidido utilizar uma arquitetura serverless para o sistema proposto. Todo o código será encapsulado em um Docker container e será executado no ambiente da AWS utilizando o serviço AWS Lambda functions.

5.2.3 AWS Lambda

Para desenvolver uma aplicação que roda em uma arquitetura serverless no ambiente da AWS, foi necessário estudar os padrões de desenvolvimento a serem seguidos na linguagem Python, como instalar bibliotecas necessárias para a execução do código e como gerar um URL para expor a aplicação para a internet para poder ser utilizada como uma API. Para esse estudo foi utilizada a própria documentação disponibilizada pela Amazon.

5.2.4 Amplify

Para servir o front-end da aplicação foi utilizado o AWS Amplify que permite que seja levantado um site desenvolvido em HTML, CSS e Javascript. Utilizando a própria interface web do Amplify é possível subir um arquivo zip contendo um HTML e arquivos adicionais. Ao subir o arquivo uma URL é automaticamente gerada para que o site possa ser acessado.

5.2.5 APIs para automação em nuvem

Além de disponibilizar uma interface web para interação com seus serviços, a AWS também permite que um usuário crie e gerencie aplicações por meio de APIs. Para interagir com esses serviços de uma forma programática foi utilizada a biblioteca Python Boto3. Para tratar do deploy da aplicação de uma forma automática, foi também utilizada a ferramenta Serverless Application Model (SAM), disponibilizada pela AWS. Esta ferramenta permite o deploy programático de aplicações serverless sem a necessidade de uma interface gráfica.

5.3 Implementação

5.3.1 Visão geral da solução desenvolvida

Para implementar o sistema proposto por esse projeto foi inicialmente desenhada uma arquitetura da solução geral definindo quais componentes deverão ser implementados. Essa arquitetura pode ser vista na figura 5.2. Para a solução completa os seguintes 4 módulos foram desenvolvidos:

1. Codigo de treinamento para a rede neural siamesa utilizando triplet loss e o modelo de kNN.

2. Um script para extrair os embeddings das imagens da base de dados e armazená-las na nuvem.

3. O back-end da aplicação, que irá expor uma API que recebe uma imagem e retorna às 10 imagens mais semelhantes em uma base de dados.

4. O front-end da aplicação, que servirá como interface gráfica para a API exposta pelo back-end.

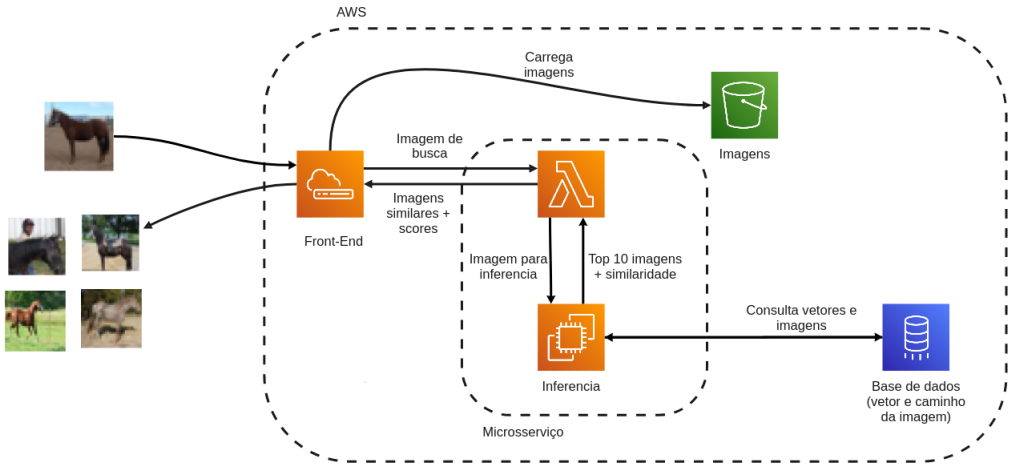

Figura 5.2: Arquitetura da solução

5.3.2 Extração e alimentação da base de dados

Após realizado o treinamento do modelo foi desenvolvido um script, utilizando a linguagem Python, para inicializar a base de dados contendo o mapeamento entre o URL da imagem já no bucket S3 e seus respectivos embeddings.

O script recebe uma pasta local contendo imagens cujo vetores serão extraídos, um modelo treinado que será utilizado para extrair os vetores e as credenciais necessárias para inicializar a arquitetura no ambiente AWS. São realizadas as seguintes tarefas pelo script:

1. Carrega o modelo siamês pré-treinado para extração dos embeddings.
2. Cria um bucket S3 público no ambiente AWS que será utilizado para armazenar as imagens.
3. Para cada imagem na base: insere a imagem no bucket S3, pré-processa a imagem e extrai o embedding utilizando o modelo carregado.
4. Cria uma entrada em uma dicionário local armazenando um mapeamento entre o URL da imagem e seu respectivo embedding.
5. Cria duas listas, uma que ira conter os URLs das imagens e outra que ira conter os respectivos embeddings.
6. Treina e armazena na máquina local o modelo de kNN que será utilizado para realizar a busca por embeddings e uma lista contendo as URLs das imagens.

Com a execução deste script são criados dois arquivos na máquina local: o kNN treinado e a lista com as URLs das imagens. Estes dois arquivos e o modelo siamês pré-treinado serão utilizados no back-end da aplicação. Vale ressaltar que os índices da lista de embeddings utilizada no treinamento do kNN deve estar condizente aos índices da lista de URLs. O kNN armazena os embeddings utilizados durante sua instanciação e ao realizar a inferência, serão retornados apenas os índices dos 10 vetores mais próximos e suas distâncias euclidianas. Para de fato ter o URL da imagem mais similar, precisamos apenas retornar os elementos da lista de URLs nos índices retornados pelo kNN.

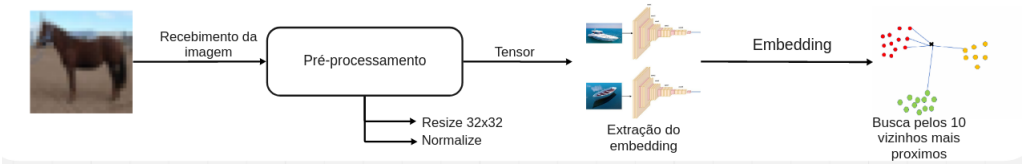

Figura 5.3: Fluxo de inferência

5.3.3 Back-end

O back-end da aplicação realiza as seguintes tarefas:

1. Recebe uma imagem através de uma requisição POST (HTTP).
2. Redimensiona a imagem recebida para 32x32, pré-processa a imagem utilizando o mesmo pré-processamento utilizado durante o treinamento e realiza a inferência sobre a imagem para obter um embedding.
3. Retorna aos URLs e os scores de similaridade das 10 imagens mais semelhantes a recebida. (Scores de similaridade são as distâncias euclidianas retornadas pelo kNN).

Este módulo foi desenvolvido para rodar como um serviço serverless então ele nem sempre estará ligado. Ao receber uma requisição e o container não estiver inicializado, é realizado o carregamento do modelo siamês treinado, a instanciação do kNN e o dicionário contendo o mapeamento entre os embeddings e as URLs públicas da imagens. Todos esses arquivos foram previamente incluídos na imagem Docker. Ao receber a imagem codificada como uma string em Base64 no corpo de um método POST, o código converte a imagem recebida para bytes e a carrega em memória. Com a imagem carregada são feitos os mesmos pré-processamentos que foram realizados nas imagens da base de treino, em seguida é feita a extração do embedding da imagem

pelo modelo siamês. Tendo o embedding da imagem recebida, é realizada a inferência sobre o kNN para obter os índices e scores dos 10 embeddings mais próximos ao da imagem recebida. Este codigo que executa no back-end pode ser visto no apêndice B.3. Os mesmos índices são então utilizados para retornar as URLs das imagens. Por fim é montado um JSON que é enviado como resposta da requisição. Este JSON de resposta pode ser visto no apêndice B.2.

5.3.4 Container Docker

Para servir o código do back-end em uma arquitetura serverless foi necessário primeiramente desenvolver um container Docker que contém todas as bibliotecas necessárias para executar o código. Além de definir as bibliotecas necessárias, também foi necessário incluir os arquivos gerados pelos scripts anteriores dentro do container. O arquivo Dockerfile para a criação deste container pode ser encontrado no apêndice B.2.

5.3.5 Front-end

Para oferecer uma interface gráfica com a API exposta pelo back-end da aplicação foi desenvolvido um front-end utilizando Javascript, HTML e CSS. Este front-end foi desenvolvido para rodar em um navegador e possui apenas dois componentes.

O primeiro componente desenvolvido apenas apresenta para o usuário uma caixa para realizar o upload das imagens que devem ser enviadas para o back-end. Em cima do componente fica especificado qual modelo está sendo utilizado para realizar a comparação de similaridade. Ao receber a imagem que será enviada, o próprio código Javascript converte a imagem para base64 e a envia para o back-end utilizando uma requisição POST, passando a imagem codificada no corpo da requisição.

O segundo componente é apenas um grid responsável por carregar e mostrar as imagens recebidas pelo front-end. Ao receber o JSON de resposta contendo dez URLs com seus respectivos scores de similaridade, o front-end realiza uma requisição GET para cada URL e exibe a imagem com seu score na tela. Esta tela com as imagens já carregadas pode ser vista no apêndice A.

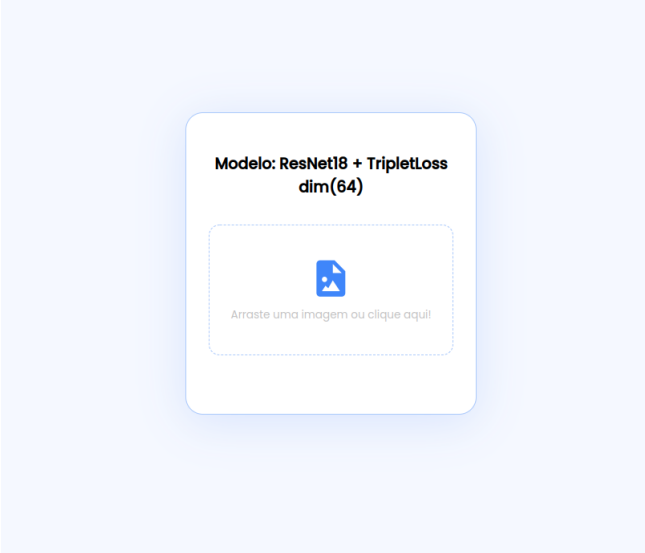

Figura 5.4: Componente para upload de imagem

6
Avaliação do sistema implementado

Este capítulo descreve os testes realizados sobre o sistema desenvolvido. Serão apresentados resultados de diferentes consultas realizadas no sistema, incluindo erros e limitações do algoritmo desenvolvido.

6.1
Testes e experimentos realizados no sistema

Como o treinamento de modelos mais profundos e com diferentes dimensões de embeddings não apresentaram ganhos significativos, como apresentado no capítulo 4, os testes deste capítulo foram realizados utilizando o modelo com a ResNet de 18 camadas, gerando embeddings de dimensão 64. A base levantada no ambiente AWS contém as 10.000 imagens do conjunto de teste do CIFAR-10.

6.1.1
Avaliação de consultas realizadas no sistema

Para avaliar se o sistema estava de fato retornando imagens similares, foram realizadas diversas consultas ao sistema utilizando imagens da própria base do CIFAR-10. Além de utilizar imagens presentes no CIFAR-10, foi também analisada a capacidade de generalização do algoritmo ao receber imagens que não estão presentes na base.

Como pode ser observado pela figura 6.2 mesmo ao receber uma imagem de um jumento que não pertence ao CIFAR-10, o sistema corretamente retornou imagens da classe mais similar presente na base. Outras consultas realizadas com imagens que pertencem e não pertencem ao CIFAR-10 podem ser vistas no apêndice A.

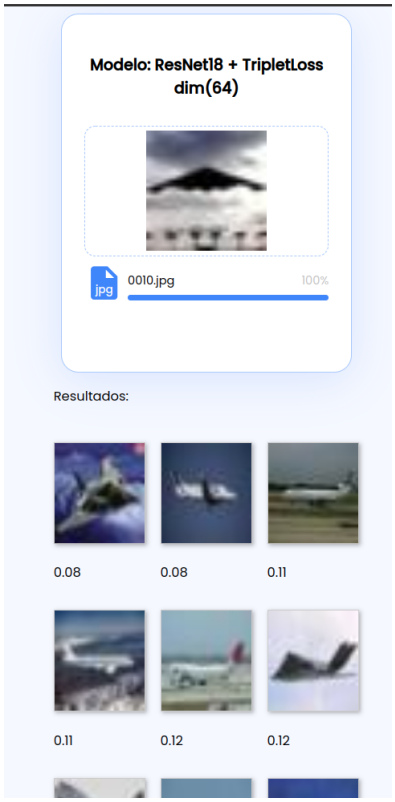

Figura 6.1: Busca utilizando uma imagem de um avião presente no CIFAR-10.

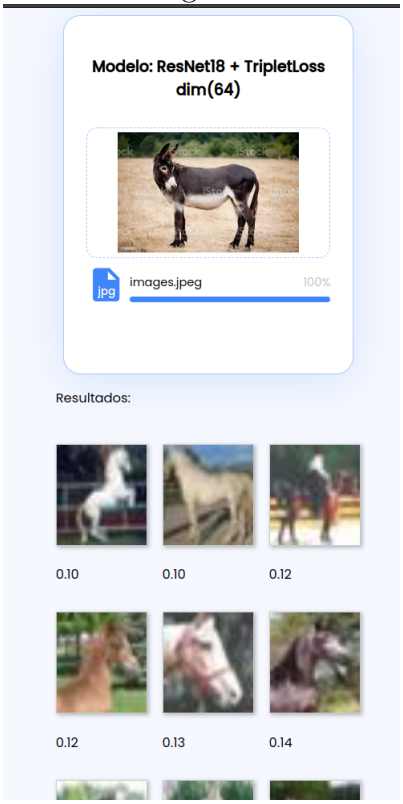

Figura 6.2: Consulta utilizando uma imagem de um jumento.

6.1.2 Erros e limitações

Foram encontrados alguns erros ao utilizar certas imagens como consulta para o sistema. Dois destes erros podem ser vistos nas figuras A.7, A.8 e A.9. Na figura A.7 utilizando uma imagem de um avião fora da base do CIFAR-10 o sistema falhou em encontrar imagens semelhantes. A Figura A.8 mostra uma outra consulta que retornou resultados que não condizem com a entrada mesmo utilizando uma imagem do CIFAR-10.

Apesar do modelo apresentar bons resultados para certas imagens fora da base do CIFAR-10, o sistema retorna resultados incorretos ao receber imagens que fogem muito do padrão das imagens do CIFAR-10. Isso apesar de ser uma limitação é algo esperado do modelo desenvolvido. Modelos de aprendizado de máquina são muito bons em capturar a distribuição da base de dados em que foram treinados, porém ao utilizar imagens que fogem muito desta distribuição os modelos acabam falhando. Esta limitação pode ser vista claramente na figura A.9 onde foi utilizada a imagem da nuca de um cavalo como parâmetro de busca. Apesar do CIFAR-10 conter diversas imagens de cavalos, essas imagens acabam sendo comportadas demais não dando variação suficiente para generalização do modelo.

7
Considerações finais

O objetivo deste projeto foi realizar o treinamento de uma rede siamesa utilizando a triplet loss e implementar um sistema de busca reversa na base de dados CIFAR-10. Para alcançar este objetivo foi necessário realizar estudos sobre como algoritmos de aprendizado por contraste funcionam e como podemos treinar um modelo que gera representações vetoriais de imagens para efetuar a comparação de similaridade. A fim de estudar a melhor configuração de treinamento do modelo siamês foram realizados diversos treinamentos.

Na segunda etapa do projeto foi estudado o ambiente computacional de nuvem AWS e quais serviços foram necessários para de fato implementar um sistema que utiliza o modelo treinado para realizar buscas por imagens. Como foi observado pelos resultados dos treinamentos, mesmo não atingindo uma precisão acima de 90% o modelo mostrou que de fato pode ser utilizado como um algoritmo eficaz por trás de um sistema de busca reversa de imagens.

Em relação a trabalhos futuros esse projeto sugere algumas opções para a continuidade do desenvolvimento do sistema e estudos adicionais para o modelo siamês:

- É evidente a limitação na capacidade de generalização do modelo desenvolvido neste projeto. Apesar do modelo apresentar bons resultados até para imagens fora da base de treinamento, trabalhos futuros podem realizar o treinamento em bases maiores como a ImageNet e estudar a capacidade de generalização do modelo. Outros trabalhos podem também realizar uma comparação entre diferentes funções de custo para aprendizado por contraste e compará-las a triplet loss.
- O sistema desenvolvido apesar de poder ser utilizado para outras bases apenas necessitando a troca do modelo siamês não possui funcionalidade de gestão sobre a base de imagens. Futuros desenvolvimentos podem adicionar novas funcionalidades ao back-end para permitir que imagens sejam incluídas ou retiradas da base.

Referências Bibliográficas

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., e Shah, R. (1993). Signature verification using a"siamese"time delay neural network. *Advances in neural information processing systems*, 6.

Cook, A. (2017). Using transfer learning to classify images with keras.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., e Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118.

Fix, E. e Hodges, J. L. (1989). Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247.

Hadsell, R., Chopra, S., e LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE.

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

Kha Vu, C. (2021). Deep metric learning: A (long) survey.

Khan, A., Sohail, A., Zahoora, U., e Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53(8):5455–5516.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., e Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Kühl, N., Goutier, M., Baier, L., Wolff, C., e Martin, D. (2020). Human vs. supervised machine learning: Who learns patterns faster? *arXiv preprint arXiv:2012.03661*.

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

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Pugliese, R., Regondi, S., e Marini, R. (2021). Machine learning-based approach: global trends, research directions, and regulatory standpoints. *Data Science and Management*, 4:19–29.

Ramzan, F., Khan, M. U., Rehmat, A., Iqbal, S., Saba, T., Rehman, A., e Mehmood, Z. (2019). A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using resting-state fmri and residual neural networks. *Journal of Medical Systems*, 44.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., e Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Salomon, G., Britto, A., Vareto, R. H., Schwartz, W. R., e Menotti, D. (2020). Open-set face recognition for small galleries using siamese networks. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 161–166. IEEE.

Schroff, F., Kalenichenko, D., e Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

Sivic, J. e Zisserman, A. (2003). Video google: A text retrieval approach to object matching in videos. In *Computer Vision, IEEE International Conference on*, volume 3, pages 1470–1470. IEEE Computer Society.

Srivastava, R. K., Greff, K., e Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Weng, L. (2021). Contrastive representation learning. *lilianweng.github.io*.

Wu, H., Xu, Z., Zhang, J., Yan, W., e Ma, X. (2017). Face recognition based on convolution siamese networks. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–5.

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., e Stolcke, A. (2018). The microsoft 2017 conversational speech recognition system. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5934–5938. IEEE.

Zhang, W. (2017). Machine learning approaches to predicting company bankruptcy. *Journal of Financial Risk Management*, 06:364–374.

A
Resultados de buscas feitas no sistema

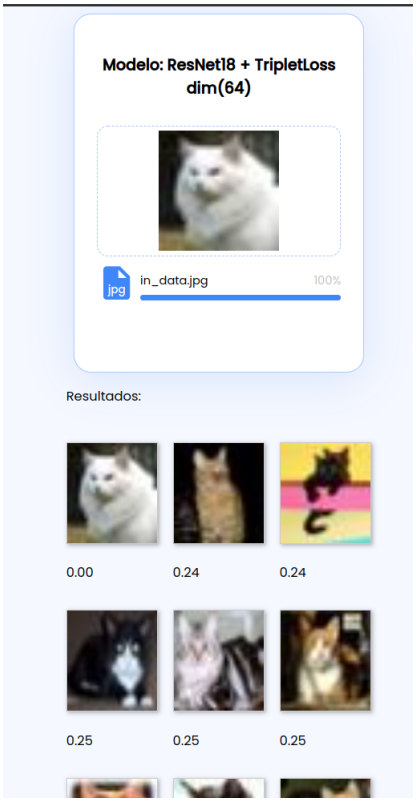

Figura A.1: Consulta utilizando uma imagem presente na base
Score de similaridade igual a 0,00 pois a distância entre do mesmo embedding para ele mesmo é 0.

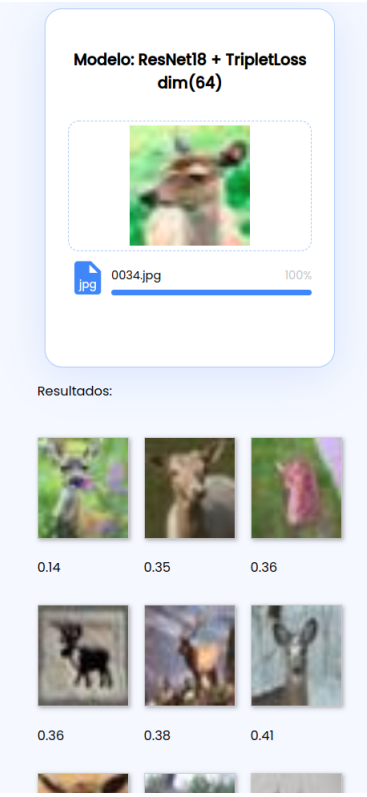

Figura A.2: Busca utilizando uma imagem de um veado que pertence ao CIFAR-10.

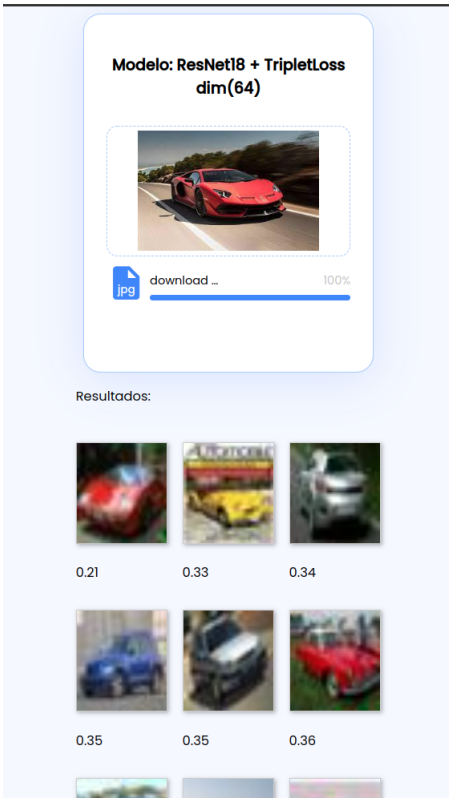

Figura A.3: Busca utilizando uma imagem de um carro que não pertence a base CIFAR-10

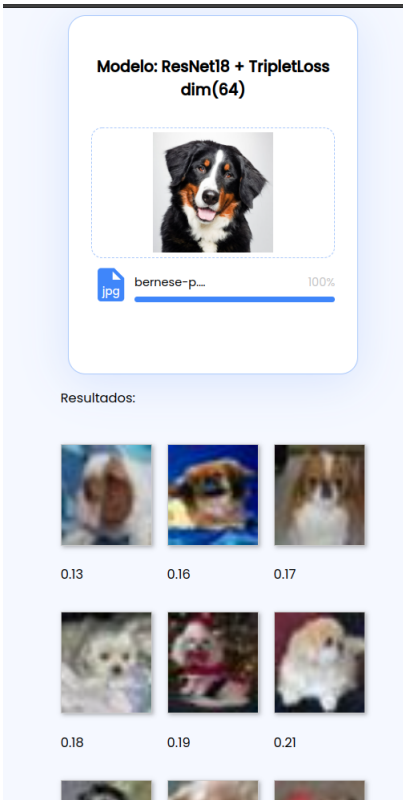

Figura A.4: Busca utilizando uma imagem de um cachorro que não pertence a base CIFAR-10.

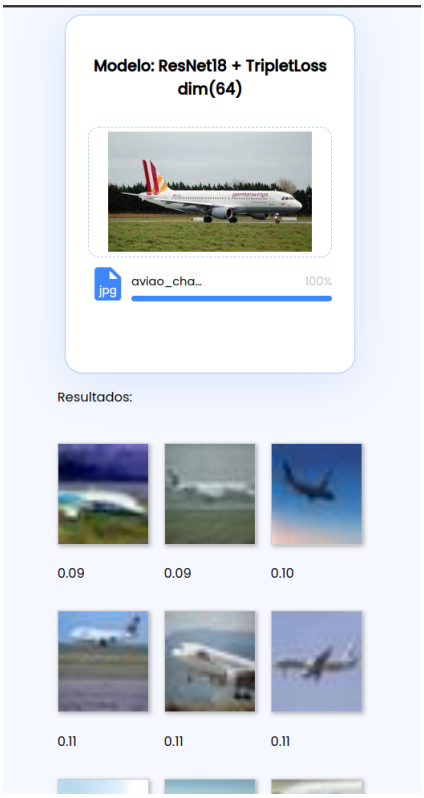

Figura A.5: Busca utilizando uma imagem de um avião que não pertence a base CIFAR-10.

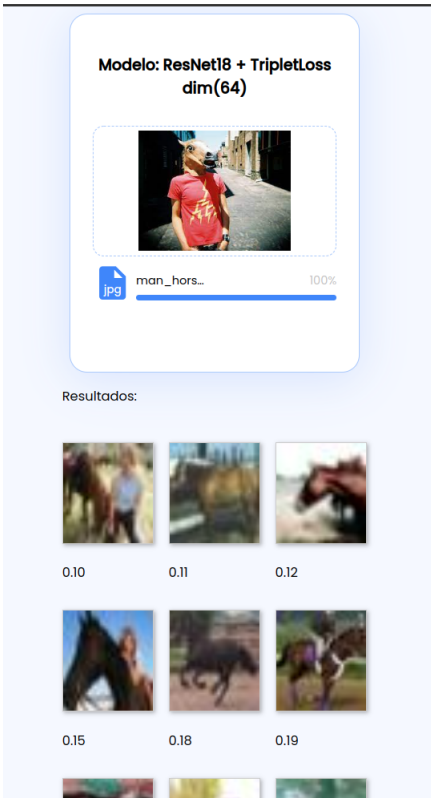

Figura A.6: Busca utilizando uma imagem de um homem usando uma mascara de cavalo.

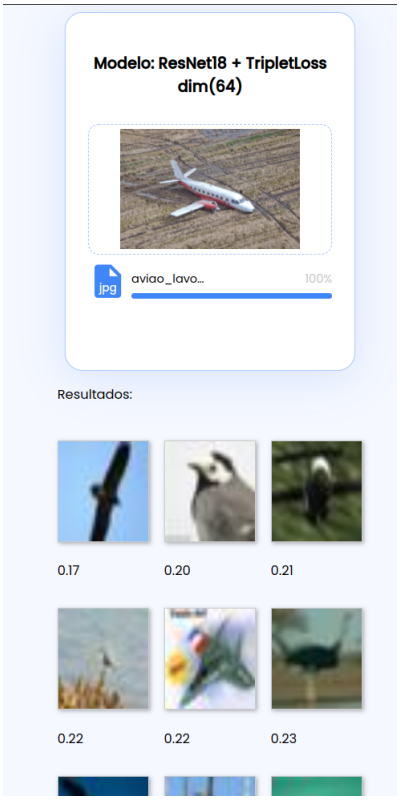

Figura A.7: Busca utilizando uma imagem de um avião que não pertence a base CIFAR-10.

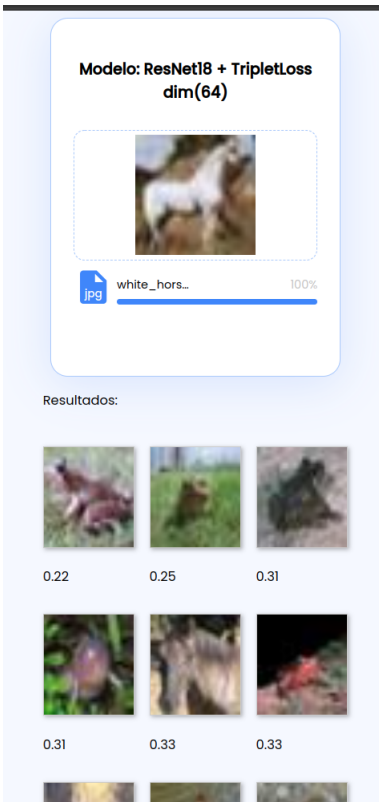

Figura A.8: Busca utilizando uma imagem de um cavalo que pertence a base CIFAR-10.

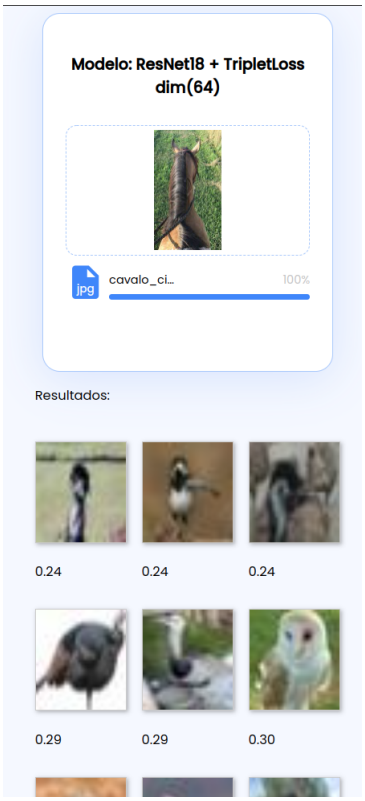

Figura A.9: Busca utilizando uma imagem de um cavalo que não pertence a base CIFAR-10.

B
Trechos de códigos utilizados neste projeto

B.1
Código para criação do container Docker

```
1
2  #Container base a ser utilizado
3  FROM public.ecr.aws/lambda/python:3.8
4
5  #Copia o codigo de inferencia e a arquitetura do modelo
6  #para dentro da imagem
7  COPY app.py resnet_cifar.py requirements.txt ./
8
9  #Copia os pesos do modelo siames
10 COPY model_trained.pth /opt/ml/model
11 #Copia a lista de urls
12 COPY url_list.pkl /opt/ml/emb_db.pkl
13 #Copia o modelo de kNN
14 COPY knn.joblib /opt/ml/knn.joblib
15
16 #Instala dependencias python
17 RUN python3.8 -m pip install -r requirements.txt -t .
18
19 #applicacao que sera executada ao iniciar este container
20 CMD ["app.lambda_handler"]
```

B.2
Exemplo de um JSON retornado pelo back-end

```
1 {
2    "results":{
3       "https://public-cifar10.s3.amazonaws.com/32545.jpg"
           :0.10253123549829335,
4       "https://public-cifar10.s3.amazonaws.com/19037.jpg"
           :0.10849990017992298,
```

```
      "https://public-cifar10.s3.amazonaws.com/36569.jpg"
         :0.12291307389610259,
      "https://public-cifar10.s3.amazonaws.com/21493.jpg"
         :0.15399842964767968,
      "https://public-cifar10.s3.amazonaws.com/12901.jpg"
         :0.1840623240175785,
      "https://public-cifar10.s3.amazonaws.com/10789.jpg"
         :0.1852586319257727,
      "https://public-cifar10.s3.amazonaws.com/25090.jpg"
         :0.18636216278722567,
      "https://public-cifar10.s3.amazonaws.com/47170.jpg"
         :0.19119351106428029,
      "https://public-cifar10.s3.amazonaws.com/13198.jpg"
         :0.19688016685886855,
      "https://public-cifar10.s3.amazonaws.com/42353.jpg"
         :0.1973455917194234
   }
}
```

B.3
Trecho de codigo de inferência executado no back-end

```
    #Converte image recebida de base64 para bytes
    image = Image.open(BytesIO(base64.b64decode(image_
       bytes)))

    #Preprocessa imagem
    input_tensor = transform(image).unsqueeze(0)

    #Realiza a infenrecia sobre o modelo siames
    output_embedding = model(input_tensor)

    #Realiza a inferencia sobre o modelo kNN
    output_embedding = output_embedding.detach().numpy()
    distances, indices = knn.kneighbors(output_embedding)

    #Busca as URLs na lista ‘path‘ pelos indices
       retornados pelo kNN
    result_paths = [paths[indice] for indice in indices
       [0]]

```

```
17      #Monta JSON em ordem decrescente de acordo com
            distancias
18      resp = {path: distances[0][i] for i, path in
            enumerate(result_paths)}
19      resp = dict(sorted(resp.items(), key=lambda item:
            item[1]))
20
21      return {
22          'statusCode': 200,
23          'body': json.dumps(
24              {
25                  "results": resp,
26              },
27          )
28      }
```