Geoguessr: Determinando estados brasileiros a partir de um modelo de visão computacional

Nauane Linhares
Engenharia Aeronáutica
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
Email: nauane.nascimento@ga.ita.br

I. Introdução

O google é uma das maiores empresas da atualidade, sendo famosa por ser nosso principal recurso para pesquisas de sites e de localização. No campo de localização, a evolução do produto Google Maps, permitiu uma cobertura de imagens ao redor do globo, como é possível visualizar na Figura 1.

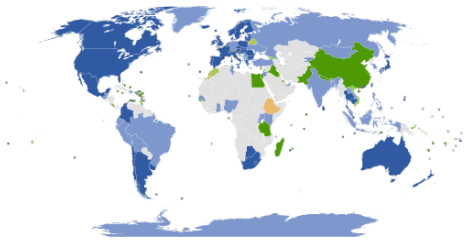

Fig. 1. Cobertura de imagens fornecidas pela google ao redor do mundo

Essa extensa cobertura de imagens deu origem a diversos aplicativos que exploram essa abordagem. Um desses aplicativos é o Geoguessr, um jogo que utiliza a base de dados do Google Maps para desafiar os jogadores a adivinharem sua localização com base nas imagens fornecidas.

A fama desse jogo evoluiu significativamente nos últimos anos, principalmente em época de pandemia e as pessoas não poderiam sair de casa. Com tal reputação, competições nacionais e internacionais foram surgindo, o que fez com que os jogadores profissionais surgissem e criassem vários tipos de estratégias para determinar sua posição exata no mapa-mundi. Por exemplo, uma estratégia famosa é detectar se estamos na Nigéria, uma vez que é comum observar a presença de carros policiais seguindo o automóvel do Google.

Fig. 2. Carro da polícia seguindo carro do Google

Com o sucesso desse jogo e a gama de estratégias existentes, surge a motivação de criar um modelo de aprendizado de máquina que consegue olhar as imagens fornecidas no jogo e identificar essas nuancias que os jogadores conseguem perceber.

Com tal motivação, este trabalho tem como objetivo o de abrir o Geoguessr em mapas brasileiros e conseguir determinar o estado brasileiro a partir de fotos disponibilizadas no jogo.

II. Metodologia

Como a maior parte dos projetos de Visão Computacional, a primeira parte consistiu de obter dados suficientes para treinar o modelo. No caso desse projeto, há a limitação de não ter nenhum banco de imagens dos estados brasileiros, limitando a execução do projeto. Para contornar tal problema, utilizamos dados disponíveis a partir da API do IBGE e do Google para obter imagens suficientes para treinamento e validação do modelo. Por meio da API do IBGE podemos obter dados relacionados aos estados Brasileiros, enquanto que por meio da API do Google Maps, é possível obter imagens a partir de dados de latitude e longitude.

A. Determinação da latitude e longitude e distribuição de imagens

Para utilizar a API do Google e obter imagens, inicialmente é necessário fornecer dados de latitude e longitude. Para tal, obtiveram-se os dados de latitude e longitude do contorno de cada estado brasileiro, conforme site do IBGE [1]. Isso foi importante já que o treinamento necessita ter uma distribuição homogênea de dados, então é necessário conseguir um número igual de imagens para cada estado.

Um outro ponto a se considerar, é que a maior partes das imagens que estão disponibilizadas no Google Maps está relacionada a vegetação, o que pode ser importante para diferir entre vegetações da região Nordeste e do Sul, mas é ruim quando é necessário perceber diferenças entre estados da mesma região, ou para o próprio Geoguessr, na qual a maior parte das imagens se encontram na região urbana.

Para contornar isso, utiliza-se tanto os dados das malhas dos muncípios brasileiros, quanto os dados de população de cada uma dessas cidades. Vale ressaltar que todos esses dados vieram via API ou site do IBGE [1].

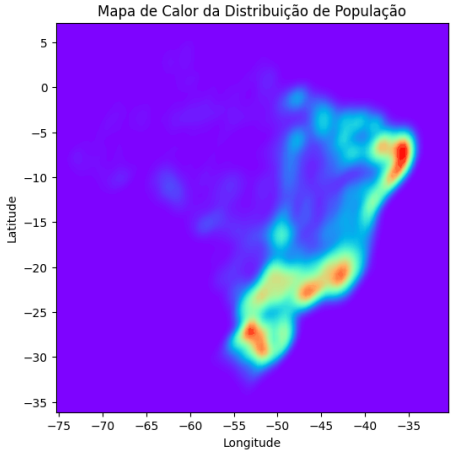

Fig. 3. Distribuição populacional no Brasil, Dados do IBGE

A estratégia aqui é selecionar latitudes e longitude se baseando na distribuição de população de cada cidade. Por exemplo, para o Estado de São Paulo, na maior parte das vezes seleciona-se latitudes e longitudes bem próximas ao centro da cidade de São Paulo, já que a população dessa cidade é muito elevada em comparação com cidades vizinhas. Nesse caso, as latitudes e longitudes selecionadas terão como base no centróide da cidade de são paulo. Na Figura 3, observa-se que há uma distribuição desigual de pessoas no território, evidenciando que é importante focalizar nas cidades, já que a maior parte mapeada é vegetação.

Com os dados dos limites das latitudes e longitudes de cada estado, e com o centróide da cidade escolhida, é possível obter imagens a partir de uma aleatorização uniforme desses dados, usando a API do google.

Vale reforçar, que a escolha da cidade se baseia na distribuição da população das cidades que estão dentro dos limites inferiores e superiores de latitude e longitude conforme pode ser visto na Figura 4, na qual o estado é limitado por um retângulo ou *bounding box*.

Fig. 4. Limites do Mato Gross que servem para selecionar uma cidade para aleatoriazação

A princípio isso pode prejudicar a identificação do estado, já que podemos sair e escolher cidades que estão fora do estado, mas, é esperado que regiões próximas ao estado sejam semelhantes, até ajudando a evitar fenômenos de overtfitting. As Equações 1 e 2 mostram uma equação matemática para a determinação aleatória dos valores de latitude e longitude, em que lat' e long' são as latitudes do centróide da cidade escolhida, σ_{lat} e σ_{long} são os desvios padrões relacionados as cidades no *bounding box* relacionado ao estado e n é o número de cidades dentro do *bounding box*.

$$lat =\sim U(lat' - \frac{\sigma_{lat}}{n}, lat' + \frac{\sigma_{lat}}{n}) \tag{1}$$

$$lat =\sim U(long' - \frac{\sigma_{long}}{n}, long' + \frac{\sigma_{long}}{n}) \tag{2}$$

B. Raio de detecção

Para fazer uma requisição na API do Google, um dos parâmetros fornecidos é o raio de detecção (R), que consiste no raio do círculo que a API usa para detectar a imagem mais próxima a partir de latitude e longitude definidas. Esse parâmetro é fundamental, principalmente para regiões do norte que possuem baixa cobertura, pois estas precisam de um raio de detecção elevado. Além disso, regiões pequenas ou regiões com alta cobertura teriam que ter um raio de detecção baixo, já que é fácil achar uma localização com imagem disponível.

Como o raio de detecção ótimo varia para cada estado, define-se o valor inicial do raio de detecção para 100 metros. Esse valor vai aumentando conforme o número de falhas de requisição de imagem para aquele R vai acontecendo. Dessa forma, estados com baixa cobertura ficarão com R alto, enquanto cidades com excelente cobertura ficarão com R baixo. O R vai atualizando de valor a cada 100 erros de requisição, conforme a Equação 3.

$$R \rightarrow R \cdot 1.2 \tag{3}$$

C. Modelo

Para o modelo utilizamos a rede neural pré-treinada ResNet50, que é uma rede neural convolucional profunda composta por 50 camadas. As imagens estão inicialmente com dimensões 600x300, posteriormente, com transformações tais como redimensionização para dimensão 224 x 224 e normalização, obtiveram-se dimensões (224,224,3), que é a entrada da rede do modelo. No caso a saída é um tensor de tamanho 27 contendo as probabilidades da imagem referentes aos estados brasileiros. Sua arquitetura é conforme Figura 5.

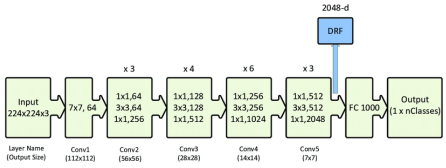

Fig. 5. Modelo Resnet50

D. Desempenho no Geoguessr

O Geoguessr funciona da mesma forma que o Google Street View, permitindo que o jogador ande ao redor de um determinado mapa. Pensando nisso, para analisar o desempenho do modelo no Geoguessr, implementa-se um código que, com o mapa aberto, pega as imagens do mapa e retorna as probabilidades. Esse código rodará iterativamente, de tal forma que as probabilidades irão convergir, sendo que a convergência irá acontecer conforme a Equação 4, em que P é o vetor de probabilidades final e p(n) é o vetor das probabilidades geradas a partir da n-ésima foto. Após essa operação, há uma normalização dos resultados a cada iteração, estas que são interrompidas até a maior probabilidade do vetor atingir ao menos 99,9%. Tal método faz com que a identificação do estado não dependa de apenas uma foto, mas de várias fotos, já que, conforme o jogador anda no mapa, mais imagens ele analisa.

$$\mathbf{P} \rightarrow \mathbf{P} \times \mathbf{p}(n) \tag{4}$$

III. Implementação Desenvolvida

Para implementação do modelo, foram criadas duas rotinas em Python: getDataSet.py e trainAndPlayGeoguessrModel.py.

A. Obtendo o dataset

O primeiro arquivo, getDataSet.py, consiste no código que cria o dataset utilizando a API do Google e do IBGE. Ao ser executado, o programa verifica quantas imagens há para cada estado. Se esse número de imagens for menor que um determinado valor (variável que é definida pelo usuário), ele irá girar imagens daquele estado até alcançar tal valor.

Um ponto a se considerar é que, para que o processo fosse acelerado, executa-se várias *threads* rodando a função *GetImageSingle* da classe GoogleMapsView.

Vale ressaltar que ele irá tentar conseguir essas imagens conforme as seções II-A e II-B, com base nas populações das cidades e variando o raio de detecção conforme erros de requisição.

Ademais, esse programa necessita dos arquivos BR_Municipios_2022.shp e BR_UF_2022.shp para rodar corretamente, tais arquivos que consistem nos dados das malhas geográficas dos municípios e dos estados brasileiros.

Por fim, é importante ressaltar que, para conseguir executar a API do Google, é necessário ter uma chave, esta que pode ser obtida a partir do site de APIs do próprio Google.

B. Treinando o modelo e executando o jogo

O segundo arquivo, 'trainAndPlayGeoguessrModel.py', consiste no código que consegue treinar ou carregar o modelo treinado e jogar Geoguessr (Ou Geostatic, uma versão gratuita do jogo).

Para saber qual ação o código irá executar, existe a variável 'train', que indica se o código irá treinar ou não.

- Se 'train' for 'True', os métodos relativos ao treinamento do modelo são instanciados, salvando um arquivo 'model.pth' após o treinamento.
- Se 'train' for 'False', utilizando a biblioteca do Selenium, o navegador do Google Chrome abre, além de que o modelo treinado, 'model.pth', é carregado.

Após o navegador abrir, o site do Geoguessr (Ou Geostatic) é aberto, precisando que o usuário logue e inicie o modo de jogo correspondente ao mapa do Brasil.

Nessa situação o código entra em um loop em que verifica se as teclas 'ESC', 'Ç' e o botão esquerdo do mouse são pressionados.

- Se o usuário apertar 'ESC', o programa é finalizado.
- Se o usuário apertar 'Ç', o jogo inicia a análise de imagens, pegando a imagem do jogo cada vez que o usuário clica com o botão esquerdo do mouse. Caso o usuário aperte novamente, ele finaliza a análise, retornando o Estado mais provável.

Cada clique com o botão esquerdo faz com que o código chame o método 'getKeyPress' da classe 'Navegador' que salva a imagem, e o método 'predictState' da classe 'GeoguessrModel', que retorna as probabilidades relativas ao estado, fazendo as operações conforme II-D. Vale dizer que para o treinamento acontecer é necessários ter a pasta 'dataSet' e as subpastas com as imagens de cada estado.

IV. Resultados e Discussões

A. Treinamento

Utilizando cerca de 1600 imagens para cada Estado, e dividindo os dados, com 80% para os dados de treinamento e 20% para validação, observa-se um acurácia de 51,83% após o fim treinamento do teste, o que é um resultado incrível dado o número de imagens fornecidas.

B. Testando o Geoguessr

Utilizando o conjunto treinado, testa-se inicialmente o modo Brasil, em que você basicamente cai em um determinado estado brasileiro. São feitos 25 testes, nos quais seus resultados estão incluídos na Tabela I, a acurácia do resultado foi de cerca 36 %, o que mostra um resultado razoável.

Analisando mais detalhadamente, observa-se que a maior parte dos erros aconteceram em regiões próximas, como São Paulo (SP) e Minas Gerais (MG), Distrito Federal (DF) e Minas Gerais (MG) e Rio Grande do Norte (RN) e Ceará (CE), o que é um indício de que o modelo conseguiu aprender razoavelmente, mas não teve dados suficientes para discernir estas regiões.

Um outro ponto é que, em muitos testes, antes de convergir, sempre regiões urbanizadas apareciam com altas probabilidades relacionadas ao Distrito Federal, o que faz sentido, já que é uma cidade urbana, então o modelo conseguiu identificar a característica dessa região. Analogamente, o modelo tinha probabilidades elevadas para São Paulo e Minas Gerais quando o mapa estava em rodovias, o que é coerente, já que são os Estados com maior número de rodovias.

Além disso, observa-se que o modelo conseguiu acertar todas as vezes que o mapa caiu no Estado do Rio de Janeiro, o que é um resultado excelente. Para observar se o resultado é uma coincidência, realiza-se o teste apenas no Estado do

TABLE I

TABELA DE RESULTADOS DA PREVISÃO DE ESTADOS BRASILEIROS

Resposta Correta	Previsto
MS	MS
SP	MG
MG	MS
BA	BA
BA	PE
SC	DF
SC	SC
RJ	SP
RS	DF
SP	SP
MA	DF
RJ	RJ
PE	PE
MG	DF
MG	DF
RS	BA
SP	MG
RJ	RJ
RJ	RJ
DF	GO
SP	SP
RN	CE
DF	TO
CE	BA
SP	MG

Rio de Janeiro. Na Tabela II, é possível ver o resultado para 10 testes.

Com uma acurácia de 90%, é possível observar que o modelo conseguiu diferenciar imagens do Rio de Janeiro, e até mesmo quando errava, era em uma região próxima.

TABLE II

TABELA DE RESULTADOS DA PREVISÃO DE MODO

Resposta Correta	Previsto
RJ	RJ
RJ	RJ
RJ	RJ
RJ	RJ
RJ	RJ
RJ	RJ
RJ	RJ
RJ	SP
RJ	RJ
RJ	RJ
RJ	RJ

Testando com outro estado, tal como o Estado de Santa Catarina, conforme Tabela III, observa-se que o resultado do Rio de Janeiro não é uma coincidência de estarmos pegando o modo de um estado específico, já que o resultado do Estado de Santa Catarina obteve 20% de acurácia. No geral, o modelo realmente identificou características daquele estado.

TABLE III

TABELA DE RESULTADOS DA PREVISÃO DE MODO

Resposta Correta	Previsto
SC	DF
SC	SC
SC	MG
SC	BA
SC	BA
SC	MG
SC	DF
SC	PR
SC	DF
SC	SC

Explicando o motivo para isso, tais hipóteses consistem da presença das favelas e também das cores dos táxis que estão sempre presentes na capital do Estado. Na Figura 6 é possível encontrar um exemplo, na qual há dois desses táxis amarelos na mesma foto e o modelo convergiu com apenas duas fotos.

Fig. 6. Figura ilustrado o padrão dos táxis no Rio de Janeiro

Padrões podem também ser vistos para situações em que o mapa está região urbana no Estado de São Paulo, ocasiões nos quais o modelo converge rapidamente e corretamente. Tais padrões são a existência de portões em barras verticais nas casas, o que pode ser algo que o modelo possa ter aprendido. Os padrões podem ser vistos na Figura 7.

Fig. 7. Portões em barra verticais nas casas do estado de São Paulo

V. CONCLUSÃO

Em resumo, o projeto aborda com sucesso a identificação de estados brasileiros a partir de imagens do jogo Geoguessr,

usando uma abordagem de aprendizado de máquina com a arquitetura ResNet50.

Conforme discutido, o modelo apresentou capacidade de aprendizado de padrões visuais, especialmente em estados como o Rio de Janeiro. Independente disso, ainda há desafios, principalmente em regiões geograficamente próximas, tais como regiões do nordeste.

Para alcançar com sucesso esses desafios, projeto oferece uma base sólida para futuras melhorias, incluindo a expansão do conjunto de dados e aprimoramentos na robustez do modelo, já que foi utilizado um modelo pré treinado e já implementado para realizar essa previsão.

REFERENCES

1 Instituto Brasileiro de Geografia e Estatística. *IBGE - Malhas Territoriais*. Acesso em 15/12/2023. Disponível em: ⟨https://www.ibge.gov.br/geociencias/organizacao-do-territorio/malhas-territoriais/15774-malhas.html⟩.