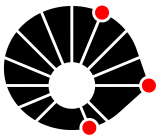

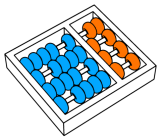

Identificação da origem de trombos em um Acidente Vascular Cerebral usando aprendizado de máquina

L. F. B. Santos *Jacques Wainer*

Relatório Técnico - IC-PFG-22-46
Projeto Final de Graduação
2022 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Identificação da origem de trombos em um Acidente Vascular Cerebral usando aprendizado de máquina

Luís Fernando Barreto dos Santos Jacques Wainer*

Resumo

Nesse trabalho, uma solução de aprendizado de máquina é desenvolvida, a fim de identificar a etiologia de trombos retirados de pacientes que sofreram um acidente vascular cerebral. Para isso, utiliza-se de métodos padronizados presentes em uma *pipeline* capaz de resolver o problema fim-a-fim, distrubuídos através de uma biblioteca Python, Slideflow. O problema e a solução são propostos no ambiente *cloud* de ciência de dados Kaggle e com isso se define as restrições de *hardware* no desenvolvimento, que impossibilitaram a submissão e avaliação do modelo na plataforma. Ainda assim, demonstra-se como a biblioteca utilizada é capaz de criar soluções no ambiente usado através de uma metodologia que divide-as em três *notebooks*. Também são relatados experimentos e a solução final, que não foi capaz de distinguir as classes, são feitas comparações com outras soluções disponíveis na plataforma e por fim se mostra como técnicas mais sofisticadas e dados mais relevantes são necessários para uma solução confiável.

1 Introdução

Em 2019, 12.2 milhões de pessoas sofreram um Acidente Vascular Cerebral (AVC) [1]. Mundialmente, o AVC corresponde à segunda maior causa de morte [2]. A alta recorrência do acidente (cerca de 23% dos acidentes são recorrentes) e as sequelas permanentes muitas vezes deixadas também fazem do AVC um ponto de atenção importante na saúde pública.

O AVC é causado por um coágulo em um vaso sanguíneo que irriga o cérebro. Tal coágulo bloqueia o fluxo de sanguíneo, ou causa uma ruptura no vaso, fazendo com que o sangue deixe de chegar ao cérebro.

Para a remoção dos coágulos, usa-se um procedimento chamado trombectomia mecânica, que consiste em levar um catéter até o vaso afetado, que então remove consigo o coágulo. A obtenção do coágulo significa que ele pode ser analisado e, ao classificá-lo de acordo com sua etiologia, é possível oferecer melhores opções para que o paciente previna um novo episódio.

Nesse contexto, a divisão Neurovascular da Mayo Clinic, uma organização sem fins lucrativos dos Estados Unidos, disponibilizou seus dados de trombectomia e propôs o problema de classificar a origem dos coágulos em duas categorias principais: Cardiembólica (CE, *Cardioembolic*) e ateroesclerose em uma artéria larga (LAA, *Large Artery Atherosclerosis*).

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

O problema foi proposto como uma competição de código na plataforma Kaggle. Uma plataforma de ciência de dados aberta, o Kaggle oferece competições de aprendizagem de máquina de forma fácil de se aderir: os dados já foram coletados, selecionados, documentados e estão disponíveis. A forma com que se avalia a solução apresentada para a competição também é definida, oferencendo um padrão de comparação para todos. Para esta competição, a solução é avaliada através de um conjunto de avaliação escondido, composto por 280 slides, utilizando como métrica a perda logarítmica multi-classe ponderada. Autores de soluções e de dados processados podem disponibilizar seu trabalho abertamente, se quiserem. Outro aspecto importante são as discussões, com os proponentes das competições e entre autores.

Ao estudar soluções para problemas de aprendizado de máquina aplicado à histopatologia, notou-se que em grande parte delas certos procedimentos são comuns, especialmente aqueles relacionados ao pré-processamento dos dados. Nesse contexto, buscou-se *pipelines* que contivessem módulos úteis ou que pudessem ser utilizados fim-a-fim na resolução do problema. Foram encontrados alguns que pudessem ser de ajuda. A solução escolhida foi o *Slideflow* [6] por quatro motivos: ela se insere de forma fácil e acessível, pois é simplesmente uma biblioteca Python, ela é capaz de resolver o problema proposto fim-a-fim, ela suporta os *backends* populares (tensorflow e pytorch) e possui boa documentação e práticas de programação, o que facilita o entendimento e uso.

2 Conceitos

2.1 Área sobre a curva ROC

A curva ROC (*Receiver Operating Characteristic*) é uma distribuição de probabilidades que introduz o grau de separabilidade que o modelo é capaz de realizar sobre as classes. Para um dado limiar de classificação, se mostra a taxa de verdadeiros positivos contra a taxa de falsos positivos. A área embaixo da curva diz a probabilidade do modelo conseguir separar corretamente as duas classes. Uma área de 0.5 nos diz que o modelo não é capaz de distinguir de forma alguma as classes. Como referência, desenha-se uma linha diagonal no gráfico e, para demonstrar a aprendizagem do modelo, precisamos de uma área embaixo da curva maior que a da linha.

2.2 Perda utilizada

Ao treinar um modelo de aprendizado de máquina, há um conflito entre seu desempenho minimizando erros do tipo I (falsos positivos) e erros do tipo II (falsos negativos). No contexto médico, há uma forte preferência por minimizar erros do tipo II já que há uma consequência muito maior em não tratar um problema do que tratar um problema que não existe.

Nesse contexto, para o problema proposto, foi adotada a perda logarítmica multi-classe ponderada dada pela fórmula na Figura 2, onde N_i é o número de imagens de determinada classe, M é a quantidade delas, y_{ij} é igual a 1 se a predição i pertence à classe j e 0 caso contrário, e p_{ij} é a probabilidade estimada da imagem i pertencer à classe j.

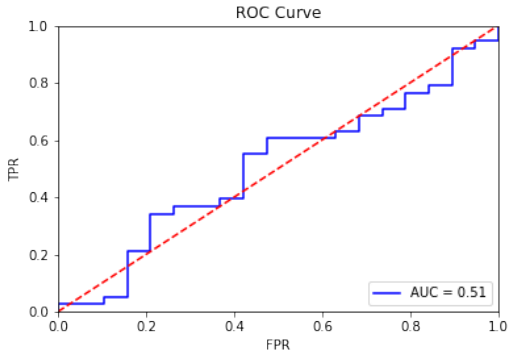

Figura 1: Exemplo de curva ROC, desenhada pela biblioteca Slideflow.

$$\text{Log Loss} = -\left(\frac{\sum_{i=1}^{M} w_i \cdot \sum_{j=1}^{N_i} \frac{y_{ij}}{N_i} \cdot \ln p_{ij}}{\sum_{i=1}^{M} w_i}\right)$$

Figura 2: Fórmula de perda adotada. Adaptado de [5]

No problema, a perda degenera-se para o caso binário com pesos iguais. A perda adotada tem as seguintes características: para uma predição aleatória em um *dataset* balanceado, o erro absoluto médio será de 0,5 para o qual a perda correspondente será de cerca de 0,69. Erros maiores que esse serão fortemente penalizados, enquanto erros menores que esse serão fracamente recompensados. Dessa forma, há um desincentivo massivo à predições confiantes e erradas, que não se balanceia quando tal confiança acerta o alvo. Isso minimiza o erro do tipo II e justifica a escolha da perda.

2.3 Arquitetura EfficientNetV2B0

A rede neural escolhida para treinamento e predição das classes é baseada na arquitetura EfficientNetV2B0 [9], com pesos imagenet [8]. A família de arquiteturas EfficientNet é focada em oferecer maior "custo-benefício", ou seja: otimizar o problema multi-objetivo composto pelo custo computacional e capacidade de aprender o padrão dos dados: quanto mais complexo o padrão que a rede neural é capaz de generalizar, mais custosa ela é computacionalmente.

A proposta da família "V2", de segunda geração, é de focar em minimizar o tempo de treino. A lógica é de que um modelo que treina mais rápido não só é mais acessível como pode comportar mais épocas e regimes de treino no geral, aumentando a eficiência do modelo na prática.

Para aumentar ainda mais a eficiência, os autores propõem o que chamam de *progressive learning*: a rede processa as informações levando em conta o regime de treino em que é submetida, em que imagens começam pequenas e com baixa regularização, depois aumenta

de tamanho e recebem regularização mais forte conforme as épocas passam.

2.4 Whole Slide Images

Os dados brutos utilizados vem são *Whole Slide Images* (WSI, ou "slides"). Os slides vêm de um processo em que se usa um *scanner* para digitalizar lâminas de vidro, permitindo sua análise por computador. Essas imagens são piramidais, o que significa que elas na verdade são um composto de várias imagens, cada uma em um "nível"com uma resolução diferente. O tamanho delas pode variar muito, chegando a centenas de milhares de pixels em uma dimensão, o que pode ocupar 3 GB de memória ou mais. Os diferentes formatos em que os slides podem ser distribuídos, bem como diferentes resoluções de *scanners* utilizados, regiões que podem ficar borradas e a separação entre o objeto de estudo e o plano de fundo são outros desafios que trabalhar com tais imagens representa.

3 Materiais

3.1 Ambiente Kaggle

Em uma competição Kaggle, pode-se desenvolver as soluções localmente, ou compor *notebooks* em linguagem Python ou R, que são executados em cloud. No momento da escrita, se disponibiliza um processador comum de 2.0GHz e quatro opções de aceleradores gráficos. Um *notebook* que utiliza algum acelerador tem disponível dois núcleos de processador e 13GB de memória RAM, caso o contrário são disponibilizados 4 núcleos e 30GB de RAM. No projeto, a opção utilizada foi a de duas GPUs T4 para o treino e inferência de modelos. Para se obter resultados intermediários, se submete o *notebook* ao Kaggle que, crucialmente, não pode exceder o limite de memória ou executar por mais de 12 horas. Após isso, é possível fazer com que *outputs* de *notebooks* intermediários se tornem *inputs* de outros *notebooks*.

3.2 Dados

Os dados utilizados [5] foram extraídos de um estudo [3] que avaliou coágulos de 1350 pacientes que passaram por trombectomia mecânica. Os coágulos, após retirados, passaram por uma coloração adequada e seus trombos foram classificados. O estudo também conta o número de plaquetas, células vermelhas e brancas do sangue e fibrina para então correlacioná-los à classificação feita. Foi mostrado que a quantidade de células vermelhas e a densidade de plaquetas têm correlação estatística com a etiologia do coágulo0

. Os dados são divididos em três conjuntos. O conjunto de treino é composto por 754 WSIs de 632 pacientes. 547 slides foram classificados como CE, outros 207 como LAA, mostrando um forte desbalanceamento entre as classes.

Os outros dois conjuntos são: *other*, composto por 396 slides com coágulo de outra origem ou origem desconhecida e *test*, composto por 280 slides que farão parte da avaliação do modelo. O conjunto *other* é anotado, o conjunto *test* não é anotado e não é possível de ser acessado. Um conjunto de "teste"acessível de 4 slides é posto como referência e é substituído pelo verdadeiro conjunto de teste quando o *notebook* com a solução é submetido.

3.3 Pipeline Slideflow

Cada base de dados utilizada precisa estar catalogada em um arquivo de anotações contendo, obrigatóriamente, uma coluna chamada *patient*, com o código do paciente, e outra chamada *slide*, com o código do WSI da base de dados. Para bases de dados a serem utilizadas no treino supervisionado, é preciso uma coluna contendo a classe anotada. Outras colunas podem ser adicionadas a fim de filtrar as imagens a serem trabalhadas. As principais funções da biblioteca se não através de um objeto chamado *Projeto*. Para configurar um projeto, basta informar caminhos para o projeto como um todo, para o arquivo de anotações e opcionalmente, para arquivos de configurações de dados, diretório de modelos e de dados de validação. O backend utilizado foi o *tensorflow*, que está habilitado por padrão. Para escolher o pytorch, é preciso mudar a variável de ambiente *SF_BACKEND* para 'torch'.

3.3.1 Extração armazenamento de imagens

As imagens de exames médicos no formato WSI podem ser muito grandes e por isso, para treinar o modelo, é necessário extrair imagens menores capazes de serem inseridas na arquitetura escolhida, chamadas *tiles*.

Dois parâmetros básicos para a extração são o tamanho da imagem, em pixels, e partir da resolução em microns por pixel ($\mu m/px$), se passa o tamanho em microns.

O tamanho leva em conta a arquitetura utilizada. Para a EfficientNetV2B0, são 224 pixels. A resolução usada de 0.5 serve para padronizar as imagens, já que WSI de aparelhos diferentes podem ter diferentes resoluções nativas. Assim, determina-se que as imagens representam 112 microns.

A partir daí, pode-se pensar no controle de qualidade. Esses são métodos para retirar regiões que interferem negativamente no treino, aplicados em todo o slide. Para executar tais métodos a biblioteca retira uma *thumbnail* do WSI analisado. Após as operações desejadas, é obtida uma máscara, que então é usada para retirar o que for indesejado. A biblioteca suporta nativamente dois controles de qualidade: um que filtra borrões através de um filtro gaussiano, e outro que retira o plano de fundo utilizando uma limiarização através do método de Otsu [4]. Devido às limitações do ambiente Kaggle e o alto consumo de memória RAM das *thumbnails* usadas, foi possível realizar apenas o controle de borrões. Uma tentativa de melhorar a eficiência de memória fez com que apenas uma parte dos slides conseguissem passar pelos dois métodos, que foi então descontinuada. Para o filtro Gaussiano, foi usado o raio $\sigma = 3$.

Outra forma de fornecer *tiles* de boa qualidade é ignorando aqueles que tiverem uma alta proporção de plano de fundo. Basta fornecer a proporção mínima de tecido com relação ao plano de fundo, e o limiar do que é considerado plano de fundo. Há duas formas de fazer isso. A primeira considera o sistema RGB, em que o brilho é utilizado no limiar. A segunda usa o valor de saturação do sistema HSV. No projeto, a segunda forma é utilizada, com a proporção de 60% e limiar de 0,05.

Após escolher quais imagens serão extraídas, pode-se obter acesso a elas por duas formas. A primeira é, de fato, guardar as imagens de forma comum, em formatos de compressão sem perdas (.png) ou com perdas (.jpg) . Alternativamente, pode-se utilizar *tfrecords*, um

formato binário que consome menos memória em execução (apenas os dados necessários do *batch* são carregados) e em disco. O registro foi feito usando *tfrecords*.

3.3.2 Treino

O treino é feito em duas partes. Primeiro, cria-se uma instância da classe *ModelParams*. Tal objeto guarda informações relativas ao modelo propriamente, bem como informações sobre o treino e *dataset* que impactam na configuração do modelo. No projeto foi usado o tamanho definido de 224 pixels e 112 microns, a instância do modelo utilizado, um batch de tamanho 32, três épocas e a perda implementada. Outros parâmetros foram definidos pelo padrão, entre os mais relevantes estão: *learning rate* de 0.0001, todas as modalidades disponíveis de *data augmentation*: viradas, rotação e compressão em qualidade aleatória e otimizador Adam [7].

A partir disso, se realiza uma chamada para o método de treino, *Train()* do objeto *Project* criado, que leva parâmetros referentes ao contexto, como qual é a coluna no arquivo de anotações que contém as classes para se treinar, o objeto *ModelParams* criado, se o treino é uma continuação de outro, bem como ao regime de treino e validação, destacando-se: se haverá multiplas GPUS (treinadas sincronamente utilizando uma estratégia espelhada), como será feita a divisão entre conjuntos de validação e treino se não houver previamente, se haverá balanceamento desses conjuntos a partir de *tiles*, pacientes, slide, categoria (classe). No projeto, se utiliza as duas GPUs T4 disponibilizadas pelo ambiente Kaggle, a divisão dos conjuntos é feita por 5 *k-folds*, com balanceamento de categoria no treino (ou seja, no treino, para cada *batch*, se fornece quantidades iguais de tiles vindos de slides classificados com LAA e CE).

4 Métodos

4.1 Contribuições ao software

A biblioteca está amadurecendo, com isso alguns bugs foram notados. O uso de uma plataforma cloud e com recursos limitados como o Kaggle também foram uma forma de testá-la. Dentro desse contexto, criou-se algumas *issues* no *github*. Das 6 *issues* criadas, 5 levaram a um aprimoramento na biblioteca.

A *loss* utilizada no problema não está implementada no *tensorflow* ou em outra biblioteca relevante. Com isso, foi necessária implementá-la. Tomando um tensor que contém a saída do modelo e um tensor contendo as anotações de classe, codificada em um inteiro, para cada dado em um *batch*, calcula-se a perda. A biblioteca, porém, não havia suporte à perdas customizadas. Ela esperava uma *string* para saber qual *loss* utilizar, usando um dicinário que pareava tal *string* com a função de perda no backend escolhido. Caso a perda escolhida não estivesse presente, notadamente se não implementada pelo *backend* utilizado, não havia como treinar o modelo. Com isso, para realizar o projeto, foi implementada a funcionalidade de perdas customizadas, onde, ao passar um dicionário contendo o tipo de perda e a função de perda implementada, consegue-se treinar e validar o modelo com ela.

4.2 Experimentos

Para realizar os experimentos, teve de se levar em conta as limitações de *hardware* e ambiente *cloud* do Kaggle. Com isso, os experimentos podem se dividir da seguinte forma: é criado um notebook para se extrair os *tiles*, outro para realizar o treinamento, outro para submeter à competição e por fim avaliar o modelo.

O notebook de extração lida com o fato de que o conjunto de dados com o WSI fornecido só permite leitura. Tal conjunto possui mais de 100GB. O diretório de trabalho, que permite escrita, contém apenas 20GB. Com isso, foi necessário iterar sobre os WSI para copiar ao diretório de trabalho, realizar o controle de qualidade, extrair os *tiles* e enfim excluí-los.

Já o *notebook* de treino cria os diretórios e o objeto de projeto necessários, cuida da formatação do arquivo de anotação, implementa a função de perda, define os *ModelParams* e chama o treino com os argumentos necessários.

Os experimentos 1, e 2 serviram para se testar hipóteses e utilizaram cerca de 1/3 dos dados.

4.2.1 Experimento 1

O primeiro experimento teve como motivação servir de base. Os parâmetros usados foram os detalhados na seção de métodos. As métricas obtidas pelos slides mostram que o modelo não conseguiu distinguir as classes. Para o subproblema de classificar *tilles*, o modelo performou melhor do que a escolha aleatória, porém pouco melhor. A pior medida é a variação entre as áreas sobre as curvas ROC, que varia tanto negativamente quanto positivamente. A variação negativa e significativa, mostra que o modelo não está aprendendo a realizar a tarefa, sobretudo quando presente na classificação de *tiles*.

4.2.2 Experimento 2

A variação nas métricas levou a pensar que o ritmo de aprendizado está rápido demais e que portanto o modelo não consegue convergir. Com isso, diminui-se o *learning rate* para um décimo do anterior, 0.00001. Isso, porém, não só deixou a aproximadamente a mesma variação da métrica como limitou o desempenho do modelo, que na média classifica pior do que aleatóriamente.

4.2.3 Experimento final

Com isso, optou-se por seguir o caminho dado no experimento 1, com a expectativa de que mais dados possam fazer com que o modelo demonstre estar aprendendo ou desempenhe melhor no geral. De fato, o modelo performa melhor na média e tem uma variação menor na métrica para *tiles*. Isso, porém, se transfere em menor medida para os slides, evidenciando a necessidade de uma análise dupla: a do subproblema e do problema geral.

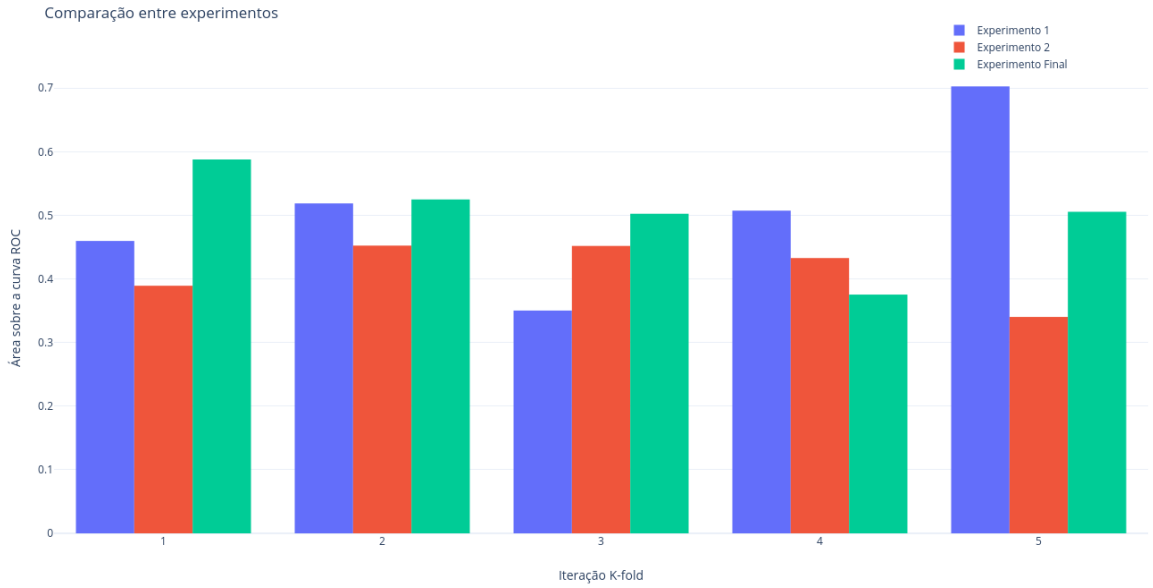

Figura 3: Área embaixo da curva ROC para slides.

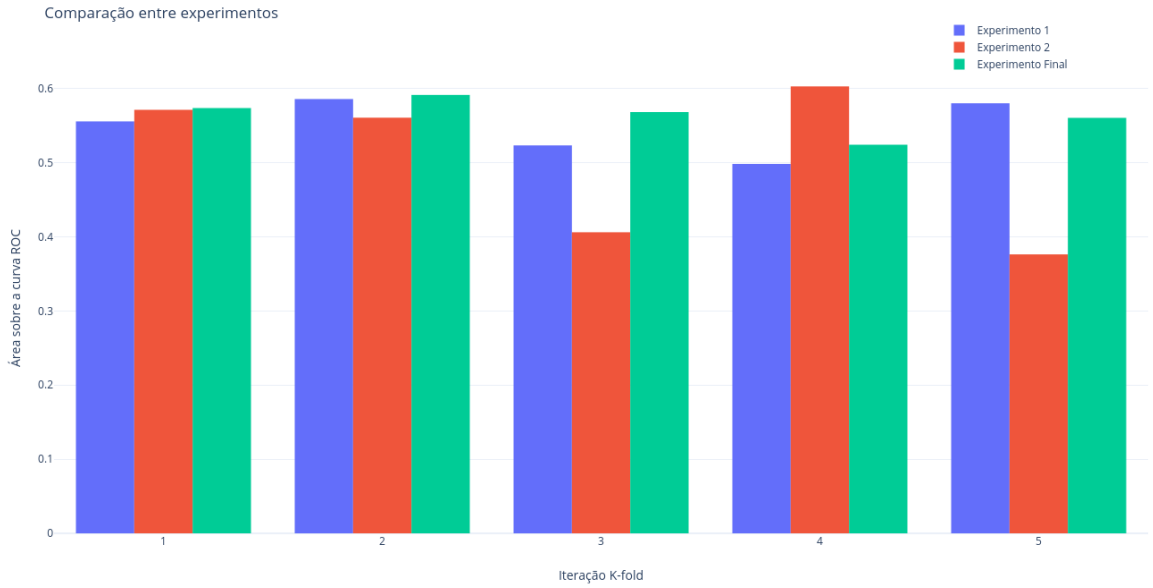

Figura 4: Área embaixo da curva ROC para tiles.

5 Resultados

5.1 Submissão à plataforma

Tomando o experimento final, o modelo gerado foi exportado para um outro *notebook*, que configurou normalmente um projeto *Slideflow* e o arquivo de anotações, trocando os dados de treino por os de teste.

A partir daí, pode-se extrair os *tiles* e guardá-los em *tfrecords*, depois realizar a inferência com o modelo, gerando um dicionário com a probabilidades de classificação para cada slide, que então é escrito em um arquivo .csv. Com isso a perda calculada e o modelo avaliado.

Isso não foi possível de ser feito, porém, devido às restrições de hardware da plataforma Kaggle. O notebook precisa realizar tarefas com alto uso de CPU, como a extração, e tarefas intensivas para a GPU na inferência. Com isso, há a necessidade de se utilizar um acelerador, que reduz a quantidade de núcleos de processamento disponíveis e memória RAM. Cada slide toma em média 5 minutos para ter seus *tfrecords* criados, excedendo o tempo de execução permitido ao aplicar o método para 280 slides. Outro fator limitante foi a memória RAM, já que sua redução implicou em uma alocação maior que a permitida para alguns slides, encerrando a execução.

5.2 Comparação e Trabalhos Futuros

Na página da competição, é possível ver alguns dos melhores resultados na competição. Em primeiro lugar, com a menor perda avaliada, obteve 0.65993, o que mostra que sua solução desempenha um pouco melhor do que uma solução aleatória. Ela pode ser melhor do que isso, porém, caso o *dataset* de teste não esteja balanceado.

De qualquer forma, a competição mostrou que a solução para o problema proposto pode ser muito complexa, isto é, são necessárias soluções de aprendizado de máquina mais sofisticadas e mais caras computacionalmente, bem como pode exigir mais metadados, mais dados ou outras características dos coágulos, como sugere a própria pesquisa que colheu o conjunto de dados [3].

6 Conclusão

No geral, foram discutidas as relações entre três questões: a resolução do problema proposto, o uso de uma *pipelinde* composta em uma biblioteca Python e o ambiente *cloud* de ciência de dados Kaggle. Se mostra como se pode costurar uma solução a um problema de histopatoligia aplicado à aprendizado de máquina contribuindo com o desenvolvimento da biblioteca, utilizando seus módulos de forma adequada ao problema e planejando o uso de *notebooks* para usar dos recursos da plataforma. Teoricamente, é possível construir uma solução fim-a-fim utilizando a biblioteca como estimado, porém são necessários recursos próprios ou de serviços pagos de computação *cloud*, o que demonstra as limitações do contexto adotado. Ainda assim, Kaggle e Slideflow são duas ferramentas importantes no sentido de se desenvolver experimentos que podem render soluções inovadoras na área.

Apesar de não ter sido possível avaliar a solução desenvolvida com um conjunto de teste fechado, pode-se dizer a partir dos dados de validação que ela apresenta um grau baixíssimo de separabilidade entre as classes anotadas, e com isso conclui-se de que é necessário um método mais sofisticado e uma reavaliação da relevância dos dados apresentados para a resolução do problema.

Referências

[1] GBD 2019 Stroke Collaborators. *Global, regional, and national burden of stroke and its risk factors, 1990-2019: a systematic analysis for the Global Burden of Disease Study 2019*. The Lancet. Neurology vol. 20,10 (2021): 795-820.

[2] Donkor E. S. (2018). *Stroke in the 21st Century: A Snapshot of the Burden, Epidemiology, and Quality of Life*. Stroke research and treatment, 2018, 3238165.

[3] Brinjikji, W., Nogueira, R. G., Kvamme, P., Layton, K. F., Delgado Almandoz, J. E., Hanel, R. A., Mendes Pereira, V., Almekhlafi, M. A., Yoo, A. J., Jahromi, B. S., Gounis, M. J., Patel, B., Abbasi, M., Fitzgerald, S., Mereuta, O. M., Dai, D., Kadirvel, R., Doyle, K., Savastano, L., Cloft, H. J., ... Kallmes, D. F. (2021).*Association between clot composition and stroke origin in mechanical thrombectomy patients: analysis of the Stroke Thromboembolism Registry of Imaging and Pathology*. Journal of neurointerventional surgery, 13(7), 594–598.

[4] N. Otsu, *A Threshold Selection Method from Gray-Level Histograms*. IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62–66 (Jan. 1979).

[5] Ashley Chow, Barbaros, HCL-kanishkaa, Ryan Holbrook, Sobhi Jabal, VikashGupta. *Mayo Clinic - STRIP AI*. Kaggle, 2022.

[6] James Dolezal, Sara Kochanny, Frederick Howard. *Slideflow: A Unified Deep Learning Pipeline for Digital Histology*. Zenodo, 2022.

[7] Kingma, Diederik P., Ba, Jimmy, *Adam: A Method for Stochastic Optimization*. arXiv, 2014.

[8] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. *Imagenet large scale visual recognition challenge*. International Journal of Computer Vision, 115(3): 211–252, 2015.

[9] Tan, Mingxing and Le, Quoc V., *EfficientNetV2: Smaller Models and Faster Training*, Computer Vision and Pattern Recognition (cs.CV), FOS: Computer and information sciences, FOS: Computer and information sciences. arXiv, 2O21.