PathoSpotter Classifier: Uma Serviço Web para Auxílio à Classificação de Lesões em Glomérulos Renais

Sarah P. Cerqueira[1], Ellen C. Aguiar[1], Angelo A. Duarte[1], Washington L.C. dos Santos[2], Luciano R. Oliveira[3], Michele F. Ângelo[4]

[1] Laboratório de Computação de Alto Desempenho
Universidade Estadual de Feira de Santana – Feira de Santana, BA – Brasil

[2]Instituto Gonçalo Moniz
Fundação Oswaldo Cruz – Salvador, BA – Brasil

[3]Laboratório iVision
Universidade Federal da Bahia – Salvador, BA – Brasil

[4]Departamento de Ciências Exatas
Universidade Estadual de Feira de Santana – Feira de Santana, BA – Brasil

{cerqueira.sarahp,chalegreaguiar}@gmail.com, angeloduarte@uefs.br

wluis@bahia.fiocruz.br, lreboucas@ufba.br, mfangelo@uefs.br

Abstract. *In recent years, the PathoSpotter Project has developed and perfected classifiers to aid in the diagnosis of lesions in digital images of renal biopsies. Among the goals of the project is the availability of these classifiers so that pathologists can use them to facilitate their medical practice and also contribute to the improvement of the system. This work presents the architecture of the PathoSpotter Classifier, the Web service created by the PathoSpotter Project, and how the challenges faced in distributing the system for use by pathologists were overcome.*

Resumo. *Nos últimos anos, o Projeto PathoSpotter desenvolveu e aperfeiçoou classificadores para auxílio ao diagnóstico de lesões em imagens digitais de biópsias renais. Entre as metas do Projeto está a disponibilização destes classificadores para que patologistas possam utiliza-los de forma a facilitarem sua prática médica e também contribuírem para o aprimoramento do sistema. Esse trabalho apresenta a arquitetura do PathoSpotter Classifier, o serviço Web criado pelo Projeto PathoSpotter, e como foram vencidos os desafios enfrentados para a distribuição do sistema para uso por patologistas.*

1. Introdução

O envelhecimento populacional somado ao aumento da incidência de hipertensão, diabetes e doenças cardiovasculares projetam a Doença Renal Crônica (DRC) como um dos maiores desafios à saúde pública mundial deste século [Silva et al. 2020]. A DRC se caracteriza por lesões nos rins que causam anormalidades estruturais ou funcionais, com ou sem diminuição da Filtração Glomerular (FG). Quando a FG atinge valores muito baixos, estabelece-se a Falência Funcional Renal (FFR) [Bastos et al. 2010].

Além do sofrimento que a FFR causa a seu portador, os custos do tratamento são onerosos tanto para seus familiares quanto para o sistema de saúde. O relatório dos Gastos do Sistema Único de Saúde Brasileiro com Doença Renal Crônica de 2018 indica que o Brasil gastou mais de 2 bilhões de reais com diálise e transplante em 2015 [Alcalde and Kirsztajn 2018]. Felizmente, essa situação pode ser prevenida ou retardada se houver um diagnóstico precoce da doença renal, evitando que ela atinja seu estado crônico [Bastos et al. 2010].

A terceira maior causa da Doença Renal Crônica são as glomerulopatias, uma família de patologias primárias que atingem os glomérulos renais. O diagnóstico de glomerulopatias é realizado a partir de dados clínicos e laboratoriais, associados à análise de imagens de biopsias renais [Sweet et al. 2011] diretamente ao microscópio ou digitalizadas. Essas análises são feitas por um patologista e muitas vezes estão sujeitas a subjetividades que interferem na avaliação da imagem e geram divergências ou dificultam o fechamento do diagnóstico.

Uma das possíveis estratégias para enfrentar o problema da variabilidade das análises de imagens de biópsias renais é desenvolver métodos automáticos de análise, mais especificamente para detecção e classificação de lesões glomerulares. Dentro dessa abordagem surgiu o projeto PathoSpotter, que se propõe desenvolver sistemas computacionais para auxiliar o processo de análise de imagens de biópsias renais. Uma das metas do projeto é desenvolver um classificador de lesões glomerulares e disponibiliza-lo para acesso a patologistas.

Até agora o projeto PathoSpotter já desenvolveu classificadores para dois tipos de lesões glomerulares, esclerose e hipercelularidade [Barros et al. 2017, Chagas et al. 2020, de Araújo et al. 2017], obtendo resultados de até $94,5\%$ de acurácia. Enquanto outros classificadores estão sendo desenvolvidos, partiu-se para o trabalho de distribuição dos classificadores para que outros patologistas possam avaliar a solução e ajudar a melhora-la.

Rapidamente observou-se que a linguagem e as bibliotecas de *softwares* usadas para o desenvolvimento dos classificadores dificultavam muito a distribuição de um programa que fosse facilmente instalável nos computadores dos patologistas. Como solução, optou-se por desenvolver um sistema Web, que permitiria que qualquer patologista pudesse acessar os recursos do Pathospotter apenas usando um navegador na Internet. Este artigo descreve todo o processo de desenvolvimento do sistema Web para o classificador de lesões, chamado de *PathoSpotter Classifier*.

2. PathoSpotter Classifier

Desde 2015 o Projeto PathoSpotter[1] vêm implementando e melhorando modelos capazes de classificar lesões glomerulares em imagens digitais de biópsias renais. As versões mais recentes dos classificadores que serão distribuídas identificam lesões do tipo hipercelularidade (mesangial, endocapilar e combinadas) e esclerose glomerular [Chagas et al. 2020, de Araújo et al. 2017].

O classificador foi desenvolvidos em linguagem *Python*, usando as bibliotecas *TensorFlow*, *Keras* e *Numpy*. Para a primeira versão foi criada uma interface gráfica sim-

[1] `http://pathospotter.bahia.fiocruz.br/`

ples, com as principais operações necessárias para o uso do sistema. A criação de um instalador para a versão distribuível do classificador foi feita usando a biblioteca *PyInstaller*, mas rapidamente concluiu-se que o tamanho final do instalador ($+100$ MiB) era inconveniente para distribuição. Além disso, os experimentos de distribuição mostraram que haveriam dificuldades durante a instalação em função da versão do sistema operacional da máquina destino, que exigiriam um nível de conhecimento de informática pouco comum a médicos patologistas.

A solução natural foi a criação de uma versão do classificador que pudesse funcionar como um serviço Web no servidor do projeto, dispensando qualquer necessidade de instalação de software por parte do patologista, o qual poderia usar o sistema através de um simples navegador na Internet. As pesquisas indicaram que a abordagem mais prática seria o desenvolvimento de um sistema Web com Python. Esse texto descreve com os modelos de classificadores do PathoSpotter foram adaptados para um serviço Web e como os desafios dessa adaptação foram vencidos.

3. Tecnologias Utilizadas

O desenvolvimento de um sistema Web com linguagem Python requer o uso de um *framework* e um *middleware* WSGI (*Web Server Gateway Interface*) para gerir a comunicação entre a aplicação Python e o servidor HTTP. No momento em que se iniciou o desenvolvimento da versão do Web classificador, os frameworks mais populares disponíveis gratuitamente eram o Django (`https://www.djangoproject.com`), o Flask (`http://flask.pocoo.org/`), o web2py (`http://www.web2py.com/`), o CherryPy (`http://www.cherrypy.org/`) e o Bootle (`https://bottlepy.org/`). Por falta de um critério mais objetivo, optou-se pelo Django por se apresentar como uma solução robusta com vasta documentação disponível.

Em termos do *middleware* WSGI, foram encontradas diversas opções gratuitas disponíveis na Internet e, novamente por falta de um critério mais objetivo, optou-se pelo Gunicorn por sua premissa de uma solução robusta com vasta documentação disponível. Embora a documentação do Django indicasse a possibilidade de sua integração direta com o Apache, na prática esse integração trouxe instabilidade ao servidor e por isso optou-se por usar o Gunicorn.

3.1. Framework para desenvolvimento Web usando Python

Django é o segundo web framework *Python* mais usado de acordo com a pesquisa *Python Developers Survey 2020* [JetBrains 2020]. Sua arquitetura é do tipo *Model*, *Template* e *View* (MTV) e visa aumentar a produtividade do programador, facilitando o reaproveitamento de códigos já escritos, evitando a repetição de funções e configurações.

Nesse modelo, as requisições do usuário são enviadas através do módulo *Template*, que assume o papel de *front-end* da aplicação. O módulo *View* é responsável pela formatação dos dados que serão enviados ao *Model* ou retornados para o *Template*. Por sua vez, o módulo *Model* fica responsável pela manipulação dos dados da aplicação, que pode ser através de uma transação com um gerenciador de banco dados ou outro tipo de processamento necessário. Essas relações são ilustradas na Figura 1.

Conforme explicado pelos desenvolvedores do Django [Mozilla 2021], o fluxo de uso começa com uma requisição HTTP que, com base na URL de solicitação e possíveis

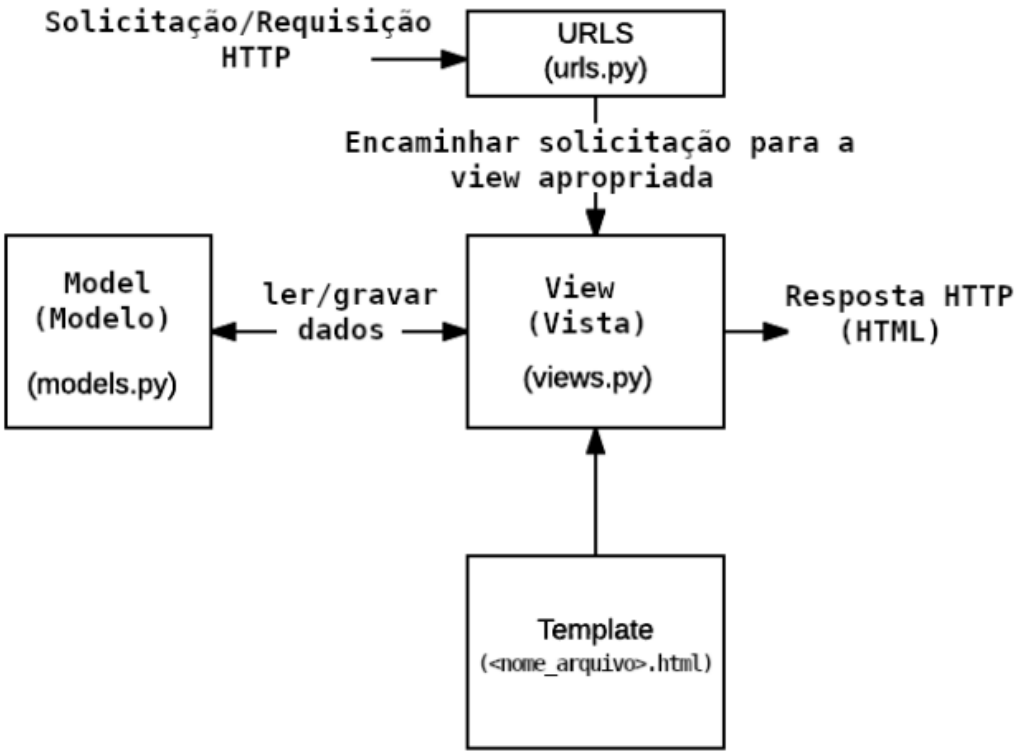

Figura 1. Arquitetura do framework Django.

dados *POST* ou *GET*, é mapeada para uma função *View* especifica, a qual realizará as manipulações destes dados de forma a satisfazer à requisição. O resultado do processamento será retornado via uma página HTML criada dinamicamente com base num *template*, na qual é inserida a resposta da solicitação. Por fim, essa página é retornada ao navegador web do usuário.

3.2. *Middleware* WSGI

O *Web Server Gateway Interface* (WSGI) é um padrão *Python* que especifica uma interface de comunicação entre o servidor Web e a aplicação, conforme detalhado no documento PEP 3333 [Eby 2010]. Resumidamente, o emprego desse padrão consiste na criação de um objeto dentro da aplicação, o qual pode ser acessado pelo servidor HTTP. Assim, quando uma requisição compatível com WSGI chega ao servidor, ela é transferida para aplicação através desse objeto. Por sua vez, a aplicação resolve a requisição e o resultado é retornado ao servidor, que transfere a resposta ao usuário.

O Gunicorn é um popular *middleware* WSGI para Unix, que usa um modelo pré-bifurcado de processos de trabalho [Chesneau 2020]. A bifurcação em sistemas Unix é uma função que cria novos processos a partir da duplicação de um processo já existente. Um modelo pré-bifurcado significa que a bifurcação acontece com antecedência, assim o processo filho está imediatamente disponível quando necessário, diminuindo a latência de resposta da aplicação.

No modelo de servidor do Gunicorn, existe um processo mestre central que gerencia uma lista de processos trabalhadores em execução, os quais atendem as demandas de processamento recebidas pela aplicação. O processo mestre pode aumentar ou diminuir

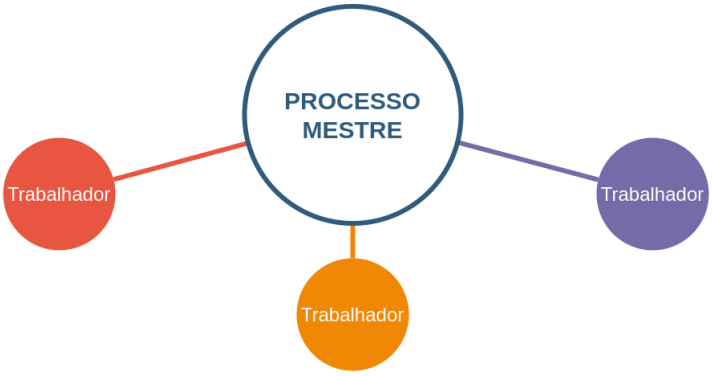

Figura 2. Esquema do funcionamento do Gunicorn.

o número de processos trabalhadores de acordo com a demanda que o sistema está recebendo, além de ser responsável por reiniciar automaticamente os processos trabalhadores em estado de erro. Cerca de 4 a 12 processos de trabalho são suficientes para lidar com centenas ou milhares de solicitação por segundo [Chesneau 2020].

Os processos trabalhadores podem ser dos seguintes tipos: Síncrono, Assíncrono, Tornado e *asyncIO*. O trabalhador Síncrono é o padrão do Gunicorn e trata somente uma requisição por vez, além de não oferecer suporte ao uso de conexões persistentes. Isto significa que toda conexão é fechada após o envio da resposta. Este tipo de trabalhador é indicado para aplicações que são limitadas a partir de recursos de CPU e largura de banda da rede.

Já o tipo Assíncrono é implementado com base em *greenlets*, que são rotinas leves construídas para permitir a execução simultânea em processos [Rigo and Tismer 2011], similar a uma *thread*. Qualquer aplicação pode usar esses tipos de trabalhadores, mas eles são mais indicados para aquelas que possuem tempo de resposta indefinido como as API, por exemplo.

O trabalhador da classe Tornado é aquele que segue a estrutura do *framework* e biblioteca de rede Python Tornado [Tornado 2020]. Ele funciona de forma assíncrona, com entrada e saída que não bloqueia o processo, sendo ideal para aplicações que possuem tempo de resposta longo. Por fim, a classe de trabalhador *asyncIO* opera usando *threads*. Cada processo de trabalho cria *threads* para lidar com as requisições que chegam no servidor, permitindo o compartilhamento de memória. Essa classe de trabalhador é indicada para aplicações que possuem grande limitação de CPU.

3.3. Servidor HTTP

O Apache é um servidor HTTP de código aberto bastante usado com plataforma base para serviços Web [Apache 2020]. Entre os diversos recursos disponibilizados pelo Apache, um dos que foi utilizado nesta solução foi o *proxy* reverso. Esse recurso funciona como um *gateway* entre a Web e os servidores internos de uma rede. Desta forma, o Apache não gera ou hospeda os dados, ele redireciona as requisições para um servidor responsável, o qual processa a requisição e envia o resultado de volta ao Apache, que o retransmite para o cliente Web. Uma implementação comum do *proxy* reverso é apresentada na Figura 3.

Embora o *proxy* reverso seja requerido por conta do uso do Gunicorn na implementação da solução, esse mecanismo também ajuda a aumentar a segurança, já

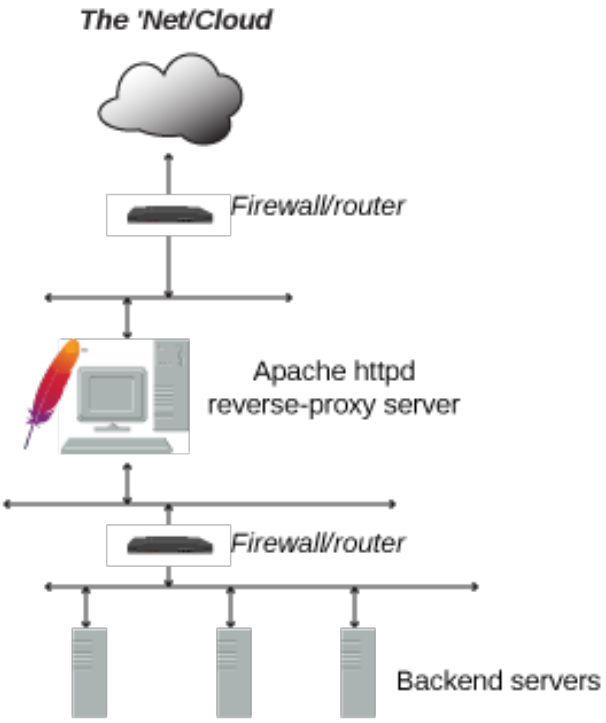

Figura 3. *Proxy* reverso no Apache [Apache 2020].

que o servidor da aplicação não fica diretamente exposto à Internet, facilita o balanceamento de carga, quando a quantidade de requisições demandar mais de um servidor de aplicação e permite alta disponibilidade no sistema, com o uso de servidores de aplicação redundante. Embora estas características ainda não estejam sendo exploradas, elas serão úteis com o aumento do uso do *PathoSpotter Classifier*.

4. *PathoSpotter Classifier*

A arquitetura final do sistema é apresentada na Figura 4. Optou-se por usar o Django apenas para a execução do código dos classificadores, enquanto as demais funcionalidades do site foram desenvolvidas usando HTML e CSS puro. O fluxo de dados ocorre da seguinte forma: primeiramente o patologista submete a imagem para classificação através de um formulário na página `http://pathospotter.bahia.fiocruz.br/pathospotterclassifier`, como demonstrado no primeiro passo da Figura 5.

Usando um navegador Web como interface, o patologista submete uma imagem para classificação. Essa requisição é processada pelo servidor HTTP (Apache), o qual, ao tratar da URL de requisição, interpreta que o requerimento é destinado ao servidor da aplicação e o encaminha para o Gunicorn.

No Gunicorn, a solicitação é tratada através de um trabalhador que carrega a aplicação Python e, usando WSGI, repassa a solicitação para o Django. Este, por sua vez, recebe a imagem que veio do servidor HTTP e a envia para a aplicação Python fazer a classificação. O resultado da aplicação Python (tipo de lesão na imagem) realiza o mesmo percurso de maneira inversa, até que é apresentado na tela do navegador do patologista.

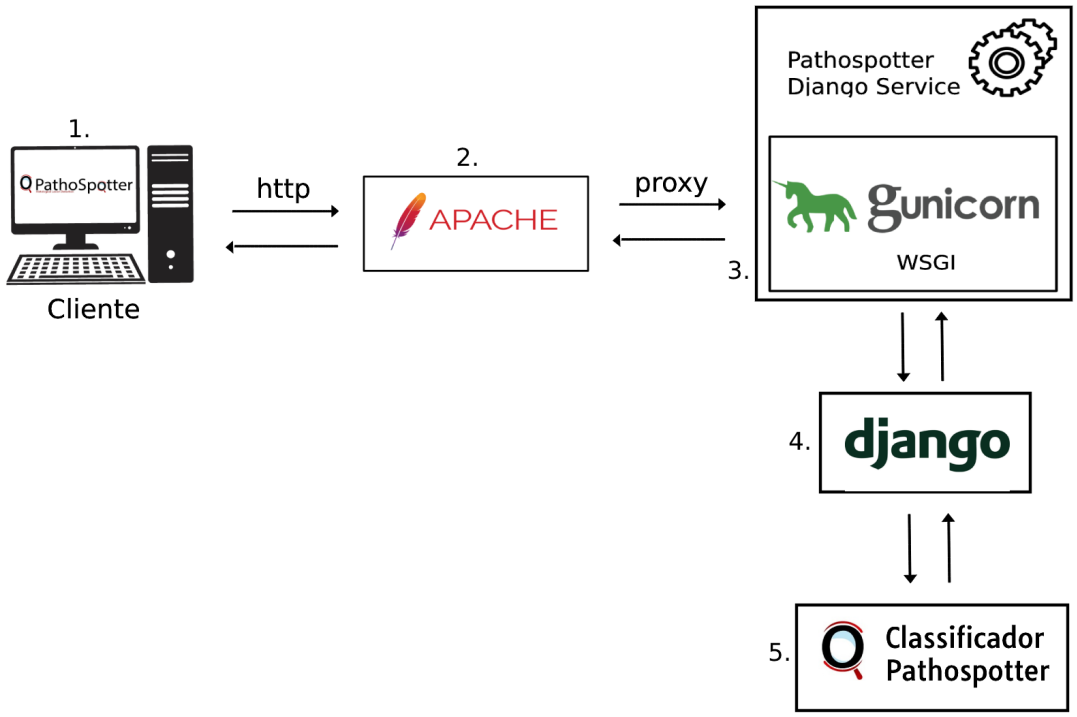

Figura 4. Arquitetura do *PathoSpotter Classifier*.

4.1. Classificadores

Os modelos de classificação de hipercelularidade e esclerose foram desenvolvidos à parte e integrados em um único classificador. Os códigos dos classificadores foram adaptados para a estrutura de classes e isolados dos demais dados, a fim de facilitar a integração e manutenção. Posteriormente se implementou a página `http://pathospotter.bahia.fiocruz.br/pathospotterclassifier`, responsável pela interface com os usuários.

4.2. Integração do Gunicorn ao Apache

O Gunicorn é simples de configurar, mas não tem módulos prontos que facilitem a integração com o Apache. Em comunidades de desenvolvedores, observou-se que o mais comum é unir o Gunicorn com o servidor Nginx (`https://www.nginx.com`) para servir aplicações construídas usando o *framework* Flask. Porém, pouco se encontra sobre o funcionamento conjunto do Apache com o Gunicorn. Tomando como referência uma implementação de Nginx com Gunicorn, construiu-se o serviço *Pathopotter Django Service* para disponibilizar o Gunicorn como um processo executando continuamente em *background* (*daemon* do Linux).

Com o Gunicorn operacional, configurou-se um *proxy* reverso no Apache, de forma a redirecionar as requisições da página Web para o servidor Gunicorn. Dado que o Gunicorn não é apropriado para servir arquivos estáticos e mídias, também foi necessário criar um *proxy* reverso para que estes arquivos fossem servidos pelo Apache.

Figura 5. Página Web para submissão de imagens.

5. Testes de funcionamento

Os testes iniciais mostraram que o sistema era bastante instável, apresentando diversos erros 502, que foram identificados como relacionados à comunicação entre o Apache e o Gunicorn, além de tempo de resposta aleatório para classificação. Para diagnosticar a origem do problema, realizaram-se diversos experimentos para coletar dados de tempo de resposta e identificar quais requisições demoravam mais.

Primeiramente, simulou-se o ambiente num computador local usando somente o classificador de hipercelularidade, para executa a classificação via terminal linux. Foram utilizadas 12 imagens de tamanhos variados, que produziram tempos de classificação muito similares entre si, com uma média de $5,216$ segundos aproximadamente (Tabela 1). Conclui-se que esse tempo seria insuficiente para provocar o erro que estávamos enfrentando.

Com base nestes resultados, a configuração do servidor Apache foi ajustada para aumentar os tempos de *timeout* e retenção da conexão e mesmo assim o problema não se resolveu. Admitiu-se a hipótese de que o tempo de processamento dos códigos Python estaria sendo superior ao tempo de *timeout*. Com base nesta hipótese, mediu-se o tempo de execução do código do classificador e conclui-se que o tempo de carregamento da biblioteca *TensorFlow* era o que mais influenciava no tempo de execução total.

Os próximos esforços foram concentrados na comparação destes tempos e observou-se que na máquina local o tempo de carregamento da biblioteca era muito redu-

Tabela 1. Teste de processamento do classificador na máquina local.

Imagem	Tamanho (KiB)	Tempo (s)
img1	25,9	5,468
img2	37,2	5,173
img3	89,2	5,307
img4	134,6	5,771
img5	257,1	5,218
img6	363,7	5,402
img7	446,0	5,394
img8	790,7	5,422
img9	855,8	5,577
img10	1100,0	5,251
img11	1600,0	5,364
img12	5300,0	5,645

zido a partir do segundo carregamento da página. Contrariamente, notou-se que ao usar o servidor HTTP remoto este comportamento não se reproduzia. A Tabela 2 mostra os resultados obtidos para um dos experimentos.

Tabela 2. Teste de importação do *TensorFlow*

Ordem	Local (s)	Servidor (s)
1^a	15,305	19,637
2^a	1,157	13,217
3^a	1,090	10,849
4^a	1,031	24,146

Com o problema identificado, o próximo passo eram as melhorias no código dos classificadores visando aumentar o desempenho de cômputo. Também foram pesquisadas formas de manter o *TensorFlow* carregado na memória do servidor, para que ele não fosse importado sempre que chegasse uma nova requisição.

Como explicado anteriormente, o processo trabalhador do tipo Síncrono é o tipo padrão do Gunicorn. Como os trabalhadores Sícronos são processos que não compartilham memória, quando um deles recebe a requisição é preciso carregar todas as bibliotecas, inclusive o TensorFlow. Supondo que a primeira requisição é entregue ao trabalhador 1, a segunda pode ser entregue ao trabalhador 1, mas também ao 2 ou 3. Considerando que o tempo de carregamento da biblioteca TensorFlow em memória foi identificado como o grande causador da latência de resposta do sistema, entendeu-se a necessidade de assegurar que a biblioteca fosse carregada apenas uma vez e mantida em memória para ser compartilhada com os demais processos Gunicorn ativos.

Uma análise detalhada dos tipos de processos do Gunicorn indicou que um processo trabalhador do tipo Assíncrono seria mais adequado para atender ao requisitos anteriormente explicitados. O processo trabalhador padrão do Gunicorn foi substituído por um único processo trabalhador do tipo *gevent*, que cria rotinas em pararelo para lidar com as requisições simultâneas que chegam no servidor.

Os mesmos experimentos foram refeitos e o resultado indicou um aumento no desempenho de classificação usando servidor, cujo tempo baixou para uma média de $7,39 \pm 1,15$s. Vale registrar que o tempo do primeiro carregamento da biblioteca TensorFlow seguiu igual ao do primeiro experimento. O que mudou foi que todas as requisições após a primeira já encontravam a biblioteca em memória. No jargão para desenvolvimento Web, o conceito de assegurar que todas as bibliotecas estão em memória antes da primeira requisição se chama de "aquecer o sistema".

6. Conclusão

Neste texto apresentou-se a necessidade que motivou o desenvolvimento de um sistema Web para auxiliar patologistas na classificação de lesões glomerulares e os desafios enfrentados ao longo do processo. Como resultado do trabalho, foi disponibilizado o serviço *PathoSpotter Classifier* (`http://pathospotter.bahia.fiocruz.br/pathospotterclassifier`) que permite a patologistas utilizarem os classificadores desenvolvidos no Projeto PathoSpotter através de uma página Web, sem a necessidade de instalação de qualquer *software* em suas estações de trabalho.

A solução foi baseada nos softwares em Django, Gunicorn e Apache e o desempenho inicial foi muito ruim (chegando até $1,4$ min para uma classificação). Isto por consequência da latência para carregamento em memória das bibliotecas Python, principalmente a TensorFlow. O problema foi resolvido configurando o Gunicorn para funcionar com apenas um processo trabalhador do tipo Assíncrono. Essa configuração viabiliza que o sistema fique aquecido após as duas primeiras classificações, minimizando o tempo de resposta do sistema.

Nesse trabalho foram utilizados dois modelos de classificação (hipercelularidade e esclerose), mas outros classificadores estão em fase de desenvolvimento e poderão ser facilmente integrados ao sistema Web tão logo estejam validados.

No momento, os pesquisadores do Projeto PathoSpotter também estão desenvolvendo uma API (*Application Program Interface*) para simplificar a integração dos classificadores a qualquer página Web.

Agradecimentos

O PathoSpotter é parcialmente patrocinado pela Fundação de Apoio à Pesquisa do Estado da Bahia (FAPESB), bolsa nº TO-P0008/15 e TO-SUS0031/2018 e Edital Inova FIOCRUZ - Ideias inovadoras. Washington Santos e Luciano Oliveira são bolsistas de pesquisa do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), bolsas 306779/2017 e 307550/2018-4, respectivamente. Ellen Aguiar é bolsista de Iniciação Científica, bolsa 3627/2020.

Referências

Alcalde, P. R. and Kirsztajn, G. M. (2018). Gastos do sistema único de saúde brasileiro com doença renal crônica. *Brazilian Journal of Nephrology*, 40(2):122–129.

Apache, S. F. (2020). Welcome! - the apache http server project. `https://httpd.apache.org`. Acessado em 19 de março de 2021.

Barros, G. O., Navarro, B., Duarte, A., and Dos-Santos, W. L. (2017). Pathospotter-k: A computational tool for the automatic identification of glomerular lesions in histological images of kidneys. *Scientific reports*, 7(1):1–8.

Bastos, M. G., Bregman, R., and Kirsztajn, G. M. (2010). Doença renal crônica: frequente e grave, mas também prevenível e tratável. *Revista da Associação Médica Brasileira*, 56(2):248–253.

Chagas, P., Souza, L., Araújo, I., Aldeman, N., Duarte, A., Angelo, M., Dos-Santos, W. L., and Oliveira, L. (2020). Classification of glomerular hypercellularity using convolutional features and support vector machine. *Artificial intelligence in medicine*, 103:101808.

Chesneau, B. (2020). Gunicorn - wsgi server. `https://docs.gunicorn.org/en/stable/#`. Acessado em 19 de março de 2021.

de Araújo, I. C., Schnitman, L., Duarte, A. A., and dos Santos, W. L. (2017). Automated detection of segmental glomerulosclerosis in kidney histopathology. In *XIII Brazilian Congress on Computational Intelligence*, page 12.

Eby, P. J. (2010). Python web server gateway interface v1.0.1. `https://www.python.org/dev/peps/pep-3333/`. Acessado em 19 de março de 2021.

JetBrains (2020). Python developers survey 2020 results. `https://www.jetbrains.com/lp/python-developers-survey-2020/`. Acessado em 19 de março de 2021.

Mozilla, D. N. (2021). Introdução ao django. `https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django/Introduction`. Acessado em 19 de março de 2021.

Rigo, A. and Tismer, C. (2011). greenlet: Lightweight concurrent programming. `https://greenlet.readthedocs.io/en/latest/`. Acessado em 19 de março de 2021.

Silva, P. A. B., Silva, L. B., Santos, J. F. G., and Soares, S. M. (2020). Brazilian public policy for chronic kidney disease prevention: challenges and perspectives. *Revista de Saúde Pública*, 54:86.

Sweet, G. M. M. et al. (2011). *Glomerulopatias prevalentes na Bahia, um estudo baseado em biópsias*. PhD thesis, Centro de Pesquisas Gonçalo Moniz.

Tornado (2020). Tornado web server. `https://www.tornadoweb.org/en/stable/`. Acessado em 19 de março de 2021.