CAPTURA DE DADOS DE PROVENIÊNCIA PARA APOIAR A ANÁLISE DE HIPERPARÂMETROS EM REDES DE APRENDIZADO PROFUNDO

Débora Barbosa Pina

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Rio de Janeiro
Abril de 2020

CAPTURA DE DADOS DE PROVENIÊNCIA PARA APOIAR A ANÁLISE DE HIPERPARÂMETROS EM REDES DE APRENDIZADO PROFUNDO

Débora Barbosa Pina

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Aprovada por: Profª. Marta Lima de Queirós Mattoso
Prof. Daniel Cardoso Moraes de Oliveira
Prof. Geraldo Bonorino Xexéo
Prof. Eduardo Soares Ogasawara

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2020

Pina, Débora Barbosa

Captura de dados de proveniência para apoiar a análise de hiperparâmetros em redes de aprendizado profundo/Débora Barbosa Pina. – Rio de Janeiro: UFRJ/COPPE, 2020.

XII, 70 p.: il.; 29, 7cm.

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2020.

Referências Bibliográficas: p. 63 – 70.

1. Aprendizado Profundo. 2. Hiperparâmetros. 3. Redes Neurais Convolucionais. I. Mattoso, Marta Lima de Queirós *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha família.

Agradecimentos

Primeiramente, agradeço imensamente aos meus pais. Por tanto. Por tudo. São as duas pessoas mais incríveis que eu já conheci na vida e eu tenho o privilégio de chamá-los de pais. Agradeço por me apoiarem em toda e qualquer decisão, por todo o apoio emocional, carinho e amor, apesar da distância física. Agradeço por cada palavra, cada olhar e cada guloseima feita pra mim. Agradeço.

À minha irmã querida, sempre pronta a me apoiar em tudo nessa vida.

À Professora Marta Lima de Queirós Mattoso, minha orientadora, que desde a graduação contribui para o meu conhecimento acadêmico. Agradeço por ser uma mulher forte, incentivadora de seus alunos e por me mostrar que computação também é "coisa de menina".

Ao Professor Daniel de Oliveira, meu orientador, pelos comentários, críticas e sugestões.

Aos membros da banca por aceitarem participar da defesa da minha dissertação de mestrado.

À Liliane Neves, que me auxiliou na escrita desta dissertação, assessorando em assuntos burocráticos, técnicos e apoio moral.

À Andréa Doreste, minha fiel escudeira. Agradeço pelo apoio, incentivo, amizade. Fico muito feliz em ver que alguns dos seus sonhos já se tornaram realidade e quero ter a oportunidade de estar ao seu lado nas próximas conquistas que certamente virão.

E aos meus amigos, que me apoiaram e acreditaram no meu potencial. E que, além disso, me ajudaram em todas as minhas crises e inseguranças.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

CAPTURA DE DADOS DE PROVENIÊNCIA PARA APOIAR A ANÁLISE DE HIPERPARÂMETROS EM REDES DE APRENDIZADO PROFUNDO

Débora Barbosa Pina

Abril/2020

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Programa: Engenharia de Sistemas e Computação

O treinamento das Redes Neurais Convolucionais (CNN) requer o ajuste de hiperparâmetros até que se encontre uma configuração em que a métrica escolhida pelo especialista seja satisfatória. Para isso, é preciso analisar dados do treinamento da CNN como a configuração dos hiperparâmetros e seus relacionamentos com a derivação dos dados. Nesse sentido, dados de proveniência podem auxiliar nessa análise por apresentarem metadados e a derivação dos dados do treinamento. As soluções existentes para a análise de configurações de hiperparâmetros não seguem padrões de representação de derivação dos dados e não permitem análise durante o treinamento. Tais abordagens requerem que o treinamento seja executado sob um portal ou ambiente de execução e/ou apresentam impacto significativo no tempo de treinamento da CNN. Apresenta-se, nesta dissertação, a CNNProv, uma solução de captura de dados de proveniência, que permite a análise de valores de hiperparâmetros durante o treinamento. A CNNProv adota o padrão W3C PROV para representar dados de proveniência e pode ser usada como serviços, independente de portal, contribuindo assim para a fase de treinamento das CNNs. Os experimentos realizados com o treinamento de CNNs em diferentes cenários usam uma CNN clássica, a AlexNet, e uma aplicação de imageamento sísmico. Os resultados mostram a adequação da CNNProv para a análise de hiperparâmetros com sobrecarga desprezível de até, no máximo, 4%.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

CAPTURING PROVENANCE DATA TO SUPPORT HYPERPARAMETER ANALYSIS IN DEEP LEARNING NETWORKS

Débora Barbosa Pina

April/2020

Advisors: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Department: Systems Engineering and Computer Science

Convolutional Neural Networks (CNN) training requires adjusting hyperparameters until a configuration is found in which the metric chosen by the specialist is satisfactory. For this, it is necessary to analyze CNN training data such as the hyperparameters configuration and their relationships with the data derivation. In this sense, provenance data can assist in this analysis by presenting metadata and the derivation of the training data. Current solutions for the analysis of hyperparameters configurations do not follow representation patterns of data derivation and do not allow analysis during training. Also, such approaches require CNN training to be executed under a portal or execution environment and/or have a significant impact on CNN's training time. In this dissertation, we present CNNProv, a provenance data capture solution, which allows the analysis of hyperparameters values during training. CNNProv adopts the W3C PROV standard to represent provenance data and can be used as services, regardless of a portal, thus contributing to the CNNs training phase. The experiments conducted with the training of CNNs in different scenarios use a classic CNN, AlexNet, and a seismic imaging application. The results show the suitability of CNNProv for the hyperparameter analysis with a negligible overhead of up to 4%.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Algoritmos de aprendizado de máquina pertencem a uma área em Inteligência Artificial que, com base em análises sobre grandes conjuntos de dados, "aprendem" seu comportamento. Esse aprendizado é fortemente baseado em uma fase chamada treinamento, em que diferentes alternativas aos relacionamentos sobre os dados são treinados e finalmente aprendidos automaticamente (MITCHELL, 1997). Os resultados desses algoritmos auxiliam a tomada de decisões, predizendo o comportamento futuro de novos dados (MITCHELL, 1997). Especialmente nos últimos anos, a comunidade científica tem presenciado a ascensão de aplicações que se baseiam em técnicas de aprendizado profundo (*Deep Learning*), que é uma área de pesquisa de aprendizado de máquina. O aprendizado profundo se caracteriza por suas camadas não serem projetadas por engenheiros, mas, aprendidas através dos dados (LECUN *et al.*, 2015). Tal ascensão se deve ao fato de essa técnica ter apresentado resultados impressionantes em problemas que, até então, não haviam sido descobertos meios para resolução (GHARIBI *et al.*, 2019b; MIAO *et al.*, 2017). Aprendizado profundo já apresentou melhorias significantes em diferentes domínios como visão computacional (HE *et al.*, 2016), reconhecimento de fala (HANNUN *et al.*, 2014), processamento de linguagem natural (SMITH, 2018) e até mesmo na área da saúde, como a utilização de técnicas de aprendizado profundo na detecção de doenças oculares diabéticas com precisão semelhante a de oftalmologistas (PENG e GULSHAN, 2016).

Aprendizado profundo, de acordo com DENG e YU (2014), comumente considera em sua definição dois aspectos: (i) modelos compostos por camadas múltiplas ou etapas de processamento de informação não linear; e (ii) métodos para aprendizado supervisionado ou não supervisionado de representação de características em camadas sucessivamente mais altas e mais abstratas. Ademais, aprendizado profundo é baseado em redes neurais artificiais.

As redes neurais artificiais podem ser definidas como estruturas compostas por elementos de processamento capazes de realizar computações paralelas para o pro-

cessamento de dados, rotulando ou agrupando dados brutos (HECHT-NIELSEN, 1989). As redes de aprendizado profundo se distinguem das redes neurais de camada única mais comuns por sua profundidade, ou seja, o número de camadas através das quais os dados devem passar em um processo de várias etapas (MUSIOL, 2016).

Nas redes de aprendizado profundo, cada camada treina em um conjunto distinto de características com base na saída da camada anterior. Quanto mais se avança na rede neural, mais complexas as características que seus nós podem reconhecer, pois agregam e recombinam características da camada anterior (LECUN *et al.*, 2015).

Uma considerável contribuição são as Redes Neurais Convolucionais (do inglês *Convolutional Neural Networks*, CNN) (LECUN *et al.*, 1998), que é uma das classes de redes neurais artificiais mais utilizadas em aprendizado profundo. As CNNs pertencem a uma categoria de algoritmos baseados em redes neurais artificiais que utilizam a convolução em, pelo menos, uma de suas camadas e são constituídas por uma arquitetura projetada para reduzir o número de parâmetros a serem aprendidos, otimizando o tempo de treinamento através de *backpropagation* (DENG e YU, 2014).

No entanto, no desenvolvimento de modelos de aprendizado profundo é necessário encontrar uma configuração de parâmetros que apresente bons resultados, ou seja, uma configuração que atinja uma métrica específica, como acurácia (ZAHARIA *et al.*, 2018). Esses parâmetros configuráveis são conhecidos como hiperparâmetros e podem ser, por exemplo, o número de iterações de treinamento, o número de camadas, a configuração das camadas e a taxa de aprendizado.

Apesar dos bons resultados obtidos com a utilização das redes neurais profundas, elas são bastante sensíveis aos hiperparâmetros (ORR e MÜLLER, 2003). O ajuste de grandes redes neurais é custoso em termos de computação e o tempo de otimização automatizada de hiperparâmetros dificulta sua adoção generalizada. Em vez disso, muitos especialistas em aprendizado profundo ainda realizam a busca manual por valores de hiperparâmetros (DOMHAN *et al.*, 2015). Usando esse conhecimento adquirido, eles costumam dizer, após algumas etapas do treinamento, se o procedimento convergirá para um modelo com desempenho satisfatório ou não. Para economizar tempo, eles encerram prematuramente as execuções com desempenho insatisfatório, o que lhes permite progredir mais rapidamente do que os métodos automatizados (que treinam modelos ruins até o final) (DOMHAN *et al.*, 2015). Dessa forma, o ciclo de vida típico de um modelo de aprendizado profundo passa por ciclos de tentativa e erro, em que se faz o ajuste dos hiperparâmetros, antes de se alcançar um resultado satisfatório (ZAHARIA *et al.*, 2018).

Em vista disso, a característica exploratória das configurações de hiperparâmetros se mostra uma tarefa intensa de se realizar, por demandar o treinamento da rede neural para cada uma das combinações de valores de hiperparâmetros, bem como seu armazenamento. Dessa forma, o espaço de busca para a melhor solução é

grande e a avaliação de cada configuração de hiperparâmetros pode ser computacionalmente intensiva. Além disso, o processo de tentativas e acertos manuais torna-se cansativo e suscetível a erros.

A avaliação das diversas configurações de hiperparâmetros exige que seja feita a relação entre vários dados, como dados de desempenho, dados de ambiente associados a épocas e outros parâmetros. No caso da análise *offline*, é necessário aguardar o fim da execução para que seja possível acessar resultados e dados de proveniência (MATTOSO *et al.*, 2015). Assim, ao final da execução, o usuário pode analisar os resultados e decidir se a rede neural deve ser treinada novamente, após o ajuste dos hiperparâmetros. Quando a análise de dados é feita em tempo de execução, o ajuste fino pode reduzir o espaço de busca dos hiperparâmetros, melhorando o desempenho geral.

Academia e indústria vêm investindo nesse ponto e, por isso, já existem soluções de análise de hiperparâmetros (ATKINSON *et al.*, 2017). Entre as vantagens das soluções existentes para os problemas mencionados estão gerência automatizada de metadados, proveniência dos artefatos produzidos durante o treinamento de redes neurais e fornecimento de métodos para consultas dos dados persistidos (GHARIBI *et al.*, 2019b; SCHELTER *et al.*, 2017).

Contudo, as soluções para apoiar a avaliação de configurações de hiperparâmetros apresentam algumas limitações. Há uma diversidade na representação dos dados de apoio ao treinamento, em que métodos usam sua própria representação do histórico de derivação de dados, e, em algumas dessas soluções, é necessário que o treinamento da rede neural seja executado sob um portal ou ambiente de execução específico.

No caso da diversidade na representação dos dados de apoio ao treinamento, essas representações proprietárias de dados podem dificultar a análise de dados e a interoperabilidade. Isso dificulta a comparação entre diferentes modelos, ainda que sejam do mesmo domínio. Em alguns casos, são utilizados portais científicos, que auxiliam na modelagem, monitoramento e análise de dados gerados em experimentos científicos (SCHELTER *et al.*, 2017; SOUZA *et al.*, 2018; TSAY *et al.*, 2018; ZAHARIA *et al.*, 2018), integrando em um mesmo ambiente uma interface para modelagem e execução de experimentos, ferramentas de monitoramento da execução e consulta aos dados produzidos. Porém, nessas soluções em que o treinamento das redes neurais é executado sob um portal ou outro sistema de controle de execução, há limitação da autonomia de execução do treinamento, podendo gerar incompatibilidade com o ambiente de execução paralela, além de criar uma dependência do portal ou ambiente de execução (PINA *et al.*, 2017).

Seja o seguinte exemplo fictício: Alice tem um problema de classificação de imagens para resolver e decide utilizar CNN. Alice configurou vários hiperparâmetros, entre eles a taxa de aprendizado com o valor 0,0005. Com a captura dos dados

de proveniência, Alice pode analisar os dados durante o treinamento da CNN. Assim, usando um recurso básico de consultas a um Sistema de Gerência de Banco de Dados (SGBD), Alice é capaz de realizar diferentes consultas. Um exemplo seria a consulta "Qual foi á acurácia para as épocas executadas até o momento atual?", com o resultado mostrado na Tabela 1.1. Através dessa consulta, Alice tem acesso a diversas informações, e com base nesses elementos, ela conclui que, dado o tempo de duração de cada época e pouca melhoria na acurácia, é necessário realizar alterações nos hiperparâmetros e decide alterar o valor da taxa de aprendizado, por exemplo. A relação entre os dados de proveniência e a configuração utilizada para os hiperparâmetros pode ajudar Alice na realização de ajustes dos valores dos hiperparâmetros durante a busca pela configuração de resultados mais satisfatórios.

Tabela 1.1: Resultado da consulta "Qual foi á acurácia para as épocas executadas até o momento atual?".

época	**acurácia**	**tempo decorrido (em segundos)**
1	0.903	102.86
2	0.916	107.59
3	0.920	106.22
4	0.922	107.72
5	0.924	101.20

Essa dissertação visa a apresentar uma alternativa ao problema de análise de dados de hiperparâmetros em CNNs. O objetivo é contornar os problemas de diversidade na representação dos dados de apoio ao treinamento e necessidade de execução sob um portal ou ferramenta. A solução desenvolvida se propõe a realizar a captura e o registro, seguindo padrões de representação de dados e relacionamentos já estabelecidos e a disponibilização de uma biblioteca de serviços a serem invocados pelo *script* de treinamento, de modo a manter a autonomia do controle de execução do treinamento.

Para evitar a diversidade na representação dos dados de apoio ao treinamento, propõe-se o uso de dados de proveniência (FREIRE *et al.*, 2008). Dados de proveniência permitem a representação de fluxos de dados registrando o histórico de derivação dos dados, isto é, dados consumidos, propagados e produzidos pelos componentes de uma aplicação geradora de dados. Uma vantagem no uso de modelo de dados de proveniência é a padronização na representação dos dados adotada pela recomendação W3C PROV (MOREAU e GROTH, 2013). Ao adotar um modelo genérico de representação, o processamento de consultas é facilitado, independente do domínio de dados da aplicação. Outra vantagem dessa representação é a possibilidade de os relacionamentos entre os dados serem registrados a partir de identificadores que fluem ao longo da execução (SILVA, 2018), o que facilita o seu registro em um SGBD qualquer. Além disso, é possível que ao invés de se inserir todos os da-

dos na base de dados de proveniência, apenas alguns valores sejam inseridos. Dessa forma, é possível acessar a maneira como os dados são transformados ao longo da execução, bem como os relacionamentos resultantes da execução. Além disso, dados de proveniência também são importantes para melhorar as soluções de análise de dados após o fim da execução da aplicação e, principalmente, para facilitar a reprodução dos resultados.

A associação da configuração de hiperparâmetros com a derivação de dados no contexto de CNNs pode trazer benefícios, contribuindo para o entendimento dos resultados do treinamento. Com a semelhança existente entre as consultas de CNNs e as de proveniência, é possível estruturar os dados de treinamento como proveniência e se beneficiar do arcabouço já existente, uma vez que consultas de proveniência refletem as de CNNs, de forma que as consultas de proveniência podem auxiliar nas de CNNs, contribuindo para a configuração de hiperparâmetros.

De fato, há um crescente uso de proveniência em aprendizado de máquina (GHARIBI *et al.*, 2019b; SCHELTER *et al.*, 2017; SOUZA *et al.*, 2018; TSAY *et al.*, 2018; ZAHARIA *et al.*, 2018). No entanto, especificamente para a fase de treinamento, não foram encontradas abordagens que sigam um padrão para representação de dados; ou que permitam consultas *online* sobre os dados; e ainda que não exijam o uso de portal ou sistema de execução. O levantamento da literatura realizado aponta que a solução (GHARIBI *et al.*, 2019b) não exige o uso de um portal ou ferramenta que mantenha a autonomia do controle de execução do treinamento. No entanto, (GHARIBI *et al.*, 2019b) usa representação para os dados própria dificultando comparações entre resultados.

Esta dissertação propõe uma representação de dados tendo como base o modelo de dados do W3C PROV. Foi desenvolvida uma solução de captura e registro de dados de proveniência seguindo W3C PROV por meio da DfAnalyzer (SILVA *et al.*, 2018b). DfAnalyzer se destaca dentre as soluções existentes para captura e registro de proveniência durante a execução de um experimento científico possibilitando uma análise *online*, uma seleção de dados de domínio a serem capturados antes ou mesmo durante a execução, por sua baixa sobrecarga, funcionamento em computação de alto desempenho e uso do W3C PROV. No entanto, por ser muito genérica, não possui apoio direto a aprendizado de máquina, o que acaba por gerar também diversidade na representação de hiperparâmetros e dificuldade em sua adoção. Sendo assim, foram realizadas extensões e uma especialização da DfAnalyzer para captura de hiperparâmetros. Além disso, foi incorporada uma ferramenta gráfica de consultas (PINA, 2018) para facilitar ainda mais a análise.

Além desta introdução, esta dissertação está organizada em outros quatro capítulos. O Capítulo 2 apresenta a fundamentação teórica necessária para o entendimento do restante da dissertação, abordando os conceitos de proveniência, redes

neurais e hiperparâmetros, além de uma avaliação das soluções existentes junto à análise de hiperparâmetros no treinamento de redes de aprendizado profundo. O Capítulo 3 descreve a abordagem desenvolvida para a análise de hiperparâmetros e dados acerca do treinamento de redes de aprendizado profundo por meio da abstração de fluxo de dados, incluindo a representação de dados de configurações de hiperparâmetros com o modelo de dados do W3C PROV, a arquitetura de *software* da CNNProv e a arquitetura do Keras-Prov. O Capítulo 4 discute os resultados dos experimentos realizados com a CNNProv, apresentando a instrumentação das duas redes neurais utilizadas, a análise de sobrecarga e a análise de configurações de hiperparâmetros. O Capítulo 5 conclui esta dissertação, além de apresentar as oportunidades de trabalhos futuros nesse tópico de pesquisa.

Capítulo 2

Hiperparâmetros em Redes Neurais Convolucionais

Este capítulo tem o propósito de apresentar a fundamentação teórica para a melhor compreensão desta dissertação. Por isso, apresenta-se aprendizado profundo descrevendo redes neurais profundas na Seção 2.1, hiperparâmetros relacionados à redes neurais na Seção 2.2 e gerência de dados de proveniência na Seção 2.3. Além disso, as abordagens existentes para análise de configurações de hiperparâmetros em redes neurais são descritas na Seção 2.4 deste capítulo.

2.1 Aprendizado profundo

2.1.1 Definição

Aprendizado profundo é um campo de aprendizado de máquina que tem foco na compreensão das características dos dados através de múltiplas camadas de abstração (YOUNG *et al.*, 2015). Segundo LECUN *et al.* (2015), métodos de aprendizado profundo são métodos que permitem que uma máquina seja alimentada com dados brutos e automaticamente descubra as representações necessárias para detecção ou classificação. Isso ocorre em diferentes níveis de representação que são obtidos por intermédio da composição de módulos não lineares que transformam a representação nível a nível, começando com a entrada bruta em nível mais baixo até um nível mais alto e um pouco mais abstrato. Dessa forma, com a composição de transformações, funções complexas podem ser aprendidas (PATTERSON e GIBSON, 2017).

O aprendizado de funções mais complexas é possível nessa realidade por conta de alguns aspectos decorrentes da evolução de redes neurais, como: mais neurônios que redes anteriores, maneiras mais complexas de conectar camadas e neurônios, recursos computacionais disponíveis para treinamento e extração automática de características, que é o processo da rede de decidir quais características de um conjunto de

dados pode ser usado como indicador para rotular esses dados de maneira confiável (PATTERSON e GIBSON, 2017). Esse processo de extração demanda muito tempo de profissionais de aprendizado de máquina, já que esses conjuntos de características são criados manualmente para a classificação de dados. Sendo isso feito de forma automática, os profissionais podem se dedicar a tarefas mais significativas.

Em suma, aprendizado profundo é uma ramificação de aprendizado de máquina, possuindo um conjunto de algoritmos de aprendizado que podem ser treinados e aprender os dados. BENGIO (2012) acredita que aprendizado profundo terá ainda muito mais sucesso, porque requer pouca engenharia manual, então pode tirar proveito do aumento na quantidade de recursos computacionais e dados disponíveis. E, a medida que tais fatores se tornam disponíveis, novos algoritmos de aprendizado e arquiteturas vêm sendo desenvolvidos para redes neurais de maneira a acelerar o progresso.

2.1.2 Redes neurais profundas

As redes neurais são um modelo computacional em que muitas unidades simples estão trabalhando paralelamente, sem nenhuma unidade de controle centralizada. Os pesos entre as unidades são os principais meios de armazenamento de informações a longo prazo em redes neurais e a atualização dos pesos é a principal maneira pela qual a rede neural aprende novas informações (NIELSEN, 2015). O comportamento das redes neurais é moldado por sua arquitetura de rede, que pode ser definida pelo número de neurônios, número de camadas, tipos de conexões entre camadas, entre outros.

Um modelo de aprendizado profundo é uma rede neural profunda (*Deep Neural Network*, DNN) consistindo de diversas camadas com funções de ativação não lineares que são capazes de representar transformações complexas entre os dados de entrada e a saída desejada (MIAO, 2018). A Figura 2.1 mostra um exemplo em que são mostradas duas camadas ocultas, podendo haver mais, além de uma camada de entrada e uma camada de saída.

Alguns exemplos de arquiteturas de DNNs são *Unsupervised Pretrained Networks* (UPNs), *Convolutional Neural Networks* (CNNs), *Recurrent Neural Networks* e *Recursive Neural Networks*.

Uma Rede Neural Convolucional (*Convolutional neural network*, CNN) é um tipo de rede neural artificial que tem como objetivo aprender características de ordem superior nos dados por meio de convoluções (PATTERSON e GIBSON, 2017). Esse tipo de rede vem sendo amplamente utilizado na classificação de imagens, podendo reconhecer rostos, indivíduos, placas e muitos outros aspectos (WU, 2019). Embora a análise de imagens tenha sido o uso mais difundido das CNNs, elas também podem

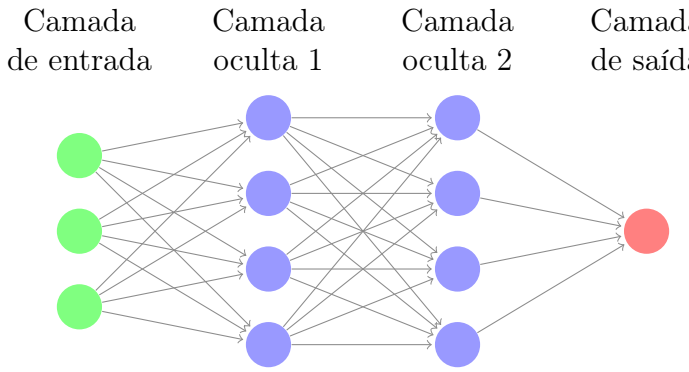

Figura 2.1: Modelo geral de uma rede neural profunda, consistindo de uma camada de entrada, duas camadas ocultas e uma camada de saída. Adaptado de (KARPATHY).

ser usadas para outros problemas de análise ou classificação de dados, como processamento de linguagem natural, descoberta de drogas, entre outros (PATTERSON e GIBSON, 2017).

As CNNs são caracterizadas pelo uso de múltiplas camadas de convolução (SZE *et al.*, 2017). Este tipo de rede neural consiste em múltiplas partes assumindo diferentes funções. Inicialmente é comum aplicar sobre o dado de entrada camadas de convolução. Uma camada de convolução é composta por diversos neurônios, cada um responsável pela aplicação de um filtro em uma parte específica da imagem.

Uma única imagem possui diferentes aspectos como arestas, formas, texturas, objetos, etc. Assim, um tipo de padrão que um filtro pode detectar pode ser as arestas. Alguns filtros podem detectar cantos, alguns podem detectar círculos, outros quadrados. Quanto mais profunda a rede, ou seja, mais camadas, mais sofisticados esses filtros se tornam. Portanto, em camadas posteriores, no lugar de bordas e formas simples, o filtro poderá detectar objetos específicos, como olhos, ouvidos e cabelos em uma imagem de rosto humano (WU, 2019). E em camadas ainda mais profundas, os filtros são capazes de detectar objetos ainda mais sofisticados, como cães, gatos, lagartos e pássaros (WU, 2019).

Comumente, após a convolução, é aplicada uma função de ativação, que está presente em cada neurônio e tem como responsabilidade aplicar uma transformação nos dados recebidos (VARGAS *et al.*, 2016). Outra camada utilizada após as camadas de convolução e ativação é a camada de *pooling*, que tem como função reduzir a dimensionalidade dos dados na rede. Na realização de classificações, é comum, após as camadas de convolução e *pooling*, o acréscimo de uma camada totalmente conectada, que é responsável por traçar um caminho de decisão a partir das respostas dos filtros vindos das camadas anteriores.

2.2 Hiperparâmetros

Como mencionado anteriormente, aprendizado profundo tem atraído muita atenção da academia e da indústria, devido ao seu excelente desempenho em várias áreas de pesquisa. No entanto, o desempenho do algoritmo de aprendizado profundo é um desafio, uma vez que depende muito da seleção de hiperparâmetros. Segundo BENGIO (2012), hiperparâmetro é *"uma variável a ser definida antes da real aplicação de um determinado algoritmo de aprendizado aos dados, sendo tal variável não diretamente selecionada pelo próprio algoritmo de aprendizado"*.

De acordo com ZHANG *et al.* (2019), comparado com algoritmos tradicionais de aprendizado de máquina, o aprendizado profundo tem um ajuste de hiperparâmetros mais desafiador, porque as redes neurais profundas: (i) têm mais hiperparâmetros a serem ajustados e (ii) possuem maior dependência da configuração dos hiperparâmetros, principalmente em modelos complexos, como redes neurais convolucionais que possuem dezenas de hiperparâmetros (HOOS e LEYTON-BROWN, 2014).

Dado o ciclo de vida típico de modelagem de aprendizado profundo apresentado na Figura 2.2, destaca-se a fase de treinamento que engloba os ajustes de hiperparâmetros. De acordo com MIAO *et al.* (2015), dada uma tarefa de predição, um especialista de redes neurais geralmente parte de modelos conhecidos que foram bem-sucedidos em domínios de tarefas semelhantes; então define dados de treinamento de entrada e funções de perda de saída e ajusta repetidamente a rede neural, ajusta os hiperparâmetros do modelo, treina e avalia o modelo e repete esse processo até que a métrica escolhida pelo especialista não apresente melhora. Ainda segundo MIAO *et al.* (2015), devido à falta de entendimento sobre o motivo pelo qual os modelos funcionam, os ajustes dentro do processo são conduzidos por heurísticas, por exemplo, ajustando hiperparâmetros que parecem ter um impacto significativo nos pesos aprendidos, aplicando novas camadas ou técnicas vistas em estudos empíricos recentes. Assim, muitos modelos semelhantes são treinados e comparados, e uma série de variantes de modelos precisa ser explorada e desenvolvida. Devido à dispendiosa fase de aprendizado/treinamento, cada iteração do processo de modelagem leva um longo período de tempo.

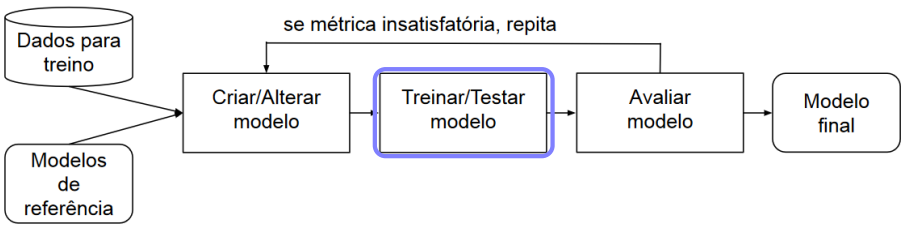

Figura 2.2: Ciclo de vida de modelagem de aprendizado profundo. Adaptado de (MIAO *et al.*, 2015).

Nesse sentido, a quantidade de hiperparâmetros pode fazer com que a necessidade de se fazer um grande número de ajustes não seja atraente (HOOS e LEYTON-BROWN, 2014). A escolha de valores de hiperparâmetros é formalmente equivalente à questão da seleção de modelos (BENGIO, 2012), isto é, "dada uma família ou um conjunto de algoritmos de aprendizado, como escolher o mais apropriado dentro do conjunto?". ZHANG *et al.* (2017) relatam que a acurácia da classificação em aprendizado profundo varia drasticamente de 32,2 % para 92,6 % devido à seleção de diferentes valores para os hiperparâmetros. Portanto, métodos que favoreçam a análise de configurações de hiperparâmetros de maneira eficiente são necessários.

Os hiperparâmetros podem ser divididos em duas categorias: hiperparâmetros de otimização e hiperparâmetros específicos do modelo. Hiperparâmetros de otimização estão relacionados ao processo de treinamento da rede e otimização das métricas, enquanto os hiperparâmetros relacionados ao modelo são mais voltados para estrutura da rede neural. Alguns hiperparâmetros associados à otimização são:

- A **taxa de aprendizado** é normalmente um dos hiperparâmetros mais importantes. Valores típicos para uma rede neural são menores que 1 e maiores que 10^{-11}. A taxa de aprendizado é a rapidez com que o vetor de parâmetros é alterado à medida que se avança no espaço de busca. Se a taxa de aprendizado se tornar muito alta, pode-se avançar em direção ao objetivo mais rapidamente (menor quantidade de erros para a função que está sendo avaliada), mas também pode ser que o passo dado seja tão grande acabando por ir além da melhor resposta para o problema. Se a taxa de aprendizado for reduzida, pode ser que o processo de treinamento demore muito mais do que o desejado para conclusão (PATTERSON e GIBSON, 2017). Uma baixa taxa de aprendizado pode tornar o algoritmo de aprendizado ineficiente. As taxas de aprendizado são complicadas porque acabam sendo específicas para o conjunto de dados e até para outros hiperparâmetros. Isso cria muita sobrecarga para encontrar a configuração correta de hiperparâmetros. Além disso, também é possível definir uma diminuição nas taxas de aprendizado ao longo do tempo, de acordo com alguma regra.

- ***Batch size*** é o número de subamostras fornecidas à rede após a atualização de parâmetros.

- O **número de épocas** é o número de vezes que todos os dados de treinamento são apresentados à rede neural durante o treinamento.

- O ***momentum*** ajuda o algoritmo de aprendizado a sair de pontos no espaço de busca (PATTERSON e GIBSON, 2017), isto é, permite que a curva de

aprendizado escape das regiões onde pode haver um mínimo local da função de perda (TAN *et al.*, 2016).

- O ***dropout*** é um mecanismo usado para melhorar o treinamento de redes neurais, omitindo uma unidade oculta. Também acelera o treinamento. O *dropout* é conduzido pela retirada aleatória de um neurônio, para que ele não contribua para o *backpropagation*, por exemplo (PATTERSON e GIBSON, 2017). A principal motivação por trás do algoritmo é impedir a coadaptação de detectores de características, ou sobreajuste, forçando os neurônios a serem robustos e confiar no comportamento da população, e não na atividade de outras unidades específicas (BALDI e SADOWSKI, 2013).

Entre os hiperparâmetros relacionados à estrutura do modelo podemos citar o número de unidades ocultas e o número de camadas, brevemente comentados a seguir.

- **Número de unidades ocultas** em uma rede neural *feed-forward* é significativo na caracterização do desempenho da rede (FUJITA, 1998), influenciando, por exemplo, a capacidade da rede.

- **Número de camadas** em uma rede neural é um indicador de sua complexidade (PATTERSON e GIBSON, 2017). Caso muitas camadas sejam definidas, isso fará com que o modelo aprenda muitas informações sobre os dados de treinamento, causando sobreajuste. No caso de poucas camadas serem definidas, pode ser que a capacidade de aprendizado do modelo seja limitada, causando subajuste.

Além desses hiperparâmetros que dizem respeito à redes neurais de maneira geral, redes neurais convolucionais possuem hiperparâmetros específicos. Alguns hiperparâmetros chaves dessa camada das CNNs são: tamanho do filtro, *stride* e *padding*.

- Um filtro é uma matriz de pesos com a qual a entrada da rede é transformada. Dessa forma, o **tamanho do filtro/kernel** se refere à largura x altura do filtro.

- ***Stride*** é usado para diminuir consideravelmente o tamanho da imagem de entrada, configurando até que ponto a janela do filtro se moverá por aplicação da função de filtro (PATTERSON e GIBSON, 2017), isto é, é o número de *pixels* que se deseja pular ao percorrer a entrada horizontal e verticalmente durante a convolução após cada multiplicação por elemento dos pesos das entradas com os do filtro. Além disso, é quase sempre simétrico nas dimensões

de altura e largura. O *stride* padrão é (1,1) para a altura e a largura, mas pode ser alterado, o que afeta tanto como o filtro é aplicado à imagem quanto o tamanho do mapa de características resultante.

- ***Padding*** é geralmente usado para adicionar colunas e linhas de forma a manter os tamanhos espaciais constantes após a convolução, o que pode melhorar a acurácia da rede, uma vez que são preservadas informações nas bordas.

2.3 Gerência de dados de proveniência

Dados de proveniência podem ajudar no registro de parâmetros relevantes à aplicações científicas e associá-los aos resultados pode melhorar tanto o ajuste fino de parâmetros quanto às análises de dados em tempo de execução (HERSCHEL *et al.*, 2017). O dicionário Cambridge[1] define proveniência como "o lugar de origem de algo" (tradução de "*the place of origin of something*"). Na Ciência da Computação, os dados de proveniência estão associados a uma composição de transformações de dados e correspondem ao registro do caminho de derivação de dados que levou ao resultado com a transformação final (FREIRE *et al.*, 2008). Para que dados de proveniência sejam efetivamente usados em análises, eles precisam ser gerenciados.

A gerência de dados de proveniência trata de modelar, capturar, representar e armazenar dados de proveniência durante a execução das transformações de dados para que sejam disponibilizados para análise (SILVA *et al.*, 2018c). Dados de proveniência incluem (ZHAO *et al.*, 2006):

- Dados sobre a definição das transformações de dados;
- Dados capturados durante a execução das transformações de dados;
- Metadados sobre as transformações de dados e conjuntos de dados gerados;
- Fluxo de dados entre as transformações de dados.

Apesar de o uso de dados de proveniência no contexto de aprendizado de máquina ainda ser recente, dados de proveniência vêm sendo usados com sucesso no contexto de transformações em SGBDs (BUNEMAN *et al.*, 2001), em *workflows* científicos (FREIRE *et al.*, 2008), dentre outras análises envolvendo transformações de dados. Especificamente em experimentos em larga escala (MATTOSO *et al.*, 2010) dados de proveniência vêm sendo acrescidos de dados de desempenho de execução e dados de domínio, enriquecendo o ajuste fino de parâmetros e a conclusão do experimento científico (COSTA *et al.*, 2013; DE A.R. GONÇALVES *et al.*, 2012;

[1] https://dictionary.cambridge.org/

DE OLIVEIRA *et al.*, 2010, 2012; OGASAWARA *et al.*, 2011; SILVA *et al.*, 2017; SOUZA e MATTOSO, 2018).

Existem diversas abordagens para gerência e análise de proveniência em experimentos computacionais de maneira geral (HERSCHEL *et al.*, 2017; SILVA *et al.*, 2018c). Na maioria dos casos, essas abordagens constroem uma base de dados para armazenar os dados de proveniência e adotam diferentes estratégias para construção, população e acesso às bases.

Um fator importante acerca dos dados de proveniência é o momento em que os dados capturados podem ser consultados. Nesse sentido, a proveniência pode ser analisada de forma *offline* ou *online* (MATTOSO *et al.*, 2015). A análise *offline* é decorrente de uma captura de dados de proveniência disponibilizada para consulta apenas após o término da execução das transformações de dados, enquanto na análise *online* os dados de proveniência são armazenados em tempo de execução, o que possibilita tal análise. Na análise *offline* só é permitido o processamento de consultas após o término do treinamento da rede, impossibilitando a investigação de resultados parciais.

A análise *online* permite que a análise de dados de proveniência seja realizada durante a execução/treinamento da rede, permitindo que especialistas utilizem a análise para monitorar, depurar e inspecionar as transformações de dados enquanto ainda estão em execução, consultando *status* e investigando a evolução de resultados intermediários (MATTOSO *et al.*, 2015). Uma questão nesse modo de análise é a sobrecarga adicionada pela persistência dos dados em tempo de execução.

DfAnalyzer (SILVA *et al.*, 2018b) é um exemplo de ferramenta que realiza captura e gerência de dados de proveniência. Esta ferramenta permite consultas sobre dados extraídos de arquivos usados no *script* instrumentado em tempo de execução a partir de uma base de dados de proveniência que contém informações sobre o fluxo de dados. Dessa forma, a DfAnalyzer provê análise de dados de proveniência de modo *online*, inclusive considerando ambientes de Processamento de Alto Desempenho (PAD) e seguindo uma abordagem assíncrona para a captura e o armazenamento dos dados de proveniência, não comprometendo, assim, o tempo de execução. Ademais, assumindo as recomendações do W3C PROV, SILVA *et al.* (2016) propôs o PROV-Df para apoiar as definições de fluxo de dados, conforme mostrado na Figura 2.3. O PROV-Df especializou as três grandes classes do W3C PROV, sendo elas, Agente (*Agent*), Atividade (*Activity*) e Entidade (*Entity*), e utiliza estereótipos para a correspondência com tais classes. O PROV-Df é composto de três partes: a estrutura do fluxo de dados, os dados de execução referentes a um fluxo de dados com os dados de domínio e as configurações do ambiente computacional.

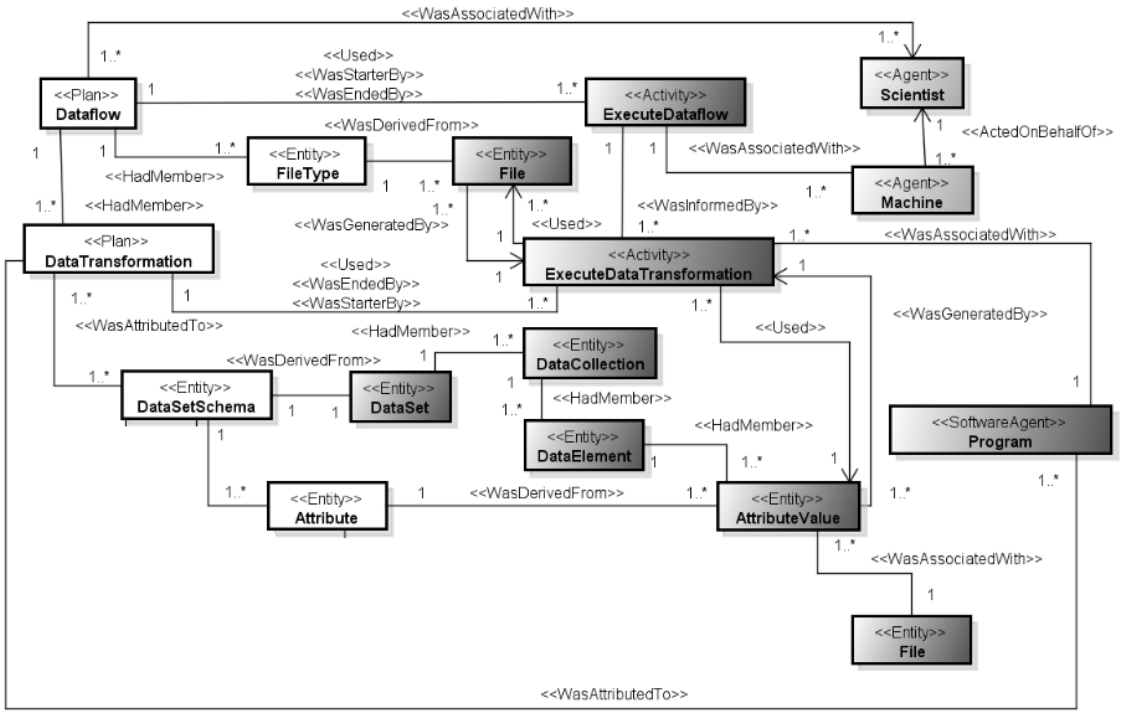

Figura 2.3: Diagrama de classe PROV-Df (SILVA *et al.*, 2016).

As classes em branco descrevem a especificação do fluxo de dados (classe *Dataflow*), que é composto de transformações de dados (classe *DataTransformation*), conjuntos de dados (classe *DataSetSchema*) e atributos (classe *Attribute*). As dependências de dados entre as transformações de dados são representadas nesse modelo de dados pelo relacionamento *WasAttributedTo* entre as classes *DataTransformation* e *DataSetSchema*.

As classes em cinza escuro descrevem os dados de execução referente a um fluxo de dados (atividade *ExecuteDataflow*), que considera as propriedades sobre a execução das transformações de dados (atividade *ExecuteDataTransformation*), os conjuntos de dados (classe *DataSet*), as coleções de dados (classe *DataCollection*) e os elementos de dados a partir de cada conjunto de dados (classe *DataElement*), os valores de atributos referentes a um determinado elemento de dados (classe *AttributeValue*), os programas invocados pelas transformações de dados (agente de software *Program*) e os arquivos consumidos e produzidos (classe *File*).

As classes em cinza claro descrevem informações sobre o ambiente computacional (agente *Machine*) e os usuários do domínio científico envolvidos na especificação e execução (agente *Scientist*).

Outra solução que realiza a captura e armazenamento de dados de proveniência é o noWorkflow (MURTA *et al.*, 2014), que é uma ferramenta que permite a análise dos dados de proveniência prospectiva e retrospectiva em *scripts* Python, sem que o usuário precise modificá-los. No entanto, pelo fato de a captura da proveniência ser

automática, o usuário não pode especificar as transformações de dados de interesse. Como consequência, o noWorkflow captura os dados de proveniência em uma granularidade muito fina, armazenando informações sobre invocações que podem não ser de interesse de análise do usuário, ao contrário da DfAnalyzer, que permite que o usuário especifique transformações ou dados de interesse. Assim como noWorkflow, existem várias outras soluções de captura de dados de proveniência, mas nenhuma delas provê flexibilidade de escolha de dados que a DfAnalyzer oferece e não estão preparadas para atuar em ambientes PAD.

Esta dissertação tem como intuito apoiar o especialista em redes neurais na análise de configurações de hiperparâmetros em CNN. Nesse sentido, objetiva-se uma solução que não seja específica de uma linguagem de programação, não seja específica a um ambiente de execução, tenha flexibilidade na escolha do que será monitorado, siga o padrão W3C PROV para representação dos dados e não tenha impacto significativo no tempo de execução. Dessa forma, vamos analisar as soluções existentes à luz dessas características.

2.4 Soluções de uso de proveniência na análise de configuração de hiperparâmetros

Apenas armazenar os dados produzidos durante o treinamento de CNNs não é suficiente para prover análise de hiperparâmetros para que o especialista faça ajustes necessários ao modelo. Assim, dados de proveniência enriquecem a análise de configurações de hiperparâmetros, dando mais base ao especialista para a tomada de decisões.

Com o recente crescimento em aprendizado profundo, esforços têm sido feitos tanto na indústria quanto na academia no desenvolvimento de métodos e ferramentas que facilitem a gerência do treinamento de redes neurais no contexto de aprendizado profundo trazendo consigo também os dados de proveniência (GHARIBI *et al.*, 2019a). Atualmente, existem na literatura soluções que apoiam a análise de configurações de hiperparâmetros. No entanto, essas soluções apresentam limitações sobre o treinamento da rede neural, como dependência de linguagens de programação e bibliotecas de treinamento, controle do treinamento por portais ou ambiente de execução e utilização de modelo próprio para representação de dados.

Esta dissertação visa oferecer uma solução para gerar dados de proveniência ligados ao treinamento de redes neurais convolucionais de modo a contribuir com a análise de hiperparâmetros sem que, com isso, o especialista precise realizar o treinamento sob um portal ou sistema de controle de execução ao mesmo tempo em que gera uma base de dados de proveniência aderente ao padrão W3C PROV.

A seguir, apresentamos as principais soluções existentes. Vale ressaltar que não foi encontrada uma solução que reunisse as características de flexibilidade na escolha de bibliotecas para o treinamento da CNN, treinamento da CNN independente do portal e padrão para representação dos dados.

O ModelHub (MIAO *et al.*, 2017) é um sistema para gerência do ciclo de vida de redes neurais profundas. Seu objetivo é armazenar pesos de modelos em diferentes versões, com foco no aprendizado profundo. São propostos um sistema de versão de modelo e um sistema de armazenamento específico de aprendizado profundo, fornecendo um repositório baseado em nuvem. O ModelHub não substitui sistemas de treinamento, mas é para ser utilizado associado a eles para gerência de artefatos produzidos durante o ciclo de vida, se assemelhando ao nosso objetivo de analisar configurações de hiperparâmetros em conjunto com dados relacionados ao treinamento. No entanto, MIAO *et al.* (2017) propõem uma linguagem específica de domínio para permitir a exploração de modelos.

O ModelDB (VARTAK *et al.*, 2016) é um sistema que visa abordar problemas de gerência de modelos. O foco é armazenar modelos treinados e seus resultados para permitir a exploração visual e a consulta de metadados e artefatos, por exemplo, hiperparâmetros e métricas de acurácia. No entanto, o ModelDB é personalizado para modelos criados no *scikit-learn* e *SparkML*, oferecendo suporte limitado para redes neurais convolucionais.

MLFlow (ZAHARIA *et al.*, 2018) gerencia e arquiva experimentos de aprendizado de máquina facilitando a reprodutibilidade, concentrando-se no rastreamento de experimentos e acompanhamento de métricas. Porém, o MLFlow requer que o especialista realize o treinamento sob essa ferramenta.

Uma solução proposta pela IBM Research é a ferramenta Runway (TSAY *et al.*, 2018). Runway contribui para gerência de artefatos de aprendizado de máquina e aprendizado profundo, como modelos, dados ou experimentos, bem como sua proveniência. Nesse sentido, a solução permite rastrear o modelo e os dados desde o uso no treinamento até a implementação, facilitando a reprodutibilidade. No entanto, além de ser uma solução proprietária, é restrita à linguagem de programação Python 3.

O ProvDB (MIAO e DESHPANDE, 2018) propõe um modelo de dados e operadores de consulta para armazenar e consultar dados de proveniência em projetos de ciência de dados. O modelo de dados específico utilizado generaliza o modelo de dados padrão W3C PROV, o que se aproxima dos nossos objetivos. ProvDB funciona ingerindo a proveniência por meio de comandos *shell* e seu foco principal é o armazenamento e processamento de consultas de maneira eficiente aos dados de proveniência do aprendizado de máquina. Entretanto, o ProvDB é para ser utilizado

como um complemento, isto é, deve ser associado a ferramentas como a proposta nesta dissertação.

ModelKB (Model Knowledge Base) (GHARIBI *et al.*, 2019b) tem como objetivo automatizar o processo de gerência do ciclo de vida de modelos de aprendizado profundo com mínima intervenção do usuário. As contribuições deste trabalho são para extrair e armazenar automaticamente os metadados do modelo e artefatos, além de visualizar, consultar, comparar experimentos e reproduzir modelos. ModelKB em si não é uma ferramenta de modelagem, mas um sistema complementar que pode gerenciar automaticamente os experimentos em suas estruturas nativas, como TensorFlow (ABADI *et al.*, 2016). Todavia, o ModelKB não disponibiliza os dados capturados para análise em tempo de execução.

Outra solução existente na linha de representação de dados de aprendizado de máquina é o W3C ML Schema [2], que pode ser usado para representar algoritmos, implementações e execuções, além de dados de entrada e saída especificados. Apesar de representar dados de proveniência, tal representação não distingue a proveniência prospectiva da proveniência retrospectiva, prejudicando os recursos de consulta, uma vez que a proveniência prospectiva fornece a camada de abstração para especificar a análise de proveniência sobre os dados gerados na execução dos fluxos de dados (SOUZA *et al.*, 2019).

Já abordagens como OpenML (VANSCHOREN *et al.*, 2014) e Kipoi (AVSEC *et al.*, 2018), fornecem serviços de compartilhamento de dados, modelos, *pipelines* e resultados experimentais de aprendizado de máquina, abordando, assim, o problema de reprodutibilidade, mas não levam em consideração a especificação de experimentos.

SCHELTER *et al.* (2017) fornecem uma ferramenta automatizada para extrair os metadados do modelo e provê-los com uma visualização interativa para consultar e comparar experimentos. É proposta uma linguagem declarativa para que os usuários especifiquem o *log* por operação do *workflow* e não por métricas individuais. Dessa forma, essa solução concentra-se no rastreamento de metadados e na proveniência dos dados de experimentação de aprendizado de máquina. Porém, a abordagem proposta não utiliza o padrão W3C PROV afetando a interoperabilidade.

SOUZA *et al.* (2018) propõem o DL-Steer, que é uma biblioteca voltada para aprendizado profundo que captura valores de entrada dos hiperparâmetros do modelo e relaciona-os aos resultados dos modelos treinados, permitindo que usuários alterem os valores dos hiperparâmetros durante o treinamento. Porém, a DL-Steer é limitada a *scripts* Python com chamadas de funções para o TensorFlow (ABADI *et al.*, 2016), além de não considerar os dados de domínio da aplicação durante o processo de extração.

[2]https://www.w3.org/community/ml-schema/

As ferramentas propostas por SCHELTER *et al.* (2017) e SOUZA *et al.* (2018) são as que mais se aproximam da nossa na questão da captura de dados independente da ferramenta na qual o treinamento é feito.

A Tabela 2.1 apresenta um resumo comparativo das soluções que usam proveniência na análise de configurações de hiperparâmetros apresentadas neste capítulo. A comparação tem como base as características identificadas como relevantes para a fase de treinamento em CNNs de diferentes domínios de aplicação, em especial a necessidade de uso de ambiente PAD. Portanto, conforme observado nas análises de cada uma das abordagens apresentadas, não encontramos nenhuma solução que reúna as características que pontuamos no início deste capítulo.

Tabela 2.1: Comparativo de soluções para análise de configurações de hiperparâmetros.

Soluções	**Flexibilidade na escolha da biblioteca**	**Dependência do ambiente de execução**	**Representação de dados**
ModelHub	sim	não	própria
ModelDB	não	não	própria
MLFlow	sim	sim	própria
Runway	não	não	própria
ModelKB	não	não	própria
Amazon	não	não	própria
DL-Steer	não	sim	W3C PROV

Capítulo 3

CNNProv - Análise de hiperparâmetros no treinamento de redes de aprendizado profundo

Este capítulo descreve a abordagem proposta por esta dissertação, destacando-se os seus objetivos e aspectos de implementação.

Esta dissertação propõe usar dados de proveniência para prover análise de configuração de hiperparâmetros no treinamento de redes de aprendizado profundo. Para prover a análise de configuração de hiperparâmetros em tempo de execução, o objetivo é uma solução leve com o intuito de prover flexibilidade na análise e ambiente de execução. Com esse apoio à análise de configuração de hiperparâmetros, espera-se aprimorar a tarefa repetitiva de treinamento de redes neurais profundas, ainda durante o treinamento. Os passos envolvidos para prover essa análise são a modelagem, a captura dos dados, a representação dos dados como proveniência e o armazenamento dos dados capturados e, por fim, as consultas.

Para prover dados de proveniência para análise, propõe-se uma solução de gerência de dados de proveniência denominada CNNProv e suas principais funcionalidades são modelar dados de análise, capturar, estruturar, armazenar e disponibilizar para consulta, para facilitar a análise de dados de proveniência de redes neurais convolucionais de maneira eficiente, em tempo de execução, seguindo um padrão para a representação de dados.

Para modelar os dados propõe-se uma representação de dados com base no W3C PROV que modela as principais classes de objetos de uma CNN junto aos dados de proveniência. Para realizar a captura dos hiperparâmetros, dados de domínio e dados relativos ao treinamento da CNN de forma eficiente, a CNNProv adota a DfAnalyzer (SILVA *et al.*, 2018b), que permite o monitoramento, a depuração e a análise do fluxo de dados durante a execução das transformações de dados, isso

porque o processo de treinamento da rede neural pode ser tratado como um fluxo de dados.

A partir do conhecimento sobre abstração de fluxo de dados, e sabendo da importância dos dados de proveniência e dos dados acerca dos hiperparâmetros, apresentamos a arquitetura CNNProv, que tem como objetivos capturar e analisar dados de proveniência em redes neurais convolucionais (CNN). A CNNProv pode ser descrita como uma especialização da DfAnalyzer (SILVA *et al.*, 2018b) para análise de configurações de hiperparâmetros no treinamento de redes de aprendizado profundo, mantendo a abordagem agnóstica à linguagem de programação ou bibliotecas como TensorFlow. Com essa abordagem, o intuito é simplificar a adaptação do código das redes neurais convolucionais, de maneira que o especialista defina quais os hiperparâmetros deseja monitorar e consultar.

Nesse sentido, decidiu-se pela adoção da DfAnalyzer na CNNProv para captura, representação e armazenamento de dados de proveniência, que, apesar de nunca ter sido utilizada no contexto de aprendizado profundo, já teve a captura de metadados e proveniência avaliada em diversos cenários, tendo obtido sucesso na análise de dados durante a execução. Além disso, a DfAnalyzer é única ao explorar a captura de dados "in-situ" ou assíncrona, não interferindo no desempenho da aplicação mesmo em ambientes de PAD. A abstração de fluxo de dados é útil para modelagem do ciclo de vida de aprendizado profundo, de maneira a permitir análise de configurações de hiperparâmetros. A DfAnalyzer captura dados de proveniência e domínio, o que também é útil no treinamento de CNNs. O fato de ser invocada como biblioteca de proveniência também propicia sua utilização no treinamento de CNNs, uma vez que as aplicações também fazem chamadas à diversas bibliotecas. Um outro fator é que aplicações de aprendizado profundo também são especificadas como *scripts*, o que se adequa a DfAnalyzer. Ademais, a base de dados da DfAnalyzer fica disponível para consultas em tempo de execução.

Desta forma, ao incorporar a DfAnalyzer à CNNProv, foi necessário estender a representação da DfAnalyzer com representações voltadas para o treinamento de CNNs. Assim, é possível registrar informações complementares quanto ao desempenho do treinamento, além de associá-las a configurações de hiperparâmetros.

As seções a seguir descrevem as contribuições técnicas presentes na CNNProv, como representação de dados de configurações de hiperparâmetros com W3C PROV (Seção 3.1), arquitetura da CNNProv (Seção 3.2), desenvolvimento da arquitetura (Seção 3.3), metodologia de uso (Seção 3.4) e um subcomponente da CNNProv, Keras-Prov (Seção 3.5).

3.1 Representação de dados de configurações de hiperparâmetros com W3C PROV

Um exemplo de fluxo de dados para o treinamento de CNNs é mostrado na Figura 3.1, em que a transformação *Training* diz respeito à criação e treinamento do modelo, a transformação *Adaptation* se refere à atualização de hiperparâmetros e a transformação *Testing* se refere à avaliação do modelo. As letras *i* e *o* nos conjuntos de dados designam *input* e *output*, respectivamente, conforme convenção da DfAnalyzer, indicando que o conjunto de dados é um conjunto de dados de entrada ou conjunto de dados de saída.

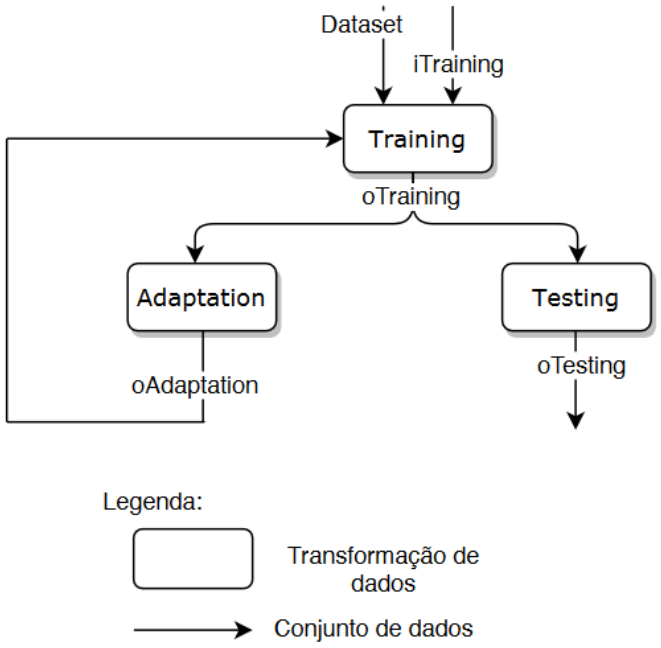

Figura 3.1: Fluxo de dados para o ciclo de vida de aprendizado profundo.

A associação dos dados de proveniência aos dados de domínio pode auxiliar os especialistas em redes neurais a realizar análises mais ricas em tempo de execução, além de permitir o monitoramento da execução. Assim, nesta dissertação, propõe-se uma solução capaz de rastrear transformações de dados que ocorrem no ciclo de vida do aprendizado profundo, fornecendo captura de proveniência eficiente e análise de dados por meio de consultas de proveniência. A abordagem que se propõe modela o ciclo de vida (Figura 2.2) como um fluxo de dados, com a captura dos dados de proveniência, em que os especialistas em redes neurais podem monitorar a evolução do desempenho do modelo durante as iterações do treinamento, além de realizar análises de proveniência mais abrangentes, unindo dados específicos de domínio com dados de aprendizado profundo gerados no ciclo de vida.

Desse modo, esta dissertação especializou o PROV-Df (Figura 2.3) para representar dados específicos da fase de treinamento com o objetivo de facilitar a análise de hiperparâmetros. A representação dos dados segue o modelo de dados baseado nas recomendações do PROV-DM do W3C PROV (MOREAU e GROTH, 2013) que é uma iniciativa para representação de diferentes tipos de dados de proveniência sem ser específico a um domínio. O diagrama da Figura 3.2 representa em termos dos conceitos Agente (*Agent*), Atividade (*Activity*) e Entidade (*Entity*) do W3C PROV as transformações de dados do processo de treinamento das redes neurais. A representação em laranja representa o conceito de Agente, em amarelo representa Entidade e em azul representa Atividade. O diagrama mostra o que foi usado diretamente do PROV-Df (com a *tag* dfanalyzer) e o que foi estendido na CNNProv (com a *tag* cnnprov).

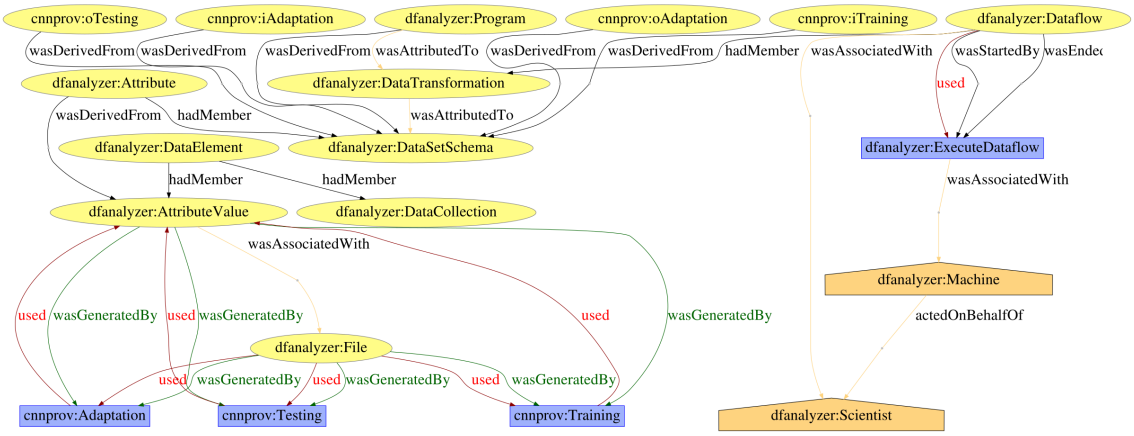

Figura 3.2: Representação da CNNProv adaptado do PROV-Df.

A representação W3C PROV é agnóstica quanto a estruturação dos dados para consulta. Como a DfAnalyzer adota o modelo relacional para a representação interna, a Figura 3.3 mostra as classes correspondentes, mantendo-se a compatibilidade com o padrão de representação W3C PROV. As classes destacadas na área verde são classes relacionadas à especificação do fluxo de dados de aprendizado profundo e compõem o *schema* específico para tal, que será explicado na Seção 3.3.

A Figura 3.4 destaca as classes e atributos que representam os dados de treinamento e hiperparâmetros. Essa representação engloba dados relativos à estrutura do fluxo de dados, o que corresponde à proveniência prospectiva, dados de execução, que são relativos à proveniência retrospectiva, e dados de domínio. Comparando-se o PROV-Df, as contribuições da representação desta dissertação se encontram na representação dos conjuntos de dados específicos ao treinamento do modelo de aprendizado profundo, a atualização/adaptação do modelo e avaliação do mesmo, de forma genérica do ponto de vista da aplicação.

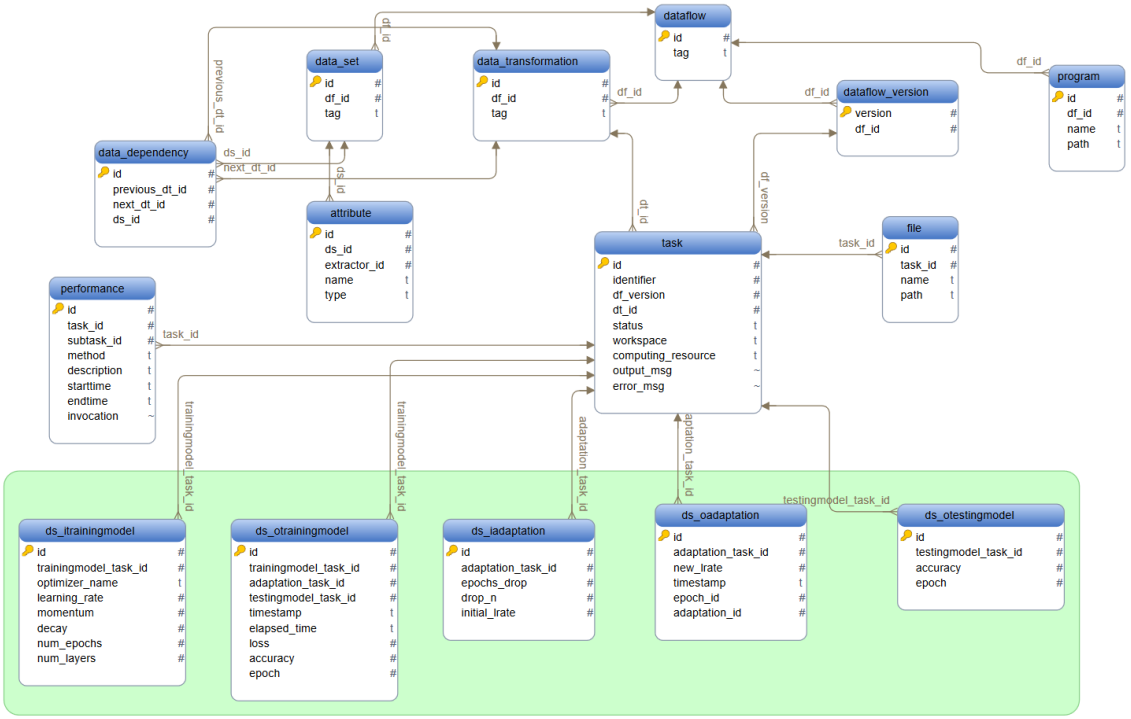

Figura 3.3: Representação de dados para os dados de proveniência e dados de aprendizado profundo capturados no treinamento de uma rede neural.

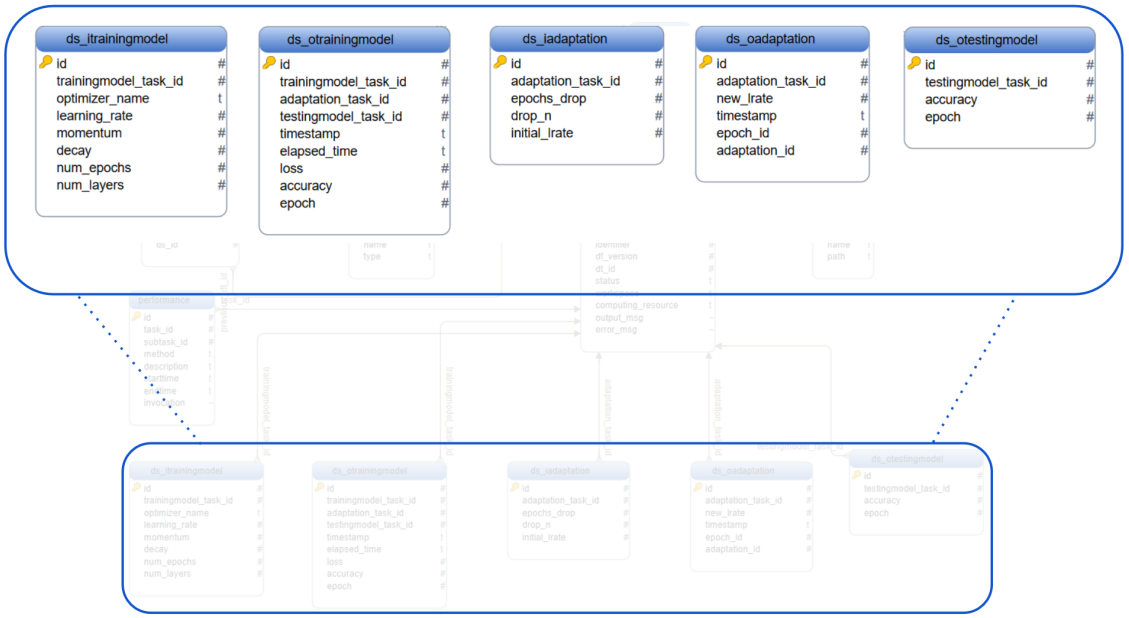

Figura 3.4: Representação de dados com dados de proveniência destacando-se as classes e atributos que representam os dados de treinamento e hiperparâmetros.

No ciclo de vida de aprendizado profundo, são escolhidos hiperparâmetros para o treinamento, por exemplo, taxa de aprendizado, *batch size*, número de épocas, *momentum* e *dropout* que serão otimizados no processo de treinamento. O processo de treinamento é intensivo em computação, geralmente executado em um ambiente de processamento de alto desempenho, e nesse processo, diferentes modelos são treinados, dentre os quais um é escolhido como o "melhor", dependendo das métricas de avaliação. Além disso, como o processo de treinamento leva muito tempo, é necessário monitorá-lo, por exemplo, inspecionando como as métricas de avaliação estão evoluindo enquanto o processo de treinamento é repetido, de maneira que pode-se esperar até a conclusão ou interromper o processo de treinamento, para alterar parâmetros e recomeçar o treinamento, até ficarem satisfeitos com os resultados.

Nesse contexto, em termos de modelagem, as classes destacadas na área da representação de dados da Figura 3.3 descrevem esse ciclo de vida. As classes relacionadas à especificação do fluxo de dados são oriundas do modelo apresentado por SILVA *et al.* (2018b), sendo elas *dataflow*, *data_transformation*, *data_set*, *data_dependency* e *attribute*. Todos os dados associados a essas classes são dados de proveniência prospectiva. A classe *dataflow* é a responsável por representar os diferentes fluxos de dados cujo conteúdo foi armazenado no banco de dados de proveniência. Essa classe, basicamente, contém um nome para cada fluxo de dados. A classe *data_transformation* representa as múltiplas transformações de dados associadas a um determinado fluxo de dados. Uma transformação está associada a um programa ou *script* que é executado e seus dados se encontram representados na classe *program*. Essas relações permitem apenas o armazenamento de dados de proveniência, ou seja, sem os dados de domínio.

Para permitir o armazenamento de dados de proveniência retrospectiva, o modelo de dados proposto por SILVA *et al.* (2018b) ainda contém duas relações, *task* e *file*, para representar as instâncias das transformações de dados (também conhecidas como tarefas) que foram executadas, assim como os dados que foram eventualmente consumidos e produzidos em tempo de execução, respectivamente.

Como descrito por MIAO *et al.* (2015), o ajuste dos hiperparâmetros ocorre após cada iteração. Com base nos resultados obtidos nas iterações anteriores, o especialista em redes neurais ajusta os parâmetros *offline*, e prepara uma nova configuração para a rede. Para decidir sobre uma próxima configuração de hiperparâmetros, o especialista em redes neurais precisa avaliar vários dados associados ao comportamento da rede e seu treinamento junto à configuração de hiperparâmetros usada naquela iteração. Exemplos desses dados são o tempo de treinamento de cada época e a perda associada a cada época, tendo todos esses dados associados a uma determinada configuração de hiperparâmetros registrada para consultas durante e após o treinamento. Nesse sentido, para que o modelo seja capaz de armazenar dados

que satisfaçam essa necessidade do especialista, o treinamento da rede foi dividido nas classes *ds_ itrainingmodel*, *ds_ otrainingmodel*, *ds_ iadaptation*, *ds_ oadaptation* e *ds_ otestingmodel*.

A classe *ds_ itrainingmodel* contém os hiperparâmetros envolvidos no modelo da rede neural. Alguns desses hiperparâmetros são taxa de aprendizado, número de épocas, otimizador, *momentum* e *decay*. Enquanto isso, a classe *ds_ otrainingmodel* contém métricas de desempenho do modelo por época. Essa classe define dados como o valor da época, da acurácia, da função de custo (*loss function*), o tempo decorrido e a data e hora do fim da execução da época. O tempo decorrido definido nesta classe possibilita, por exemplo, que especialistas verifiquem se uma época está levando mais tempo que o usual.

Muitas vezes, são feitas adaptações durante o treinamento de uma CNN. Por exemplo, uma adaptação pode gerar uma nova taxa de aprendizado ao fim de uma época, utilizando uma função que diminui significativamente o valor da taxa de aprendizado a cada n épocas em um fator m. Dessa forma, a transformação *Adaptation* recebe como dados de entrada o conjunto produzido pela transformação anterior, transformação *Training*, e um conjunto de dados com informações como o fator m, o valor de n e a taxa de aprendizado inicial. Esse conjunto de dados é representado pela classe *ds_ iadaptation*. O conjunto de dados produzido por essa transformação, representado pela classe *ds_ oadaptation*, contém a nova taxa de aprendizado, o valor da época e a data e hora em que a adaptação ocorreu, além de uma identificação para a adaptação.

Por último, a classe *ds_ otesting* está relacionada à transformação *Testing* e provê dados acerca da avaliação do modelo. Dessa forma, assim como a classe *ds_ otrainingmodel*, *ds_ otesting* contém métricas de desempenho, como valores de acurácia e função de custo.

Dessa forma, o esquema de dados da CNNProv representa os dados dos hiperparâmetros com proveniência e pode ser estendido com dados de domínio, de maneira que outros hiperparâmetros possam ser capturados durante o treinamento da rede neural.

3.2 Arquitetura de *software* da CNNProv

Devido às peculiaridades das redes neurais profundas, algumas adaptações foram realizadas no modelo de dados da DfAnalyzer, de modo a facilitar sua integração junto ao treinamento da rede e a análise de dados (PINA *et al.*, 2019). As principais adaptações realizadas foram nas definições de transformações de dados e hiperparâmetros que serão rastreados.

Para uma solução de proveniência e análise de configurações de hiperparâmetros, é necessário um conjunto rico de dados que estejam relacionados de maneira a serem analisados. Dessa forma, uma vez modelados os dados da rede neural e do domínio, a arquitetura atua para prover automaticamente a captura dos dados, a representação e o armazenamento no SGBD.

Nesse sentido, a CNNProv é composta pelos componentes (i) Instrumentação, (ii) Captura, (iii) Análise/Consultas e (iv) Armazenamento, conforme mostrado na Figura 3.5, sendo componentes estendidos da DfAnalyzer [1]. Os dois primeiros componentes são invocados ao instrumentar o código da rede neural, enquanto os outros têm interfaces independentes para o especialista consultar e analisar dados em tempo de execução. As extensões foram realizadas no esquema do banco de dados que armazena os modelos treinados e na captura do histórico de derivação dos dados.

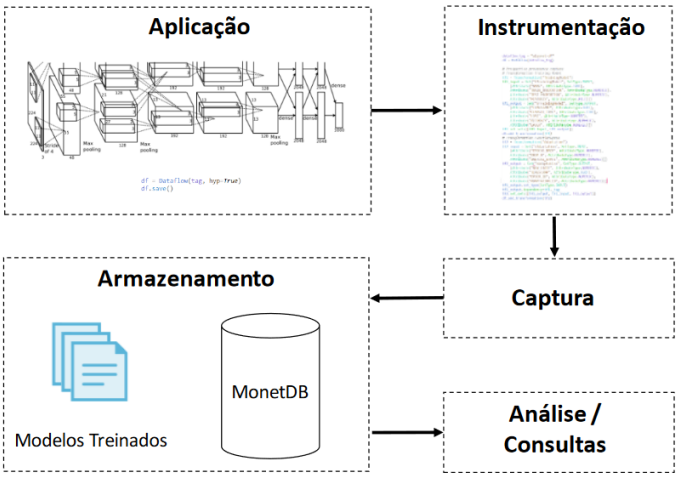

Figura 3.5: Arquitetura da CNNProv.

O componente Instrumentação é responsável pela definição da especificação das classes descritas na representação da Figura 3.4, que dizem respeito aos hiperparâmetros inerentes ao modelo de aprendizado profundo de forma que o especialista não precise adaptar seu código. Os detalhes de implementação deste componente são descritos na Seção 3.3.

O componente Captura é responsável pela captura dos dados de proveniência prospectiva e retrospectiva. São capturados os hiperparâmetros não inerentes ao modelo de aprendizado profundo, que, nesta dissertação, são considerados como dados de domínio, dados das métricas do treinamento da CNN e dados de proveniência junto à execução da rede neural profunda. Esses dados são representados seguindo o formalismo de fluxo de dados apresentado em (SILVA, 2018).

[1] https://gitlab.com/ssvitor/dataflow_analyzer

O componente Armazenamento realiza a carga dos dados capturados em uma base de dados estendido para modelos de CNN utilizando o SGBD relacional orientado a coluna MonetDB (BONCZ *et al.*, 2008). Optamos por manter o mesmo SGBD da arquitetura da DfAnalyzer, por ser eficiente na inserção de dados, além de facilitar as análises de dados sobre os valores de um mesmo atributo.

Já o componente Análise/Consultas consiste em uma interface gráfica que fornece uma visão do fluxo de dados a partir da perspectiva do conjunto de dados, conforme mostrado na Figura 3.6, que apresenta a representação da especificação do fluxo de dados do ciclo de vida de aprendizado profundo definida na representação de dados da Figura 3.4. O grafo é criado dinamicamente a partir da especificação do fluxo de dados. Usualmente, os vértices representam as transformações de dados e as arestas direcionadas representam os conjuntos de dados entre as transformações, porém, como na especificação da consulta o especialista em redes neurais está interessado em elementos de dados presentes em conjuntos de dados, esta dissertação assume que os vértices do grafo direcionado acíclico (DAG) correspondem aos conjuntos de dados entre as transformações. Além disso, a cor verde nos vértices da Figura 3.6 indica os conjuntos de dados de entrada, enquanto os vértices em azul são conjuntos de dados intermediários (saída de uma transformação e entrada de outra). Esse componente permite que os especialistas consultem as especificações de fluxos de dados que já foram registradas na base de dados, através da disponibilização de uma lista desses fluxos para que o especialista opte por um. Além disso, os especialistas são capazes de clicar nos conjuntos de dados para acessarem seus esquemas, visualizando os atributos pertencentes ao conjunto de dados que foi selecionado.

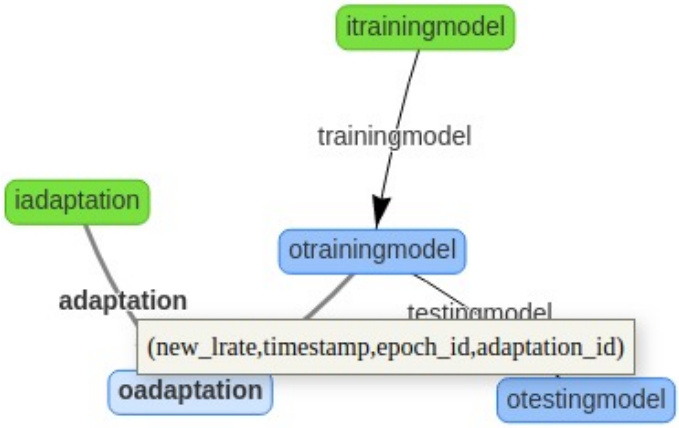

Figura 3.6: Exemplo de grafo apresentado pelo componente de análise.

Tal componente permite o processamento de consultas por meio da edição e submissão da consulta através de uma interface gráfica, apresentada em (PINA, 2018). É necessário especificar os conjuntos de dados que serão utilizados na análise, os atributos a serem projetados e as seleções de atributos a serem consideradas.

Assim, esse componente utiliza esses argumentos para a especificação e execução da consulta.

Vale ressaltar que a CNNProv é uma solução para permitir a análise de configurações de hiperparâmetros, isto é, registrar hiperparâmetros bem como dados relacionados ao treinamento da CNN e não uma solução para realização do ajuste e otimização de hiperparâmetros.

3.3 Desenvolvimento da arquitetura

Para contornar a questão de especificidade à linguagem de programação, a solução é disponibilizada na forma de biblioteca de serviços a serem invocados independente de linguagem de programação. Assim, o código fonte pode invocar os serviços disponibilizados pela CNNProv. Vale ressaltar que a CNNProv pode ser executada em qualquer ambiente, estando inclusive preparada para execução em ambiente paralelo ou distribuído, assim, se resolve a questão de ser específica a um ambiente de execução. Para a flexibilidade na escolha do que será monitorado, o especialista instrumenta seu código de treinamento de maneira a especificar esses dados. Para a representação de dados, seguimos o padrão W3C PROV, para disponibilizar os dados durante a execução e evitar impacto significativo no tempo de execução, a solução caracteriza-se por ser assíncrona, utilizando recursos de processamento e armazenamento que não competem com o treinamento da rede.

Além disso, para que os componentes da CNNProv sejam agnósticos à linguagem de programação adotada pelas bibliotecas de treinamento de CNNs, essa solução foi disponibilizada como um conjunto de serviços RESTful que auxiliam o especialista na captura de dados de proveniência, hiperparâmetros e outros dados relacionados ao treinamento da CNN, que podem ser invocados a partir do código fonte do treinamento. Para permitir a captura e o armazenamento dos dados de proveniência prospectiva na base de dados da CNNProv, a aplicação RESTful recebe e processa requisições HTTP com o método POST em que o conteúdo ou a mensagem dessa requisição deve apresentar esses dados de proveniência.

Muitas ferramentas exigem a especificação do fluxo de dados antes do início da execução, momento em que a ferramenta cria uma base de dados de proveniência. A especificação prévia é utilizada para a composição da estrutura da base de dados de proveniência (criação de tabelas, colunas e relacionamentos, por exemplo) e especificar o fluxo de dados pode ser uma tarefa árdua para o especialista do domínio científico, visto que precisa ser detalhada e correta para que a base de dados seja corretamente construída. Para lidar com essa especificação, implementamos o componente Instrumentação.

Para o desenvolvimento desse componente foi utilizado a DfAnalyzer, implementada em Java e RESTful com Spring Boot, e o banco de dados MonetDB. Consideramos os dados inerentes ao modelo de aprendizado profundo como dados imprescindíveis para a análise de configurações de hiperparâmetros. Nesse sentido, a CNNProv instrumenta automaticamente tal especificação de transformações de dados e conjuntos de dados. A especificação é responsável pela construção dinâmica do esquema. A Figura 3.7 apresenta um trecho de código para essa modelagem.

Desse modo, ao invés de especificar tal fluxo de dados, é necessário acrescentar ao *script* de treinamento apenas o código descrito na Figura 3.8, informando a *tag* para identificação do fluxo de dados e o valor *True* para a captura de hiperparâmetros. Assim, a CNNProv identificará que o especialista deseja capturar os dados dos hiperparâmetros definidos na representação de dados da Figura 3.4.

O especialista também pode definir outras transformações de dados, inclusive para captura de hiperparâmetros que sejam específicos de sua CNN (dados de domínio). Para isso, é preciso, juntamente com o código da Figura 3.8, especificar as transformações de dados, os conjuntos de dados e as dependências existentes. Dessa forma, a CNNProv identifica que, além das transformações definidas pela representação (Figura 3.4), as transformações definidas pelo especialista também fazem parte do fluxo de dados.

Além da extensão da DfAnalyzer para captura dos hiperparâmetros, a biblioteca DFA-LIB-PYTHON (CAMPOS, 2018), voltada para captura de dados de proveniência em *scripts* escritos em Python, também foi modificada para seguir a captura de hiperparâmetros inerentes à rede.

Além disso, foram feitas modificações para o armazenamento dos hiperparâmetros e dados de treinamento das CNNs no banco de dados. A CNNProv considera um *schema* para os dados de proveniência e um *schema* para os hiperparâmetros e dados de treinamento das CNNs, modelados como fluxo de dados. Nesse sentido, a CNNProv armazena dados sobre a definição do fluxo de dados, isto é, transformações de dados e conjuntos de dados, em um *schema* global, enquanto os hiperparâmetros são armazenados em um *schema* específico para o fluxo de dados definido pelo especialista. Assim, cada fluxo de dados possui suas tabelas acerca do treinamento da CNN, como destacado em verde na Figura 3.4.

3.4 Metodologia de uso da CNNProv

A utilização da CNNProv implica seguir uma metodologia que auxilia na modelagem de dados das aplicações, no uso dos serviços para realizar a captura de dados, na geração dos dados de proveniência e nas consultas aos dados em geral. Assim, é preciso acrescentar ao *script* do especialista a chamada para a CNNProv. É

```
tf1 = Transformation("TrainingModel")
tf1_input = Set("iTrainingModel", SetType.INPUT,
    [Attribute("OPTIMIZER_NAME", AttributeType.TEXT),
    Attribute("LEARNING_RATE", AttributeType.NUMERIC),
    ...
    Attribute("MOMENTUM", AttributeType.NUMERIC),
    Attribute("DECAY", AttributeType.NUMERIC)])
tf1_output = Set("oTrainingModel", SetType.OUTPUT,
    [Attribute("TIMESTAMP", AttributeType.TEXT),
    Attribute("ELAPSED_TIME", AttributeType.TEXT),
    Attribute("LOSS", AttributeType.NUMERIC),
    Attribute("ACCURACY", AttributeType.NUMERIC),
    Attribute("EPOCH", AttributeType.NUMERIC)])
tf1.set_sets([tf1_input, tf1_output])
df.add_transformation(tf1)
tf2 = Transformation("Adaptation")
tf2_output = Set("oAdaptation", SetType.OUTPUT,
    [Attribute("NEW_LRATE", AttributeType.NUMERIC),
    Attribute("TIMESTAMP", AttributeType.TEXT),
    Attribute("EPOCH_ID", AttributeType.NUMERIC),
    Attribute("ADAPTATION_ID", AttributeType.NUMERIC)])
tf1_output.set_type(SetType.INPUT)
tf1_output.dependency=tf1._tag
tf2.set_sets([tf1_output, tf2_output])
df.add_transformation(tf2)
tf3 = Transformation("TestingModel")
tf3_output = Set("oTestingModel", SetType.OUTPUT,
    [Attribute("ACCURACY", AttributeType.NUMERIC),
    Attribute("EPOCH", AttributeType.NUMERIC)])
tf1_output.set_type(SetType.INPUT)
tf1_output.dependency=tf1._tag
tf3.set_sets([tf1_output, tf3_output])
df.add_transformation(tf3)
```

Figura 3.7: Código utilizado pela CNNProv para captura de hiperparâmetros e dados de proveniência.

```
df = Dataflow(dataflow_tag, True)
df.save()
```

Figura 3.8: Trecho de código a ser inserido para utilização da CNNProv.

importante ressaltar que são incluídas chamadas, mas não é necessário reprogramar o código do treinamento.

Para isso, inicialmente trabalha-se na especificação do fluxo de dados, seguindo o fluxo proposto na representação de dados (Figura 3.4) para o ciclo de vida do treinamento de redes neurais e acrescentando dados e transformações, quando necessário. Por conseguinte, os especialistas precisam realizar os ajustes necessários no código do treinamento da rede para capturar os dados de proveniência. Nessa seção, apresentamos os passos envolvidos na modelagem dos dados de proveniência, considerando um exemplo de *script* de rede neural em Python usando Keras [2] para classificação de imagens, conforme mostra a Figura 3.9.

Dessa forma, a primeira etapa do uso da CNNProv é a modelagem da aplicação de aprendizado profundo como um fluxo de dados. Nessa etapa, são identificadas as transformações de dados, os conjuntos de dados, os atributos existentes em cada conjunto de dados que devem ser capturados, e as dependências de dados entre as transformações. Assim, obtemos a especificação do fluxo de dados para o treinamento da rede neural de interesse. Nessa primeira fase, o especialista em redes neurais define as transformações de dados a serem rastreadas e quais hiperparâmetros serão considerados.

Para os dados de proveniência prospectiva, seguindo a representação proposta em 3.4, a Figura 3.10 apresenta um *script* em Python que seria invocado antes do *script* para o treinamento da rede, com o objetivo de armazenar a especificação do fluxo de dados na base de dados com a utilização da DfAnalyzer.

Com a utilização da CNNProv, é necessário especificar um identificador para o fluxo de dados e *True* para que a CNNProv possa utilizar a especificação da representação de dados da Figura 3.4 para o fluxo de dados. A Figura 3.11 apresenta esse código que é invocado antes do treinamento da rede, com o objetivo de armazenar a especificação do fluxo de dados na base de dados da CNNProv. Além disso, os especialistas podem considerar a inclusão de outras transformações. Um exemplo é apresentado no Capítulo 4.

A partir da especificação do fluxo de dados, é realizada a adaptação do código que modela o treinamento com o intuito de capturar os dados de proveniência produzidos, além dos dados relacionados aos hiperparâmetros. Nessa segunda etapa, o especialista indica, no código, onde obter os valores dos hiperparâmetros, para serem capturados em tempo de execução. A cada instância de hiperparâmetros em que o treinamento da rede é feito, novos dados são adicionados ao banco de dados.

Com isso, notamos que enquanto a especificação do fluxo de dados considera dados de proveniência prospectiva, a segunda etapa é importante no apoio a captura de dados de proveniência retrospectiva. Para mais, essa segunda etapa é valiosa na

[2] https://keras.io/

```
x, y = oxflower17.load_data(one_hot=True)
# learning rate schedule
def step_decay(epoch):
    lrate = initial_lrate * math.pow(drop,
       math.floor((1+epoch)/epochs_drop))
    return lrate

lrate = LearningRateScheduler(step_decay, verbose=1)

model = Sequential()
model.add(Conv2D(filters=96, input_shape=(224,224,3),
   kernel_size=(11,11), strides=(4,4), padding='valid'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
   padding='valid'))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(11,11), strides=(1,1),
   padding='valid'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
   padding='valid'))
model.add(BatchNormalization())
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
   padding='valid'))
model.add(Activation('relu'))
model.add(BatchNormalization())
# outras camadas
model.summary()
optimizer = tf.keras.optimizers.Adam(learning_rate=lerate)
model.compile(loss='categorical_crossentropy',
   optimizer='adam',metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=64,epochs=epochs,
   callbacks=[lrate], verbose=1, validation_split=0.2, shuffle=True)

model.evaluate(x_test, y_test)
```

Figura 3.9: *Script* em Python da AlexNet para classificação de imagens.

obtenção das dependências de dados durante a execução, uma vez que se especifica no código fonte quais tarefas são responsáveis pela produção e pelo consumo de elementos de dados em um determinado conjunto.

Uma vez adaptado o código da CNN e seu treinamento tenha sido iniciado, o componente Captura é invocado assincronamente. Para cada chamada da CNNProv, dados das transformações, dos hiperparâmetros e dos modelos treinados são armazenados no banco de dados de proveniência usando o sistema colunar MonetDB e

```
df = Dataflow("alexnet-df")
tf1 = Transformation("TrainingModel")
tf1_input = Set("iTrainingModel", SetType.INPUT,
    [Attribute("OPTIMIZER_NAME", AttributeType.TEXT),
    Attribute("LEARNING_RATE", AttributeType.NUMERIC),
    Attribute("NUM_EPOCHS", AttributeType.NUMERIC),
    Attribute("NUM_LAYERS", AttributeType.NUMERIC)])
tf1_output = Set("oTrainingModel", SetType.OUTPUT,
    [Attribute("TIMESTAMP", AttributeType.TEXT),
    Attribute("ELAPSED_TIME", AttributeType.TEXT),
    Attribute("LOSS", AttributeType.NUMERIC),
    Attribute("ACCURACY", AttributeType.NUMERIC),
    Attribute("EPOCH", AttributeType.NUMERIC)])
tf1.set_sets([tf1_input, tf1_output])
df.add_transformation(tf1)
tf2 = Transformation("Adaptation")
tf2_output = Set("oAdaptation", SetType.OUTPUT,
    [Attribute("NEW_LRATE", AttributeType.NUMERIC),
    Attribute("TIMESTAMP", AttributeType.TEXT),
    Attribute("EPOCH_ID", AttributeType.NUMERIC),
    Attribute("ADAPTATION_ID", AttributeType.NUMERIC)])
tf1_output.set_type(SetType.INPUT)
tf1_output.dependency=tf1._tag
tf2.set_sets([tf1_output, tf2_output])
df.add_transformation(tf2)
tf3 = Transformation("TestingModel")
tf3_output = Set("oTestingModel", SetType.OUTPUT,
    [Attribute("ACCURACY", AttributeType.NUMERIC),
    Attribute("EPOCH", AttributeType.NUMERIC)])
tf1_output.set_type(SetType.INPUT)
tf1_output.dependency=tf1._tag
tf3.set_sets([tf1_output, tf3_output])
df.add_transformation(tf3)
df.save()
```

Figura 3.10: Código da DfAnalyzer para captura de hiperparâmetros e dados de proveniência.

seguindo a representação adaptada do PROV-Df (SILVA *et al.*, 2018a). Esses dados incluem o histórico de derivação dos dados, erros que tenham ocorrido, a associação entre os valores de hiperparâmetros e os arquivos de modelo treinados. Finalmente, a interface de consulta permite ao especialista analisar os dados de proveniência capturados, seja por meio da submissão de uma consulta ao CNNProv.

A captura de dados de proveniência e seu registro em bancos de dados permite a realização de consultas como por exemplo: "Qual o valor de perda associado a

```
df = Dataflow("meu_fluxo_de_dados", True)
df.save()
```

Figura 3.11: Trecho de código para utilização da CNNProv.

cada época e o tempo decorrido de treinamento para essa época?". A resposta desta consulta é apresentada na Tabela 3.1, que mostra uma tabela com o tempo decorrido em segundos, o número da época e o valor da perda (PINA *et al.*, 2019). Ao analisar os dados da tabela junto à configuração utilizada para os hiperparâmetros, o especialista em redes neurais pode realizar os ajustes com mais segurança e confiança e os dados dessa base de dados também podem auxiliar em futuras configurações.

Tabela 3.1: Exemplo de consulta que mostra o tempo decorrido em cada época por ordem decrescente de perda.

época	**tempo decorrido (em segundos)**	**valor de perda**
20	106.55	0.50
19	110.92	0.53
18	113.86	0.66
17	100.06	0.71
16	109.21	0.73

3.5 Keras-Prov

Para que a proveniência seja armazenada em uma base de dados, é necessário fornecer uma definição do fluxo de dados. Com essa especificação, a ferramenta é capaz de construir as estruturas de dados adequadas para armazenar os dados que serão produzidos durante o treinamento da rede neural. A necessidade de especificação da proveniência prospectiva e retrospectiva, isto é, a modelagem do fluxo de dados e adaptação do código para obtenção dos dados, respectivamente, pode ser uma barreira para a adoção de ferramentas de proveniência, visto o esforço que pode ser exigido do especialista ao longo do ciclo de vida da modelagem de CNNs.

Nesse sentido, essa parte da dissertação introduz um subcomponente da CNNProv adaptado do Keras [3]. O principal objetivo desse subcomponente é reduzir o esforço de instrumentação do código quando o Keras é utilizado para o treinamento da rede neural, minimizando, assim, as ações que o especialista deve realizar para captura de dados de proveniência. O Keras foi escolhido por ser uma biblioteca de rede neural de código aberto muito utilizado e com muitas informações disponíveis a respeito de suas características, estruturas e funcionamento.

[3] https://keras.io/

Dessa forma, introduzimos o Keras-Prov, que lida com a criação das transformações e conjuntos de dados relacionados aos hiperparâmetros e métricas seguindo a representação da Figura 3.4 e com o armazenamento desses dados. A arquitetura da CNNProv com esse subcomponente é apresentada na Figura 3.12 e conta com os componentes de captura de hiperparâmetros e dados de treinamento, armazenamento e análise/consultas, como na CNNProv. O componente de captura conta com o subcomponente Keras-Prov. Esses subcomponentes são extensíveis, de forma que diferentes bibliotecas de treinamento de CNNs podem ser modificadas para automaticamente capturar e armazenar hiperparâmetros e outros dados de treinamento.

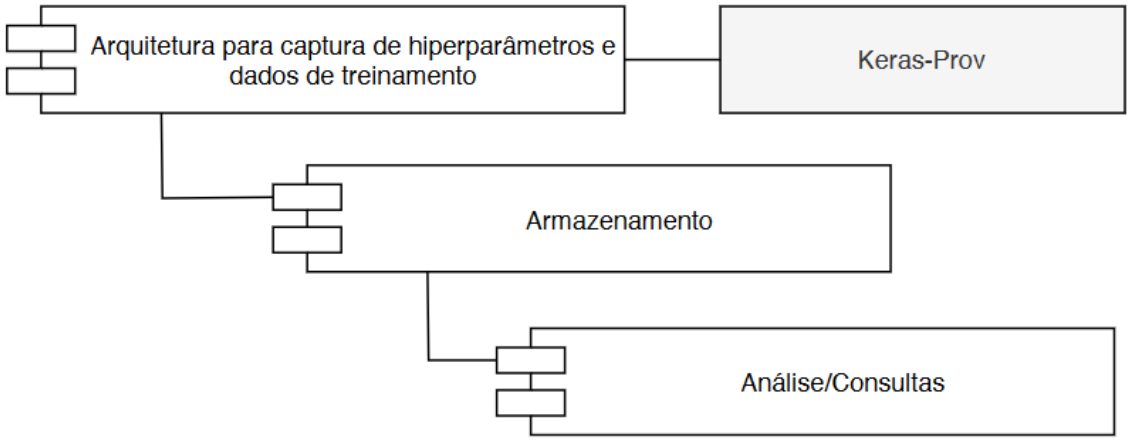

Figura 3.12: Arquitetura da CNNProv com o Keras-Prov.

Na implementação desse subcomponente, foram feitas modificações no código fonte do Keras [4]. Uma classe *Provenance* foi criada contendo métodos para lidar com a criação das transformações que acompanham o ciclo de vida de aprendizado profundo, seguindo a representação apresentada na Figura 3.4.

Na classe *Model* foi criada um método *provenance* que deve ser utilizado para capturar dados de proveniência. Nesse sentido, esse método recebe uma *tag* para identificação do fluxo de dados, se existe adaptação dos hiperparâmetros durante o treinamento (atualização da taxa de aprendizado, por exemplo), isto é, utilização de métodos como *LearningRateScheduler* e *ReduceLROnPlateau* oferecidos pelo Keras, e a lista de hiperparâmetros que se deseja capturar. O método definido é mostrado na Figura 3.13.

Os dados recebidos pelo método *provenance* são definidos pelo especialista no código da aplicação de aprendizado profundo seguindo o formato apresentado na Figura 3.14. Assim, nota-se que é necessário acrescentar ao código as seguintes linhas, colocando *True* para aqueles hiperparâmetros que devem ser armazenados e *False*

[4]https://github.com/keras-team/keras

```
    def provenance(self,
                   dataflow_tag="",
                   adaptation=False,
                   hyperparameters={},
                   **kwargs):

        self.dataflow_tag = dataflow_tag
        p = Provenance(dataflow_tag)
```

Figura 3.13: Código inserido no Keras para definição dos hiperparâmetros a serem capturados.

para aqueles que não devem ser armazenados, seguido da chamada do método *provenance* com as informações necessárias. Desse modo, Keras-Prov automaticamente criará as transformações *Training*, *Adaptation* e *Testing*.

```
hyps = {"OPTIMIZER_NAME": True,
        "LEARNING_RATE": True,
        "DECAY": False,
        "MOMENTUM": False,
        "NUM_EPOCHS": True,
        "BATCH_SIZE": False,
        "NUM_LAYERS": True}

model.provenance(dataflow_tag="tag-para-df",
                 adaptation=True,
                 hyperparameters = hyps)
```

Figura 3.14: Código que deve ser adicionado à aplicação para captura de hiperparâmetros e dados de proveniência usando o Keras-Prov.

Para a obtenção dos valores dos dados e armazenamento na base de dados, acrescentou-se em pontos do código do Keras, as chamadas para a CNNProv com esses valores, como apresentado na Figura 3.15. Nessa parte do código, dados relativos à transformação *Training* são adicionados aos seus respectivos conjuntos de dados. A variável *values_to_store* insere os valores para os hiperparâmetros que foram selecionados na etapa anterior (Figura 3.14).

Portanto, como as chamadas para a captura de hiperparâmetros e métricas são incluídas diretamente no Keras, ao invés de chamadas no código da aplicação de aprendizado profundo, é possível fazer esses acréscimos de chamadas em outras

```
def fit_loop(model, fit_function, fit_inputs,
             out_labels=None,
             batch_size=None,
             epochs=100,
             verbose=1,
             callbacks=None,
             val_function=None,
             val_inputs=None,
             shuffle=True,
             initial_epoch=0,
             steps_per_epoch=None,
             validation_steps=None,
             validation_freq=1):
    ...
    t1_input = DataSet("iTrainingModel", [Element(values_to_store)])
    t1.add_dataset(t1_input)
    t1.begin()
    ...
    t1_output = DataSet("oTrainingModel", [Element([now, now -
        start, loss, acc, epoch])])
    t1.add_dataset(t1_output)
    if(epoch==final_epoch):
        t1.end()
    else:
        t1.save()
```

Figura 3.15: Código adicionado ao Keras para inserção dos dados.

ferramentas de redes neurais para captura dos dados, desde que tenham o código aberto.

Capítulo 4

Avaliação Experimental

Este capítulo apresenta os resultados obtidos utilizando a CNNProv para a análise de configurações de hiperparâmetros no treinamento de redes de aprendizado profundo através da utilização de dados de proveniência. A adoção da CNNProv para analisar configurações dos hiperparâmetros forma um histórico para enriquecer as análises de maneira a auxiliar o especialista em redes neurais na tomada de decisão. Para obtenção dos resultados, duas redes neurais convolucionais foram executadas no *cluster* Lobo Carneiro da COPPE/UFRJ. Os experimentos realizados tiveram o objetivo de (i) analisar a sobrecarga da CNNProv no treinamento das redes neurais consideradas e (ii) analisar, por meio de consultas, as configurações de hiperparâmetros e métricas armazenadas.

As CNNs escolhidas para os experimentos são a AlexNet e DenseED. A AlexNet foi escolhida por ser uma CNN bastante popular e por ser algo como um *benchmark*. Já a DenseED foi escolhida por ser um caso real considerado complicado no ramo de imageamento sísmico e que pode tirar proveito da proveniência.

Uma vez tendo o código de treinamento das redes neurais AlexNet e DenseED instrumentados para a captura de dados e registro na base de dados com a CNNProv, foram feitas execuções variando-se hiperparâmetros como o número de épocas, a taxa de aprendizado e o otimizador. O otimizador foi variado de maneira a explorar outros hiperparâmetros como *momentum* e *decay*. Cada combinação de hiperparâmetros foi executada cinco vezes e a média de tempo de execução foi considerada.

A Seção 4.1 apresenta o ambiente de experimentos, enquanto as Seções 4.2 e 4.3 apresenta os casos de estudo.

4.1 Ambiente computacional

Os experimentos descritos neste capítulo foram executados no *cluster* de computadores Lobo Carneiro, também conhecido como LoboC, do Núcleo Avançado de Computação de Alto Desempenho (NACAD) da COPPE/UFRJ. O LoboC é um cluster

SGI ICE-X Linux com 504 CPUs Intel Xeon E5-2670v3 (Haswell), totalizando 6.048 processadores. Os processadores contam com a tecnologia Hyper-Threading (HT), oferecendo 48 threads de processamento por nó, com 64GB de memória RAM. Os nós computacionais são interconectados com a tecnologia InfiniBand FDR –56 Gbs (Hypercube). O *cluster* funciona sob a arquitetura de disco compartilhado, com um sistema de arquivos paralelo Intel Lustre com capacidade de armazenamento de 500 TB.

Para a utilização da CNNProv nos experimentos, o SGBD MonetDB foi instanciado em um nó computacional dedicado para armazenar os dados de proveniência e os hiperparâmetros. A CNNProv recebe as requisições HTTP com os dados a serem armazenados e, em seguida, estabelece uma conexão com a base de dados do MonetDB por meio do driver JDBC para carregar os novos dados.

4.2 Estudo de caso: AlexNet

AlexNet (KRIZHEVSKY *et al.*, 2012) é uma CNN voltada para o reconhecimento de imagens e que venceu em 2012 o desafio anual do *ImageNet*, que consiste num desafio voltado para a criação de métodos e arquiteturas de aprendizado para a classificação da base de dados *ImageNet* (DENG *et al.*, 2009). Dessa forma, a AlexNet foi a arquitetura que popularizou CNNs para visão computacional.

Arquitetura

A evolução da capacidade das placas gráficas junto com implementações altamente otimizadas de convoluções possibilitou a criação dessa arquitetura que, na época, era uma das maiores já existentes, composta por oito camadas, sendo as cinco primeiras camadas convolucionais e as três camadas restantes completamente conectadas, como pode ser visto na Figura 4.1.

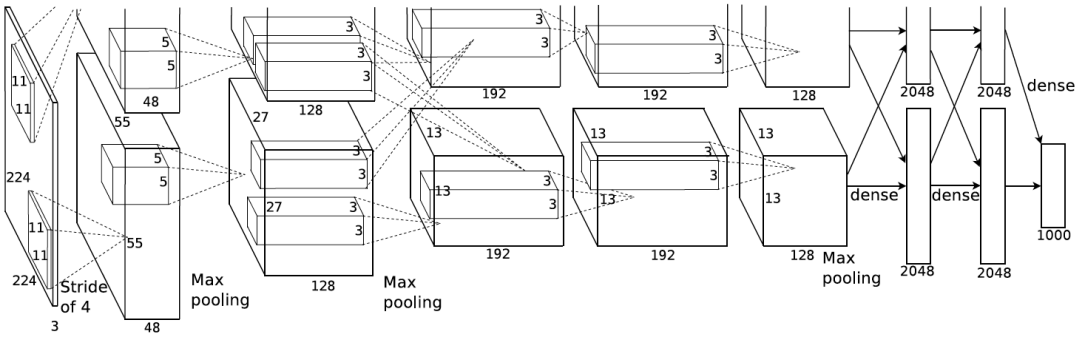

Figura 4.1: Arquitetura da AlexNet (KRIZHEVSKY *et al.*, 2012).

A AlexNet recebe como entrada imagens de 224×224 pixels por canal. Na primeira camada de convolução utiliza um filtro de 11×11×3, na segunda 5×5×3 e da terceira em diante 3×3×3 (ALOM *et al.*, 2018). Além disso, a terceira, a quarta e a quinta camadas são conectadas sem utilização de *pooling*. Os núcleos da terceira camada convolucional são conectados a todos os mapas do *kernel* na segunda camada. Os neurônios nas camadas totalmente conectadas são conectados a todos os neurônios da camada anterior. As camadas de normalização de resposta seguem a primeira e a segunda camadas convolucionais. As camadas de *pool* máximo seguem as camadas de normalização de resposta e a quinta camada convolucional. Por fim, a rede possui duas camadas totalmente conectadas com 2048 neurônios cada e uma camada de saída com 1000 neurônios.

A redução na quantidade de parâmetros e no tempo de treinamento é alcançada ao conectar os filtros da segunda, quarta e quinta camadas convolucionais apenas aos mapas de filtro das camadas anteriores que estão na mesma GPU – os filtros da terceira camada são conectados a todos os mapas de filtros da segunda camada. Para reduzir o problema da dissipação do gradiente, após cada camada convolucional, aplica-se a função de ativação *Relu*. Para reduzir sobreajuste, antes da primeira e segunda camadas completamente conectadas, é incluída uma operação de *dropout* (KRIZHEVSKY *et al.*, 2012).

Conjunto de dados

O conjunto de dados *Oxford Flower* (NILSBACK e ZISSERMAN, 2006) consiste em dezessete espécies de flores com 80 imagens para cada classe (Figura 4.2). As flores escolhidas são algumas flores comuns no Reino Unido. Todas as imagens são em cores, formato JPEG e o tamanho médio da imagem é 560×560 pixels. As imagens têm variações de grande escala, pose e luz e, além disso, também existem classes com grandes variações de imagens dentro da classe e semelhança próxima a outras classes.

As categorias de flores deste conjunto de dados foram escolhidas deliberadamente para ter alguma ambiguidade em cada aspecto. Por exemplo, algumas classes não podem ser distinguidas apenas na cor, como dente-de-leão e ranúnculos, outras não podem ser distinguidas apenas na forma, como narcisos e anêmona silvestre. As imagens das flores foram recuperadas de vários sites, com algumas imagens complementares de fotografias próprias dos autores (NILSBACK e ZISSERMAN, 2006).

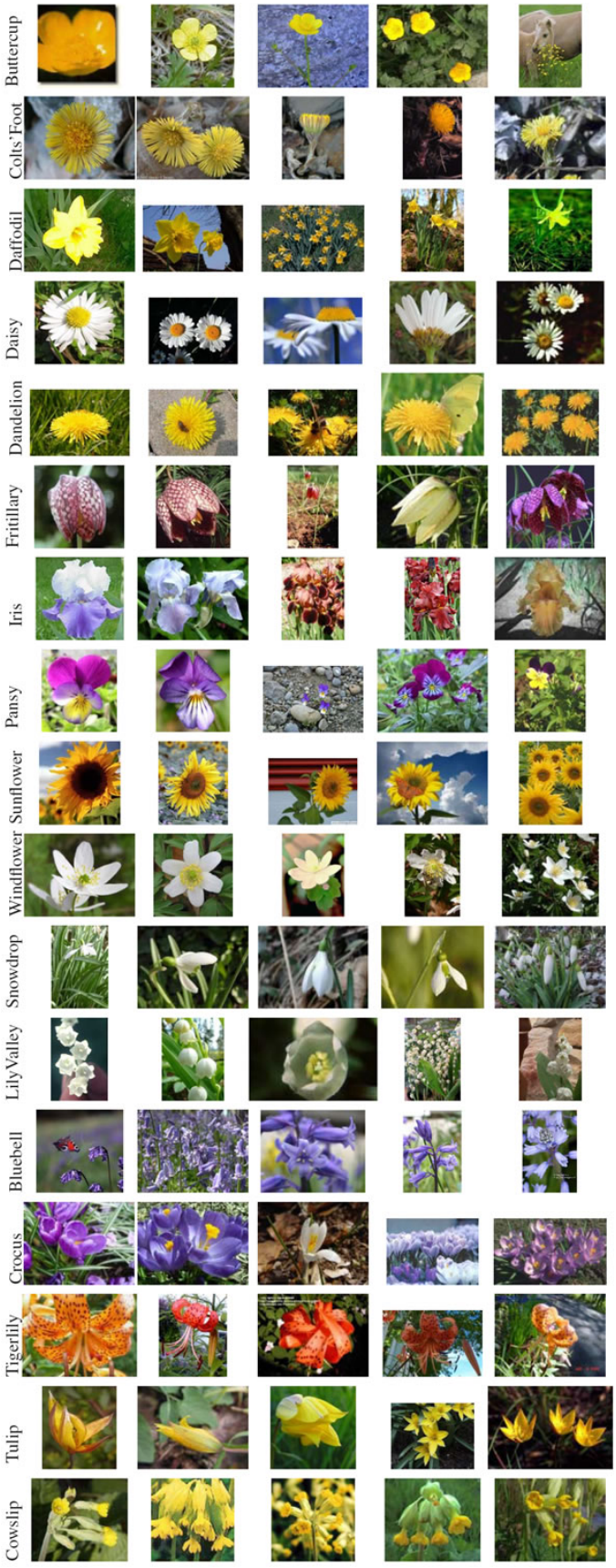

Figura 4.2: Imagens do conjunto de dados (NILSBACK e ZISSERMAN, 2006).

4.2.1 Instrumentação com a CNNProv

A implementação da AlexNet para esta dissertação foi feita utilizando o Keras [1]. Para a captura e registro de dados do treinamento, são identificadas as transformações de dados envolvidas no processo de treinamento da AlexNet: *Training*, *Adaptation* e *Testing*. Essas transformações já são contempladas pelo modelo de dados da CNNProv. Nesse sentido, a transformação *Training* consome o nome do otimizador utilizado no treinamento, os hiperparâmetros taxa de aprendizado, número de épocas e número de camadas da rede, e produz um conjunto de dados que define os dados gerados durante o treinamento, como o valor da época, da acurácia, da função de custo (*loss function*), o tempo decorrido e a data e hora do fim da execução da época. A adaptação que é realizada pela AlexNet gera uma nova taxa de aprendizado (α') ao fim da época utilizando a função *Scheduler Step Decay* que diminui significativamente o valor da taxa de aprendizado (α) a cada n épocas em um fator m. Dessa forma, a transformação *Adaptation* recebe como dados de entrada o conjunto produzido pela transformação anterior, *Training*, e um conjunto de dados com informações como o fator m, o valor de n e a taxa de aprendizado inicial. O conjunto de dados produzido por essa transformação contém a nova taxa de aprendizado, o valor da época e a data e hora em que a adaptação ocorreu, além de uma identificação para a adaptação e *oAdaptation* representa o conjunto de dados de saída da transformação *Adaptation*. Por último, a transformação *Testing* provê uma avaliação do modelo de acordo com o conjunto de dados de treinamento e que tem como saída os valores de acurácia e da função de custo da CNN treinada.

A Figura 4.3 mostra o texto de código que é incluído no programa de treinamento da rede para a definição dos dados a serem capturados com a CNNProv. Esses dados correspondem aos dados definidos no esquema de banco de dados da Figura 3.4. Como as transformações utilizadas no treinamento da AlexNet já estão modeladas na CNNProv, durante a fase de adaptação do código da AlexNet para definir os dados a serem capturados (proveniência prospectiva), é necessário apenas informar a *tag* para identificação do fluxo de dados e o valor *True* para a captura de hiperparâmetros. Assim, a CNNProv identificará que o especialista deseja capturar os dados dos hiperparâmetros definidos no modelo.

```
df = Dataflow("alexnet-df", True)
df.save()
```

Figura 4.3: Trecho adaptado do código da AlexNet para utilização da CNNProv para definição da proveniência prospectiva.

[1] https://www.mydatahack.com/building-alexnet-with-keras/

A Figura 4.4 mostra o conjunto de dados produzido pela transformação *Adaptation*, onde *t2* representa a tarefa de uma transformação que terá seus dados extraídos e armazenados e *oAdaptation* representa o conjunto de dados de saída da transformacão *Adaptation*.

```
def on_epoch_begin(self, epoch, logs=None):
    old_lr = float(K.get_value(self.model.optimizer.lr))
    new_lr = self.schedule(epoch, lr)
    if (old_lr != new_lr):
        self.adaptation_id += 1
        K.set_value(self.model.optimizer.lr, new_lr)
        t2_output = DataSet("oAdaptation", [Element([new_lr,
        datetime.now().strftime('%Y-%m-%d %H:%M:%S'), epoch,
        str(self.adaptation_id)])])
        t2.add_dataset(t2_output)
        t2.save()
```

Figura 4.4: Trecho adaptado do código da AlexNet para utilização da CNNProv para proveniência retrospectiva.

Como o código foi implementado utilizando o Keras, o especialista poderia ter utilizado o Keras-Prov, que faz a captura dos dados sem que o usuário precise instrumentar o código com as chamadas como o exemplo mostrado na Figura 4.4. Para tal, ao invés dos códigos apresentados nas Figuras 4.3 e 4.4, o especialista precisaria incluir no programa de treinamento da rede o código mostrado na Figura 4.5. A Figura 4.5 mostra que os hiperparâmetros *momentum* e *decay* estão como *False*, isso porque, neste caso, o especialista não utiliza esses hiperparâmetros. Caso optasse pela utilização de um otimizador que faz uso desses hiperparâmetros, seria apenas necessário mudar esses valores para *True*.

```
hyps = {"OPTIMIZER_NAME": True,
        "LEARNING_RATE": True,
        "DECAY": False,
        "MOMENTUM": False,
        "NUM_EPOCHS": True,
        "BATCH_SIZE": False,
        "NUM_LAYERS": True}

model.provenance(dataflow_tag="alexnet-df",
                 adaptation=True,
                 hyperparameters = hyps)
```

Figura 4.5: Código que deve ser adicionado à aplicação para captura de hiperparâmetros e dados de proveniência usando o Keras-Prov.

4.2.2 Análise de desempenho

Com o código da AlexNet adaptado para a utilização da CNNProv, a AlexNet foi treinada variando-se o número de épocas, taxa de aprendizado, *momentum*, *decay* e o otimizador. As execuções foram feitas com 20, 50 e 100 épocas, a taxa de aprendizado foi variada com os valores 0,0005, 0,001 e 0,002, o *decay* foi variado com os valores 0,0001 e 0,000001 e o *momentum* foi variado com os valores 0,5 e 0,9. Cada configuração de hiperparâmetros foi executada cinco vezes e a média do tempo de execução foi considerada. O tempo de execução foi considerado a partir do início da aplicação até sua finalização, a CNNProv já estava inicializada antes da execução da aplicação e o MonetDB já estava com o *schema* global definido. As Figuras 4.6, 4.7, 4.8, 4.9 e 4.10 apresentam as médias de treinamento, em minutos, para essas diferentes configurações, sem a utilização da CNNProv e com a utilização da CNNProv.

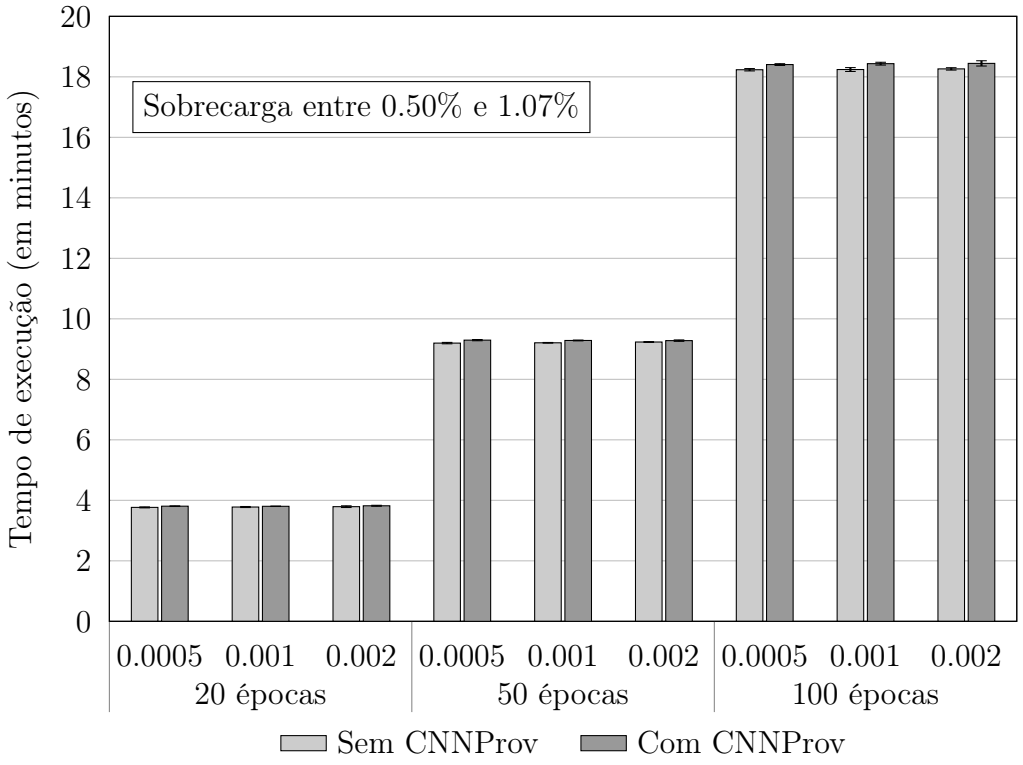

Figura 4.6: Tempo de treinamento (em minutos) da AlexNet com otimizador Adam.

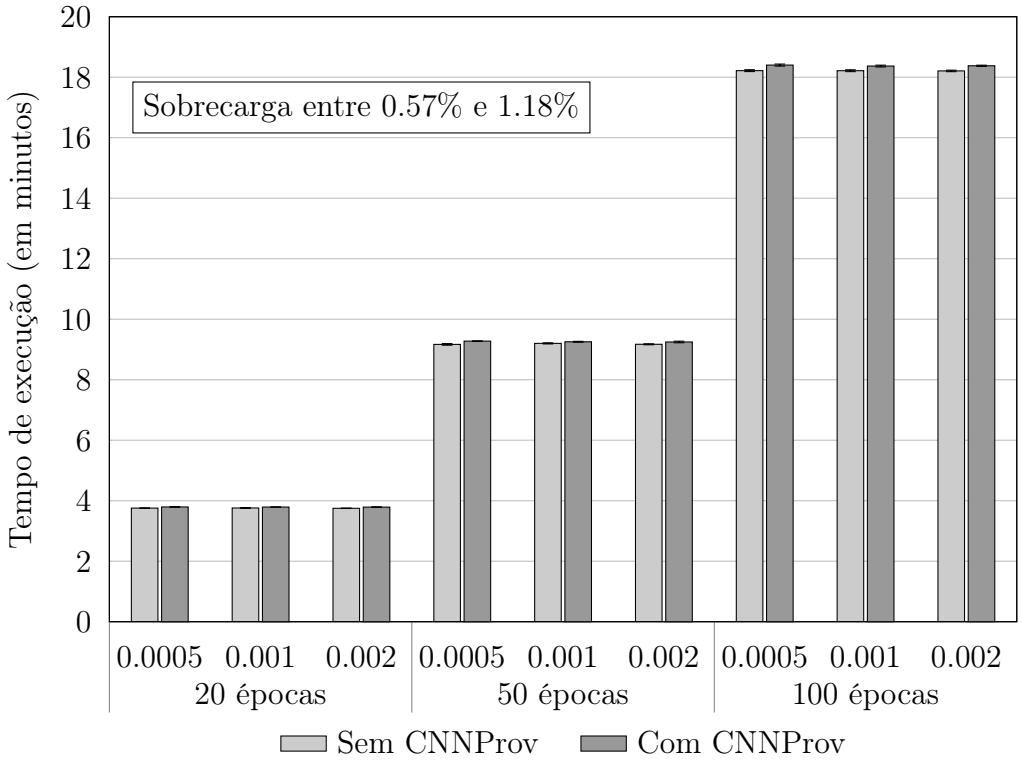

Figura 4.7: Tempo de treinamento (em minutos) da AlexNet com otimizador *SGD*, *momentum* 0,5, *decay* 0,0001.

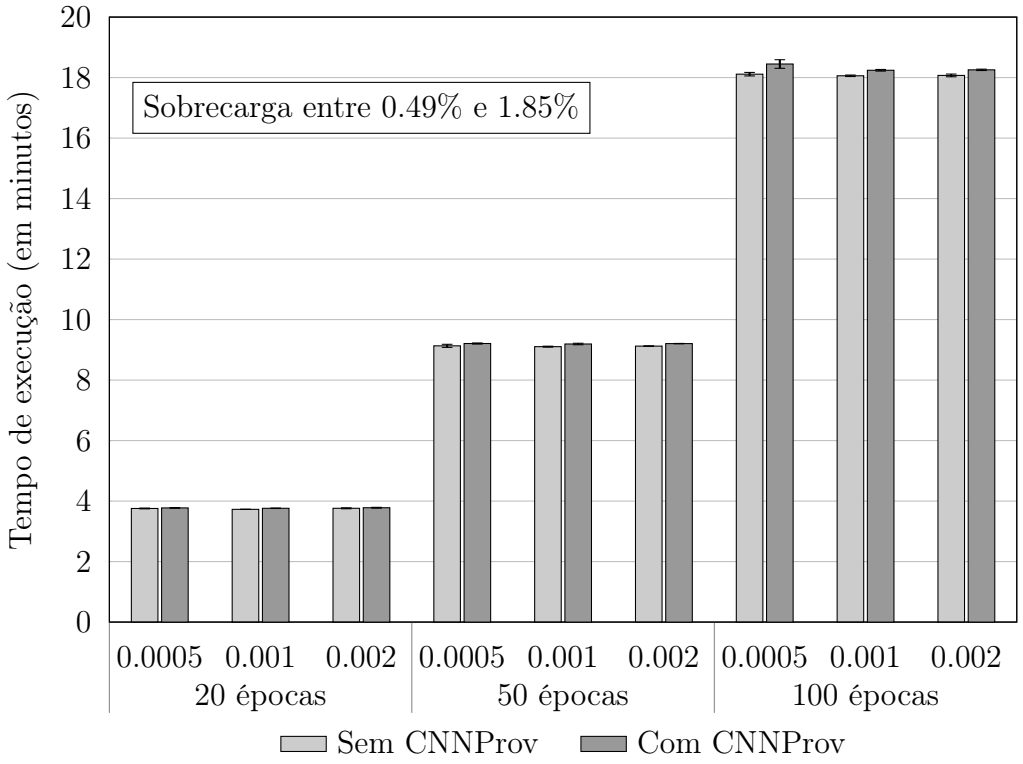

Figura 4.8: Tempo de treinamento (em minutos) da AlexNet com otimizador *SGD*, *momentum* 0,5, *decay* 0,000001.

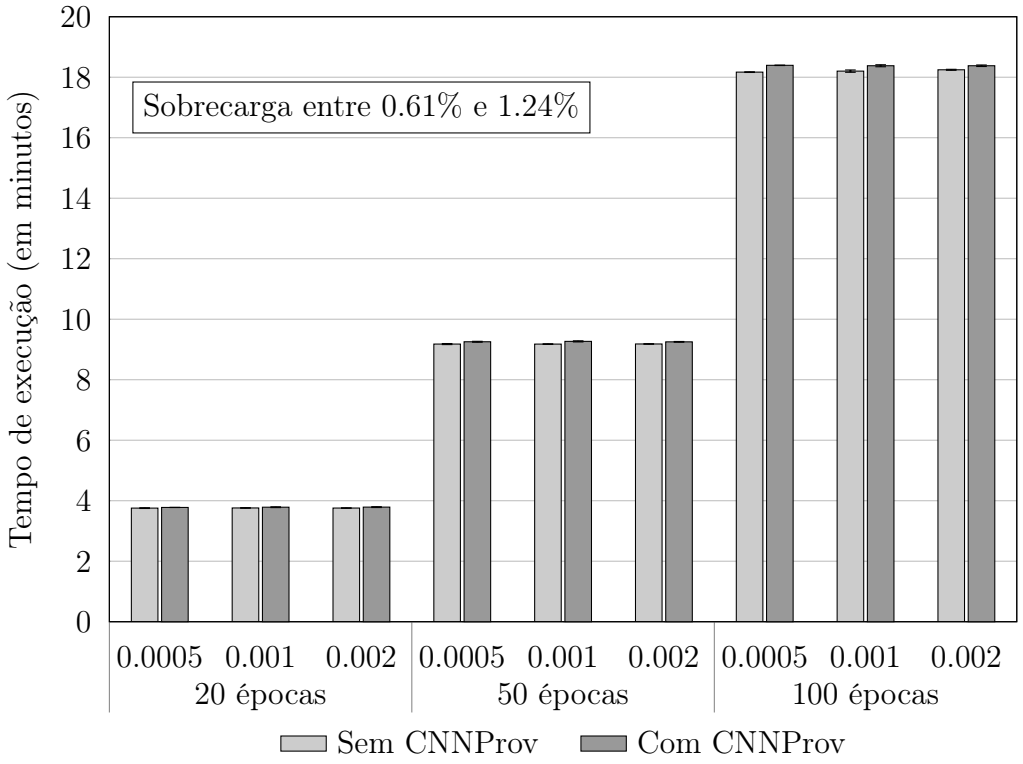

Figura 4.9: Tempo de treinamento (em minutos) da AlexNet com otimizador *SGD*, *momentum* 0,9, *decay* 0,0001.

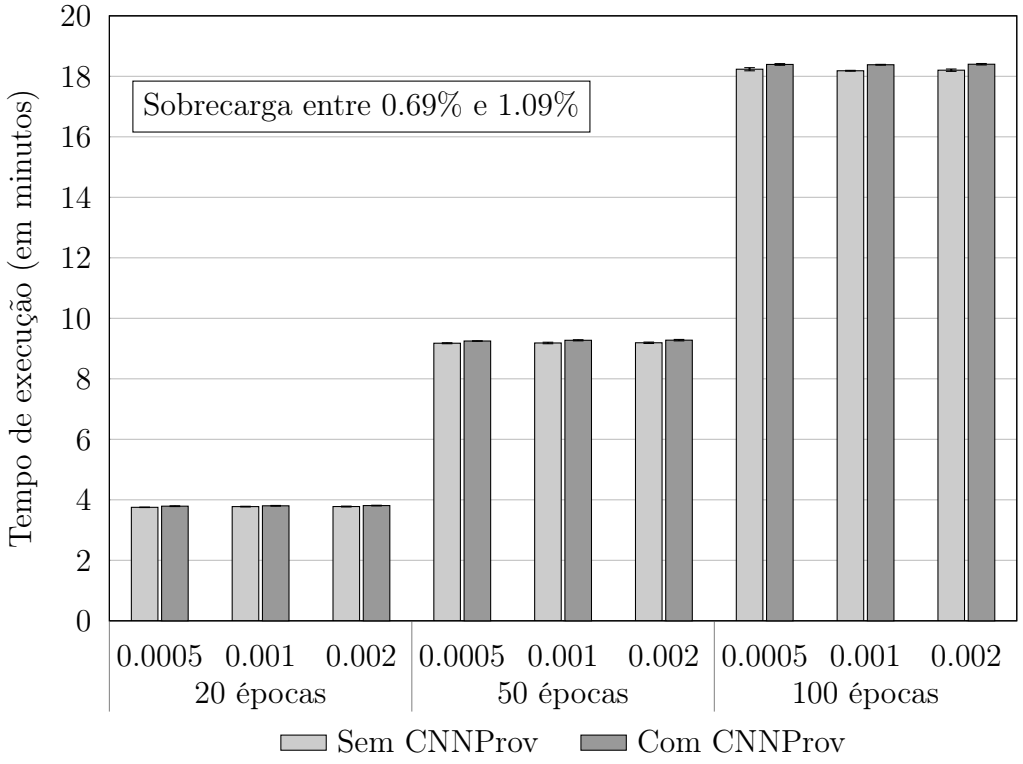

Figura 4.10: Tempo de treinamento (em minutos) da AlexNet com otimizador *SGD*, *momentum* 0,9, *decay* 0,000001.

O desvio padrão das cinco medições para as diferentes configurações de hiperparâmetros para a AlexNet ficou entre 0,004 e 0,142. Além disso, também observamos a sobrecarga introduzida pela CNNProv. O objetivo desta medição foi avaliar o impacto das chamadas à CNNProv durante o treinamento da AlexNet com dados reais. Observarmos que a sobrecarga da solução proposta corresponde a um aumento de menos de 2% no pior caso sobre o tempo total da aplicação. Os valores mínimos e máximos para as diferentes configurações estão ressaltados nos gráficos.

4.2.3 Análise de configurações de hiperparâmetros

Diferentemente da seção anterior, que apresentou uma análise quantitativa acerca do treinamento das redes neurais convolucionais modeladas para capturar hiperparâmetros e outros dados utilizando a CNNProv, esta subseção apresenta o potencial analítico da arquitetura CNNProv através do processamento de consultas. Basicamente, executamos consultas com o propósito de demonstrar a análise de configurações de hiperparâmetros, de forma que, ao analisar os dados de métricas sobre o treinamento das redes neurais junto à configuração utilizada para os hiperparâmetros, o especialista em redes neurais pode realizar os ajustes com mais segurança e confiança. Os dados dessa base de dados também podem auxiliar em futuras configurações.

Nesse sentido, foram submetidas as seguintes consultas para análise de dados de proveniência retrospectiva:

- Qual foi o tempo de execução de cada época da rede neural convolucional?
- Quanto tempo levou o treinamento da época com o menor valor de perda para o treinamento?
- Qual a taxa de aprendizado e época em que obtivemos o melhor valor de acurácia, e que valor foi esse?
- Quais foram as adaptações realizadas ao longo da execução da aplicação?

As consultas definidas anteriormente foram submetidas à base de dados do treinamento da AlexNet com a configuração de hiperparâmetros: 20 épocas e taxa de aprendizado 0,001 com o otimizador *Adam*. A Tabela 4.1 apresenta o resultado do processamento da consulta "Qual foi o tempo de execução de cada época da rede neural convolucional?". A relação descrita na Tabela 4.1 apresenta o número da época (*epoch*), o tempo decorrido (*elapsed time*) em segundos.

Tabela 4.1: Resultado da consulta “Qual foi o tempo de execução de cada época da AlexNet?”.

época	**tempo decorrido (em segundos)**
1	14.005
2	11.458
3	10.647
4	10.548
5	10.720
6	12.779
7	10.394
8	10.619
9	10.695
10	10.572

A Tabela 4.2 apresenta o resultado do processamento da consulta “Quanto tempo levou o treinamento da época com o menor valor de perda para o treinamento?” e os atributos mostrados são o número da época, o tempo decorrido e o valor de perda. Mesmo com esses exemplos simples, podemos observar que, com as adaptações no código da AlexNet, foi plenamente viável consultar quanto tempo levou o treinamento em cada época e qual o valor de perda associado. Com essas consultas, é possível verificar, por exemplo, se alguma época está levando mais tempo do que o normal, além disso, selecionado o atributo de valor de perda juntamente com a identificação da época, o especialista pode verificar se o aumento do número de épocas deixa de contribuir para uma melhor acurácia a partir de uma determinada época.

Tabela 4.2: Resultado da consulta “Quanto tempo levou o treinamento da época com o menor valor de perda para o treinamento da AlexNet?”.

época	**tempo decorrido (em segundos)**	**valor de perda**
20	10.889	0.1954

A Tabela 4.3 apresenta o resultado do processamento da consulta “Qual a taxa de aprendizado e época em que obtivemos o melhor valor de acurácia, e que valor foi esse?”, com os valores para a taxa de aprendizado de acordo com a atualização sofrida por ela durante o treinamento da CNN, bem como a época e a acurácia para a época em que essa atualização ocorreu. Essas consultas agrupam resultados que favorecem a comparação e uma análise da evolução do treinamento. Sem esse registro, a obtenção dessa evolução junto a dados de execução e demais configurações seria custoso e sujeito a erros.

Tabela 4.3: Resultado da consulta "Qual a taxa de aprendizado e época em que obtivemos o melhor valor de acurácia, e que valor foi esse?".

época	**taxa de aprendizado**	**acurácia**
20	0.00025	0.9402299

A Tabela 4.4 mostra a resposta da consulta "Quais foram as adaptações realizadas ao longo da execução da aplicação?", com os valores da taxa de aprendizado atualizados ao longo do treinamento bem como o valor da época em que essas atualizações ocorreram associados à uma identificação para a adaptação (id), que, assim como a consulta anterior, favorece a comparação e a análise da evolução do treinamento da CNN.

Tabela 4.4: Resultado da consulta "Quais foram as adaptações realizadas ao longo da execução da AlexNet?".

id	**taxa de aprendizado**	**época**
1	0.0005	10
2	0.00025	20

4.3 Estudo de caso: DenseED

Para analisar as incertezas das grandezas de interesse de um modelo de Migração Reversa no Tempo (MRT) devido às incertezas nos campos de velocidade, é necessário um grande número de simulações computacionais, tal que se possa analisar os momentos estatísticos através do método de Monte Carlo com uma determinada precisão. A fim de viabilizar computacionalmente a estratégia de incorporação de incertezas no imageamento sísmico, modelos substitutivos são utilizados para propagação dessas incertezas (XIU e KARNIADAKIS, 2002). No entanto, a maioria dos modelos substitutivos falham quando as entradas do modelo apresentam alta dimensionalidade (MA e ZABARAS, 2009), como é o caso do campo de velocidades associado às propriedades de meios heterogêneos. Este é o caso de uma onda acústica se propagando no fundo marinho através de camadas no solo, onde a dimensionalidade do campo de velocidade pode ser igual ao número de pontos do domínio espacial.

Nesse sentido, a utilização de redes neurais tem sido proposta pelo fato de o cálculo da equação da onda (MRT) ser muito custoso (MO *et al.*, 2019; TRIPATHY e BILIONIS, 2018). Quando se associa uma incerteza à equação, várias MRTs têm que ser calculadas e isso praticamente inviabiliza a aplicação de quantificação de incerteza. Dessa forma, a utilização de redes neurais é considerada, para não ser preciso resolver a equação de onda para cada incerteza.

As CNNs abordam o problema de alta dimensionalidade através de projeções não-lineares dos campos de entrada em espaços latentes (MO *et al.*, 2019). Nesse sentido, a arquitetura "encoder-decoder" proposta por ZHU e ZABARAS (2018) atua como modelo substitutivo para o problema básico de MRT, a propagação de ondas acústicas em meios heterogêneos, para viabilizar a abordagem de quantificação de incertezas no imageamento.

Arquitetura

A arquitetura DenseED usa uma rede bayesiana de codificador-decodificador totalmente convolucional para resolver uma tarefa de regressão de imagem para imagem (TRIPATHY e BILIONIS, 2018). Para essa arquitetura, ZHU e ZABARAS (2018) seguem redes totalmente convolucionais para segmentação de imagem sem usar camadas totalmente conectadas e propõem usar uma arquitetura codificador-decodificador.

Na codificação, as realizações do campo de entrada são alimentadas na primeira camada de convolução. Em seguida, os mapas de características extraídos são passados através de uma cascata alternativa de blocos densos (*dense block*) e camadas de codificação (*encoder*). O bloco denso após a última camada de codificação gera os mapas de características grosseiros de alto nível extraídos da entrada, como mostrado em roxo na extremidade direita da rede na Figura 4.11, que são posteriormente alimentados no caminho do decodificador (*decoder*). A rede de decodificação consiste na análise de blocos densos e camadas de decodificação, com a última camada de decodificação levando diretamente à previsão dos campos de saída.

A primeira camada da rede neural é constituída de uma convolução, sendo utilizada para extrair informações do campo de alta dimensionalidade. A rede neural apresenta um número total de parâmetros a serem estimados igual a 228.761.

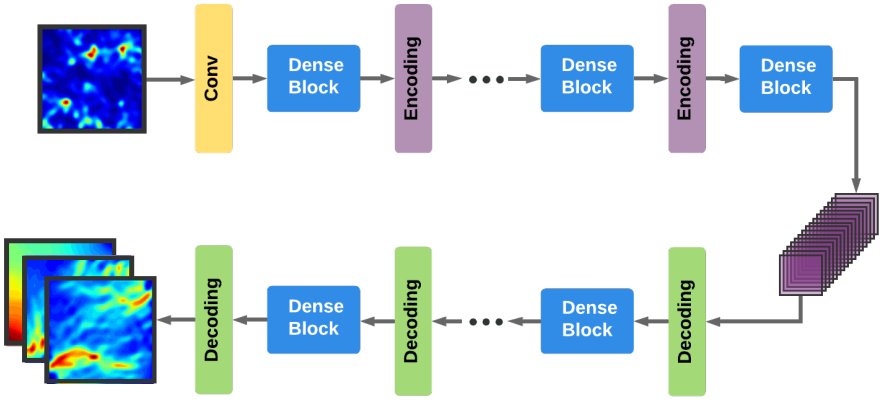

Figura 4.11: Arquitetura da DenseED (ZHU e ZABARAS, 2018).

Conjunto de dados

Os dados utilizados no treinamento desta arquitetura são campos de velocidade homogêneos e estocásticos, na magnitude do valor do campo, gerados por uma distribuição normal com valores de ordem de grandeza similares a velocidade de propagação da onda compressional em rochas sedimentares. Foi gerado um total de 100 mil amostras. Para cada conjunto de amostras, 80% dos dados foram separados para o treino e 20% foi separado para teste, no processo de treinamento da rede neural.

4.3.1 Instrumentação com a CNNProv

A implementação da DenseED foi feita com algumas modificações e, para esta dissertação, foi feita utilizando o Keras.

A instrumentação desta segunda rede neural foi realizada utilizando a mesma especificação da AlexNet, isto é, as mesmas transformações e conjunto de dados, sendo elas as transformações *Training*, *Adaptation* e *Testing*, em que a transformação *Training* consome o nome do otimizador utilizado no treinamento, os hiperparâmetros taxa de aprendizado, número de épocas e número de camadas da rede, e produz um conjunto de dados que define os dados gerados durante o treinamento, como o valor da época, da acurácia, da função de custo (*loss function*), o tempo decorrido e a data e hora do fim da execução da época. A transformação *Adaptation* recebe como dados de entrada o conjunto produzido pela transformação anterior (*Training*) e um conjunto de dados com informações para a adaptação ocorrida e o conjunto de dados de saída contém a nova taxa de aprendizado, o valor da época e a data e hora em que a adaptação ocorreu, além de uma identificação para a adaptação. A transformação *Testing* provê dados sobre a avaliação do modelo de acordo com o conjunto de dados de treinamento e tem como saída os valores de acurácia e da função de custo. Porém, nesse caso, foram acrescentadas transformações para captura de informações acerca das camadas *encoder*, *decoder* e *dense block*.

A Figura 4.12 mostra o texto de código que é incluído no programa de treinamento da rede para a definição dos dados a serem capturados com a CNNProv. Como no caso anterior, parte dos dados corresponde aos dados definidos no esquema de banco de dados da Figura 3.4. No entanto, como mencionado, nesse caso é necessária a captura de outros hiperparâmetros como *strides*, *padding* e *kernel_ size*. Dessa forma, para a definição dos dados a serem capturados (proveniência prospectiva) é necessário informar a *tag* para identificação do fluxo de dados e o valor *True* para a captura de hiperparâmetros já definidos no modelo de dados e as especificações das novas transformações.

```
df = Dataflow("dense-df", True)
tf1 = Transformation("Encoder")
...
tf1_output = Set("oEncoder", SetType.OUTPUT,
    [Attribute("PADDING", AttributeType.TEXT),
    Attribute("KERNEL_SIZE1", AttributeType.NUMERIC),
    Attribute("KERNEL_SIZE2", AttributeType.NUMERIC),
    Attribute("STRIDES", AttributeType.NUMERIC),
    Attribute("X1", AttributeType.NUMERIC),
    Attribute("X2", AttributeType.NUMERIC)])
tf1.set_sets([tf1_input, tf1_output])
df.add_transformation(tf1)

tf2 = Transformation("Decoder")
...
tf2_output = Set("oDecoder", SetType.OUTPUT,
    [Attribute("KERNEL_SIZE", AttributeType.NUMERIC),
    Attribute("PADDING", AttributeType.TEXT),
    Attribute("X1", AttributeType.NUMERIC),
    Attribute("X2", AttributeType.NUMERIC)])
tf2.set_sets([tf2_input, tf2_output])
df.add_transformation(tf2)

tf3 = Transformation("DenseBlock")
tf3_input = Set("iDenseBlock", SetType.INPUT,
    [Attribute("K", AttributeType.NUMERIC),
    Attribute("L", AttributeType.NUMERIC),
    Attribute("X1", AttributeType.NUMERIC),
    Attribute("X2", AttributeType.NUMERIC),
    Attribute("KERNEL", AttributeType.NUMERIC)])
tf3_output = Set("oDenseBlock", SetType.OUTPUT,
    [Attribute("X1", AttributeType.NUMERIC),
    Attribute("X2", AttributeType.NUMERIC)])
tf3.set_sets([tf3_input, tf3_output])
df.add_transformation(tf3)

df.save()
```

Figura 4.12: Trecho adaptado do código da DenseED para utilização da CNNProv para proveniência prospectiva.

A Figura 4.13 mostra o conjunto de dados produzido pela transformação *Training*, onde *t2* representa a tarefa de uma transformação que terá seus dados extraídos e armazenados e *oTraining* representa o conjunto de dados de saída da transformação *Training*.

```
def on_epoch_end(self, epoch, logs=None):
  elapsed_time = time.time()-self.start_time
  t2_output = DataSet("oTrainingModel", [Element([time.time(),
     elapsed_time, logs['loss'], logs['acc'], epoch])])
  t2.add_dataset(t2_output)
  t2.save()
```

Figura 4.13: Trecho adaptado do código da DenseED para utilização da CNNProv para proveniência retrospectiva.

4.3.2 Análise de desempenho

Com o código da DenseED adaptado para a utilização da CNNProv, foi realizado o treinamento variando-se o número de épocas, taxa de aprendizado, *momentum*, *decay* e o otimizador. As execuções foram feitas com 20, 50 e 100 épocas, a taxa de aprendizado foi variada com os valores 0,0005, 0,001 e 0,002, *decay* foi variado com os valores 0,0001 e 0,000001 e *momentum* foi variado com os valores 0,5 e 0,9. Cada configuração de hiperparâmetros foi executada cinco vezes e a média do tempo de execução foi considerada. O tempo de execução foi considerado a partir do início da aplicação até sua finalização, a CNNProv já estava inicializada antes da execução da aplicação e o MonetDB já estava com o *schema* global definido. As Figuras 4.14, 4.15, 4.16, 4.17 e 4.18 apresentam as médias de treinamento, em minutos para essas diferentes configurações sem a utilização da CNNProv e com a utilização da CNNProv.

O desvio padrão das cinco medições para as diferentes configurações de hiperparâmetros ficou entre 0,009 e 0,265. Assim como para a AlexNet, foi feita a medição da sobrecarga introduzida pela utilização CNNProv. Observamos que a sobrecarga da solução proposta corresponde a um aumento de menos de 4% no pior caso sobre o tempo total da aplicação. Além disso, a sobrecarga introduzida pela utilização da CNNProv foi ressaltada nos gráficos de acordo com as configurações. Nota-se que a sobrecarga apresentada pela DenseED é maior que a sobrecarga apresentada pela AlexNet. Isso se deve ao fato de mais chamadas à CNNProv serem feitas nessa rede.

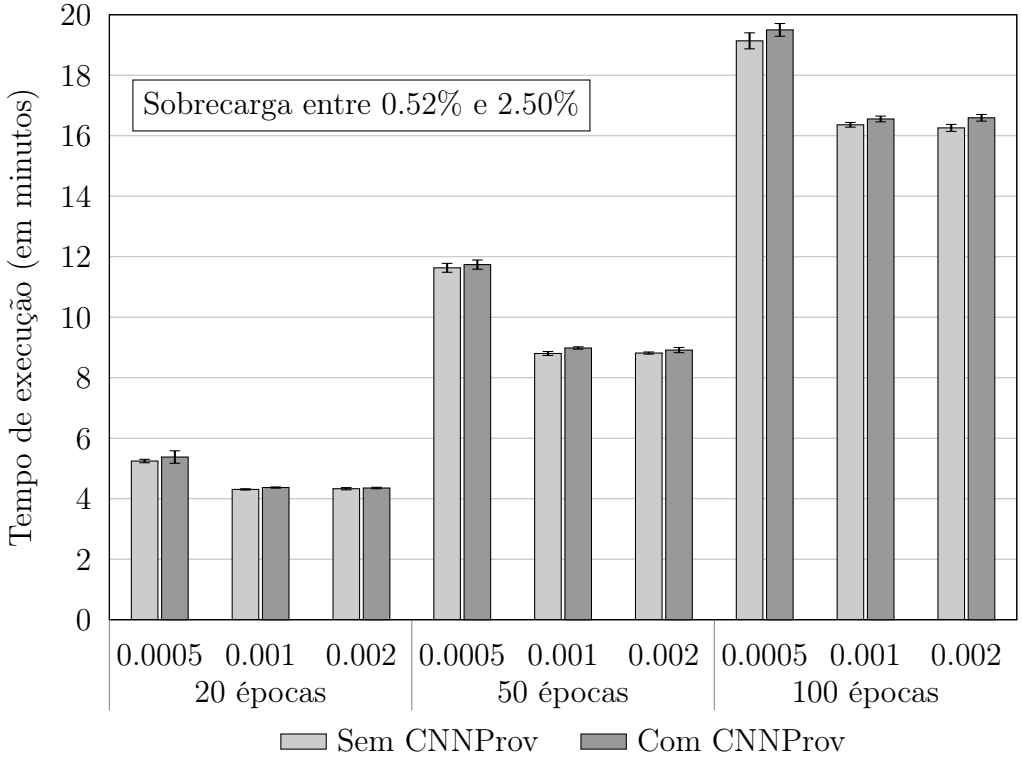

Figura 4.14: Tempo de treinamento (em minutos) da DenseED com otimizador Adam.

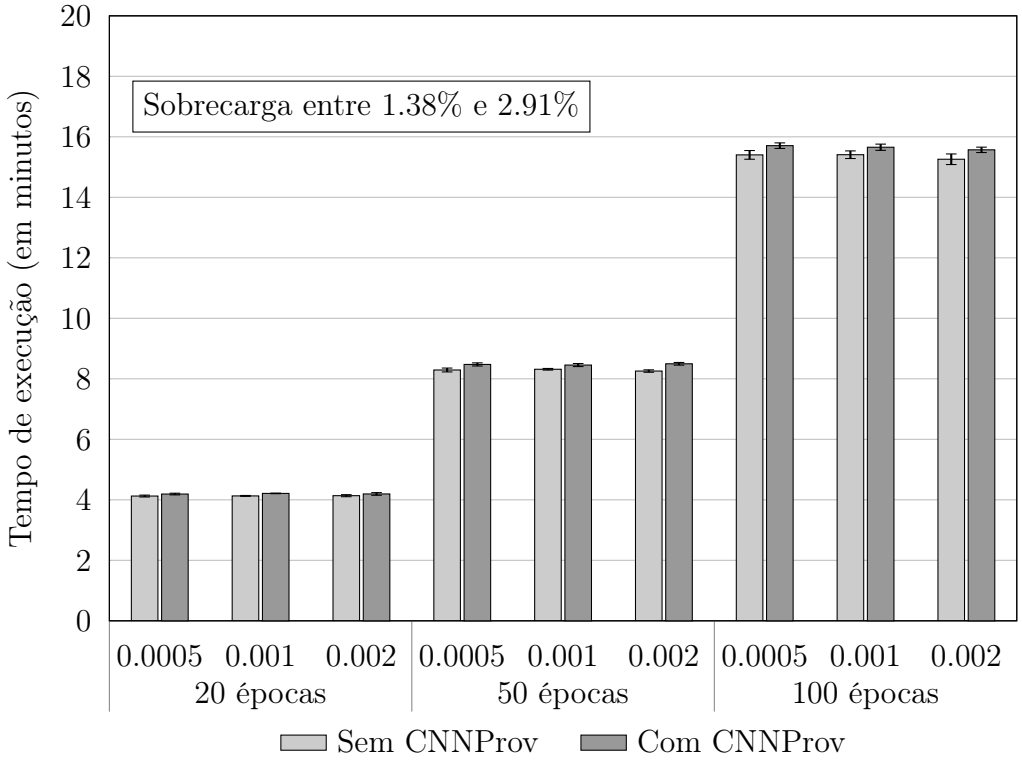

Figura 4.15: Tempo de treinamento (em minutos) da DenseED com otimizador *SGD*, *momentum* 0,5, *decay* 0,0001.

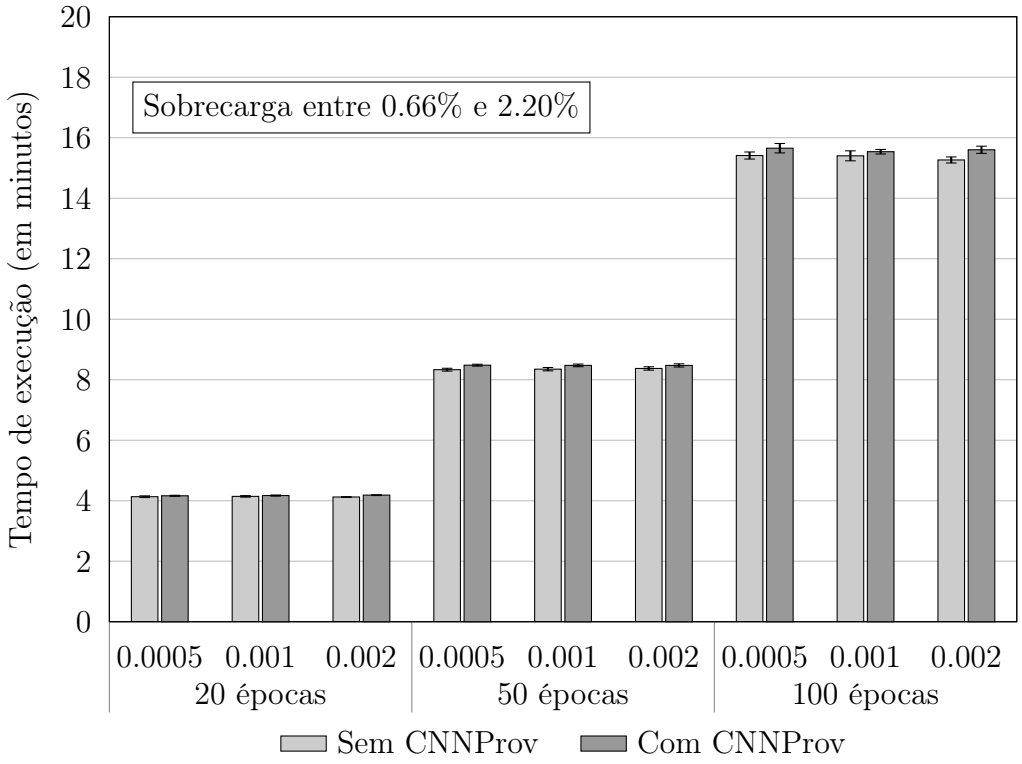

Figura 4.16: Tempo de treinamento (em minutos) da DenseED com otimizador *SGD*, *momentum* 0,5, *decay* 0,000001.

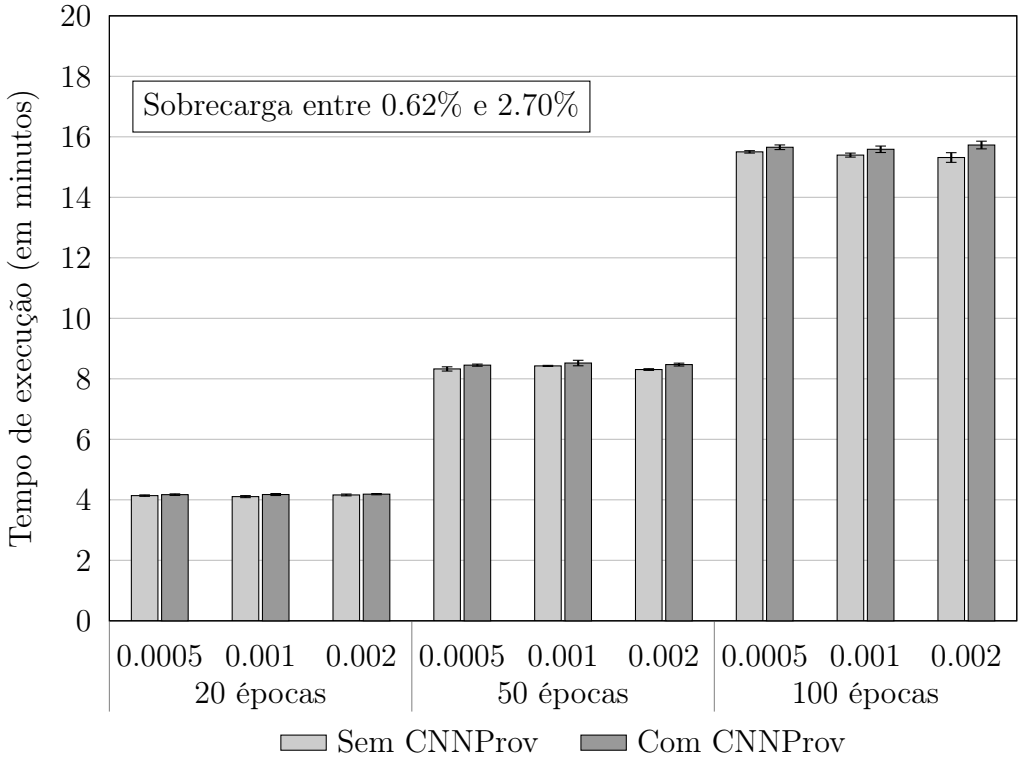

Figura 4.17: Tempo de treinamento (em minutos) da DenseED com otimizador *SGD*, *momentum* 0,9, *decay* 0,0001.

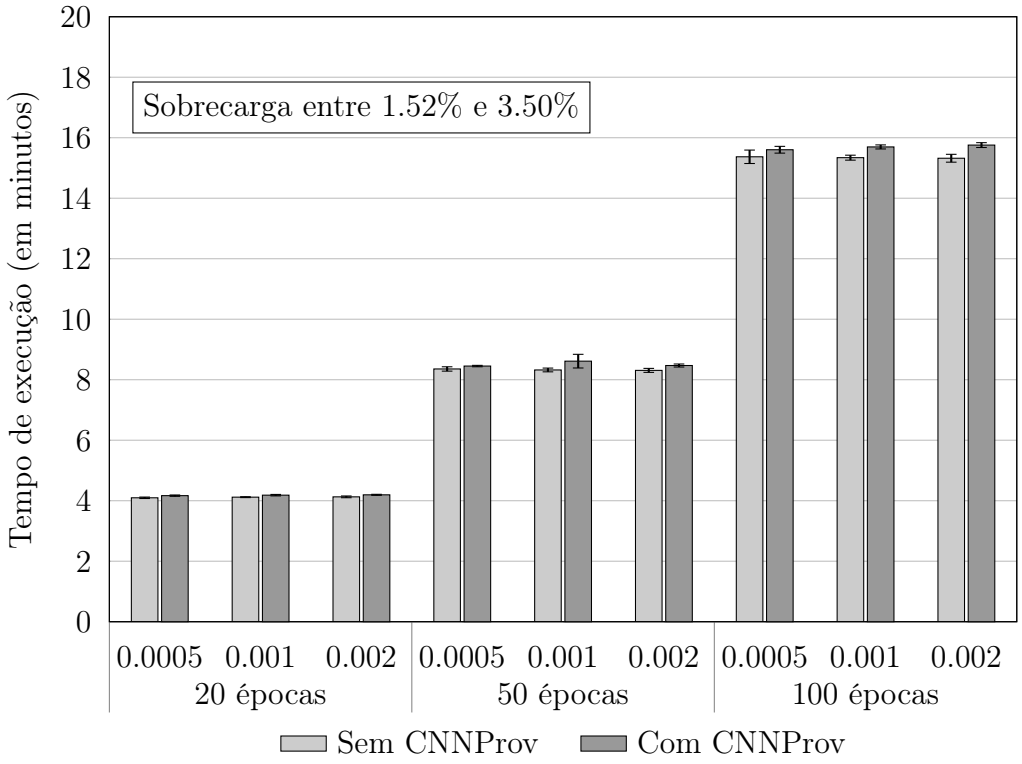

Figura 4.18: Tempo de treinamento (em minutos) da DenseED com otimizador *SGD*, *momentum* 0,9, *decay* 0,000001.

4.3.3 Análise de configurações de hiperparâmetros

Assim como na análise de configurações de hiperparâmetros da AlexNet, nesta seção apresentamos uma análise sobre os dados de treinamento da DenseED associados aos hiperparâmetros. Nesse sentido, foram submetidas as seguintes consultas:

- Qual foi o tempo de execução de cada época para as 5 primeiras épocas da rede neural convolucional?
- Quanto tempo levou o treinamento da época com o menor valor de perda para o treinamento?

Para o processamento dessas consultas, utilizamos a base de dados do treinamento da DenseED com o otimizador *Adam*, 20 épocas e taxa de aprendizado 0,001. A Tabela 4.5 apresenta o resultado para a consulta "Qual foi o tempo de execução de cada época para as 5 primeiras épocas da rede neural convolucional?" e a Tabela 4.6 apresenta a resposta da consulta "Quanto tempo levou o treinamento da época com o menor valor de perda para o treinamento?".

Além disso, consideramos, aqui, um exemplo sobre como o especialista em redes neurais age para chegar ao modelo final através de tentativa e erro, mas com o auxílio da CNNProv para a análise de configurações de hiperparâmetros. Nesse sentido, dois hiperparâmetros considerados importantes para serem analisados são

Tabela 4.5: Resultado da consulta "Qual foi o tempo de execução de cada época para as 5 primeiras épocas da DenseED?".

época	**tempo decorrido (em segundos)**
1	48.398
2	14.896
3	15.332
4	15.413
5	14.885

Tabela 4.6: Resultado da consulta "Quanto tempo levou o treinamento da época com o menor valor de perda para o treinamento da DenseED?".

época	**tempo decorrido (em segundos)**	**valor de perda**
20	15.103	0.004

capturados na transformação *DenseBlock*: K e L, que são a taxa de crescimento e o número de camadas no bloco denso, respectivamente. A taxa de aprendizado foi 0,001 com o otimizador *Adam*.

Esse processo é iniciado com o treinamento da DenseED com K=32 e L=4. Os hiperparâmetros definidos no início do treinamento são inseridos na base de dados e a medida que o treinamento da DenseED acontece, os dados sobre as métricas são adicionados, de forma que o especialista pode acompanhar o treinamento. Para a análise dos dados, é feita uma consulta à base para que se saiba "Qual o valor de perda para a DenseED com K=32 e L=4 no momento atual?". A resposta desta consulta está na Tabela 4.7. Para demonstração, colocamos os valores para as dez últimas épocas até o momento atual.

Tabela 4.7: Resultado da consulta "Qual o valor de perda para a DenseED com K=32 e L=4 no momento atual?".

época	**valor de perda**
1	0.431
2	0.079
3	0.053
4	0.047
5	0.045
6	0.045
7	0.044
8	0.044
9	0.044
10	0.043

O especialista continua realizando consultas à base a medida que o treinamento progride. Ao se chegar em 200 épocas, o especialista analisa os valores de métricas, como valor de perda, e decide finalizar a execução, porque esses valores não estão

satisfazendo seus critérios. O resultado dessa nova consulta é mostrado na Tabela 4.8.

Tabela 4.8: Resultado da consulta "Qual o valor de perda para a DenseED com K=32 e L=4 no momento atual (época 200)?".

época	**valor de perda**
200	0.0184

Supõe-se que o especialista opta por alterar o valor de L para 8. Esse valor de L atualizado é armazenado na base de dados. Novamente o treinamento da DenseED é iniciado, os primeiros dados acerca das métricas de treinamento são apresentadas na Tabela 4.9. Para demonstração, colocamos os valores para as duas últimas épocas até o momento atual.

Tabela 4.9: Resultado da consulta "Qual o valor de perda para a DenseED com K=32 e L=8 no momento atual?".

época	**valor de perda**
1	2.168
2	0.268
3	0.136
4	0.115
5	0.108
6	0.106
7	0.105
8	0.105
9	0.105
10	0.104

Na centésima época essa consulta é repetida e o especialista identifica que os resultados dessa configuração estão melhores do que os resultados da configuração anterior e decide continuar o treinamento. Durante esse processo, ele pode também realizar consultas levando em consideração o tempo de cada época. Por manter os dados na base de dados, o usuário pode consultar também dados sobre outros fluxos de dados, que podem ajudá-lo na tomada de decisão em relação à configuração de hiperparâmetros.

Enquanto o treinamento para uma configuração é realizado, é possível fazer consultas com junções de configurações de hiperparâmetros com as métricas obtidas durante o treinamento de uma determinada configuração. Por exemplo, "Qual o menor valor de perda e o número de camadas para a configuração em que taxa de aprendizado=0,001, K=24 e L=4?". A Tabela 4.10 apresenta o resultado dessa consulta, que agregado ao conhecimento do especialista pode ajudar na determinação de uma nova configuração de hiperparâmetros.

Tabela 4.10: Resultado da consulta “Qual o menor valor de perda e o número de camadas para a configuração em que taxa de aprendizado=0,001, K=24 e L=4?”.

época	**valor de perda**	**número de camadas**
200	0.012	64

Capítulo 5

Conclusão

A abordagem desenvolvida nesta dissertação tem como objetivo apoiar a análise de configurações de hiperparâmetros no treinamento de CNNs através do registro de informações relevantes. Apresentamos a CNNProv, que adota um padrão para representação de dados, não é específica a linguagens de programação e bibliotecas, e não exige que o especialista utilize um determinado ambiente de execução. Essa abordagem permite armazenar dados para consultas durante e após a execução, podendo auxiliar na tomada de decisão e formar um histórico para enriquecer as análises. Dessa forma, a CNNProv se destaca por capturar e registrar os dados durante a execução, possibilitando uma análise *online*, favorecendo a adaptação dos hiperparâmetros, e com funcionamento em ambientes de computação de alto desempenho.

A CNNProv foi implementada como uma biblioteca, que pode ser incorporada aos *scripts* através de requisições HTTP para o serviço RESTful, não obrigando que o treinamento seja executado em um ambiente específico. Ademais, para lidar com a diversidade de representação dos dados, a CNNProv adota o padrão W3C PROV com a DfAnalyzer. No sentido de facilitar a inclusão de chamadas à CNNProv, foi, também, implementado o subcomponente Keras-Prov, em que a captura dos dados contemplados pelo modelo é feita de maneira praticamente automática.

Foram feitos experimentos utilizando uma CNN clássica, a AlexNet, e uma aplicação científica não convencional, a DenseED. Essa rede neural (DenseED) é um exemplo de uso de CNN por pesquisadores que não são da área de ciência da computação, uma situação que tende a aumentar e necessitar ainda mais de auxílio na análise e configuração de hiperparâmetros. Os experimentos realizados evidenciam a adequação do uso de proveniência nas atividades de análise e monitoramento, contribuindo para um padrão de consultas que pode acessar, de forma integrada, as configurações de hiperparâmetros associadas aos dados de proveniência.

Com isso, concluímos que, de fato, a proveniência em uma solução como a CNNProv possibilitou a captura de dados de proveniência contribuindo com a análise de

configurações de hiperparâmetros em redes de aprendizado profundo, sendo possível associar dados de proveniência com dados de treinamento e assim oferecer um panorama do comportamento dos hiperparâmetros no treinamento de CNNs.

Assim, as principais contribuições da CNNProv para o estado da arte das ferramentas de análise de configurações de hiperparâmetros de redes neurais convolucionais são contribuições que contornam os problemas de diversidade na representação dos dados de apoio ao treinamento e necessidade de execução sob um portal ou ferramenta. Além disso, observou-se, nos experimentos realizados nesta dissertação, que a sobrecarga da solução proposta pode ser considerada desprezível, principalmente em treinamentos mais demorados, considerando que o especialista terá o benefício das consultas e visualizações aos dados de proveniência capturados.

Ainda há muitos desafios a serem explorados no que diz respeito ao ciclo de vida de aprendizado profundo. Esta dissertação tem foco na análise de configurações de hiperparâmetros de maneira a apoiar análise *online*. A CNNProv conta com o conhecimento do especialista para identificar as relações entre os dados (transformações, conjuntos de dados e dependências), além de dados de interesse. Apesar disso ser interessante por conta do especialista escolher apenas os dados de seu interesse, a necessidade de adaptação do código, ainda que isso tenha sido reduzido nesta abordagem, é uma tarefa que demanda tempo. Nesse sentido, soluções para mitigar esse problema podem ser exploradas.

Além disso, capturar dados acerca dos recursos computacionais, como CPU e memória durante a execução da aplicação, podem enriquecer ainda mais as análises feitas pelo especialista, possibilitando, por exemplo, que pontos de maior demanda de recursos sejam identificados.

Referências Bibliográficas

ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., ZHENG, X., 2016, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems", *arXiv preprint arXiv:1603.04467*.

ALOM, M. Z., TAHA, T. M., YAKOPCIC, C., WESTBERG, S., SIDIKE, P., NASRIN, M. S., VAN ESESN, B. C., AWWAL, A. A. S., ASARI, V. K., 2018, "The history began from alexnet: A comprehensive survey on deep learning approaches", *arXiv preprint arXiv:1803.01164*.

ATKINSON, M., GESING, S., MONTAGNAT, J., TAYLOR, I., 2017. "Scientific workflows: Past, present and future". .

AVSEC, Z., KREUZHUBER, R., ISRAELI, J., XU, N., CHENG, J., SHRIKUMAR, A., BANERJEE, A., KIM, D., URBAN, L., KUNDAJE, A., STEGLE, O., GAGNEUR, J., 2018, "Kipoi: accelerating the community exchange and reuse of predictive models for genomics", p. 375345.

BALDI, P., SADOWSKI, P. J., 2013, "Understanding dropout". In: *Advances in neural information processing systems*, pp. 2814–2822.

BENGIO, Y., 2012, "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*, Springer, pp. 437–478.

BONCZ, P. A., KERSTEN, M. L., MANEGOLD, S., 2008, "Breaking the memory wall in MonetDB", *Communications of the ACM*, v. 51, n. 12, pp. 77–85.

BUNEMAN, P., KHANNA, S., WANG-CHIEW, T., 2001, "Why and where: A characterization of data provenance". In: *International conference on database theory*, pp. 316–330. Springer.

CAMPOS, V. S., 2018, *DfA-lib-Python: Uma biblioteca para a extração de dados científicos usando a DfAnalyzer*. Projeto de Graduação, Universidade Federal do Rio de Janeiro.

COSTA, F., SILVA, V., DE OLIVEIRA, D., OCAÑA, K., OGASAWARA, E., DIAS, J., MATTOSO, M., 2013, "Capturing and querying workflow runtime provenance with PROV: a practical approach". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pp. 282–289.

DE A.R. GONÇALVES, J. C., DE OLIVEIRA, D., OCAÑA, K. A., OGASAWARA, E., MATTOSO, M., 2012, "Using domain-specific data to enhance scientific workflow steering queries". In: *International Provenance and Annotation Workshop*, pp. 152–167. Springer.

DE OLIVEIRA, D., OGASAWARA, E., BAIÃO, F., MATTOSO, M., 2010, "Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows". In: *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 378–385. IEEE.

DE OLIVEIRA, D., OCAÑA, K. A., BAIÃO, F., MATTOSO, M., 2012, "A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds", *Journal of grid Computing*, v. 10, n. 3, pp. 521–552.

DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., FEI-FEI, L., 2009, "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee.

DENG, L., YU, D., 2014, "Deep learning: methods and applications", *Foundations and Trends® in Signal Processing*, v. 7, n. 3–4, pp. 197–387.

DOMHAN, T., SPRINGENBERG, J. T., HUTTER, F., 2015, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

FREIRE, J., KOOP, D., SANTOS, E., SILVA, C. T., 2008, "Provenance for computational tasks: A survey", *Computing in Science & Engineering*, v. 10, n. 3, pp. 11–21.

FUJITA, O., 1998, "Statistical estimation of the number of hidden units for feed-forward neural networks", *Neural networks*, v. 11, n. 5, pp. 851–859.

GHARIBI, G., WALUNJ, V., ALANAZI, R., RELLA, S., LEE, Y., 2019a, "Automated Management of Deep Learning Experiments". In: *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, p. 8. ACM, a.

GHARIBI, G., WALUNJ, V., RELLA, S., LEE, Y., 2019b, "ModelKB: towards automated management of the modeling lifecycle in deep learning". In: *Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pp. 28–34. IEEE Press, b.

HANNUN, A., CASE, C., CASPER, J., CATANZARO, B., DIAMOS, G., ELSEN, E., PRENGER, R., SATHEESH, S., SENGUPTA, S., COATES, A., NG, A., 2014, "Deep speech: Scaling up end-to-end speech recognition", *arXiv preprint arXiv:1412.5567*.

HE, K., ZHANG, X., REN, S., SUN, J., 2016, "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

HECHT-NIELSEN, R., 1989, "Neurocomputer applications". In: *Neural computers*, Springer, pp. 445–453.

HERSCHEL, M., DIESTELKÄMPER, R., LAHMAR, H. B., 2017, "A survey on provenance: What for? What form? What from?" *The VLDB Journal*, v. 26, n. 6, pp. 881–906.

HOOS, H., LEYTON-BROWN, K., 2014, "An efficient approach for assessing hyperparameter importance". In: *International conference on machine learning*, pp. 754–762.

KARPATHY, A. "Convolutional Neural Networks for Visual Recognition". Disponível em: <`http://cs231n.github.io/neural-networks-1/`>.

KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., 2012, "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

LECUN, Y., BOTTOU, L., BENGIO, Y., HAFFNER, P., 1998, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, v. 86, n. 11, pp. 2278–2324.

LECUN, Y., BENGIO, Y., HINTON, G., 2015, "Deep learning", *nature*, v. 521, n. 7553, pp. 436.

MA, X., ZABARAS, N., 2009, "An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations", *Journal of Computational Physics*, v. 228, n. 8, pp. 3084–3113.

MATTOSO, M., WERNER, C., TRAVASSOS, G. H., BRAGANHOLO, V., OGASAWARA, E., OLIVEIRA, D., CRUZ, S., MARTINHO, W., MURTA, L., 2010, "Towards supporting the life cycle of large scale scientific experiments", *International Journal of Business Process Integration and Management*, v. 5, n. 1, pp. 79.

MATTOSO, M., DIAS, J., OCAÑA, K. A., OGASAWARA, E., COSTA, F., HORTA, F., SILVA, V., DE OLIVEIRA, D., 2015, "Dynamic steering of HPC scientific workflows: A survey", *Future Generation Computer Systems*, v. 46, pp. 100–113.

MIAO, H., 2018, *Provenance Management for Collaborative Data Science Workflows*. Tese de Doutorado.

MIAO, H., DESHPANDE, A., 2018, "ProvDB: Provenance-enabled Lifecycle Management of Collaborative Data Analysis Workflows." *IEEE Data Eng. Bull.*, v. 41, n. 4, pp. 26–38.

MIAO, H., LI, A., DAVIS, L. S., DESHPANDE, A., 2015, "ModelHub: Lifecycle Management for Deep Learning", *Univ. of Maryland*.

MIAO, H., LI, A., DAVIS, L. S., DESHPANDE, A., 2017, "Towards unified data and lifecycle management for deep learning". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 571–582. IEEE.

MITCHELL, T. M., 1997, *Machine Learning*. 1 ed. USA, McGraw-Hill, Inc. ISBN: 0070428077.

MO, S., ZHU, Y., ZABARAS, N., SHI, X., WU, J., 2019, "Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media", *Water Resources Research*, v. 55, n. 1, pp. 703–728.

MOREAU, L., GROTH, P., 2013, "Provenance: an introduction to PROV", *Synthesis Lectures on the Semantic Web: Theory and Technology*, v. 3, n. 4, pp. 1–129.

MURTA, L., BRAGANHOLO, V., CHIRIGATI, F., KOOP, D., FREIRE, J., 2014, "noWorkflow: capturing and analyzing provenance of scripts". In: *International Provenance and Annotation Workshop*, pp. 71–83. Springer.

MUSIOL, M., 2016, "Speeding up Deep Learning", .

NIELSEN, M. A., 2015, *Neural networks and deep learning*, v. 2018. Determination press San Francisco, CA, USA:.

NILSBACK, M.-E., ZISSERMAN, A., 2006, "A visual vocabulary for flower classification". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, v. 2, pp. 1447–1454. IEEE.

OGASAWARA, E., DIAS, J., OLIVEIRA, D., PORTO, F., VALDURIEZ, P., MATTOSO, M., 2011, "An algebraic approach for data-centric scientific workflows", *Proc. of VLDB Endowment*, v. 4, n. 12, pp. 1328–1339.

ORR, G. B., MÜLLER, K.-R., 2003, *Neural networks: tricks of the trade*. Springer.

PATTERSON, J., GIBSON, A., 2017, *Deep learning: A practitioner's approach*. "O'Reilly Media, Inc.".

PENG, L., GULSHAN, V., 2016, "Deep learning for detection of diabetic eye disease", *Google Research Blog*.

PINA, D., CAMPOS, V., SILVA, V., OCAÑA, K., DE OLIVEIRA, D., MATTOSO, M., 2017, "BioSciCumulus: um portal para análise de dados de proveniência em workflows de biologia computacional", *XXXVII Congresso da Sociedade Brasileira de Computação*, (Agosto). Disponível em: <`http://csbc2017.mackenzie.br/public/files/11-bresci/8.pdf`>.

PINA, D. B., NEVES, L., PAES, A., DE OLIVEIRA, D., MATTOSO, M., 2019, "Análise de Hiperparâmetros em Aplicações de Aprendizado Profundo por meio de Dados de Proveniência". In: *Anais do XXXIV Simpósio Brasileiro de Banco de Dados*, pp. 223–228. SBC.

PINA, D. B., 2018, *Uma interface para a análise de fluxo de dados em simulações computacionais intensivas em dados*. Projeto de Graduação, Universidade Federal do Rio de Janeiro.

SCHELTER, S., BÖSE, J.-H., KIRSCHNICK, J., KLEIN, T., SEUFERT, S., 2017, "Automatically tracking metadata and provenance of machine learning experiments". In: *Machine Learning Systems workshop at NIPS*.

SILVA, V., 2018, *Análise de dados científicos sobre múltiplas fontes de dados ao longo da execução de simulações computacionais*. Tese de Doutorado.

SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., MATTOSO, M., 2016, "Analyzing related raw data files through dataflows", *Concurrency and Computation: Practice and Experience*, v. 28, n. 8, pp. 2528–2545.

SILVA, V., LEITE, J., CAMATA, J. J., DE OLIVEIRA, D., COUTINHO, A. L., VALDURIEZ, P., MATTOSO, M., 2017, "Raw data queries during data-intensive parallel workflow execution", *Future Generation Computer Systems*, v. 75, pp. 402–422.

SILVA, V., DE OLIVEIRA, D., MATTOSO, M., VALDURIEZ, P., 2018a, "DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance", *PVLDB*, v. 11, n. 12, pp. 2082–2085. doi: 10.14778/3229863.3236265. Disponível em: <`http://www.vldb.org/pvldb/vol11/p2082-sousa.pdf`>.

SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., MATTOSO, M., 2018b, "DfAnalyzer: runtime dataflow analysis of scientific applications using provenance", *Proceedings of the VLDB Endowment*, v. 11, n. 12, pp. 2082–2085.

SILVA, V., SOUZA, R., CAMATA, J., DE OLIVEIRA, D., VALDURIEZ, P., COUTINHO, A. L., MATTOSO, M., 2018c, "Capturing provenance for runtime data analysis in computational science and engineering applications". In: *International Provenance and Annotation Workshop*, pp. 183–187. Springer, c.

SMITH, L. N., 2018, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay", *arXiv preprint arXiv:1803.09820*.

SOUZA, R., MATTOSO, M., 2018, "Provenance of dynamic adaptations in user-steered dataflows". In: *International Provenance and Annotation Workshop*, pp. 16–29. Springer.

SOUZA, R., NEVES, L., AZEREDO, L., LUIZ, R., TADY, E., CAVALIN, P. R., MATTOSO, M., 2018, "Towards a Human-in-the-Loop Library for Tracking Hyperparameter Tuning in Deep Learning Development." In: *LADaS@ VLDB*, pp. 84–87.

SOUZA, R., AZEVEDO, L., LOURENÇO, V., SOARES, E., THIAGO, R., BRANDÃO, R., CIVITARESE, D., BRAZIL, E. V., MORENO, M., VALDURIEZ, P., MATTOSO, M., CERQUEIRA, R., NETTO, M. A. S., 2019,

"Provenance Data in the Machine Learning Lifecycle in Computational Science and Engineering". In: *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pp. 1–10. IEEE.

SZE, V., CHEN, Y.-H., YANG, T.-J., EMER, J. S., 2017, "Efficient processing of deep neural networks: A tutorial and survey", *Proceedings of the IEEE*, v. 105, n. 12, pp. 2295–2329.

TAN, W., ZHAO, C., WU, H., 2016, "Intelligent alerting for fruit-melon lesion image based on momentum deep learning", *Multimedia Tools and Applications*, v. 75, n. 24, pp. 16741–16761.

TRIPATHY, R. K., BILIONIS, I., 2018, "Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification", *Journal of computational physics*, v. 375, pp. 565–588.

TSAY, J., MUMMERT, T., BOBROFF, N., BRAZ, A., WESTERINK, P., HIRZEL, M., 2018. "Runway: machine learning model experiment management tool". .

VANSCHOREN, J., VAN RIJN, J. N., BISCHL, B., TORGO, L., 2014, "OpenML: networked science in machine learning", *ACM SIGKDD Explorations Newsletter*, v. 15, n. 2, pp. 49–60.

VARGAS, A. C. G., PAES, A., VASCONCELOS, C. N., 2016, "Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres". In: *Proceedings of the XXIX Conference on Graphics, Patterns and Images*, v. 1.

VARTAK, M., SUBRAMANYAM, H., LEE, W.-E., VISWANATHAN, S., HUSNOO, S., MADDEN, S., ZAHARIA, M., 2016, "Model DB: a system for machine learning model management". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, p. 14. ACM.

WU, B., 2019, *Influence of Different Color Spaces on the Classification Performance of Convolutional Neural Network*. Tese de Mestrado.

XIU, D., KARNIADAKIS, G. E., 2002, "The Wiener–Askey polynomial chaos for stochastic differential equations", *SIAM journal on scientific computing*, v. 24, n. 2, pp. 619–644.

YOUNG, S. R., ROSE, D. C., KARNOWSKI, T. P., LIM, S.-H., PATTON, R. M., 2015, "Optimizing deep learning hyper-parameters through an evolutio-

nary algorithm". In: *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, p. 4. ACM.

ZAHARIA, M., CHEN, A., DAVIDSON, A., GHODSI, A., HONG, S. A., KONWINSKI, A., MURCHING, S., NYKODYM, T., OGILVIE, P., PARKHE, M., XIE, F., ZUMAR, C., 2018, "Accelerating the Machine Learning Lifecycle with MLflow", *IEEE Data Eng. Bull.*, v. 41, pp. 39–45.

ZHANG, X., YAO, L., HUANG, C., SHENG, Q. Z., WANG, X., 2017, "Intent recognition in smart living through deep recurrent neural networks". In: *International Conference on Neural Information Processing*, pp. 748–758. Springer.

ZHANG, X., CHEN, X., YAO, L., GE, C., DONG, M., 2019, "Deep Neural Network Hyperparameter Optimization with Orthogonal Array Tuning". In: *International Conference on Neural Information Processing*, pp. 287–295. Springer.

ZHAO, Y., WILDE, M., FOSTER, I., 2006, "Applying the virtual data provenance model". In: *International Provenance and Annotation Workshop*, pp. 148–161. Springer.

ZHU, Y., ZABARAS, N., 2018, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification", *Journal of Computational Physics*, v. 366, pp. 415–447.