UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Rafael Gauna Trindade

ANÁLISE DE DESEMPENHO DE BIBLIOTECAS DE *DEEP LEARNING* EM ARQUITETURAS HÍBRIDAS COM ACELERADORES

Santa Maria, RS
2017

Rafael Gauna Trindade

ANÁLISE DE DESEMPENHO DE BIBLIOTECAS DE *DEEP LEARNING* EM ARQUITETURAS HÍBRIDAS COM ACELERADORES

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

ORIENTADOR: Prof. João Vicente Ferreira Lima

437
Santa Maria, RS
2017

Rafael Gauna Trindade

ANÁLISE DE DESEMPENHO DE BIBLIOTECAS DE *DEEP LEARNING* EM ARQUITETURAS HÍBRIDAS COM ACELERADORES

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

Aprovado em 12 de dezembro de 2017:

João Vicente Ferreira Lima, Dr. (UFSM)
(Orientador)

Andrea Schwertner Charão, Dra. (UFSM)

Benhur Oliveira Stein, Dr. (UFSM)

Santa Maria, RS
2017

DEDICATÓRIA

Dedico esse trabalho exclusivamente aos meus pais, Raul e Santa, que mesmo possuindo origem simples e humilde deram o melhor de si para que seus filhos conseguissem correrem atrás dos seus sonhos, nesse país que, apesar de belo por natureza, sofre com injustiças sociais. Sem seu apoio, o trajeto até aqui teria sido exponencialmente mais difícil.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais e irmãos, pelo amor, atenção e apoio incondicional que prestaram durante a graduação e também durante toda a minha vida.

Agradeço aos meus professores, que contribuíram de forma insubstituível para a minha formação e a de meus colegas, com didáticas e maneiras de promoção à busca pelo conhecimento variadas, mas em sua grande maioria eficazes.

Agradeço ao companheirismo e a força passada pelos meus colegas de curso e hoje amigos, com os quais passei incontáveis horas no Núcleo de Ciência da Computação (NCC), aulas, eventos e em competições de programação, nosso hobby. Agradecimentos especiais aos mais próximos: Ana, Cassiano, João, Lucas, Otávio, Pedro e Vinícius.

Agradeço aos meus amigos fora do curso que também, de forma insubstituível, contribuíram para que eu concluísse o curso com um ombro amigo para aliviar a pressão de provas e compromissos. Agradecimentos especiais aos mais próximos: Gabriela, Everton, Felipe, Iago, Igor, Joel, Márian e Raul.

Agradeço à Universidade Federal de Santa Maria, pelo ambiente de ensino de excelência, amigável e acolhedor que proporciona aos seus alunos e que proporcionou a mim durante esses oito semestres de graduação.

Ofereço agradecimentos também ao programa NVIDIA Hardware Grant Program que cedeu a GPU e a plataforma embarcada utilizada nos testes realizados neste trabalho. Assim como ofereço agradecimentos à Colfax Research, que cedeu acesso remoto ao cluster também utilizado neste trabalho.

E por fim, como me permite a ocasião, agradeço ao Grêmio, equipe de futebol que exibiu raça e dedicação na campanha do ano de realização deste trabalho na Copa Libertadores da América, trazendo um tri-campeonato da américa para nosso estado e motivando a mim como torcedor nessa reta final de desenvolvimento do trabalho.

AI scientists tried to program computers to act like humans without first understanding what intelligence is and what it means to understand. They left out the most important part of building intelligent machines: the intelligence... before we attempt to build intelligent machines we have to first understand how the brain thinks, and there is nothing artificial about that.

(Jeff Hawkins)

RESUMO

ANÁLISE DE DESEMPENHO DE BIBLIOTECAS DE *DEEP LEARNING* EM ARQUITETURAS HÍBRIDAS COM ACELERADORES

AUTOR: Rafael Gauna Trindade
ORIENTADOR: João Vicente Ferreira Lima

Deep Learning, ou Aprendizagem Profunda, é uma subcategoria de algoritmos de aprendizado de máquina, sendo um tema de estudos relevantes na área de Inteligência Artificial. Caracterizando-se na maior parte dos casos como Redes Neurais Artificiais de múltiplas camadas, redes de aprendizagem profunda apresentam-se como um meio de alcançar melhorias em inúmeras tarefas computacionais, como reconhecimento de fala, processamento de linguagem natural e identificação de objetos em imagens, item presente no campo de visão computacional. Sua importância cresce cada vez mais nos últimos anos, e sua popularidade aumenta conforme se tornam acessíveis bancos de dados vastos em informações e dispositivos com alta capacidade computacional. Empresas investem no ramo de pesquisa associado, e novas aplicações ficam disponíveis aos usuários finais, além da forte esperança de eficiência da sua aplicação na área da saúde. Este trabalho se propôs a analisar o desempenho e a forma como os valores de perda evoluem até convergirem, em um cenário de sobreajuste inevitável, de duas bibliotecas de *Deep Learning* relativamente populares entre desenvolvedores e pesquisadores do ramo: Caffe, desenvolvida pela Universidade de Berkley, e TensorFlow, desenvolvida pela Google. Foram conduzidas execuções de duas redes convolucionais conhecidas (AlexNet e GoogLeNet), como forma de *benchmarking*, em arquiteturas híbridas que fazem uso de aceleradores e em um cluster, variando hiperparâmetros das redes em um cenário de sobreajuste inevitável. Os resultados levaram a constatação que a biblioteca TensorFlow apresentou um melhor desempenho na maioria dos casos, e que tende a consumir menos memória para armazenar as informações da rede. Entretanto, uma porção desse desempenho se deve em parte ao uso de instruções vetorizadas, e em um cenário contrário a biblioteca Caffe pode obter mais desempenho que a concorrente, apesar de algumas deficiências técnicas. Além disso, a biblioteca Caffe apresenta um problema ao atingir o sobreajuste com valores negativos, fato que não deve acontecer em uma rede neural artificial.

Palavras-chave: Aprendizagem Profunda. Redes Neurais. Computação Heterogênea. Benchmarking. Caffe. TensorFlow

ABSTRACT

PERFORMANCE ANALYSIS OF DEEP LEARNING LIBRARIES IN HYBRID ARCHITECTURES WITH ACCELERATORS

AUTHOR: Rafael Gauna Trindade
ADVISOR: João Vicente Ferreira Lima

Deep Learning is a subcategory of machine learning algorithms and is a subject of relevant studies in the area of Artificial Intelligence. Characterized in most cases as multi-layered Artificial Neural Networks, deep learning networks present themselves as a means of achieving improvements in numerous computational tasks, such as speech recognition, natural language processing, and object identification in images, item present in the field of computer vision. Its importance has grown steadily in recent years, and its popularity increases as vast databases of information and devices with high computational capacity become accessible. Companies invest in the field of associated research, and new applications are available to end users, in addition to the strong hope of efficiency in their application in the health area. This work proposes to analyze the performance and the way that the loss values evolve until it converge, in a scenario of inevitable overfitting, of two relatively popular Deep Learning libraries among developers and researchers: Caffe, developed by the University of Berkley, and TensorFlow, developed by Google. Executions of two known convolutional networks (AlexNet and GoogLeNet) were conducted as benchmarking in hybrid architectures that use accelerators and in a cluster, varying hyperparameters of the networks in a scenario of unavoidable overfitting. The results lead to conclusion that the TensorFlow library presented a better performance in most cases, and tends to consume less memory to store network information. However a portion of this performance is due in part to the use of vectorized instructions, and in a contrary scenario, the Caffe library may outperform the competitor, despite some technical deficiencies. Besides that, the Caffe library presents a problem by reaching overfitting with negative values, a fact that should not happens in a artificial neural network.

Keywords: Deep Learning. Neural Networks. Heterogeneous Computing. Benchmarking. Caffe. TensorFlow

LISTA DE FIGURAS

Figura 3.1 – Exemplo da demonstração online de classificação de objetos com Caffe 39
Figura 3.2 – Estrutura dos sistemas local e distribuído do TensorFlow 42
Figura 4.1 – Algumas imagens do conjunto de dados empregado 49

LISTA DE GRÁFICOS

Gráfico 1.1 – Porcentagem relativa de buscas relacionadas ao termo 'Deep Learning' no buscador Google 15
Gráfico 1.2 – Porcentagem relativa de buscas relacionadas ao termo 'Deep Learning' comparada com outros termos tidos como tendências em pesquisas científicas recentes 16
Gráfico 5.1 – Evolução da perda com a biblioteca Caffe no ambiente lsc5 52
Gráfico 5.2 – Evolução da perda com a biblioteca TensorFlow no ambiente lsc5 53
Gráfico 5.3 – Evolução da perda com a biblioteca Caffe no ambiente tx1 54
Gráfico 5.4 – Evolução da perda com a biblioteca Caffe no ambiente colfax 55
Gráfico 5.5 – Evolução da perda com a biblioteca TensorFlow no ambiente colfax 56
Gráfico 5.6 – Estatísticas de tempo de execução no ambiente lsc5 57
Gráfico 5.7 – Porcentagem de uso de GPU e memória dedicada no ambiente lsc5, para a rede GoogLeNet com lotes de 128 imagens. 58
Gráfico 5.8 – Perfil de execução da biblioteca Caffe na GPU Titan X 59
Gráfico 5.9 – Perfil de execução da biblioteca TensorFlow na GPU Titan X 60
Gráfico 5.10 – Estatísticas de tempo de execução no ambiente tx1 61
Gráfico 5.11 – Estatísticas de tempo de execução no ambiente colfax 62

LISTA DE ILUSTRAÇÕES

Ilustração 1.1 – Infográfico ilustrando como uma RNA identifica um cachorro em uma imagem 17
Ilustração 2.1 – Rede neural de 4 camadas 24
Ilustração 2.2 – Neurônio Biológico e Neurônio Artificial. 25
Ilustração 2.3 – Perceptron 26
Ilustração 2.4 – Exemplo de rede alimentada para frente 28
Ilustração 2.5 – Ilustração listando os passos do estímulo visual pelo corpo 29
Ilustração 2.6 – Exemplo de classificação ajustada (parábola preta) e com sobreajuste (linha verde). 31
Ilustração 2.7 – Convolução de um sinal 2D por um filtro de tamanho 3x3 32
Ilustração 2.8 – Organização espacial dos neurônios em CNNs 32
Ilustração 4.1 – Organização de camadas na rede AlexNet 46
Ilustração 4.2 – Organização de camadas na rede GoogleNet 47

LISTA DE TABELAS

Tabela 3.1 – Propriedades das bibliotecas estudadas 43
Tabela 4.1 – Ambientes de Execução 50
Tabela 5.1 – Tempos de execução e chamadas para cada método SGEMM utilizado pelas bibliotecas 59

LISTA DE ABREVIATURAS E SIGLAS

CNN	*Convolutional Neural Network* (Rede Neural Convolucional)
CPU	*Central Processing Unit* (Unidade Central de Processamento)
$CUDA$	*Compute Unified Device Architecture* (Arquitetura de Computação em Dispositivos Unificada)
DL	*Deep Learning* (Aprendizagem Profunda)
$FLOP$	*Floating Point Operation* (Operação de Ponto Flutuante)
$GPGPU$	*General-Purpose Graphic Processing Unit* (Unidade de Processamento Gráfico de Propósito Geral)
GPU	*Graphic Processing Unit* (Unidade de Processamento Gráfico)
HPC	*High Performance Computing* (Computação de Alta Performance)
IA	Inteligência Artificial
MLP	*Multilayer Perceptrons* (Multicamadas de Percétprons)
RNA	Rede Neural Artificial
$SGEMM$	*Single precision floating point General Matrix Multiply* (Multiplicação Genérica de Matrizes de ponto flutante de precisão Simples)

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVO 18
1.1.1 Objetivos Específicos 19
1.2 JUSTIFICATIVA 19
1.3 ORGANIZAÇÃO DO TEXTO 20
2 DEEP LEARNING 21
2.1 FUNDAMENTOS 22
2.1.1 Rede Neural Artificial 23
2.1.1.1 Percéptron 25
2.1.2 Redes Neurais Alimentadas para Frente 27
2.1.3 Retropropagação 29
2.1.4 Sobreajuste 30
2.2 REDES NEURAIS CONVOLUCIONAIS 31
2.2.1 Camadas de uma CNN 33
2.2.2 Hiperparamêtros de uma CNN 34
2.2.3 Padrões de organização de camadas em uma CNN 34
3 BIBLIOTECAS 36
3.1 CAFFE 36
3.1.1 Características 37
3.1.2 Arquitetura 38
3.1.3 Aplicações 38
3.1.4 Bibliotecas de Algebra Linear 39
3.2 TENSORFLOW 40
3.2.1 Modelo de Programação 40
3.2.2 Implementação 41
3.2.3 Bibliotecas de Algebra Linear 42
3.3 TRABALHOS RELACIONADOS 42
4 METODOLOGIA 45
4.1 PLANEJAMENTO 45
4.1.1 Modelos de Redes Neurais Convolucionais 45
4.1.2 Configurações de Hiperparâmetros dos Modelos 47
4.1.3 Conjunto de Dados de Entrada 48
4.2 AMBIENTES DE EXECUÇÃO 49
5 RESULTADOS E DISCUSSÃO 51
5.1 EVOLUÇÃO DA PERDA 51
5.1.1 Ambiente de Execução lsc5 52
5.1.2 Ambiente de Execução tx1 54
5.1.3 Ambiente de Execução colfax 55
5.2 TEMPOS E ACELERAÇÃO 55
6 CONSIDERAÇÕES FINAIS 63
REFERÊNCIAS BIBLIOGRÁFICAS 65
ANEXO A – SCRIPT DE EXECUÇÃO DA BIBLIOTECA CAFFE 67
ANEXO B – SCRIPT DE EXECUÇÃO DA BIBLIOTECA TENSORFLOW 70
ANEXO C – SCRIPT PARA GERAÇÃO DE ARQUIVOS .CSV 72

1 INTRODUÇÃO

O termo *Deep Learning* (DL) vem ganhando destaque nos últimos anos. Há uma grande probabilidade do leitor já ter escutado, lido, participado de alguma pesquisa relacionada ou tido contato direto com alguma tecnologia que utilize técnicas de *Deep Learning*, como assistentes pessoais ou algum recurso interessante de alguma rede social, por exemplo. A popularidade do termo nos últimos anos aumentou devido a um crescimento recente de pesquisas na área e o surgimento de modelos de redes profundas que realizam relativamente bem o trabalho a que lhes é proposto.

Tais modelos vieram a tona por meados de 2012 (Gráficos 1.1 e 1.2), ano em que a rede convolucional AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) ganhou o *ImageNet Large Scale Visual Recognition Challenge*[1], competição que anualmente avalia algoritmos para detecção de objetos e classificação de imagens em grande escala. A rede em questão teve uma diferença de aproximadamente 10% a menos de erros em suas classificações quando comparada ao segundo colocado. Depois da AlexNet, muitas outras redes surgiram, popularizando a área e contribuindo com seus modelos para pesquisa.

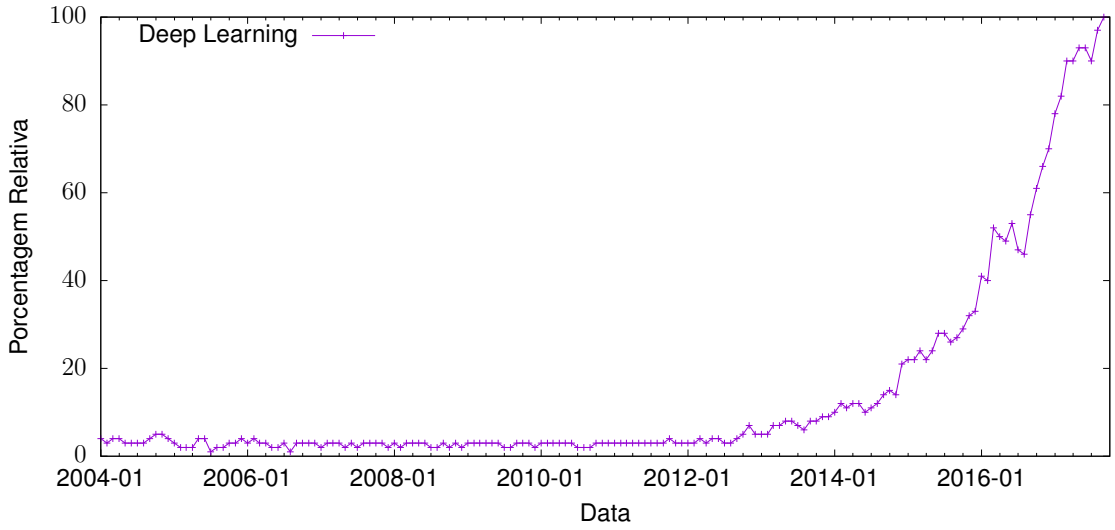

Gráfico 1.1 – Porcentagem relativa de buscas relacionadas ao termo 'Deep Learning' no buscador Google. A taxa mensal de buscas começa a crescer a partir de 2012, destacando a popularização do DL

Fonte: Adaptado de <https://trends.google.com>

Programas que utilizam *Deep Learning* geralmente são escritos utilizando **redes neurais artificiais profundas**, que podem aprender características e padrões de forma não supervisionada. Muito trabalho clássico em aprendizagem de máquinas para aplicações práticas (como reconhecimento de fala, classificação de imagem) envolveu recursos

[1] http://www.image-net.org/challenges/LSVRC/

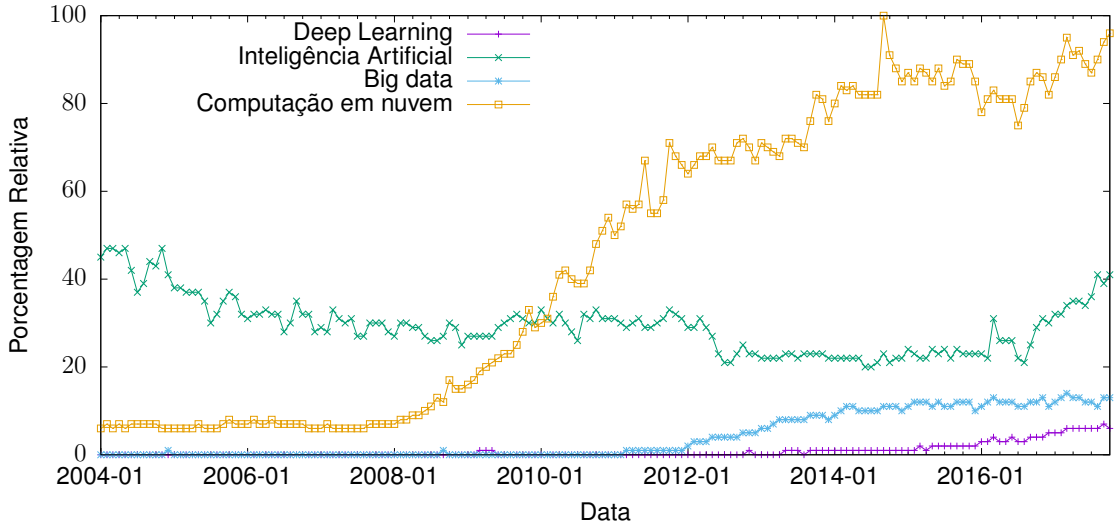

Gráfico 1.2 – Porcentagem relativa de buscas relacionadas ao termo 'Deep Learning' comparada com outros termos tidos como tendências em pesquisas científicas recentes. Note como o termo 'Inteligência Artificial' retoma uma breve estabilidade, seguida de crescimento após um período de queda constante, com o crescimento de buscas por DL. Seguindo também um padrão semelhante está o termo 'Big Data', o qual é frequentemente associado como um dos fatores que impulsionam as pesquisas na área de DL

Fonte: Adaptado de <https://trends.google.com>

criados artesanalmente para aplicações específicas. O DL leva a engenharia de recursos além disso: com dados suficientes e uma boa arquitetura de rede (existem várias heurísticas que podem ser usadas para criar boas arquiteturas para um problema específico), os neurônios em uma rede neural profunda podem aprender características abstratas – quão mais fundo se avança na rede, mais abstratas as características serão (Ilustração 1.1). Então, para muitas aplicações, redes profundas conseguem aprender a classificar características por conta própria.

Além disso, a informação característica é espalhada por vários neurônios. A rede neural não só aprende como aprender essas características, mas também sabe como combiná-las de maneira adequada. Isso é porque ela sabe o quão importante são algumas características comparadas com outras para a tarefa de classificação em mãos. Essa representação de características distribuída é, portanto, muito poderosa em comparação com outras representações de aprendizagem. Entretanto, para aprender tal representação, são necessários muitos dados. Mas como o mundo gera mais e mais dados todos os dias – estima-se que até 2024, servidores empresariais processarão por ano o equivalente digitalmente a uma pilha de livros que teria uma altura maior que 4.37 anos-luz de distância, distância suficiente para alcançar outro sistema estelar (WAGNER, 2014) – torna-se possível usar tecnologias que possam aprender características de forma automatizada.

Com as recentes melhorias na tecnologia de GPUs, uma grande quantidade de

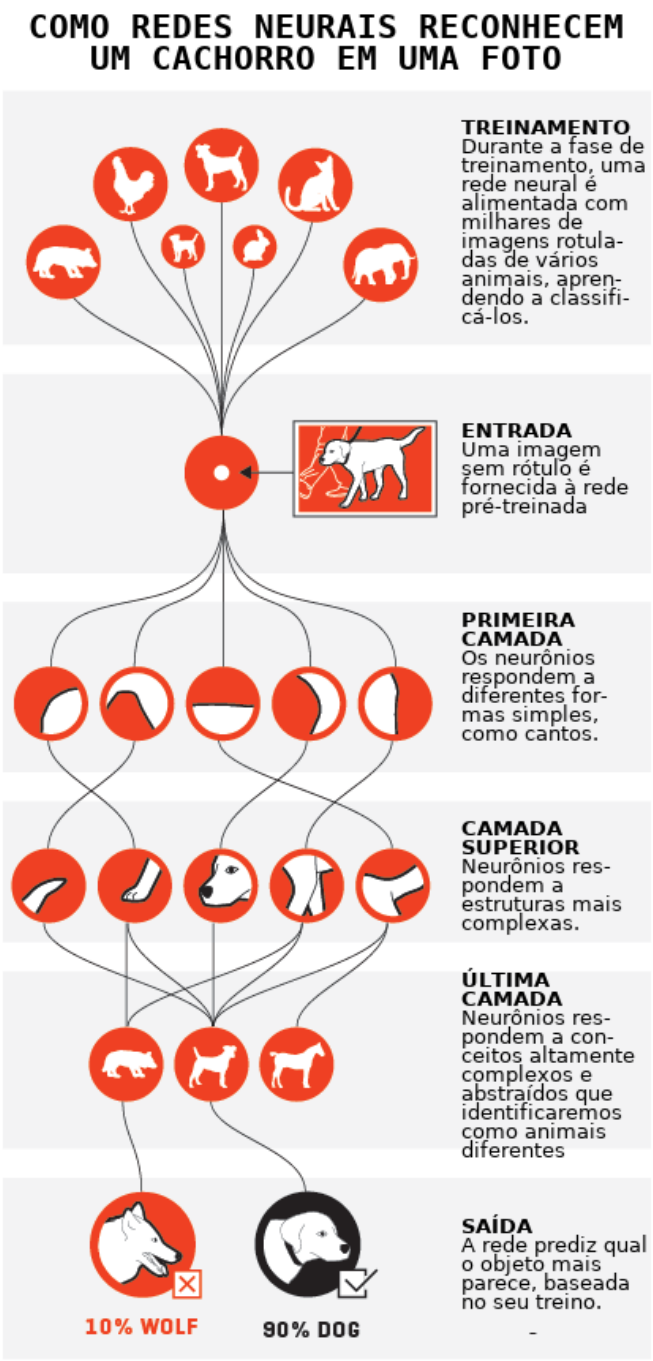

Ilustração 1.1 – Infográfico ilustrando de forma abstraída como uma rede neural identifica um cachorro em uma imagem

Fonte: Adaptado de Parloff (2016).

computações vetoriais – sempre citadas como gargalos computacionais – pode ser feita de forma muito eficiente em paralelo. Portanto, treinar uma rede profunda – que necessita de muitos cálculos vetoriais – não é tão demorado quanto costumava ser há um par de décadas atrás.

Entretanto, a implementação de softwares que treinem redes profundas acabam ressaltando a necessidade da utilização eficiente dos recursos computacionais providos por CPUs e GPUs modernas, tarefa que facilmente demandaria a escrita de uma quantia consideravelmente grande de código para tal. Com o intuito de facilitar o desenvolvimento de redes profundas e buscando reduzir o trabalho de codificação para representar tais redes – evitando, como diz um jargão bem conhecido, 'reinventar a roda' –, bibliotecas de *Deep Learning* foram criadas, simplificando as tarefas do projetista desse tipo de rede, possibilitando que ele possa focar somente no modelo da rede, e não em pormenores de desenvolvimento a um nível um pouco mais baixo. Este trabalho aborda duas dessas bibliotecas que estão bem difundidas na comunidade científica:

- **Caffe**, desenvolvida pela *Berkeley Vision and Learning Center* (BVLC), uma biblioteca desenvolvida em C++ que provê um *framework* simples e customizável para DL e uma coleção de modelos de referência. Provê ligações (*bindings*) para outras linguagens, como Python e MATLAB, simplificando o treinamento e inferência de redes neurais convolucionais de propósito geral e outros modelos profundos eficientemente em arquiteturas comuns (JIA et al., 2014).
- **TensorFlow**, desenvolvida pela Google, é uma interface para expressar algoritmos de aprendizagem de máquina, e uma implementação para executar esses algoritmos. TensorFlow permite que computações possam ser executadas com poucas modificações em uma ampla variedade de sistemas heterogêneos, desde dispositivos móveis até sistemas distribuídos em grande escala. O sistema é flexível e pode ser usado para expressar uma grande variedade de algoritmos, como de treinamento e inferência para modelos de redes neurais profundas (ABADI et al., 2016). Assim como a Caffe, é implementado em C++ e possui Python como ligação oficialmente suportada, além de possuir ligações para diversas outras linguagens, implementadas pela comunidade.

Ambas as bibliotecas suportam o uso de arquiteturas híbridas, como máquinas com CPUs e GPUs, permitindo que o projetista escolha em qual(is) unidade(s) de processamento treinará sua rede. Assim como possuem semelhanças, as bibliotecas possuem diferenças entre si ao nível de implementação e representação de modelos de rede, podendo levar a tempos diferentes para o treinamento e inferência de uma mesma rede.

1.1 OBJETIVO

Este trabalho tem como objetivo analisar o desempenho de duas bibliotecas do estado da arte de Deep Learning, buscando identificar pontos positivos e negativos de ambas

em diferentes tipos de arquiteturas com aceleradores e diferentes modelos de redes neurais profundas. Combinações variadas de hiperparâmetros, ambientes e modelos serão alternados entre treinamentos em ambas as bibliotecas, buscando cobrir uma maior variedade de cenários para que conclusões sejam tiradas a respeito do desempenho e da eficiência dessas bibliotecas.

1.1.1 Objetivos Específicos

- Estudar e relatar as características arquiteturais de duas bibliotecas de Deep Learning: Caffe e TensorFlow;
- Selecionar modelos de redes neurais convolucionais, de eficácia previamente comprovada, como meios para avaliar as bibliotecas;
- Definir variações de hiperparâmetros nas redes com o objetivo de criar diferentes cenários para avaliação, e observar comportamentos esperados e inesperados das bibliotecas perante tais mudanças;
- Empregar as rotinas de treinamento das diferentes configurações de rede e bibliotecas em diferentes categorias de arquiteturas hibridas, com capacidades de processamento e armazenamento diferentes, de modo a avaliar como as bibliotecas se comportam;
- Avaliar variáveis resultantes do processo de treinamento, como evolução da perda e tempo por imagem, buscando tirar conclusões a respeito do desempenho e corretude das bibliotecas com os diferentes tipos de redes, arquiteturas e propósitos;

1.2 JUSTIFICATIVA

A crescente demanda em pesquisas na criação de redes profundas e aplicações de DL evidenciam o grau de importância da área no mundo contemporâneo, cada vez mais tecnológico. A disponibilização das bibliotecas de forma livre serve como motivação tanto para estudantes iniciantes e pesquisadores experientes da área para modelarem e implementarem suas próprias soluções no formato de redes neurais profundas.

Entretanto, a maior parte desses indivíduos não possuem *cluster*s nem *grid*s computacionais de alto desempenho para que possam treinar suas redes de forma distribuída e rápida, possibilitando em consequência acelerar conclusões a respeito de seus estudos.

Buscando mudar essa barreira, tais bibliotecas propõem possibilitar o treinamento intensivo de redes com necessidade mínima de hardware, como apenas uma CPU e/ou GPU potentes o suficiente. Entretanto, um interesse também pode ser esperado de setores da indústria com o poder computacional que falta aos pesquisadores citados anteriormente: a diminuição do tempo total de prototipação e de execução para treinamentos e inferências de redes com propósito comercial pode afetar na forma como as empresas lucram, produzindo mais em um tempo menor.

Estudos como o proposto por este trabalho visam analisar e destacar pontos de eficiência ou deficiência dessas implementações, contribuindo para que melhorias efetivas possam ser implementadas pelas equipes de desenvolvimento por trás das bibliotecas.

1.3 ORGANIZAÇÃO DO TEXTO

A organização do presente trabalho se dá na seguinte maneira: o Capítulo 2 aborda a fundamentação teórica do *Deep Learning*, apresentando detalhes a respeito de sua popularização e especificando as redes neurais artificiais que normalmente compõe suas aplicações. O Capítulo 3 apresenta as bibliotecas escolhidas por este trabalho e apresenta alguns pormenores de suas implementações e da gama de soluções onde as quais podem ser empregadas. O Capítulo 4 detalha a etapa de elaboração e a metodologia empregada nos treinamentos que foram conduzidos nas redes em combinação com as bibliotecas. O Capítulo 5 apresenta os resultados dos treinamentos submetidos as redes nas seguintes formas: evolução da perda no decorrer dos treinamentos, tempos médios gastos por imagem pra cada configuração e a aceleração provida pelos dispositivos heterogêneos como GPUs, bem como discute os resultados obtidos considerando todas as informações levantadas. Por fim, o Capítulo 6 apresenta uma visão geral do trabalho que fora conduzido e considerações finais sobre o mesmo.

2 DEEP LEARNING

Deep Learning (Aprendizagem Profunda, em português) refere-se a um conjunto importante de algoritmos computacionais de aprendizado de máquina. Essa classe de algoritmos possui origens datadas da década de 1960, onde, segundo Schmidhuber (2014), pesquisadores começaram a modelar redes profundas baseadas em descobertas da época sobre o córtex visual de gatos. Entretanto, sua popularidade começou a crescer consideravelmente entre pesquisadores e inclusive entre o público geral na última década. Tal destaque, segundo Vasconcelos e Clua (2017), Deng e Yu (2014), se deve basicamente a 3 fatores:

1. A grande quantidade de dados disponíveis (*Big Data*), fundamentais para alimentar os algoritmos de aprendizado de máquina, servindo como entrada rotulada ou não-rotulada;

2. Os avanços em pesquisas sobre aprendizado de máquina e processamento de sinal/informação, aliado as baixíssimas taxas de erros alcançadas por redes profundas famosas como a AlexNet;

3. A popularização do uso de *hardware* com alto poder de processamento, como GPGPUs (*General-Purpose Graphical Processing Units*), que nos últimos anos têm exibido um crescimento notável de desempenho conforme novos dispositivos são lançados no mercado.

Empresas de renome tem investido financeiramente em pesquisas na área de aprendizado de máquina, dentre elas: Intel, NVIDIA, Microsoft, Amazon, Facebook, Google, IBM, entre outras. Segundo Deng e Yu (2014), esses esforços de pesquisa atingiram sucessos empíricos em aprendizado profundo com diversas aplicações de visão computacional, reconhecimento fonético, pesquisa de voz, reconhecimento de fala conversacional, codificação de função de fala e imagem, classificação de expressão semântica, compreensão de linguagem natural, reconhecimento de escrita manual, processamento de áudio, recuperação de informações, robótica, e inclusive na análise de moléculas que podem levar à descoberta de novos medicamentos. Vasconcelos e Clua (2017) também ressalta a importância das bibliotecas de *deep learning* desenvolvidas pelas empresas e por Universidades que aceleram a pesquisa na área e contribuem para o aumento no número de aplicações existentes. Algumas dessas bibliotecas serão abordadas mais à frente neste capítulo.

Dentre inúmeras definições formais sobre o que é aprendizado profundo encontradas na literatura, geralmente dois aspectos-chave estão presentes na maioria delas (DENG; YU, 2014):

1. São modelos que consistem em múltiplas camadas ou estágios de processamento não-linear de informação;

2. São métodos de aprendizado supervisionado (ou não) de representação de características em camadas sucessivamente superiores e mais abstratas.

Deng e Yu (2014) explicam que os fatores de popularização citados anteriormente permitiram que os métodos de DL explorassem funções de composição complexas e não-lineares, para aprender a representar características de forma distribuída e hierárquica, e para fazer uso efetivo de dados de entrada rotulados e não-rotulados. O uso de GPUs modernas, por exemplo, permite que sejam realizadas trilhões de operações aritméticas por segundo. Segundo Vasconcelos e Clua (2017), uma única placa NVIDIA com a tecnologia Pascal consegue realizar mais que o dobro de operações com pontos flutuantes que o melhor supercomputador disponível no ano 2000.

Este capítulo está organizado da seguinte maneira. A Seção 2.1 aborda sobre os fundamentos do Deep Learning e os componentes de suas aplicações mais comuns, que são formadas por redes neurais artificiais profundas. A Seção 2.2 introduz o tipo de rede profunda mais difundido atualmente – rede neural convolucional –, apresentando detalhes sobre seu conceito de convolução e tipos de camadas.

2.1 FUNDAMENTOS

Goodfellow, Bengio e Courville (2016) relatam que muitos projetos de inteligência artificial dependiam de conhecimento sobre o mundo traduzido manualmente para código usando linguagens formais, em um processo praticamente artesanal, tornando possível para um computador raciocinar automaticamente sobre afirmações nessas linguagens usando regras de inferência lógica. Entretanto, as dificuldades enfrentadas pelos sistemas que dependem de conhecimento transformado em código sugerem que os sistemas de IA precisam ter a capacidade de adquirir seus próprios conhecimentos, extraindo padrões de dados brutos. Esta capacidade é conhecida como aprendizado de máquina. A chegada da aprendizagem de máquina permitiu que computadores abordassem problemas envolvendo conhecimento do mundo real e tomassem decisões que pareciam subjetivas.

A abordagem clássica do aprendizado de máquina é geralmente composta de duas etapas: extração de dados de entrada a partir de dados brutos e a extração de características por modelos matemáticos formalizados para aquele assunto em específico, geralmente usando conhecimentos especialistas. Entretanto, a criação desses modelos não é uma tarefa trivial (VASCONCELOS; CLUA, 2017) e a gama de variações que um modelo pode cobrir pode tornar a criação desses modelos ainda mais complexos: por exemplo, um sistema especialista pode conseguir identificar um gato em uma imagem. Mas, e se atributos

de um gato como cor dos pelos, escala na imagem, pose e iluminação variarem de forma considerável? A formalização de um modelo matemático que identifique um gato acaba se tornando uma tarefa muito mais complexa. Assim como no campo de reconhecimento de imagens, esse problema se estende facilmente para qualquer outro tipo de sinal, como áudio. E nesse ponto que redes de aprendizado profundo apresentam-se como uma solução: Vasconcelos e Clua (2017) explicam que a etapa de descrição formal do modelo pela formulação matemática e implementação via código do conhecimento especialista para extração de características é substituída por um processo de treinamento de redes neurais de maneira integrada à etapa de tomada de decisão. Esse treinamento permite que a rede aprenda através de treinos exaustivos sobre grandes volumes de dados brutos, identificando diretamente quais padrões são relevantes e assim ajustando os próprios parâmetros para que posteriormente possa fornecer um mapeamento entre um novo dado de entrada e sua saída no problema modelado.

Logo, para ter-se um entendimento maior sobre redes de aprendizado profundo, primeiro precisa-se conhecer sua base: redes neurais artificiais (RNA). A subseção a seguir aborda o conceito de RNA, a origem de seu nome e como geralmente são compostas. Também apresentará tipos diferentes de redes e como são projetadas.

2.1.1 Rede Neural Artificial

Redes neurais artificiais são modelos computacionais inspirados pelas redes neurais biológicas que constituem o cérebro de animais. Esses sistemas aprendem a realizar tarefas a partir de exemplos, geralmente sem programação específica para tal. São extremamente úteis em aplicações complexas demais para serem expressadas em algoritmos de computador tradicionais, usando programação imperativa. Segundo Goodfellow, Bengio e Courville (2016), essas redes são chamadas assim porque são inspiradas pela neurociência. Cada camada da rede normalmente é vetorial, e cada elemento do vetor pode ser interpretado como um papel análogo ao neurônio. Cada unidade se parece a um neurônio no sentido de que recebe entrada de muitas outras unidades e calcula seu próprio valor de ativação. A ideia de usar muitas camadas de representação vetorial é tirada também da neurociência: a escolha de funções para calcular essas representações também é guiada vagamente por observações neurocientíficas sobre as funções que os neurônios biológicos calculam.

Segundo Schmidhuber (2014), uma rede neural padrão consiste em muitas unidades processadoras simples e conectadas, chamadas neurônios, cada uma produzindo uma sequência de ativações. Uma rede geralmente é constituída de, pelo menos, neurônios de entrada e neurônios de saída. Os neurônios de entrada são ativados através de sensores que percebem o meio ambiente, enquanto outros neurônios são ativados através

de conexões ponderadas (sinapses) de neurônios previamente ativos. Tais sensores geralmente fornecem conjuntos de sinais aos neurônios, não se limitando a sensores físicos. A *aprendizagem* se dá em encontrar pesos para essas sinapses que façam com que a rede neural atinja o comportamento desejado. Dependendo do problema e do modo como os neurônios estão conectados, esse comportamento pode exigir longas cadeias causais de estágios computacionais, onde cada estágio transforma – geralmente de forma não-linear – a ativação agregada da rede.

Normalmente os neurônios são organizados em camadas, sendo comum classificá-las em 3 grupos: camadas de entrada, ocultas (interior) e de saída. A Ilustração 2.1 apresenta uma visualização gráfica de uma rede neural com camadas ocultas. Camadas diferentes podem executar diferentes tipos de transformações em suas entradas. Os sinais viajam desde a primeira (entrada) até a última camada (saída), possivelmente depois de atravessar as camadas várias vezes, podendo inclusive, dependendo do modelo, atravessar camadas ocultas no interior da rede.

Apesar da área de RNA ter sido inicialmente inspirada principalmente no objetivo de modelar sistemas neurais biológicos, com o passar do tempo divergiu consideravelmente da definição biológica e se tornou uma questão de engenharia, obtendo bons resultados nas tarefas de aprendizagem de máquinas (KARPATHY, 2014). Essa divergência se deve em grande parte ao enfoque em habilidades mentais específicas, como redes com retropropagação ou passagem de informações no sentido inverso. Segundo Goodfellow, Bengio e Courville (2016), a pesquisa moderna de redes neurais é orientada por muitas disciplinas matemáticas e de engenharia, e o objetivo das redes neurais não é modelar perfeitamente o cérebro.

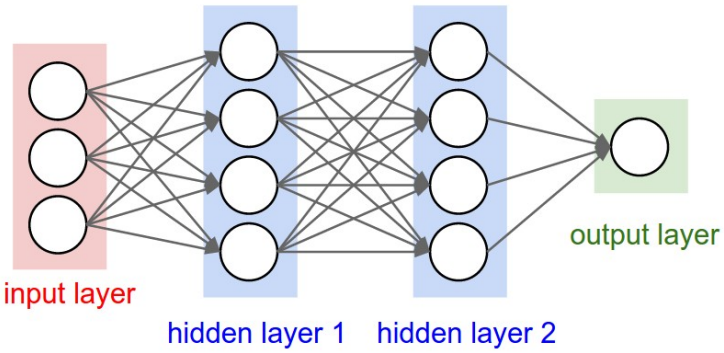

Ilustração 2.1 – Uma rede neural de 3 camadas com 3 entradas, duas camadas ocultas de 4 neurônios cada e uma camada de saída. Perceba que há conexões (sinapses) entre neurônios através das camadas, mas não dentro de uma camada.

Fonte: Retirado de Karpathy (2014)

A subseção a seguir (2.1.1.1) detalha o componente principal para redes neurais artificiais: o percéptron.

2.1.1.1 Percéptron

O modelo de neurônio usado pela maioria dos projetos de redes neurais é baseado em um modelo proposto na década de 1950 por Frank Rosenblatt, inspirado por trabalhos anteriores de Warren McCulloch e Walter Pitts, denominado percéptron (ROSENBLATT, 1962). Baseado no conceito de neurônio artificial derivado de neurônios biológicos (Figura 2.2), o percéptron é composto de 3 partes (Figura 2.3):

- **Canais de entrada**: Conjuntos de sinais repassados ao núcleo do percéptron, análogos a estrutura de dendritos de um neurônio biológico. No modelo de Rosenblatt, cada canal possui um peso associado, responsável por ponderar a influência do sinal para determinada tarefa exercida pelo percéptron.
- **Núcleo**: Etapa de processamento do neurônio, estrutura análoga ao núcleo do neurônio biológico. Rosenblatt compõe o núcleo do neurônio em duas etapas:
  1. A combinação linear dos sinais de entrada, associados aos seus respectivos pesos;
  2. A aplicação de uma função de ativação sobre a combinação gerada, geralmente uma transformação não-linear, de maneira a aumentar o poder de expressão do neurônio (VASCONCELOS; CLUA, 2017).
- **Saída**: Por fim, um canal de saída do resultado da transformação aplicada pelo núcleo no sinal de entrada. Essa saída pode ser replicada a diversos outros neurônios, tendo uma finalidade análoga à de um axônio de um neurônio biológico.

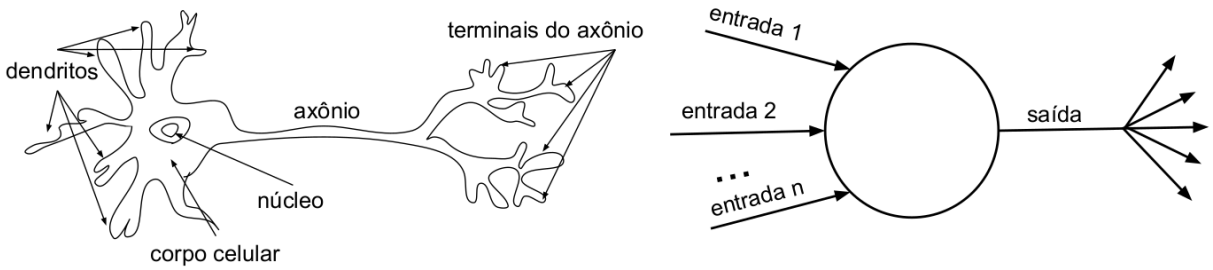

Ilustração 2.2 – Neurônio Biológico e Neurônio Artificial.

Fonte: Retirado de Vasconcelos e Clua (2017)

Assim, segundo Vasconcelos e Clua (2017), um neurônio que possui N conexões de entrada, associadas respectivamente aos pesos $W = \{w_i | 1 < i \leq N\}$, produz uma combinação linear z do sinal de entrada $X = \{x_i | 1 < i \leq N\}$, descrita pela equação:

$$z = W.X + b = \sum_{i=1}^{N} w_i x_i + b \tag{2.1}$$

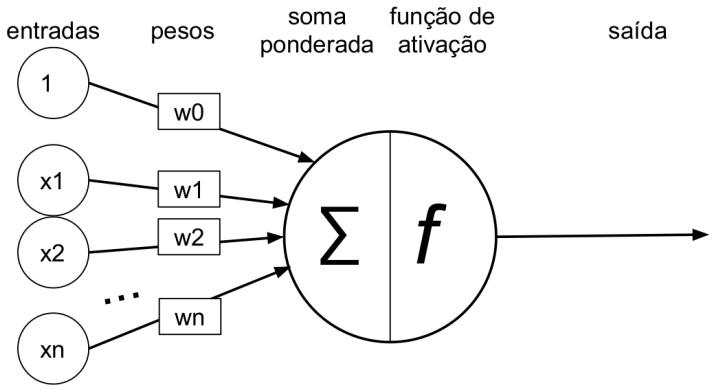

Ilustração 2.3 – Perceptron: a combinação linear das entradas x_i ponderadas pelos pesos w_i é transformada pela função de ativação f na saída y emitida pelo neurônio.

Fonte: Retirado de Vasconcelos e Clua (2017)

O termo b na equação acima refere-se ao *bias*, um valor independente dos sinais de entrada que aumenta os graus de liberdade da fronteira de decisão representada pelo neurônio. É comum encontrarmos na literatura a inclusão do valor constante 1 como primeiro elemento do conjunto de entrada, de maneira que b passa a ser representado pelo peso associado a tal entrada (w_0) . A Equação 2.1 pode então ser reescrita como:

$$z = W.X = \sum_{i=0}^{N} w_i x_i = w_0 1 + w_1 x_1 + ... + w_N x_N \tag{2.2}$$

O treinamento de percéptrons inicia-se com uma submissão de conjuntos de amostras de entrada que possuem uma saída conhecida. Essas amostras são compostas por um par (X, Y), no qual X é um vetor n-dimensional descrevendo um sinal de entrada, e Y é a resposta correspondente desejada (VASCONCELOS; CLUA, 2017), que disponibilizam o necessário para que a rede aprenda e possa criar fronteiras de decisão que classificam as informações aprendidas: seja na forma de retas, para informações em entradas usando duas dimensões, planos para três dimensões, ou hiperplanos para mais dimensões. Essas fronteiras são configuradas pelos parâmetros W e b, que são aprendidos durante o treinamento. Seus valores são tipicamente inicializados com valores aleatórios para o conjunto W e com zero para o *bias*. Uma vez com o conjunto de amostras e os parâmetros iniciais definidos, o treinamento em si consiste de duas etapas a serem alternadas repetidas vezes. A cada iteração t:

1. processa o sinal de entrada (X), produzindo um valor de saída $o(t)$;
2. ajusta os parâmetros do neurônio de acordo com a a saída obtida e o resultado desejado (Y).

O ajuste realizado na segunda etapa é diretamente relacionado ao aprendizado

obtido pela rede. O erro obtido na iteração t é calculado como $e(t) = Y - o(t)$ e é utilizado para atualizar os parâmetros, multiplicado por um fator α que determina a taxa de aprendizado (*learning rate*), escolhido antes do treinamento, utilizando diferentes políticas, indicando a influência da amostra em questão para o erro obtido. Esses treinamentos podem ser repetidos até que o erro obtido seja menor que um valor estipulado, ou por uma quantidade delimitada de vezes (VASCONCELOS; CLUA, 2017).

2.1.2 Redes Neurais Alimentadas para Frente

Dentre os muitos tipos de RNA estudados desde o início dos esforços de pesquisa, há uma espécie de rede na qual, quando se aceita uma entrada X para produzir uma saída Y, a informação flui sempre '*para frente*' – leia-se em uma direção, da entrada para a saída. Essas redes são denominadas redes alimentadas para frente, ou *feedforward networks*, em inglês. Geralmente são composta de múltiplas camadas sequenciais de percéptrons, recebendo a sigla MLP (*multilayer perceptrons*), e são a quintessência para modelos de *deep learning* (GOODFELLOW; BENGIO; COURVILLE, 2016).

Goodfellow, Bengio e Courville (2016) explica que o objetivo de uma rede alimentada para frente é aproximar-se de alguma função f^*. Por exemplo, para um classificador, $Y = f^*(X)$ mapeia uma entrada X para uma categoria Y. Uma rede alimentada para a frente define um mapeamento $Y = f(X; \theta)$ e aprende o valor dos parâmetros θ que resultem na melhor aproximação possível da função. A informação (ou sinal) flui através da função sendo avaliada a partir de X, através de computações intermediárias usadas para definir f, e finalmente chega à saída Y. Não há conexões para trás (*feedback*) nas quais as saídas dos modelos alimentam a si próprios ou às camadas anteriores na sequência. Quando se dá a existência desse tipo de comportamento, a nomenclatura empregada para a rede é de *rede neural recorrente*. A Ilustração 2.4 exibe um exemplo de rede alimentada para frente desenhada em dois estilos.

Segundo Vasconcelos e Clua (2017), redes alimentadas para frente se destacam por possuírem formulações de modelos simples e intuitivos: a configuração de camadas é diretamente relacionada à dimensionalidade dos dados e da solução. A escolha do número de camadas e do número e disposição de neurônios geralmente é realizada por experimentação. Goodfellow, Bengio e Courville (2016) mencionam sobre a extrema importância das redes alimentadas para frente para praticantes de aprendizado de máquina: essas redes formam a base de muitas aplicações comerciais importantes, citando como exemplo o uso especializado de redes convolucionais para reconhecimento de objetos em imagens.

As redes alimentadas para frente são classificadas como rede porque são normalmente representadas pela composição de muitas funções diferentes. O modelo está associado a um grafo acíclico dirigido descrevendo como as funções são compostas em

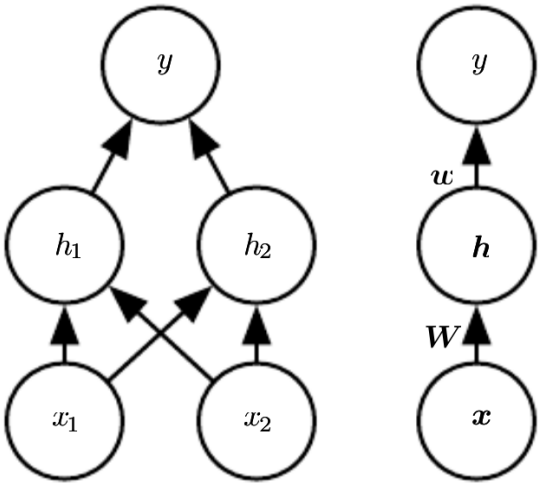

Ilustração 2.4 – Um exemplo de rede alimentada para frente desenhada em dois estilos diferentes. (*Esquerda*)Nesse estilo, cada unidade (neurônio) é um nó no grafo. Este estilo é bem explícito e não-ambíguo, mas para redes maiores que esse exemplo, o grafo pode vir a ocupar muito espaço. (*Direita*)Nesse estilo, cada nó representa a ativação de uma camada. Esse estilo é bem mais compacto, e as arestas nesse grafo podem descrever o relacionamento entre duas camadas.

Fonte: Retirado de Goodfellow, Bengio e Courville (2016)

conjunto. Por exemplo, podemos ter três funções $f(1)$, $f(2)$ e $f(3)$ conectadas em cadeia, para formar a equação 2.3. Essas estruturas de cadeia são as estruturas mais comumente usadas nas redes neurais (GOODFELLOW; BENGIO; COURVILLE, 2016). A profundidade do modelo é determinado pelo comprimento total da rede. É dessa terminologia que surge o nome aprendizado profundo. A camada final de uma rede alimentada para frente é chamada de camada de saída. Durante a formação da rede neural, conduzimos $f(X)$ para corresponder $f^*(X)$ (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$f(X) = f(3)(f(2)(f(1)(X))) \tag{2.3}$$

Ainda segundo Goodfellow, Bengio e Courville (2016), os dados de treinamento nos fornecem exemplos ruidosos e aproximados de $f^*(X)$ avaliados em diferentes pontos de treinamento. As amostras de treinamento especificam exatamente o que a camada de saída deve fazer em cada ponto X: produzir um valor muito próximo de Y. Entretanto, o comportamento das outras camadas não é especificado diretamente pelos dados da amostra. O algoritmo de aprendizagem deve então decidir como usar as camadas internas da rede para produzir a saída Y desejada, mas os dados de treinamento não dizem o que cada camada dessas deve fazer. Em vez disso, o algoritmo de aprendizagem deve decidir como usar essas camadas para implementar melhor uma aproximação de f^*.

A Ilustração 2.5 mostra como um sinal visual desencadeia uma reação humana,

com o sinal cerebral seguindo um caminho em somente uma direção, assim como uma rede alimentada para frente. Apesar de sua semelhança com redes neurais biológicas,é melhor pensar em redes alimentadas para frente como máquinas de aproximação de funções que são projetadas para alcançar a generalização estatística, ocasionalmente desenhando algumas ideias sobre o que sabemos sobre o cérebro e não como modelos de função cerebral.

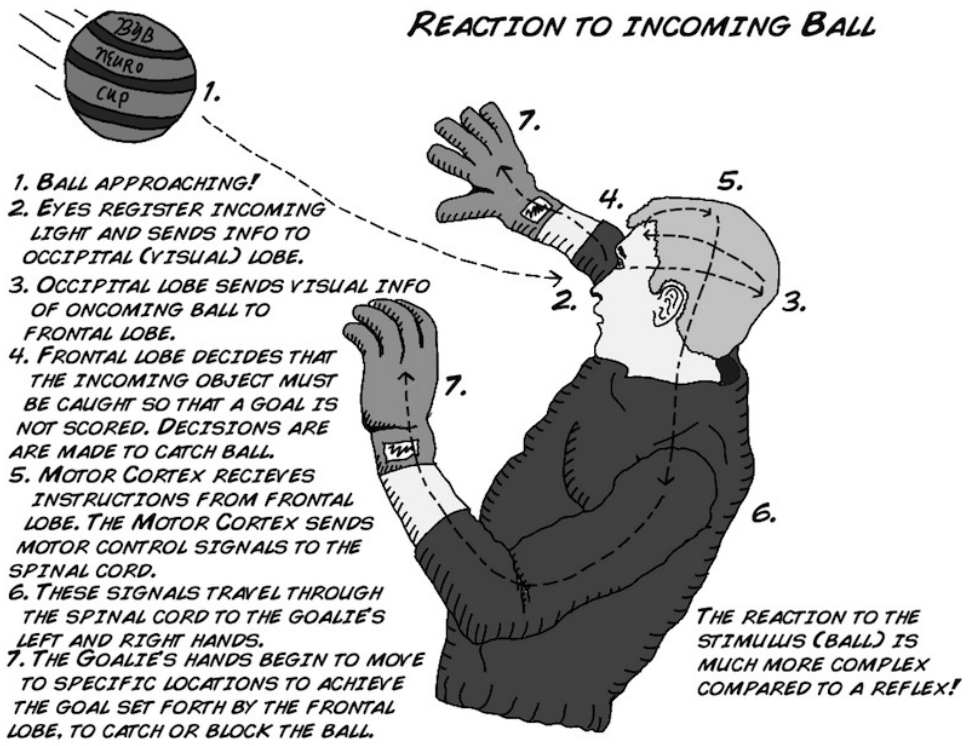

Ilustração 2.5 – Ilustração listando os passos do estímulo visual gerado pelos olhos, através do cérebro até chegar aos músculos do corpo, desencadeando uma reação quase que instantânea.

Fonte: Retirado de https://backyardbrains.com/experiments/reactiontime

2.1.3 Retropropagação

O algoritmo de retropropagação (*backpropagation*, em inglês) permite que a informação de erro flua para trás através da rede, a fim de calcular o gradiente para a sua otimização e o posterior aprendizado. O custo computacional para computar uma expressão analítica para o gradiente é consideravelmente alto, entretanto, o algoritmo de retropropagação faz isso usando um procedimento simples e menos custoso (GOODFELLOW; BENGIO; COURVILLE, 2016).

Logo, retropropagação refere-se ao método para computar o gradiente, o qual é

provido para o algoritmo de *gradiente descendente estocástico* realizar o aprendizado na rede, alterando pesos e *biases*. Esse método pode computar derivadas de qualquer função, podendo ser aplicado para redes com mais de uma camada. Goodfellow, Bengio e Courville (2016) especificam que um algoritmo de retropropagação descreve como computar o gradiente $\nabla f(X, Y)$ para uma função arbitrária f, onde X é um conjunto de variáveis as quais suas derivadas são desejadas, e Y o conjunto adicional de variáveis que são entradas para a função, mas que suas derivadas não são necessárias. Em algoritmo de aprendizado, o gradiente que geralmente é procurado é o gradiente da função de erro com respeito aos parâmetros, $\nabla_\theta J(\theta)$, referente ao custo escalar $J(\theta)$, produzido pela propagação direta contínua (propagação do sinal da camada de entrada em direção à de saída) durante o treinamento de uma rede neural alimentada para frente (GOODFELLOW; BENGIO; COURVILLE, 2016). Muitas tarefas de aprendizado de máquina envolvem computar outras derivadas, seja como parte do processo de aprendizado, ou para analisar o modelo aprendido, e para estas o algoritmo de retropropagação também pode ser aplicado.

O que torna o algoritmo de retropropagação pouco custoso computacionalmente é o fato dele necessitar de apenas uma passagem pela rede a cada treinamento: uma vez completada uma passagem de propagação direta, o algoritmo executa uma passada por toda a rede no sentido oposto – da camada de saída em direção a camada de entrada –, repropagando o erro e fazendo o cálculo do gradiente.

2.1.4 Sobreajuste

Sobreajuste (do termo em inglês, *overfitting*) é um problema estatístico que se aplica sobre redes neurais: Um modelo pode se ajustar tão bem na tarefa de classificação de um conjunto de dados que pode acabar se tornando ineficaz para classificar dados novos (Ilustração 2.6.)

Em RNAs, o problema de sobreajuste pode levar a rede a não aprender mais do que já aprendeu. O modelo aprende os detalhes e os ruídos dos dados de treinamento na medida em que afeta negativamente o desempenho do modelo em novos dados. Esse problema pode aparecer por diversos motivos (SARLE, 1995), que normalmente devem ser evitados, como conjunto de dados de entrada reduzido, altas taxas de aprendizado – valor que determina com que velocidade a rede aprende – e ausência de etapas de validação durante o treinamento.

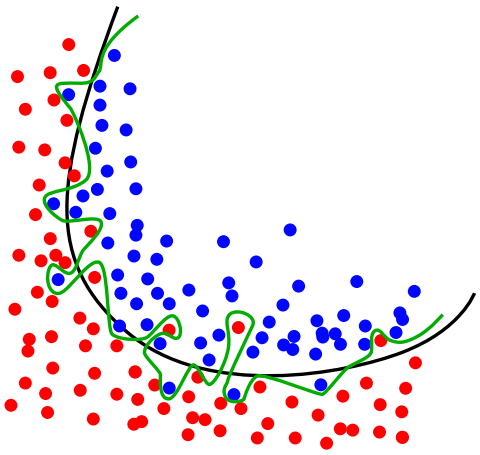

Ilustração 2.6 – Exemplo de classificação ajustada (parábola preta) e com sobreajuste (linha verde).

Fonte: Retirado de Cárdenas-Monte (2015)

2.2 REDES NEURAIS CONVOLUCIONAIS

Dentro da classe de redes neurais alimentadas para frente, nos últimos anos a classe de rede neural convolucional (*Convolution Neural Network*, em inglês), também chamada de CNN ou ConvNet, vêm ganhando muito destaque na indústria e nos campos de pesquisa. Esse destaque se deve ao fato desta obter excelentes resultados em tarefas de classificação de objetos em visão computacional, podendo esse tipo de rede possuir características que a possibilite realizar aprendizagem profunda. O termo 'convolução' adotado no nome vem da definição matemática de convolução: uma integral que expressa a quantidade de sobreposição de uma função g à medida que é deslocada sobre outra função f, 'combinando' uma função com outra e resultando numa terceira que mede a área subentendida pela superposição delas em função do deslocamento existente entre elas (WEISSTEIN, 2017). A partir dessa definição pode-se facilmente identificar a convolução em uma CNN pelo fato da rede aplicar o mesmo 'filtro' em diversas regiões menores de um sinal de entrada, como se estivesse deslizando sobre a área do sinal, nas suas chamadas camadas de convolução (Figura 2.7).

Assim como uma rede alimentada para frente, o fluxo de informações em uma CNN flui em uma direção – e também pode aprender utilizando retropropagação, com uma segunda passada na direção oposta. Esse tipo de rede possui diversas camadas, as quais são baseadas em analogias a comportamentos de células do córtex visual de animais: células simples, que têm seu disparo maximizado quando seu campo receptivo apresenta arestas em orientações particulares (em uma CNN, camadas de convolução); células complexas, cujo disparo é insensível a posição exata das arestas no campo observado, e que cobrem campos receptivos maiores do que os das células simples (em uma CNN, camadas de agrupamento) (VASCONCELOS; CLUA, 2017). Cada camada de uma CNN produz

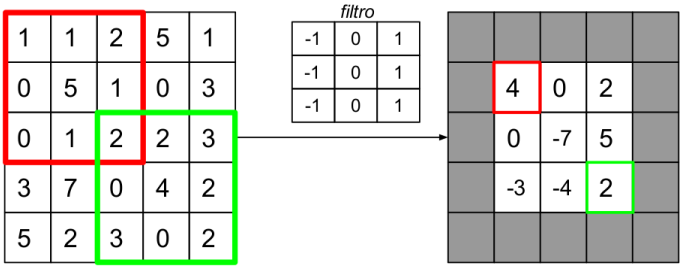

Ilustração 2.7 – Convolução de um sinal 2D por um filtro de tamanho 3x3. O filtro é aplicado deslizando-se sobre o sinal de entrada, aplicando uma convolução propriamente dita. Note o redimensionamento do sinal após aplicação do filtro. O mesmo pode ser evitado com a aplicação de uma técnica que preenche as laterais do sinal de entrada com zeros (*zero padding*).

Fonte: Retirado de Vasconcelos e Clua (2017)

um volume de ativações (Figura 2.8) que servirá de entrada para a próxima camada, podendo esse volume ser interpretado como a saída de um neurônio que olha somente uma região pequena da entrada e compartilha parâmetros (pesos e biases) com seus neurônios vizinhos – uma vez que muitos dos valores gerados no volume de saída são frutos da aplicação do mesmo filtro (KARPATHY, 2014).

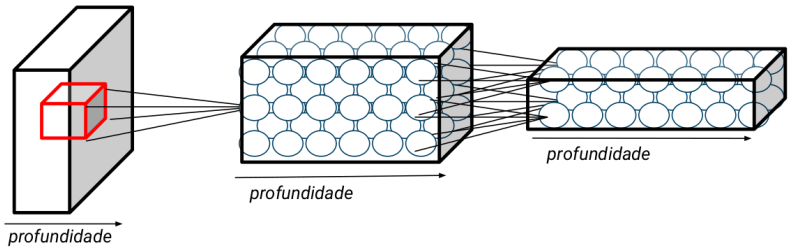

Ilustração 2.8 – Organização espacial dos neurônios em CNNs. Cada bloco representa uma camada, por onde o sinal de entrada passa gerando novos volumes de ativação, por vezes de tamanhos diferentes. Note o destaque em vermelho da região avaliada por um neurônio.

Fonte: Retirado de Vasconcelos e Clua (2017)

Os dados de entrada de uma CNN geralmente constituem-se de imagens, dada sua aplicação no campo de classificação de objetos em imagens. Arquivos de imagens geralmente fornecem dados de entrada em três dimensões, levando-se em consideração o posicionamento dos *pixels* e os valores dos canais de cores (vermelho, verde e azul). Essas três dimensões são, consequentemente, largura, altura e profundidade. Essa última dimensão tem vital importância em CNNs, uma vez que ela pode ditar a precisão e o custo computacional da rede. Assim como os dados de entradas comumente possuem 3 dimen-

sões, os volumes de ativações intermediários, gerados por cada camada da rede, também possuem 3 dimensões. É comum implementações de CNNs manterem em memória esses dados intermediários, entre passadas, para auxiliar o algoritmo de retropropagação. Dito isto, é possível associar o consumo excessivo de memória aos potenciais problemas que uma implementação de CNN pode possuir.

2.2.1 Camadas de uma CNN

CNNs possuem diferentes tipos de camadas, mas são poucas as variações frequentemente utilizadas. Podem ser listadas como tipos de camadas comumente encontradas em CNNs (KARPATHY, 2014):

- **Camada de entrada** (INPUT): Como geralmente CNNs trabalham com imagens, essa camada contém os dados brutos sobre informação das cores de cada *pixel* da imagem. Geralmente possui três dimensões: largura, altura e profundidade;
- **Camada de convolução** (CONV): Computa a saída de neurônios que estão conectados a regiões menores no sinal de entrada, cada um computando um produto vetorial entre seus pesos e as regiões ao qual estão conectados. Seus neurônios não possuem uma função de ativação não-linear;
- **Camada de ativação** (RELU): Nas definições vistas anteriormente, era a função não-linear integrada nos neurônios. Em CNNs, essas funções tem uma camada própria, vindas logo após camadas que aplicam somente combinações lineares, como as de convolução;
- **Camada de agrupamento** (POOL): Do inglês *pooling*, realiza uma operação de redimensionamento do volume de ativações, diminuindo-o. Geralmente associada a uma função MAX que reduz em 75% o volume de ativações, são importantes para controle de sobreajuste;
- **Camada completamente conectada** (FC): Nesse tipo de camada, cada neurônio possui conexões com todos os neurônios da camada anterior. Geralmente são utilizadas para computar as taxas de classificação (o quão próximo de um resultado aquela entrada fora).

2.2.2 Hiperparamêtros de uma CNN

Segundo Karpathy (2014), camadas do tipo CONV e FC geralmente contém parâmetros, enquanto que RELU e POOL geralmente não possuem, destinando-se somente a aplicar diretamente uma função sobre o sinal de entrada. Camadas do tipo CONV, FC e POOL geralmente também possuem hiperparamêtros, enquanto camadas RELU não. Hiperparamêtros são valores utilizados para controlar o tamanho do volume de saída, e os mesmos são fixos, não podendo ser aprendidos. Dentre os hiperparamêtros mais comuns, estão:

- **Profundidade** (*depth*): corresponde ao número de filtros contidos na camada, cada qual buscando algo diferente no sinal de entrada. Por exemplo, se a primeira camada convolucional tomar como entrada uma imagem, então diferentes neurônios ao longo da dimensão da profundidade podem se ativar na presença de várias arestas orientadas, ou bolhas de cor. Refere-se a um conjunto de neurônios que estão olhando a mesma região da entrada como uma coluna de profundidade, também recebendo o nome de fibra. A profundidade de uma camada determina a profundidade do volume de saída;

- **Passada** (*stride*): porção de passos a serem deslizados pelo filtro. Em uma imagem, quando o *stride* é 1, move-se o filtro de pixel em pixel. Se o *stride* for 2 (ou 3, apesar de ser raro), então o filtro pula 2 pixeis cada vez que o filtro é deslizado. Isso produz volumes de saída menores espacialmente;

- **Preenchimento com zeros** (*zero-padding*): ato de preencher as bordas do sinal de entrada com zeros. A vantagem do *zero-padding* é permitir controlar o tamanho espacial dos volumes de saída, tendo seu uso geralmente focado para manter o tamanho dos volumes entre camadas convolucionais;

- **Extensão espacial**: tamanho do campo receptivo de um neurônio. Um neurônio de extensão espacial de tamanho 3 é aplicado sobre uma região de 3x3 pixeis em uma imagem, por exemplo.

2.2.3 Padrões de organização de camadas em uma CNN

Um dos pontos chave na hora de arquiteturar uma CNN é definir a organização de suas camadas. A ordem com que elas são dispostas dentro da rede afeta muito a capacidade de aprendizado, e a quantidade de camadas aliada ao número de parâmetros por filtro implicam no desempenho em relação ao processamento e memória. Segundo Karpathy (2014), a forma mais comum de arquitetura em uma CNN consiste em empilhar

algumas camadas CONV e RELU, seguidas por camadas de POOL, e repetir esse padrão até que o sinal de entrada seja mesclado espacialmente para um tamanho pequeno. Em algum ponto da rede, é comum transitar para camadas FC. A última camada FC contém a saída, como, por exemplo, as classificações de tipos de objetos presentes em imagens. Em outras palavras, o estilo mais comum de CNNs segue o padrão:

$$\text{INPUT} \rightarrow [[\text{CONV} \rightarrow \text{RELU}] * N \rightarrow \text{POOL}?] * M \rightarrow [\text{FC} \rightarrow \text{RELU}] * K \rightarrow \text{FC}$$

Onde $*$ indica repetição, e POOL$?$ indica uma camada de agrupamento opcional. Também geralmente assume-se $N \geq 0$, $M \geq 0$ e $K \geq 0$ (e usualmente $N \leq 3$ e $K \leq 3$). Empilhar algumas camadas CONV antes de camadas POOL geralmente constitue-se de uma boa ideia para redes largas e profundas, porque múltiplas camadas CONV empilhadas podem desenvolver características mais complexas do volume de entrada antes das operações de POOL, que possuem um caráter destrutivo (KARPATHY, 2014).

3 BIBLIOTECAS

Dadas as especificidades dos tipos de rede que comumente formam redes profundas, é válido salientar a importância da existência de ferramentas que auxiliem o trabalho do projetista na tarefa de criar e modelar sua rede neural. Realizando uma breve pesquisa sobre as ferramentas disponíveis, é possível encontrar uma variedade de bibliotecas que ofereçam esse serviço. Segundo Vasconcelos e Clua (2017), avanços em ferramentas e bibliotecas para o desenvolvimento de redes neurais profundas permitem a engenheiros, cientistas e entusiastas explorar soluções para diferentes aplicações na área de Aprendizado de Máquina, tais como classificação de imagens e vídeo, processamento natural de linguagem e reconhecimento de áudio. Tais bibliotecas permite que projetistas treinem, desenvolvam e testem redes neurais profundas tirando proveito de todo poder computacional proporcionado por GPUs e CPUs modernas. As bibliotecas listadas por este trabalho apresentam uma forma de interação onde o projetista não deve se preocupar com etapas de baixo nível de programação e otimização computacional, fornecendo um nível de abstração que o permite focar especificamente na modelagem da rede em si.

Este capítulo aborda duas bibliotecas de DL vastamente difundidas pelas comunidades de pesquisa: Caffe (Seção 3.1) e TensorFlow (Seção 3.2). Ambas as bibliotecas são *software* livre, isto é, podem ser executadas, copiadas, modificadas e redistribuídas pelos seus usuários gratuitamente, além de possuírem livre acesso aos seus códigos-fonte. Este capítulo focará em características das bibliotecas como especificidades das implementações, adaptabilidade a diferente tipos de redes neurais, modelos de representação e suporte a ambientes computacionais heterogêneos. Ao final do capítulo, a Seção 3.3 apresenta alguns trabalhos relacionados encontrados na literatura.

3.1 CAFFE

Caffe é um *framework* de aprendizado profundo desenvolvido pelo *Berkeley Vision and Learning Center* (BVLC) de código aberto, escrito em C++, com a capacidade de realizar computações em GPU usando CUDA. Jia et al. (2014) denota que Caffe provê a capacidade para o projetista treinar, testar, otimizar e publicar modelos, com exemplos bem documentados para todas essas tarefas. Apresenta-se como uma opção para pesquisadores e demais desenvolvedores que buscam o estado da arte do aprendizado de máquina, devido modularidade de seu código e a separação alegadamente limpa entre implementação e definição de uma rede, enquanto que ao mesmo tempo, apresenta-se como uma das implementações mais rápidas disponíveis para algoritmos de DL, tornando-a atrativa para a indústria ao alcançar velocidades de processamento de mais de 40 milhões de imagens

por dia em uma única GPU NVIDIA K40 ou Titan (JIA et al., 2014).

As subseções a seguir abordam detalhes sobre as características principais da biblioteca (Seção 3.1.1), sua arquitetura (Seção 3.1.2), suas diferentes aplicações em projetos (Seção 3.1.3) e bibliotecas de álgebra linear das quais fornece suporte (Seção 3.2.3).

3.1.1 Características

Segundo Jia et al. (2014), as principais características de Caffe que podem ser listadas são:

- **Modularidade**: Caffe busca ser bastante modular, permitindo fácil extensibilidade para novos formatos de dados, camadas de rede e funções de erro;
- **Separação de representação e implementação**: A definição dos modelos empregada em Caffe faz uso da linguagem *Protocol Buffer*[1], da Google, possibilitando a modelagem de arquiteturas de redes na forma de grafos acíclicos dirigidos. Caffe abstrai do projetista a alocação, quantidade e localização da memória consumida pela rede neural, permitindo, por exemplo, que o usuário troque a unidade de processamento principal (CPU ou GPU) com apenas uma chamada de função – todas as transferências de dados necessárias são realizadas de forma implícita;
- **Cobertura de testes**: Cada módulo de Caffe tem um teste associado, e nenhum módulo novo é aceito no projeto sem seu respectivo teste;
- **Ligações para Python e MATLAB**: Caffe provê oficialmente ligações para Python e MATLAB, possibilitando uma rápida prototipagem e integração com código de pesquisa pré-existente. Ambas as linguagens podem ser utilizadas para construir redes e classificar entradas;
- **Modelos de referência pré-treinados**: Caffe disponibiliza em seu *Model Zoo*[2] modelos de referência para tarefas visuais, incluindo o aclamado modelo da ImageNet, AlexNet, com algumas variações e o modelo de detecção R-CNN (*Region-Convolutional Neural Network*). Esses modelos são aprendidos e aplicados para problemas que vão desde regressão simples, até classificação visual em larga escala, para redes Siamesas (arquiteturas de RNA que contém duas ou mais sub-redes idênticas) para similaridade de imagens, aplicações de fala e robótica.

[1] https://code.google.com/p/protobuf/
[2] https://github.com/BVLC/caffe/wiki/Model-Zoo

3.1.2 Arquitetura

A arquitetura do *framework* pode ser dividida principalmente nos seguinte componentes (JIA et al., 2014):

- **Armazenamento de dados**: Caffe armazena e comunica dados em conjuntos de 4 dimensões denominados *blobs*. Blobs provêm uma interface de memória unificada, carregando lotes de imagens, parâmetros e atualizações de parâmetros. Blobs sincronizam os dados entre CPU e GPU automaticamente, alocando memória dinamicamente conforme necessário. Modelos são salvos em disco como *Protocol Buffers*, que possuem características importantes: tamanho mínimo de strings, serialização eficiente, formato de texto humanamente legível compatível com as versões binárias e implementações de interface para diversas linguagens;

- **Camadas**: Caffe provê um conjunto com tipos de camadas variados, incluindo: convolução, agrupamento, produto interno, retificada-linear (RELU), operações elementares e camadas de perda como *softmax* e *hinge*. Cada camada possui duas responsabilidades: uma passagem para a frente, que recebe um ou mais *blobs* como entrada e produz um ou mais *blobs* como saída, e a passagem de retropropagação, que pega o gradiente com respeito a saída, e computa os gradientes com respeito aos parâmetros da camada e à entrada.

- **Modo de execução**: Caffe faz toda a contabilidade para qualquer gráfico acíclico dirigido de camadas, garantindo a corretude das passadas em ambos os sentidos. Uma rede típica começa com uma camada de dados que é carregada a partir do disco e termina com uma camada de perda que calcula o objetivo de uma tarefa, como classificação ou reconstrução. A rede é executada em CPU ou GPU, tendo as rotinas correspondentes pra cada unidade de processamento que produzem resultados idênticos. A alternância entre CPU/GPU é abstrata e independente da definição do modelo.

3.1.3 Aplicações

Caffe vem sendo utilizada por um grande número de projetos de pesquisa em diversas universidades e empresas. Abaixo, Jia et al. (2014) listam algumas das possíveis aplicações do *framework* em projetos:

- **Classificação de objetos**: A biblioteca possui um exemplo online[3] que mostra a classificação de objetos nas imagens fornecidas pelos usuários, inclusive via celular, tentando classificá-las em uma das 1.000 categorias da ImageNet (Figura 3.1);

[3] http://demo.caffe.berkeleyvision.org/

- **Aprendizagem de recursos semânticos**: Além do treinamento de ponta a ponta, a biblioteca também pode ser usada para extrair recursos semânticos de imagens usando uma rede pré-treinada. Esses recursos podem ser usados em outras tarefas de visão com grande sucesso, sendo úteis para muito mais do que categorização de objetos, como por exemplo, classificação de estilos artísticos (JIA et al., 2014);
- **Detecção de objetos**: Caffe pode ser combinada com técnicas como pesquisa seletiva para realizar localização e reconhecimento simultâneos efetivamente em imagens naturais (JIA et al., 2014).

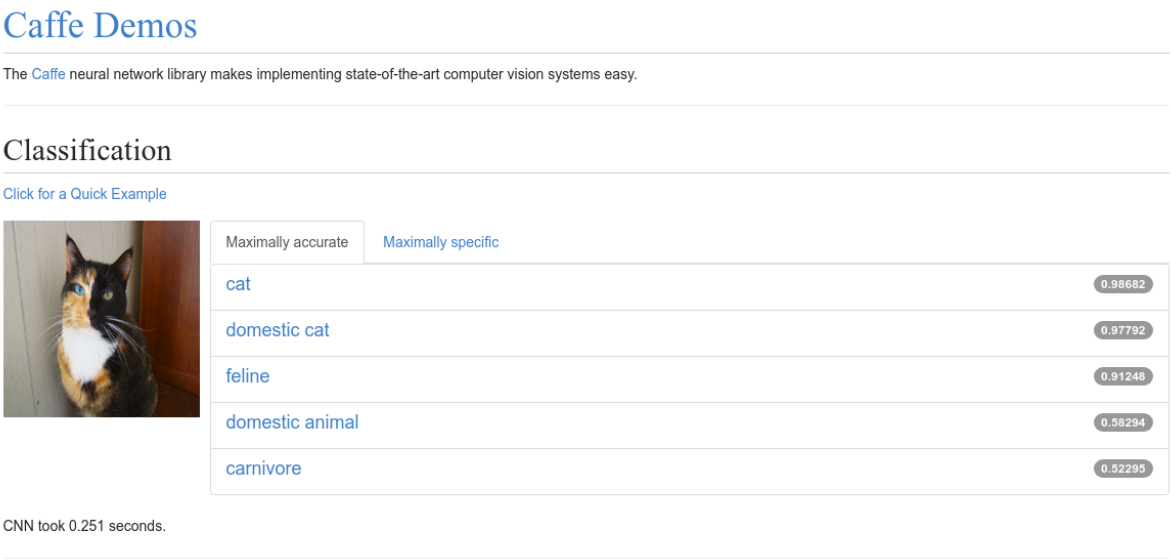

Figura 3.1 – Um exemplo da demonstração online de classificação de objetos com Caffe. Note como a rede avaliou a imagem fornecida como entrada com alta precisão: foram estimadas aproximadamente 98.7% de chances do objeto ser um gato, apesar da variação rara de raça presente na imagem.

3.1.4 Bibliotecas de Algebra Linear

Tanto CNNs quanto diversos outros tipos de redes neurais possuem uma característica de implementação em comum: a maior parte das computações realizadas durante as etapas de treinamento e validação são referentes a combinações lineares, ou mais precisamente no caso de CNNs, multiplicações de matrizes. Buscando oferecer uma solução para tal tarefa e tornar-se um pouco mais dinâmica, Caffe usa, por decisão de projeto, bibliotecas de álgebra linear de terceiros, que devem ser especificadas em seu processo de construção.

Caffe tem suporte à integração das seguintes bibliotecas de álgebra linear: OpenBLAS[4], Atlas[5] e Intel MKL[6] para uso em CPU, e cuBLAS[7] e OpenCL para uso em GPUs.

3.2 TENSORFLOW

Criado pela Google com base no seu sistema legado DistBelief, fruto do projeto Google Brain, TensorFlow é uma interface que pode ser utilizada para expressar e executar algoritmos de aprendizado de máquina, necessitando de pouca ou nenhuma alteração de código-fonte para execução em uma ampla variedade de sistemas heterogêneos, de dispositivos móveis a sistemas distribuídos em grande escala com milhares de GPUs. Segundo Abadi et al. (2016), sua flexibilidade permite que seja usado para projetar diversos algoritmos diferentes, incluindo modelos de redes neurais profundas e sistemas de aprendizagem de máquinas em produção em diversos campos como reconhecimento de fala, visão por computador, robótica, recuperação de informações, processamento de linguagem natural, extração de informações geográficas e descoberta computacional de medicamentos.

TensorFlow recebe computações descritas pelo uso de um modelo de grafo de fluxo de dados e mapeia-as em uma larga variedade de plataformas de *hardware*. Abadi et al. (2016) afirmam que possuir um sistema único que pode abranger uma ampla gama de plataformas simplifica significativamente o uso de sistemas de aprendizagem de máquina na realidade. Os cálculos de TensorFlow são expressos como grafos de fluxo de dados com estado, sendo o sistema suficientemente flexível para experimentações rápidas com modelos diferentes para fins de pesquisa e com alto desempenho e robustez para treinamento e implantação de algoritmos de aprendizado de máquina. Para dimensionar o treinamento das redes neurais para aplicações maiores, o TensorFlow permite que os projetistas expressem com facilidade vários tipos de paralelismo através da replicação e execução paralela do grafo central, onde muitos dispositivos computacionais diferentes podem colaborar para atualizar um conjunto de parâmetros compartilhados (ABADI et al., 2016).

3.2.1 Modelo de Programação

Como introduzido anteriormente, uma computação em TensorFlow é expressa por um grafo dirigido, o qual é composto por um conjunto de nodos. Esse grafo representa uma computação de fluxo de dados, com extensões permitindo alguns tipos de nós manterem e

[4] http://www.openblas.net/
[5] http://math-atlas.sourceforge.net/
[6] https://software.intel.com/en-us/mkl
[7] https://developer.nvidia.com/cublas

atualizarem estados persistentes e para controle de estruturas de ramificação e repetição dentro do grafo. Tais grafos podem ser projetados com o uso das linguagens oficialmente suportadas pela biblioteca (C++ e Python). Cada nodo possui zero ou mais entradas e zero ou mais saídas, e representa a instanciação de uma *operação*. Valores que fluem através de arestas no grafo são chamadas de *tensores* – vetores de dimensões arbitrárias inferidos no momento da construção do grafo. *Dependências de controle* são arestas especiais que não possuem fluxo de dados, mas indicam uma dependência entre o nodo de destino e o nodo de origem – um nodo não pode ser computado sem que o outro seja computado antes (ABADI et al., 2016).

- **Operações**: possuem um nome e representam uma computação abstraída (p.ex.: "multiplicação de matrizes", ou "soma"), além de poder possuir atributos. Uma operação com implementação particular para um tipo específico de arquitetura (CPU, GPU, etc.) é chamada de *kernel*;
- **Sessões**: Forma de programas clientes interagirem com a biblioteca. Disponibiliza o método `Run`, que permite que a sessão seja executada;
- **Variáveis**: Tipo de tensor que tem seus dados persistidos entre computações (ao contrário de tensores comuns). Para aplicações de aprendizagem em máquina, por exemplo, os parâmetros dos modelos são normalmente armazenados em variáveis.

3.2.2 Implementação

Segundo Abadi et al. (2016), os principais componentes em um sistema TensorFlow são o *cliente*, que usa a interface de sessão para se comunicar com o *mestre* e um ou mais *processos trabalhadores*, com cada trabalhador responsável por regular o acesso a um ou mais *dispositivos* computacionais (como threads de CPU ou placas GPU) e para executar nodos de grafos nesses dispositivos conforme instruído pelo mestre. TensorFlow possui tanto implementação local como distribuída, sendo a local usada quando o cliente, o mestre e o trabalhador são executados em uma única máquina no contexto de um único processo do sistema operacional. A implementação distribuída compartilha a maior parte do código com a implementação local, mas o estende com suporte para um ambiente em que todos os componentes possam estar em processos diferentes em máquinas diferentes. A Figura 3.2 mostra a estrutura dos sistemas local e distribuído.

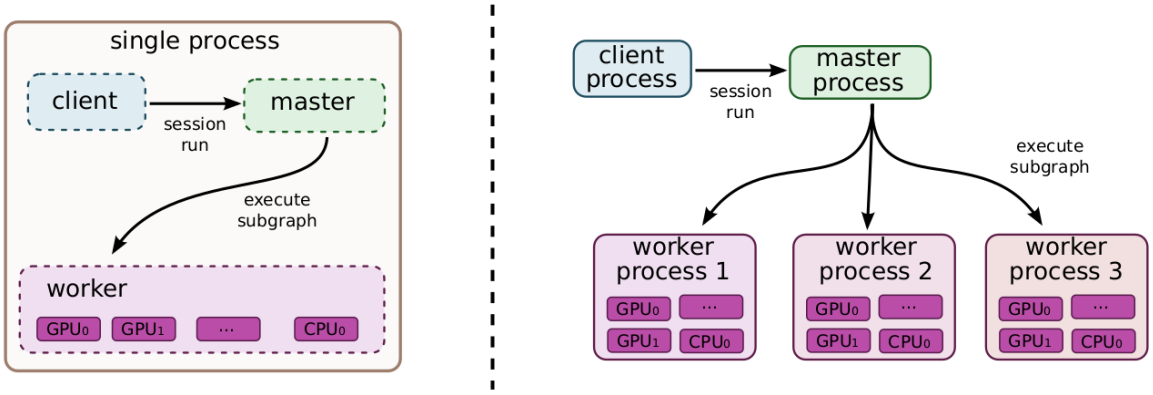

Figura 3.2 – Estrutura dos sistemas local e distribuído do TensorFlow. O algoritmo de posicionamento de nodos primeiro executa uma execução simulada do grafo, então escolhe um dispositivo para cada nodo usando heurísticas gulosas.

Fonte: Adaptado de Abadi et al. (2016)

3.2.3 Bibliotecas de Algebra Linear

Diferentemente da Caffe, TensorFlow traz em seu código o uso da biblioteca de álgebra linear Eigen[8] para computações em CPU. A biblioteca está profundamente integrada dentro do TensorFlow, tornando a tarefa de substituição por outra biblioteca de álgebra linear uma tarefa hercúlea, e não suportada oficialmente pela Google. A biblioteca é compilada com o TensorFlow, que por sua vez oferece a opção de compilá-la para que utilize instruções vetorizadas suportadas pela arquitetura da CPU, como instruções SSE e AVX. Para GPUs NVIDIA, TensorFlow também faz uso da biblioteca cuBLAS, assim como a Caffe.

A Tabela 3.1 apresenta um resumo das propriedades das duas bibliotecas de Deep Learning estudadas.

3.3 TRABALHOS RELACIONADOS

Alguns poucos trabalhos relacionados a avaliação de desempenho de bibliotecas de DL podem ser encontrados na literatura, dadas as datas recentes de lançamentos das mesmas. Praticamente todos sempre avaliam as duas bibliotecas avaliadas nesse trabalho: Caffe e TensorFlow, destacando assim suas popularidades. Diferentes propósitos podem ser encontrados, como a medição de desempenho em diversas arquiteturas diferentes e o consumo energético de cada biblioteca. É comum encontrar avaliações com versões antigas da biblioteca TensorFlow, uma vez que ela possui uma frequência relativa-

[8] http://eigen.tuxfamily.org

Tabela 3.1 – Propriedades das bibliotecas estudadas

Biblioteca	**Caffe**	**TensorFlow**
Desenvolvedor	BVLC	Google
Linguagem de Implementação	C++	C++
Interfaces Suportadas	C++, Python, Matlab	C++, Python
Paradigma de Programação	Declarativo e Imperativo	Imperativo
CPU	✓	✓
BLAS em CPU	OpenBLAS, Atlas, Intel MKL	Eigen
Múltiplos Nodos	✗	✓
GPU	✓	✓
BLAS em GPU	cuBLAS, OpenCL	cuBLAS
Múltiplas GPUs	✓	✓
Modelos Suportados	CNN, RNN, DBN, LNN	CNN, RNN, DBN, LNN, LSTM
Modelo de Sincronização	Síncrono	Síncrono ou Assíncrono
Modelo de Comunicação	✗	Servidor de Parâmetros
Tolerância à Falhas	*checkpoint-and-resume*	*checkpoint-and-recovery*

Fonte: Adaptado de Bahrampour et al. (2015) e Fox, Zou e Qiu (2016)

mente alta de lançamento de novas versões.

Shams et al. (2017) avaliam sobre diferentes arquiteturas de Computação de Alta Performance (HPC) o desempenho das bibliotecas Caffe, Apache SINGA e TensorFlow, tendo essa última sido avaliada com uma versão relativamente antiga (versão 0.12). Os autores atem-se a mensurar tempos gastos variando-se métricas como tamanhos de lotes, não buscando investigar os motivos referentes as diferenças em desempenho. As bibliotecas foram compiladas e submetidas à avaliação em ambientes com múltiplas GPUs – interligadas pelas tecnologias PCIe3 e NVLink – e ambientes com um ou mais nodos computacionais, com o objetivo de avaliar a escalabilidade das bibliotecas. Dentre os *hardwares* em uso destacados pelo trabalho estão a GPU NVIDIA Tesla P100 e a CPU Intel Xeon Phi Knights Landing (KNL), tendo esta última feito uso do recurso Omni-Path para comunicação entre múltiplos nodos. Cinco tipos diferentes de modelos de redes foram avaliados, recebendo maior destaque os modelos AlexNet, GoogLeNet e VGG-19. Após avaliar os resultados dos tempos obtidos pelas diferentes configurações de métricas, redes, bibliotecas e ambientes, Shams et al. (2017) avaliam que a biblioteca Caffe possui tempos de treinamento expressivamente menores em quase todos as métricas utilizadas, entretanto indica que para redes grandes como a VGG-19, a biblioteca TensorFlow consiga atingir tempos semelhantes à biblioteca Caffe. Todas as bibliotecas apresentaram bons resultados com relação à escalabilidade, apesar da biblioteca TensorFlow não apresentar bons resultados quando da utilização da CPU KNL em conjunto com Omni-Path. Quanto aos resultados em GPU, o uso da tecnologia NVLink apresentou resultados melhores que com PCIe3.

Seguindo uma linha semelhante a de Shams et al. (2017), entretanto de maneira mais aprofundada, Bahrampour et al. (2015) avaliam três aspectos – extensibilidade, uti-

lização de *hardware* e velocidade – de cinco diferentes bibliotecas de DL: Caffe, Neon, TensorFlow, Theano, e Torch, em um ambiente de execução com uma CPU (Intel Xeon CPU E5-1650 v2) e uma GPU (NVIDIA GeForce GTX Titan X). Diferentes tipos de redes foram avaliadas, não contando com somente CNNs: LeNet, AlexNet, LSTM e *autoencoders* empilhados. Dos resultados obtidos, Bahrampour et al. (2015) concluem que as bibliotecas Theano e Torch são as mais facilmente extensíveis, sendo Torch também a mais aconselhável para uso em CPU e também em GPU, atingindo melhores resultados que seus concorrentes, entretanto seguida de perto pela Theano. Bahrampour et al. (2015) também classificam Caffe como a biblioteca mais simples para as tarefas de modelagem e submissão de treinamentos. TensorFlow é novamente avaliada de forma negativa quanto a desempenho em comparação as demais, entretanto os autores também fizeram uso de uma versão antiga (versão 0.6.0) e provavelmente sem otimizações para a arquitetura da CPU, uma vez que não foi compilada nativamente, mas descarregada pelo *software* pip (gerenciador de módulos da linguagem Python).

Shi e Chu (2017) também avaliam o desempenho de bibliotecas de DL em diferentes ambientes de execução: com uma ou quatro GPUs e quatro nodos computacionais. As bibliotecas avaliadas foram as seguintes: Caffe-MPI (extensão para execução distribuída), CNTK, MXNet e TensorFlow (versão 1.2.1). Três modelos de rede foram utilizados na avaliação: AlexNet, GoogLeNet e ResNet-50, com tamanhos de lote fixos para cada uma. Os autores identificaram diferentes sobrecustos e gargalos que podem ser alvos de otimizações. O trabalho também oferece descrições dos diferentes algoritmos de otimização de gradiente utilizados, além de uma análise das formas de paralelismo empregadas pelas bibliotecas para a leitura dos cinjuntos de entrada, bem como o formato de armazanamento que cada biblioteca utiliza. Como resultado dos tempos obtidos com os treinamentos, Caffe-MPI novamente apresenta tempos menores em todos os ambientes para quase todas as redes utilizadas, além de apresentar melhores valores de escalabilidade tanto no uso de múltiplas GPUs quanto no uso de múltiplos nodos.

Seguindo uma linha diferente dos demais, Pena et al. (2016) avaliam a combinação de desempenho aliado a consumo energético do uso de bibliotecas de DL para treinamento de CNNs em arquiteturas embarcadas. As bibliotecas avaliadas foram Caffe e TensorFlow (versão 1.0.1), para as CNNs AlexNet, GoogLeNet, NiN, VGG_F e Cifar 10. Os testes foram conduzidos em três tipos diferentes de sistemas embarcados: USB Neural Compute Stick (NCS), Intel Joule 570X e Raspberry Pi 3 Model B. O tamanho de lote empregado foi 1, um número relativamente pequeno e geralmente não aconselhado para treinamentos reais devido a grande variância da perda através do treinamento. Como resultados, a biblioteca Caffe apresenta médias de consumo energético menores que TensorFlow – até 2.8 vezes no Raspberry Pi –, entretanto aliado a tempos de inferência maiores – até 4.7 vezes na Intel Joule.

4 METODOLOGIA

Este capítulo apresenta as atividades realizadas com o objetivo de planejar um ambiente e configurações de execução, além de escolher modelos de redes neurais para tal, de modo a medir a evolução da taxa de perda e tempos consumidos pelas redes em cada biblioteca, dependendo da variação de configuração de hiperparâmetros das mesmas. Serão apresentados os seguintes tópicos neste capítulo: a determinação dos hiperparâmetros e seleção dos modelos de redes candidatos (seção 4.1) e dos ambientes computacionais escolhidos para a execução (seção 4.2).

4.1 PLANEJAMENTO

Para atingir o objetivo estipulado pelo trabalho, foi necessário definir quais modelos de rede seriam usadas para avaliação, e quais hiperparâmetros sofreriam variações entre treinamentos. O objetivo das execuções era a extração de informações da rede como tempo gasto por iteração (ou imagem) e a taxa de perda entre essas iterações. Tais informações foram usadas para posterior análise de desempenho das bibliotecas conforme a rede e configuração.

A execução do treinamento e extração de informações dessas bibliotecas, entretanto, não são tarefas triviais: um determinado número de modificações em arquivos de modelagem (Caffe) ou arquivos de código (TensorFlow) são necessárias para tal, além da tarefa de filtrar as informações desejadas pelo trabalho dentre as muitas providas pelas bibliotecas. Com o intuito de automatizar essas tarefas, foram criados scripts na linguagem Python, customizados de acordo com as características dos sistemas de cada ambiente de execução. Tais scripts podem ser encontrados nos Anexos A, B e C

Para consultar consumo de memória dedicada e utilização da GPU, foram utilizadas ferramentas da NVIDIA para tal, como o *NVIDIA Profiler*[1] e o *NVIDIA System Management Interface*[2].

4.1.1 Modelos de Redes Neurais Convolucionais

Foram escolhidas duas CNNs, bem difundidas no campo de pesquisa de Deep Learning, para a etapa de avaliação das bibliotecas: AlexNet e GoogLeNet. Ambas as redes possuem implementações de seus modelos criados pelas equipes de desenvolvimento de

[1] http://docs.nvidia.com/cuda/profiler-users-guide/index.html
[2] https://developer.nvidia.com/nvidia-system-management-interface

ambas as bibliotecas[3][4], permitindo que possam ser treinadas e avaliadas por quem deseja estudar suas arquiteturas.

1. **AlexNet**: Criada por Krizhevsky, Sutskever e Hinton (2012), foi a primeira CNN a vencer o ILSVRC, no mesmo ano de sua criação, com uma taxa de erro de 15.3% (10% maior que a segunda colocada). Apesar de ser considerada uma rede profunda, possui um número reduzido de camadas quando comparada com campeãs mais recentes do desafio (HE et al., 2015): a rede possui em torno de 60 milhões de parâmetros e 650 mil neurônios artificiais, dispostos em 11 camadas: 5 camadas de convolução, algumas seguidas por camadas de agrupamento (*max-pooling*) (3 ao total) e 3 camadas completamente conectadas ao final, classificando até 1000 categorias (Ilustração 4.1) (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

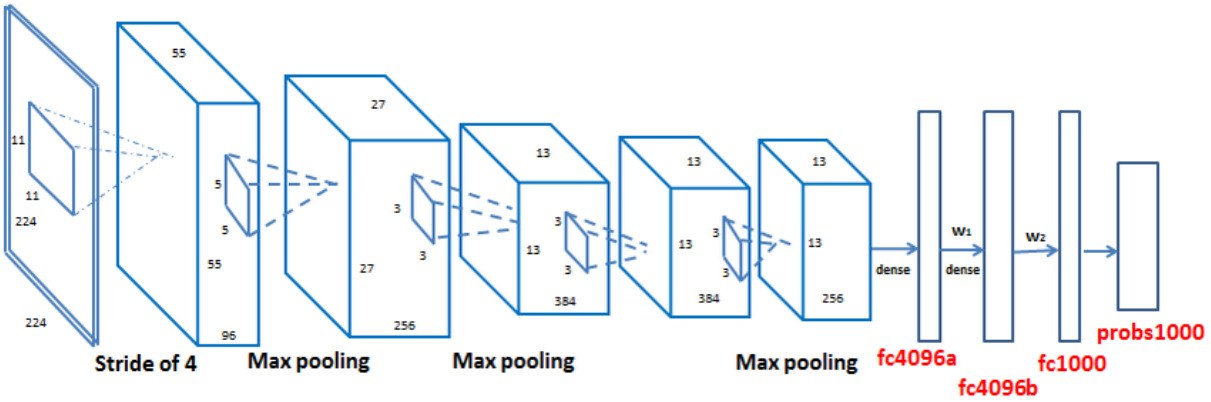

Ilustração 4.1 – Representação gráfica tridimensional da organização de camadas na rede AlexNet. Note a presença de operações de agrupamento (*max pooling*) entre as camadas: nos modelos utilizados, essas operações possuem camadas próprias.

Fonte: Retirado de (WANG et al., 2015)

2. **GoogLeNet (Inception)**: Vencedora do ILSVRC de 2014 com uma taxa de erro de 6.67% (56.5% menor que a AlexNet), a rede GoogLeNet fora desenvolvida por Szegedy et al. (2015), que à epoca trabalhavam no Google. Relativamente mais profunda que a AlexNet, a rede conta com 79 camadas (Figura 4.2), sendo dessas 63 convolucionais. A GoogLeNet fora projetada utilizando grupos de camadas com uma lógica de blocos encaixáveis, chamados 'Inceptions', onde grupos de camadas podem ser enfileirados no decorrer da rede. Esses grupos contém camadas de convolução com tamanhos de filtros 1x1, 3x3 e 5x5, buscando evitar problemas de alinhamento, e camadas de agrupamento sendo executadas em paralelo (SZEGEDY et al., 2015).

   A maior profundidade da rede acaba requisitando uma maior disponibilidade de recursos de memória para ambas as bibliotecas analisadas, que armazenam informa-

[3]https://github.com/BVLC/caffe/tree/master/models
[4]https://github.com/tensorflow/models/tree/master/research/slim/nets

ções entre as camadas da rede em memória (chamados de *blobs* para o Caffe e tensores para o TensorFlow).

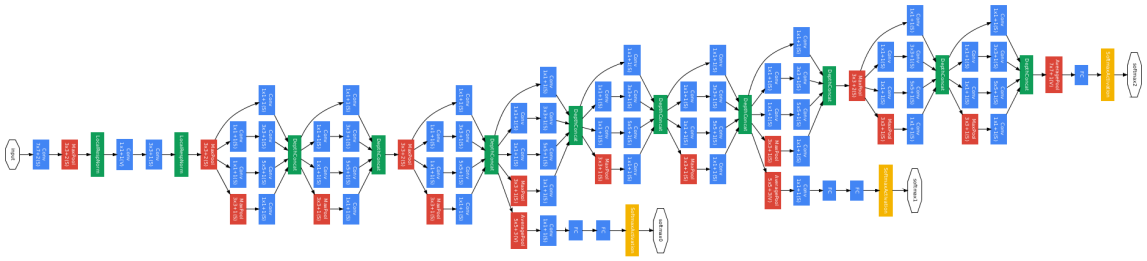

Ilustração 4.2 – Grafo acíclico dirigido apresentando a organização de camadas na rede GoogleNet. Perceba o padrão de grupos de camadas: os chamados módulos *Inception*.

Fonte: Retirado de (MARINO, 2017)

4.1.2 Configurações de Hiperparâmetros dos Modelos

Os seguintes hiperparâmetros tiveram valores especificamente adotados para os treinamentos:

- **Tamanho de lote**: Talvez o hiperparâmetro com maior influência sobre o resultado: lotes (ou *batches*, em inglês) correspondem a um subconjunto de imagens que serão fornecidas a rede por iteração. O tamanho do lote exerce influência sobre o tempo total demandado pelo treinamento, sobre o uso total de memória necessário para armazenamento dos dados intermediários entre as camadas, e sobre como a taxa de perda varia no decorrer do treinamento: tamanhos menores de lotes podem acarretar uma variância maior nos índices de perda através das iterações, uma vez que determinadas iterações podem conter lotes de imagens dos quais a rede não consiga aprender muito em comparação com outros (SEIDE; LI; YU, 2011).

  Os treinamentos foram executados com variações no tamanho dos lotes: variando, em potências de 2, de 1 a 512. Dependendo do ambiente de execução, o limite superior foi menor, pois contava com menos memória para armazenamentos dos *blobs* do Caffe. A relação de limite máximo com ambiente usado pode ser encontrada na seção 4.2;

- **Épocas**: Uma época corresponde a um intervalo de iterações em que todas as imagens do conjunto de entrada tenham passado uma vez pela rede. Para diminuir o

tempo necessário para execução, este trabalho adotou o uso de 10 épocas de treinamento.

- **Iterações**: O número de iterações máximo depende diretamente do tamanho do lote e da quantidade de épocas. A fórmula utilizada para determinar o número de iterações é a seguinte:

$$\text{ITER} = (\text{INPUT}/\text{BATCH}) * \text{EPOCHS}$$

  Onde ITER corresponde ao total de iterações necessárias, INPUT ao total de imagens no conjunto de entrada, BATCH ao tamanho do lote empregado e EPOCHS ao número de épocas que serão percorridas.

- **Intervalo para informações**: Este é um parâmetro específico das bibliotecas – responsável por determinar a cada quantas iterações informações sobre a rede são exibidas. Para que fosse recolhido o máximo de informações possível, o valor fora definido como 1 (exibição a cada 1 iteração);

- **Unidade de Processamento (Modo)**: Hiperparâmetro responsável por determinar em qual unidade de processamento a rede teria seu espaço de memória necessário alocado e seria treinada. As opções disponibilizadas pelos ambientes de execução e bibliotecas foram 2: CPU e GPU. A diferença de desempenho entre cada unidade é relevante para o objetivo do trabalho: determinar o quão bem cada biblioteca realiza sua tarefa nas mesmas unidades de processamento.

4.1.3 Conjunto de Dados de Entrada

Para a realização dos treinamentos das CNNs faz-se necessário um conjunto de imagens para alimentar as redes. Optou-se por usar o ImageNet, conjunto de dados conhecido por ser empregado na competição que ambas as redes utilizadas neste trabalho foram vencedoras. Entretanto, tal conjunto excede as capacidades de armazenamento dos ambientes de execução utilizado pelo trabalho – possui cerca de 14 milhões de imagens divididas em 22 mil categorias[5], chegando a necessitar em torno de 500 GB de espaço de armazenamento –, fator que torna seu uso inviável (confira a seção 4.2 abaixo). Para compensar esse problema, optou-se por usar um subconjunto bem reduzido do conjunto original: 1024 imagens de apenas uma categoria (Figura 4.1), o que pode ser considerado uma quantidade relativamente pequena.

[5] http://image-net.org/about-stats

Figura 4.1 – Algumas imagens do conjunto de dados empregado. O conjunto utilizado possui 1024 imagens diferentes contendo peixes Tinca Tinca – peixe bastante utilizado para aquacultura na europa.

O tamanho e a diversidade do conjunto também influenciam consideravelmente no quanto a rede consegue aprender: fornecer um número reduzido de imagens classificadas em um número pequeno de categorias faz com que a rede aprenda muito rápido – seu tamanho reduzido implica em pouco conteúdo a ser aprendido, o que por consequência pode levar ao problema de sobreajuste. Parte deste trabalho consiste em avaliar também como a biblioteca auxilia a rede a diminuir a taxa de redução da perda que leva inevitavelmente ao sobreajuste – uma vez que as definições das redes confiam completamente na implementação das camadas fornecidas pelas APIs das bibliotecas – o que indica uma implementação eficiente na etapa de retropropagação.

4.2 AMBIENTES DE EXECUÇÃO

Os ambientes escolhidos para a avaliação das bibliotecas possuem algumas peculiaridades em comum, como o fato de serem arquiteturas heterogêneas, dispondo por vezes de uma GPU ou um coprocessador. A Tabela 4.1 apresenta esses ambientes em maiores detalhes, nos quesitos *hardware* e *software*.

As versões empregadas do TensorFlow para os ambientes tx1 e lsc5 e o Caffe para a tx1 foram compiladas nativamente, enquanto a versão do Caffe para a lsc5 fora descarregada do repositório de aplicações de seu sistema operacional[6].

O ambiente colfax, que consiste em um cluster de 64 nós contendo processadores Xeon Phi[7], configura um ambiente no qual compilações que dependam de bibliotecas não instaladas previamente não possam ser realizadas. Entretanto, o ambiente possuía sua própria versão da biblioteca Caffe – configurada com a biblioteca Intel MKL e compilada com o compilador C++ da Intel[8] –, e para o TensorFlow fora descarregado e instalado um

[6] https://packages.debian.org/buster/caffe-cuda
[7] https://colfaxresearch.com/about/
[8] https://software.intel.com/en-us/c-compilers

wheel pré-compilado e alegadamente otimizada pela própria Intel[9].

Tabela 4.1 – Ambientes de Execução

Requisito	**Ambiente**		
	lsc5	**tx1**	**colfax**
Modelo CPU	Intel(R) Xeon E5620	Quad ARM(R) A57	Intel(R) Xeon Phi
Arquitetura CPU	x86_64	ARM v7	x86_64
Núcleos CPU	8	4	272
Frequência dos Núcleos CPU	2.4 GHz	1.73 GHz	1.4 GHz
Modelo GPU	NVIDIA GTX Titan X	NVIDIA Maxwell	-
Arquitetura GPU	Pascal	Maxwell	-
Núcleos GPU (CUDA)	3072	256	-
Frequência dos Núcleos GPU	1000 MHz	998 MHz	-
Memória Dedicada GPU	12 GB GDDR5	(compartilhada)	-
Memória RAM	11 GB DDR3	4 GB LPDDR4	94 GB DDR3
Memória swap	14 GB	6 GB	4 GB
Sistema Operacional	Debian Testing	Ubuntu 14.04	Red Hat 4.8.5
Caffe	1.0.0	1.0.0	1.0.0-rc3 (Intel)
TensorFlow	1.4.0	1.3.0	1.3.0

[9]https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture

5 RESULTADOS E DISCUSSÃO

Dada a metodologia a ser empregada e os ambientes a serem utilizados, uma série de execuções fora realizada com a finalidade de gerar dados a serem avaliados conforme suas características. Este capítulo aborda a evolução da perda até o sobreajuste nas diferentes combinações de redes e bibliotecas (Seção 5.1), e os tempos gastos por imagem em diferentes tamanhos de lotes e diferentes unidades de processamento, com a aceleração obtida pelos mesmos com o uso efetivo das arquiteturas híbridas (Seção 5.2).

5.1 EVOLUÇÃO DA PERDA

Esta seção apresenta gráficos que exibem a evolução da perda até um visível sobreajuste nas diferentes configurações de execução, com variações de ambiente, biblioteca, rede, unidade de processamento e tamanho de lote. As Seções 5.1.1, 5.1.2 e 5.1.3 apresentam os valores obtidos nos ambientes lsc5, tx1 e colfax, respectivamente.

Os gráficos apresentados possuem algumas similaridades:

- Os nomes dos rótulos de cada dado e suas formatações – adotou-se o seguinte padrão de nomenclatura: `PU (BATCH)`, onde `PU` representa a unidade de processamento principal empregada (CPU ou GPU), e `BATCH` representa o tamanho de lote utilizado (de 1 a 512);
- As linhas referentes às perdas em CPU se diferem visualmente das linhas referentes às perdas em GPU: para valores de CPU, as linhas apresentam formato tracejado e pontos com formas geométricas não preenchidas; para GPU, as linhas são inteiriças e mais largas, além de contar com pontos em formas geométricas completamente ou parcialmente preenchidas. Ambos os estilos apresentam exceções nas formas geométricas para valores referentes a tamanho de lote 1;
- O eixo horizontal – correspondente a quantidade de épocas de treinamento – está em escala de potência de 10. Tal configuração destaca o início da primeira época, onde praticamente toda queda no valor das perdas ocorre, antes da rede atingir um sobreajuste;
- Algumas combinações de unidade de processamento e tamanho de lote não estão presentes em alguns dos gráficos. Isso se deve exclusivamente ao fato da configuração, em combinação com a biblioteca empregada, exceder o limite de memória disponível para uso na unidade de processamento do ambiente escolhido;

- Algumas combinações apresentam interrupção prematura de suas linhas de valores no decorrer dos gráficos. Tal característica se deve exclusivamente ao fato da rede ter seu treinamento interrompido manualmente por demandar muito tempo para sua conclusão.

5.1.1 Ambiente de Execução lsc5

O ambiente lsc5 possui como principal característica a presença de uma GPU relativamente potente com elevada quantidade de memória dedicada. Tal fator em suma deve fazer prevalescer em questão de tempo e aceleração os treinamentos executados em GPU, porém, nada diz a respeito da evolução da perda: nesse caso, o diferencial é obtido exclusivamente através da forma como as partes de inferência e retropropagação foram implementadas para posterior uso via API da biblioteca escolhida. Os Gráficos 5.1 e 5.2 apresentam respectivamente a evolução da perda – até o inevitável sobreajuste – nas bibliotecas Caffe e TensorFlow.

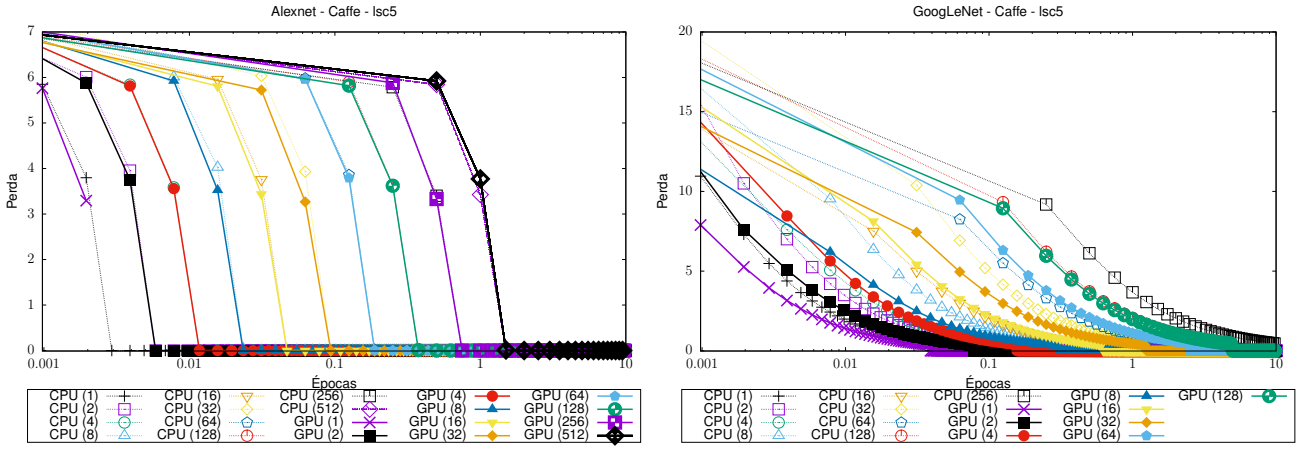

Gráfico 5.1 – Evolução da perda com a biblioteca Caffe no ambiente lsc5. Na esquerda, a perda com o modelo de rede AlexNet, e na direita com o GoogLeNet.

A partir dos gráficos acima é possível chegar as seguintes constatações:

- O modelo da rede GoogLeNet apresenta uma taxa de diminuição da perda mais compassada que a AlexNet, e inversamente proporcional ao tamanho do lote – quanto maior o lote, menor a redução da perda através das iterações. Tal fator é parte do motivo da rede ter alcançado uma maior taxa de precisão no ILSVRC;

- Com a biblioteca TensorFlow, a rede GoogLeNet não conseguiu completar suas execuções em tamanhos de lote iguais a 512. Esse fato se deve exclusivamente ao fato da rede ser mais profunda e necessitar de uma quantidade excessiva de memória

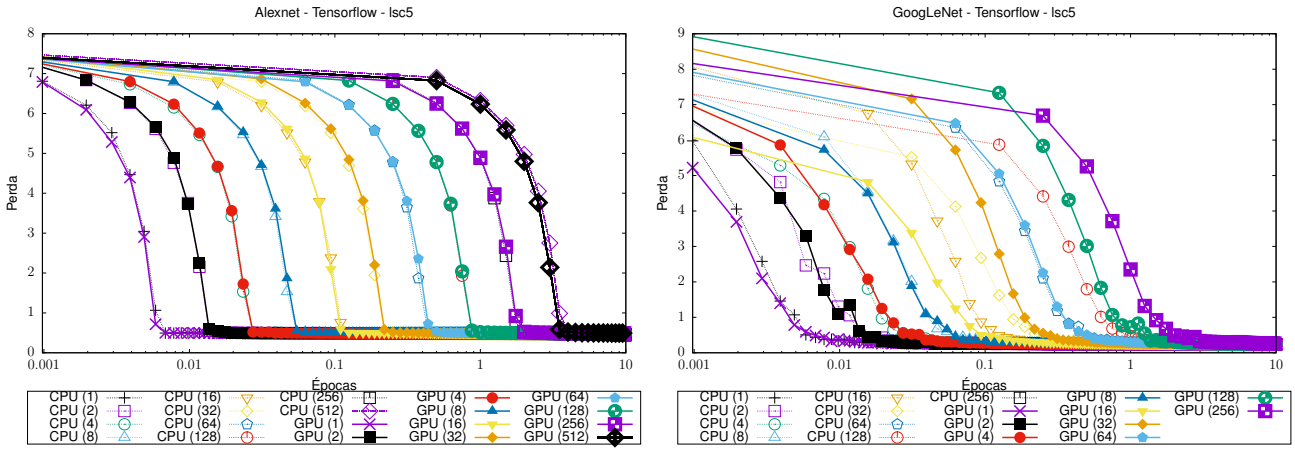

Gráfico 5.2 – Evolução da perda com a biblioteca TensorFlow no ambiente lsc5. Na esquerda, a perda com o modelo de rede AlexNet, e na direita com o GoogLeNet.

para o armazenamento dos tensores do TensorFlow, ultrapassando o limite disponível no ambiente, tanto para CPU quanto para GPU;

- Com a biblioteca Caffe, a rede GoogLeNet não conseguiu completar suas execuções em tamanhos de lote maiores ou iguais a 256, dependendo da unidade de processamento. Tal característica ressalta uma maior necessidade de memória disponível para a biblioteca em comparação com a TensorFlow. Em contrapartida, a taxa de redução de perda para a rede foi relativamente menor que a da biblioteca concorrente. O mesmo fator, entretanto, não se observa para a rede Alexnet, tendo a TensorFlow menores taxas de redução, além de uma relativa continuidade suavizada, não obtendo quedas bruscas de perda;

- A biblioteca Caffe apresenta um problema grave de sobreajuste: quando a perda converge, os valores ficam negativos, em todos os casos para ambas as redes.

- A diferença de tamanho dos lotes não implica em diferenças na evolução da taxa de perda como um todo, graças ao fato do conjunto de entrada ser pequeno e constituído de apenas uma categoria, minimizando os ruídos que seriam percebidos pela rede devido à existência de mais classes que divergem entre si. As diferenças percebidas em épocas para a convergência de lote para lote se deve então ao tamanho do lote e não à sua influência, denotando como as configurações geralmente necessitam de aproximadamente a mesma quantidade de iterações para convergirem, porém, com parcela de tamanho em relação à épocas diferentes, devido ao tamanho do lote.

  Dada tal constatação, formulou-se a hipótese que tamanhos maiores de lote são benéficos para as redes em geral, uma vez que elas podem aprender um maior número de imagens com uma taxa de perda menos agressiva.

5.1.2 Ambiente de Execução tx1

O ambiente tx1 constitui-se basicamente de módulo NVIDIA Jetson TX1[1], que possui como principal característica a presença de uma GPU relativamente potente para um sistema embarcado, oferecendo economia de energia aliada à capacidade razoável de processamento. O tx1 também conta com um sistema de memória unificada, como pode ser constatado na Tabela 4.1. Os Gráficos 5.3 apresentam respectivamente a evolução da perda na biblioteca Caffe.

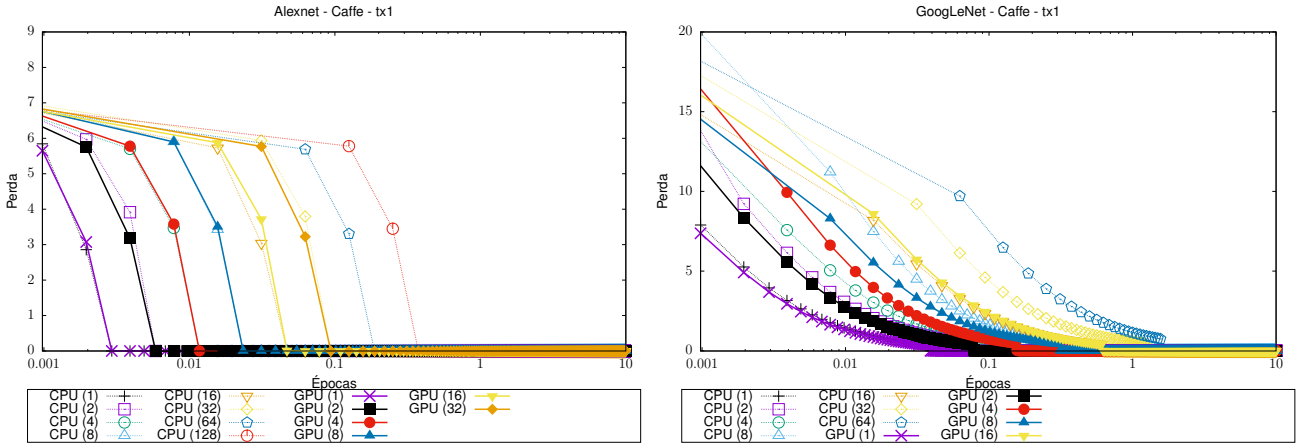

Gráfico 5.3 – Evolução da perda com a biblioteca Caffe no ambiente tx1. Na esquerda, a perda com o modelo de rede AlexNet, e na direita com o GoogLeNet.

As evoluções das perdas no ambiente tx1 foram praticamente as mesmas obtidas na lsc5, o que é esperado: independente das configurações de *hardware*, a biblioteca apresentou o mesmo comportamento para entradas idênticas. Uma característica visível da tx1, por outro lado, é a não-execução de algumas configurações de rede:

- Para a AlexNet, a Caffe conseguiu realizar o treinamento com lotes de até 32 imagens em GPU. Em CPU, o número foi estendido até 128. Apesar do sistema de memória ser unificado, execuções em CPU podiam fazer proveito do uso da memória SWAP, enquanto que a GPU não oferecia suporte para uso de memória além dos 4 GB providos pela memória unificada. Esse uso da SWAP, entretanto, trouxe problemas de desempenho, como pode ser conferido na Seção 5.2;

- A GoogLeNet em combinação com o Caffe, repetindo o padrão visto na lsc5 de consumo superior de memória, não conseguiu executar treinamentos com lotes maiores que 16 para GPU e 64 para CPU. Entretanto, assim como na AlexNet, a capacidade extra de armazenamento provida pela memória SWAP à versão com CPU trouxe custos de desempenho.

[1] http://www.nvidia.com.br/object/embedded-systems-br.html

- Ambas as redes, quando usando a biblioteca Caffe, apresentaram a convergência para um valor negativo de perda, assim como ocorrido na lsc5;
- Imprevistos nas configurações dos hiperparâmetros aliados a problemas como erros na alocação de memória em GPU acarretaram em execuções não válidas das redes na biblioteca TensorFlow, e por esse motivo não estão presentes neste trabalho.

5.1.3 Ambiente de Execução colfax

As execuções no ambiente colfax também não trouxeram grandes surpresas quando comparados aos demais ambientes. Compensando a ausência de uma GPU nesse ambiente, todos as execuções em CPU fizeram bom proveito dos 94 GB de memória disponível, conseguindo executar todos os treinamentos com todos os tamanhos de batch. Os Gráficos 5.4 e 5.5 apresentam respectivamente a evolução da perda nas bibliotecas Caffe e TensorFlow.

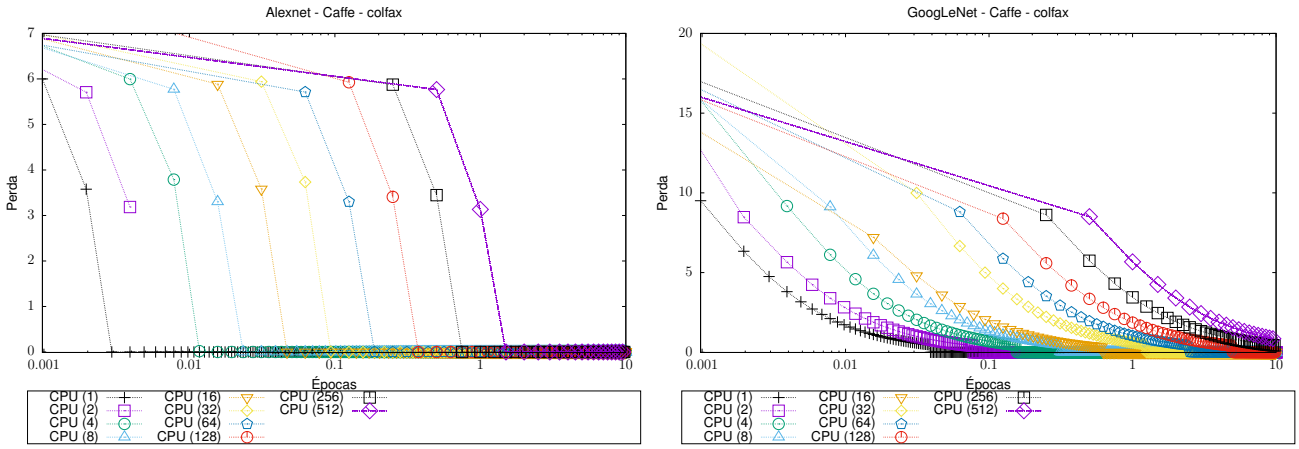

Gráfico 5.4 – Evolução da perda com a biblioteca Caffe no ambiente colfax. Na esquerda, a perda com o modelo de rede AlexNet, e na direita com o GoogLeNet.

5.2 TEMPOS E ACELERAÇÃO

O uso de um acelerador como uma GPU sempre é recomendado para a realização de computações massivas e intensivas: suas arquiteturas com centenas ou milhares de núcleos simples possibilitam acelerar em muitas vezes uma aplicação. Para CNNs isso não é diferente: em torno de 90% do tempo computacional gasto é com camadas convolucionais nas redes utilizadas neste trabalho (DAULTANI; OHNO; ISHIZAKA, 2017).

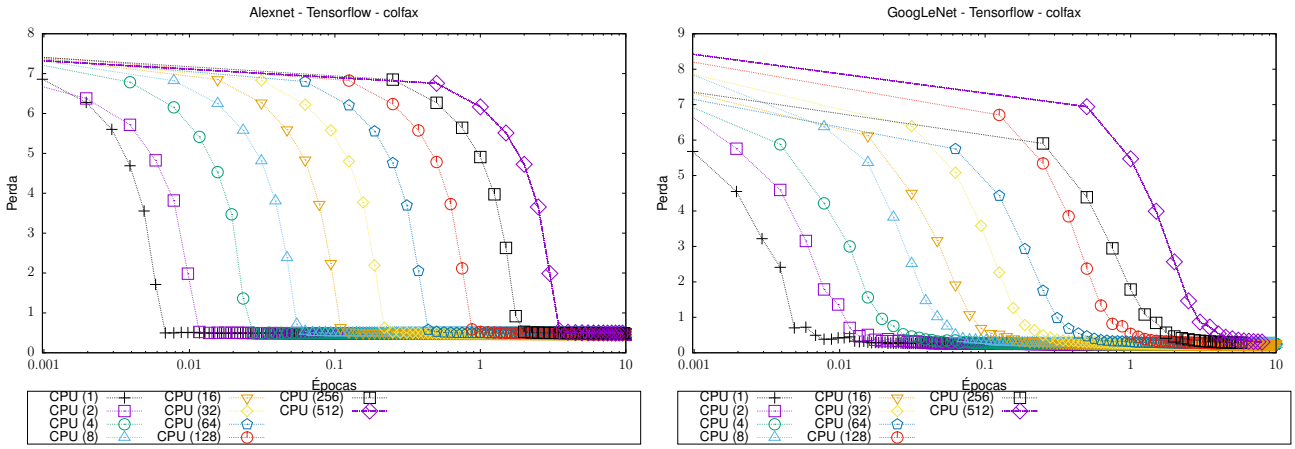

Gráfico 5.5 – Evolução da perda com a biblioteca TensorFlow no ambiente colfax. Na esquerda, a perda com o modelo de rede AlexNet, e na direita com o GoogLeNet.

Como a aplicação de filtros convolucionais pode ser resumido computacionalmente a produtos vetoriais sobre o sinal de entrada, ou mais especificamente multiplicações de matrizes em imagens, essas multiplicações de matrizes podem ser (e são) realizadas individualmente em paralelo buscando extrair o máximo de poder computacional que as bibliotecas de álgebra linear utilizadas pelo Caffe e pelo TensorFlow conseguem. Dadas as diferentes configurações de unidades de processamento dos ambientes empregados, com números de núcleos distintos (confira Tabela 4.1), é cabível mensurar quão rápido as bibliotecas conseguem deixar o treinamento dessas redes, e quantas vezes consegue-se acelerar essas execuções com o uso das GPUs disponíveis, com relação aos CPUs.

O Gráfico 5.6, referente ao ambiente lsc5, exibe o tempo médio por imagem gasto pelas redes, em combinação com as diferentes bibliotecas utilizadas, variando de acordo com o tamanho do lote utilizado, e a aceleração obtida pelo uso de GPU em comparação com o uso de CPU.

Com esses gráficos é possível visualizar a superioridade em poder computacional da GPU (Titan X) com relação a CPU (Xeon E5620). Algumas constatações podem ser realizadas:

- O tempo por imagem mantém-se praticamente constante em CPU, enquanto que em GPU ele apresentou diminuições conforme o tamanho do lote aumentava;
- Para praticamente todos as combinações de rede e unidade de processamento o TensorFlow apresentou tempos menores em maior parte dos tamanhos de lote. Algumas exceções podem ser observadas, como os tempos relativamente semelhantes para a rede AlexNet rodando em CPU;
- Nas execuções da rede GoogLeNet em CPU, para ambas as bibliotecas, é possível visualizar um aumento no tempo por imagem com um lote de tamanho 256, pois,

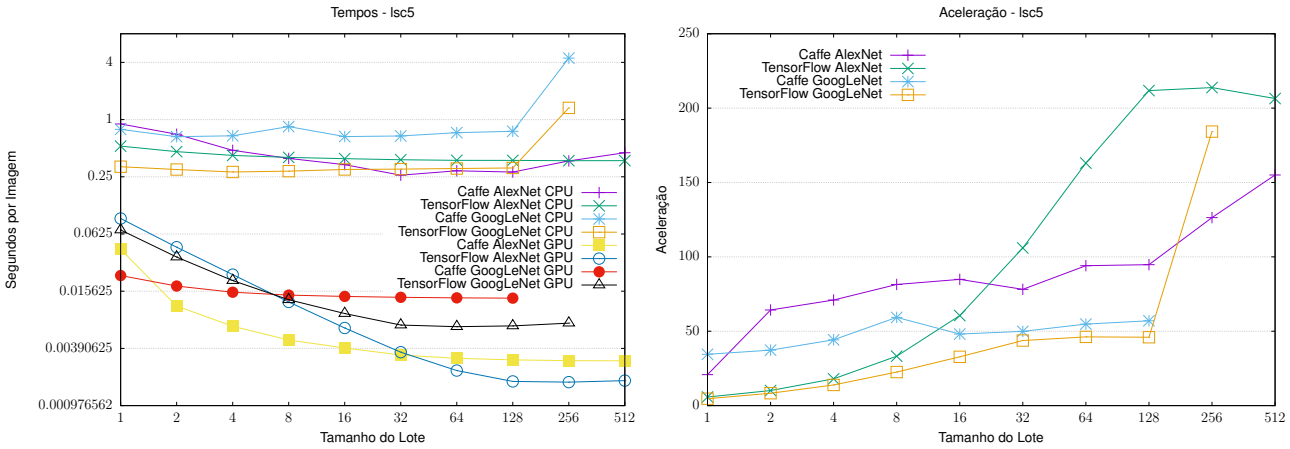

Gráfico 5.6 – Estatísticas de tempo de execução no ambiente lsc5. A esquerda, os tempos por imagem para todas as combinações de redes, bibliotecas e unidades de processamento. À direita, as acelerações em comparação ao uso de GPU contra o de CPU pelas combinações de redes e bibliotecas.

houve a necessidade das bibliotecas recorrerem à memória SWAP para armazenar os dados internos da rede, quadruplicando o tempo médio por imagem;

- Nas acelerações, a combinação de AlexNet com TensorFlow destaca-se das demais, apresentando uma curva em maior parte do tempo crescente e atingindo um pico onde se apresenta 200 vezes mais rápido que a versão com CPU, enquanto as demais tem alguns pontos de redução de aceleração em determinados tamanhos de lote, e dificilmente ultrapassam uma taxa de 150 vezes o tempo de execução, com exceção do caso abaixo;

- É possível visualizar uma aceleração acentuada para a combinação GoogLeNet com TensorFlow, graças ao fato da versão com CPU ter necessitado fazer uso da memória SWAP.

- Em suma, a biblioteca TensorFlow necessita de menos recursos de memória para armazenar os dados necessários para o treinamento. Esse fato fica parcialmente visível no gráfico, onde a biblioteca consegue realizar o treinamento com um lote de tamanho 256, enquanto a biblioteca Caffe chegou ao limite com um lote de tamanho 128. Entretanto, isso nada se refere à quantia de alocação de memória realizada pelas bibliotecas: a biblioteca TensorFlow aloca por padrão o máximo de memória possível em GPU, mesmo sem necessitar de todo o espaço para o armazenamento de seus tensores. Tal fato pode ser melhor visualizado no Gráfico 5.7.

  Essa alocação se deve ao fato do TensorFlow possuir uma política padrão de alocar todo o espaço disponível em GPU, independente do dispositivo computacional a ser utilizado. Ou seja, por padrão, mesmo que a computação ocorra totalmente em CPU, a GPU terá toda sua memória alocada.

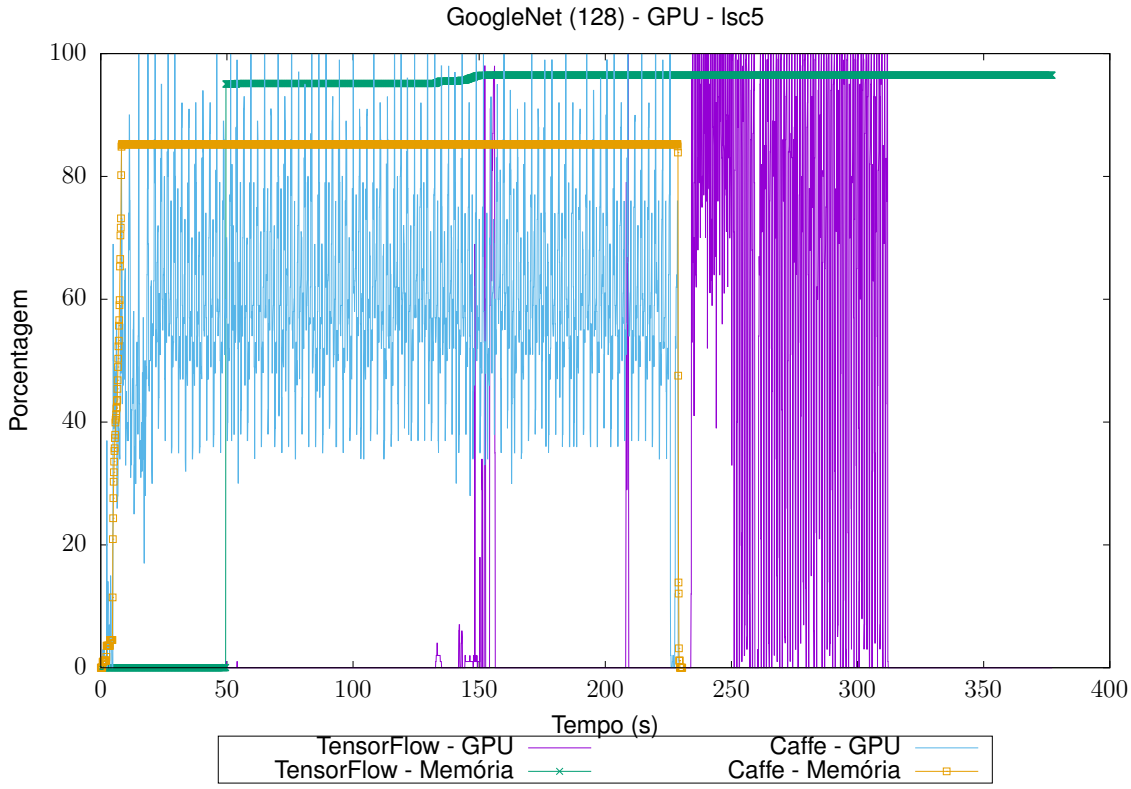

Gráfico 5.7 – Porcentagem de uso de GPU e memória dedicada no ambiente lsc5, para a rede GoogLeNet com lotes de 128 imagens. Os dados referentes foram extraídos no formato csv com a ferramenta *nvidia-smi*.

- O Gráfico 5.7 também fornece pistas de porque as execuções em GPU com TensorFlow são, em geral, mais rápidas que com Caffe. Apesar da biblioteca levar mais tempo durante sua execução, a maior parte do tempo envolve a preparação da rede, e o treinamento de fato ocorre em um intervalo de tempo muito reduzido em comparação ao Caffe. Outro ponto é que a biblioteca atinge, em uma frequência alta, picos de aproximadamente 100% de uso de GPU durante um curto espaço de tempo, enquanto a Caffe fica alternando entre 100 e 40%, com concentrações maiores entre 50 e 70%, em um espaço de tempo relativamente maior.

- Os Gráficos 5.8 e 5.9 sugerem que o espaço de tempo reduzido utilizado pela biblioteca TensorFlow se deve ao uso de multiplicações de matrizes (SGEMM) em lote (*Batched SGEMM*), habilidade fornecida pela biblioteca de algebra linear cuBLAS e que confere a habilidade de realizar múltiplas multiplicações de matrizes pequenas de uma vez só (CECKA, 2017), enquanto que a biblioteca Caffe faz uso de instruções SGEMM simples. A Tabela 5.1 exibe o número de chamadas e o tempo total gasto pelas instruções.

- O Gráfico 5.9 apresenta um gasto de quase 40% do tempo de execução para taxas

Tabela 5.1 – Tempos de execução e chamadas para cada método SGEMM utilizado pelas bibliotecas

Método SGEMM	Caffe		TensorFlow			
	Comum		Comum		Batched	
	Tempo (s)	Chamadas	Tempo (s)	Chamadas	Tempo (s)	Chamadas
128x64_nn	6,90060	75264	0,016219	21	0,14928	26
128x64_nt	4,02506	71680			0,089606	21
128x64_tn	5,02266	71701				
128x128_tn	1,22427	10282	0,18607	60		
128x128_nn	1,00165	10812	0,16892	42	0,31289	45
128x128_nt	0,20359	60	0,16529	60	0,30616	42
64x64_nt					0,39549	64
64x64_nn					0,11230	21

de transferência de dados entre memória RAM e memória dedicada. Em tempo absoluto, a biblioteca consumiu quase o dobro de segundos com transferência quando comparada a biblioteca Caffe. As razões para tal comportamento são motivos para investigações futuras;

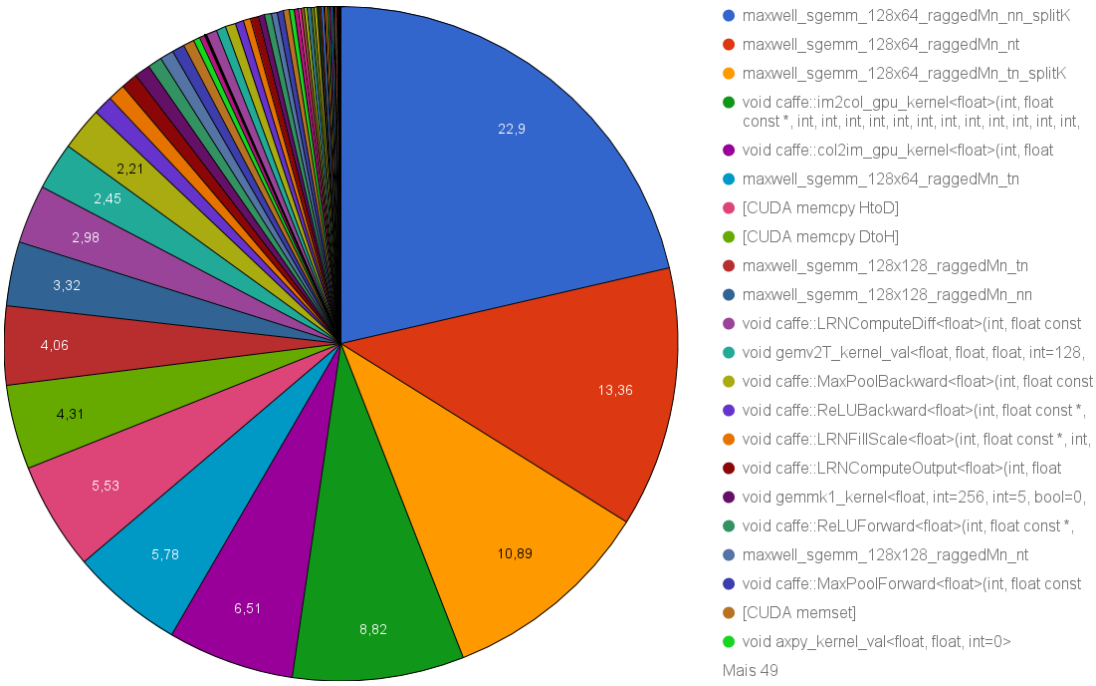

Gráfico 5.8 – Perfil de execução da biblioteca Caffe na GPU Titan X. A maior parte do tempo de execução se concentrou em operações SGEMM.

O Gráfico 5.10 a seguir exibe o tempo médio e a aceleração obtida pelo ambiente de execução embarcado tx1. Os resultados obtidos nesse ambiente novamente ressaltam a discrepância entre os tempos de execução entre CPU e GPU para treinamento de CNNs, mesmo tratando-se de um sistema embarcado onde a economia de energia é um dos focos principais. As seguintes constatações foram obtidas:

- Os tempos por imagem ficaram muito semelhantes aos obtidos pelo ambiente lsc5, com pouco menos que o dobro do tempo em GPU;

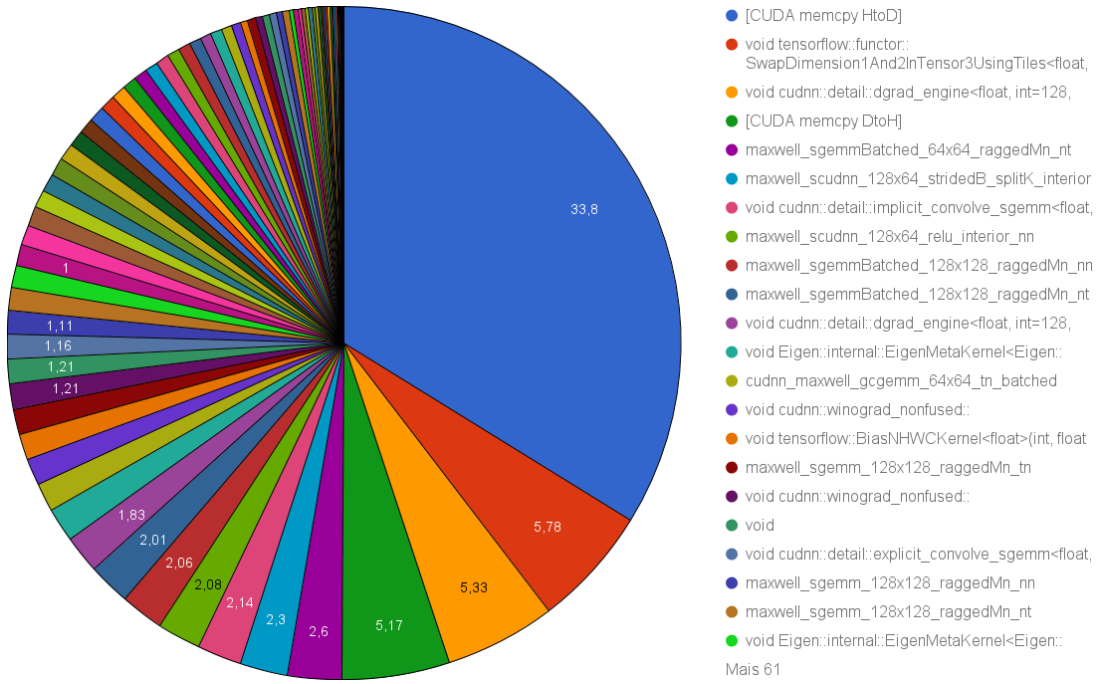

Gráfico 5.9 – Perfil de execução da biblioteca TensorFlow na GPU Titan X. A maior parte do tempo de execução se concentrou em operações de cópia de dados.

- O gráfico afirma novamente o impacto da baixa disponibilidade de memória no ambiente para o treinamento das redes: determinadas combinações não puderam ser executadas em alguns tamanhos de lote;
- Assim como ocorrido no ambiente lsc5, quando uma combinação necessitou recorrer à memória SWAP, teve seu tempo de execução comprometido: aumento em torno de 32 vezes para a rede GoogLeNet com tamanho de lote 64;
- O ambiente apresentou taxas moderadas de aceleração quando comparado à lsc5, entretanto, não diminui a importância da aceleração obtida, alcançando um fator de 24 para Alexnet usando Caffe e lotes de 32 imagens.

Por fim, o Gráfico 5.11 apresenta os tempos obtidos pela execução no ambiente colfax. O ambiente, dotado de uma CPU moderna e de bibliotecas otimizadas para a mesma apresentou desempenho superior em alguns casos a GPU Titan X no ambiente lsc5. O ambiente conseguiu atingir um tempo médio menor que 1 milissegundo por imagem na combinação da rede AlexNet com a biblioteca Caffe. O desempenho superior da biblioteca nesse caso se deve ao uso da biblioteca de álgebra linear Intel MKL, otimizada para aplicações de Deep Learning por incluir primitivas para tal (RODRIGUEZ, 2016). O sistema também conseguiu atingir valores menores que o ambiente lsc5 para a rede GoogLeNet com o Caffe.

Os tempos com TensorFlow, em contrapartida, apesar de reduzirem conforme o tamanho de lote aumenta, não são melhores que os obtidos pela Titan X. Parte da culpa pode

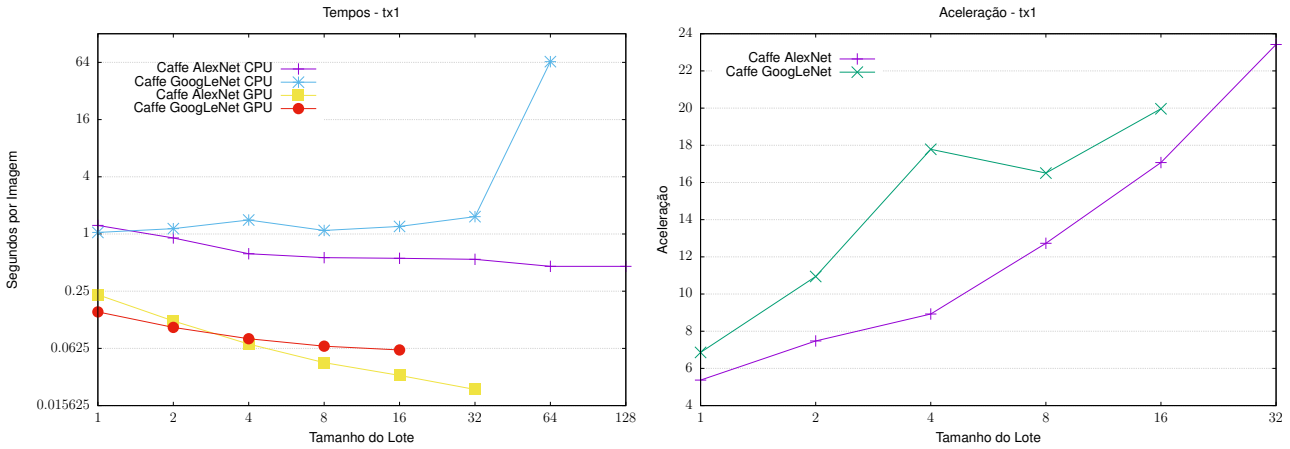

Gráfico 5.10 – Estatísticas de tempo de execução no ambiente tx1. A esquerda, os tempos por imagem para todas as combinações de redes, bibliotecas e unidades de processamento. À direita, as acelerações em comparação ao uso de GPU contra o de CPU pelas combinações de redes e bibliotecas.

se dever ao fato da biblioteca não ser realmente otimizada, ao contrário do informado[2]. No início da execução, a biblioteca TensorFlow emite um alerta sobre a não compatibilidade da biblioteca instalada com recursos de instruções vetorizadas como AVX e SSE, como na seguinte mensagem:

```
Your CPU supports instructions that this TensorFlow binary was not
compiled to use:  SSE4.1 SSE4.2 AVX AVX2 FMA
```

[2]https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture

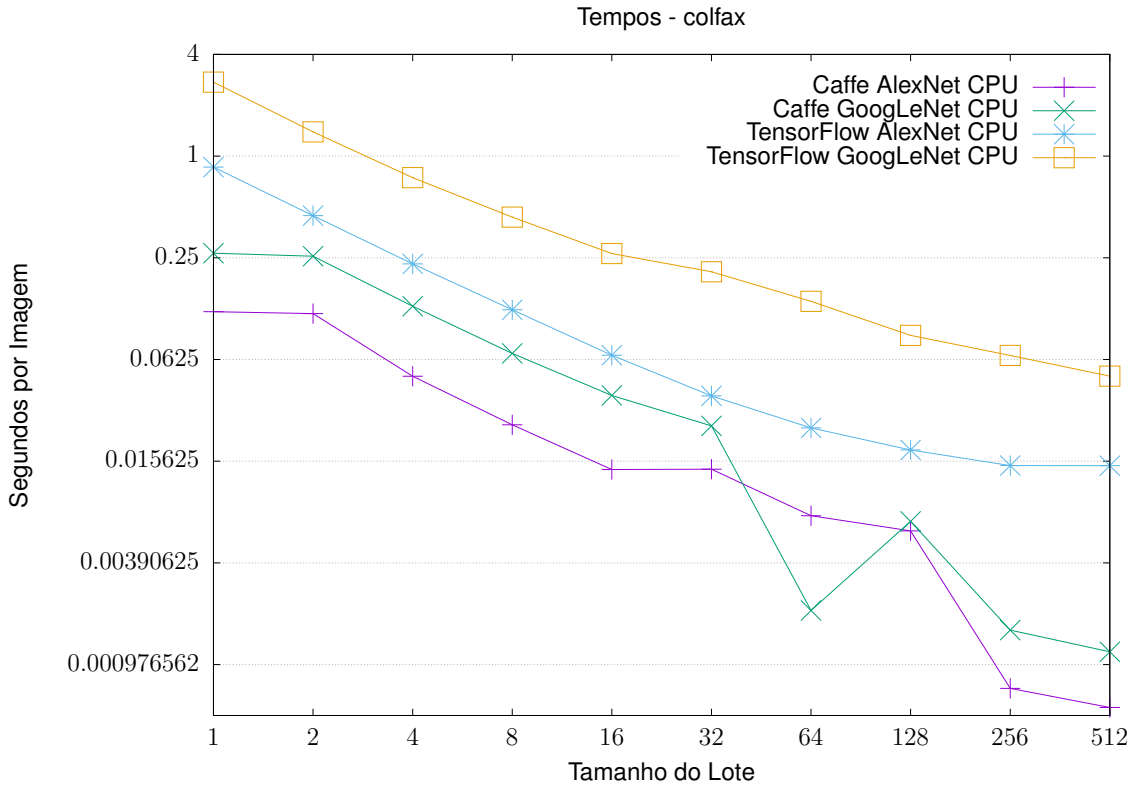

Gráfico 5.11 – Estatísticas de tempo de execução no ambiente colfax.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como propósito avaliar o desempenho em arquiteturas híbridas computacionais, e a confiabilidade em casos de sobreajuste inevitável, das bibliotecas de Aprendizagem Profunda Caffe e TensorFlow, sobre dois modelos de redes consagrados com variações dos tamanhos de lote e dispositivos computacionais empregados.

Parte do trabalho envolveu um estudo sobre a arquitetura de redes neurais profundas e suas características, dando destaque a redes neurais convolucionais, consagradas para tarefas de classificação de imagens. Logo após realizou-se uma pesquisa sobre as características das duas bibliotecas que o trabalho avalia: Caffe, da BVLC, e TensorFlow, da Google.

A etapa de desenvolvimento consistiu-se no planejamento de configurações variadas de hiperparâmetros para treinamento dos modelos de rede e posterior avaliação das bibliotecas, incluindo a definição de ambientes de execução que consistissem em sua maioria de arquiteturas híbridas, que possuíssem a disposição dispositivos aceleradores como GPGPUs.

Ao final dessa etapa, os treinamentos foram realizados e os dados desejados foram coletados dessa execução e avaliados, levando-se em consideração como se dava a evolução da perda nas redes até o inevitável sobreajuste, e como as bibliotecas tiravam proveito das arquiteturas *multicore* e *manycore* providas pelas unidades de processamento.

Com esses resultados foi possível obter-se algumas considerações acerca do desempenho provido pelas bibliotecas:

- Constataram-se os motivos que levam o tamanho do lote a exercer significativa influência sobre o processo de treinamento:
  - **tempo total de execução**: tamanhos menores de lote implicam em matrizes de entradas menores para a rede, evitando o uso de parte das capacidades de processamento de dispositivos como GPUs, uma vez que uma parcela pequena de seu número total de núcleos é utilizada nas operações de multiplicação de matrizes requeridas pelas camadas convolucionais;
  - **evolução da perda pela rede**: uma vez que ambas as bibliotecas calculam a perda uma única vez sobre cada lote, um número semelhante de iterações são necessárias para se atingir um valor estipulado de perda independente do tamanho do lote;
  - **uso de memória**: lotes maiores permitem que a rede avalie e aprenda mais sobre uma quantia maior de imagens. Entretanto, lotes maiores requerem capacidades de memória maiores, uma vez que mais imagens estão sendo tratadas pela rede de uma vez só, e a velocidade de acesso dessas memórias é crucial

para o desempenho das bibliotecas;

- Avaliou-se a biblioteca Caffe como mais amigável em questões de prototipação que a biblioteca TensorFlow: nenhuma linha de código em linguagem de programação é necessária para que seja realizado o treinamento de redes com camadas conhecidas como camadas convolucionais bidimensionais e camadas de agrupamento. A semelhança da linguagem utilizada para modelagem, *Google Protocol Buffer*, com a linguagem de notação JSON atribui pontos positivos nesse quesito;

- Em contrapartida, a API da biblioteca TensorFlow permite que redes mais complexas sejam prototipadas, pois, permite que operações menores – que podem ser relacionadas ao trabalho de neurônios individuais – sejam descritas via código;

- A biblioteca Caffe apresenta um sério erro de sobreajuste: valores de perda podem ficar negativos, o que em suma é incorreto – valores de perda se referem a diferença da inferência obtida pelo lote atual e o resultado esperado –. O trecho de código problemático está presente há um tempo considerável na biblioteca e nenhuma correção fora aplicada, apesar da comunidade de usuários já ter relatado o problema aos desenvolvedores[1];

- TensorFlow apresentou na maioria dos casos um consumo de memória menor que a biblioteca Caffe, possibilitando que tamanhos maiores de lotes pudessem ser utilizados em situações que o ambiente de execução não oferecia memória suficiente para lotes muito grandes. Entretanto, TensorFlow aloca por padrão toda a memória disponível em GPU;

- TensorFlow apresentou tempos de execução menores no ambiente lsc5 para todas as redes, em CPU e GPU. Em CPU, um dos indicativos de maior desempenho é o fato de sua biblioteca de álgebra linear fazer uso de instruções vetorizadas oferecidas pelas arquiteturas Intel, como SSE e AVX, atingindo tempos até 3 vezes menores por imagem que o Caffe; Em GPU, o indicativo é a realização de multiplicações de matrizes em lote, como constatado nos perfis de execução;

- No ambiente colfax, graças ao cenário oposto (biblioteca Caffe com suporte a instruções de vetorização e TensorFlow não), o resultado foi o oposto: Caffe consegue tempos de 4 a 8 vezes menores que com TensorFlow;

Trabalhos futuros podem explorar com mais profundidade o quão rápido as arquiteturas aqui empregadas conseguem treinar outros modelos de redes como VGG(HE et al., 2015) e Microsoft ResNet(SIMONYAN; ZISSERMAN, 2014), além de não somente explorar cenários ruins como o de sobreajuste inevitável, como abordado por este trabalho.

[1] https://github.com/BVLC/caffe/issues/3029

REFERÊNCIAS BIBLIOGRÁFICAS

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **CoRR**, abs/1603.04467, 2016. Acesso em 17 out. 2017. Disponível em: <http://arxiv.org/abs/1603.04467>.

BAHRAMPOUR, S. et al. Comparative study of caffe, neon, theano, and torch for deep learning. **CoRR**, abs/1511.06435, 2015. Disponível em: <http://arxiv.org/abs/1511.06435>.

CECKA, C. **Pro Tip: cuBLAS Strided Batched Matrix Multiply**. NVIDIA, 2017. Acesso em 7 dez. 2017. Disponível em: <https://devblogs.nvidia.com/parallelforall/cublas-strided-batched-matrix-multiply/>.

CáRDENAS-MONTE, M. **Sobreajuste - Overfitting**. 2015. Course notes. Acesso em 21 set. 2017. Disponível em: <http://wwwae.ciemat.es/~cardenas/docs/lessons/sobreajuste.pdf>.

DAULTANI, V.; OHNO, Y.; ISHIZAKA, K. Sparse direct convolutional neural network. In: ____. **Advances in Neural Networks - ISNN 2017: 14th International Symposium, ISNN 2017, Sapporo, Hakodate, and Muroran, Hokkaido, Japan, June 21–26, 2017, Proceedings, Part I**. Cham: Springer International Publishing, 2017. p. 293–303. ISBN 978-3-319-59072-1. Disponível em: <https://doi.org/10.1007/978-3-319-59072-1_35>.

DENG, L.; YU, D. **Deep Learning: Methods and Applications**. [S.l.], 2014. Acesso em 20 set. 2017. Disponível em: <https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/>.

FOX, J.; ZOU, Y.; QIU, J. **Software Frameworks for Deep Learning at Scale**. [S.l.], 2016. Acesso em 20 dez 2017. Disponível em: <http://dsc.soic.indiana.edu/publications/DLFrameworks.pdf>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. Acesso em 27 set. 2017. Disponível em: <http://www.deeplearningbook.org>.

HE, K. et al. Deep residual learning for image recognition. **CoRR**, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. **CoRR**, abs/1408.5093, 2014. Acesso em 17 out. 2017. Disponível em: <http://arxiv.org/abs/1408.5093>.

KARPATHY, A. **Convolutional Neural Networks for Visual Recognition (CS231n)**. 2014. Course notes. Acesso em 21 set. 2017. Disponível em: <http://cs231n.github.io/neural-networks-1/>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). **Advances in Neural Information Processing Systems 25**. Curran Associates, Inc., 2012. p. 1097–1105. Acesso em 13 nov. 2017. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

MARINO, J. L. **GoogLeNet in keras**. Github, 2017. Acesso em 28 nov. 2017. Disponível em: <http://joelouismarino.github.io/blog_posts/blog_googlenet_keras.html>.

PARLOFF, R. **Why Deep Learning is suddenly changing your life**: Decades-old discoveries are now electrifying the computing industry and will soon transform corporate

america. Fortune, 2016. Acesso em 21 set. 2017. Disponível em: <http://fortune.com/ai-artificial-intelligence-deep-machine-learning/>.

PENA, D. et al. Benchmarking of cnns for low-cost, low-power robotics applications. In: . [S.l.: s.n.], 2016.

RODRIGUEZ, A. **Intel® Xeon Phi Delivers Competitive Performance For Deep Learning – And Getting Better Fast**. Intel, 2016. Acesso em 6 dez. 2017. Disponível em: <https://software.intel.com/en-us/articles/intel-xeon-phi-delivers-competitive-performance-for-deep-learning-and-getting-better-fast>.

ROSENBLATT, F. **Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms**. Washington: Spartan Books, 1962.

SARLE, W. S. Stopped training and other remedies for overfitting. In: **Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics**. [S.l.: s.n.], 1995. p. 352–360.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **CoRR**, abs/1404.7828, 2014. Acesso em 20 set. 2017. Disponível em: <http://arxiv.org/abs/1404.7828>.

SEIDE, F.; LI, G.; YU, D. Conversational speech transcription using context-dependent deep neural networks. In: **Interspeech 2011**. International Speech Communication Association, 2011. Acesso em 1 dez. 2017. Disponível em: <https://www.microsoft.com/en-us/research/publication/conversational-speech-transcription-using-context-dependent-deep-neural-networks/>.

SHAMS, S. et al. Evaluation of deep learning frameworks over different hpc architectures. In: **2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)**. [S.l.: s.n.], 2017. p. 1389–1396. ISSN 1063-6927.

SHI, S.; CHU, X. Performance modeling and evaluation of distributed deep learning frameworks on gpus. **CoRR**, abs/1711.05979v2, 2017. Disponível em: <http://arxiv.org/abs/1711.05979v2>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **CoRR**, abs/1409.1556, 2014. Disponível em: <http://arxiv.org/abs/1409.1556>.

SZEGEDY, C. et al. Going deeper with convolutions. In: **2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2015. p. 1–9. ISSN 1063-6919.

VASCONCELOS, C. N.; CLUA, E. W. G. Deep learning - teoria e prática. In: Sociedade Brasileira de Computação - SBC (Ed.). **Jornadas de Atualização em Informática 2017**. 1. ed. Porto Alegre/RS: Sociedade Brasileira de Computação - SBC, 2017. cap. 6, p. 212–260. ISBN 978-85-7669-374-1. Acesso em 13 out. 2017. Disponível em: <http://csbc2017.mackenzie.br/public/files/all/livro-jai.pdf>.

WAGNER, D. **The importance of big data analytics in business**. TechRadar, 2014. Acesso em 15 out. 2017. Disponível em: <http://www.techradar.com/news/world-of-tech/the-importance-of-big-data-analytics-in-business-1267606>.

WANG, H. et al. Deep Learning for Image Retrieval: What Works and What Doesn't. p. 1576–1583, 11 2015.

WEISSTEIN, E. W. **Convolution.** 2017. From MathWorld–A Wolfram Web Resource. Acesso em 13 out. 2017. Disponível em: <http://mathworld.wolfram.com/Convolution.html>.

ANEXO A – SCRIPT DE EXECUÇÃO DA BIBLIOTECA CAFFE

```
"""
Script para rodar todos os treinamentos necessarios para o trabalho em CPU e
GPU, para a biblioteca Caffe

Para sua correta execucao, aconselha-se que o script esteja localizado na raiz
do diretorio git do caffe, no mesmo nivel dos diretorios 'models', 'examples'
e 'data'.
Alem disso, a base de dados lmdb do conjunto de entrada deve ter sido
previamente criada. Para tal, confira instrucoes em
http://caffe.berkeleyvision.org/gathered/examples/imagenet.html

Nota: As configuracoes deste script se destinam especificamente ao ambiente de
execucao utilizado neste trabalho denominado lsc5. Para ambientes diferentes e
versoes do Caffe diferentes, modifique as constantes utilizadas.

Uso: python3 <este_script>.py

Autor:      Rafael Gauna Trindade
Linguagem:  Python 3
Data:       22/12/2017
"""

import subprocess as sp
import os, shutil

def replace_in_file(_from, _to, _file):
    """
    Substitui a primeira ocorrencia de '_from' para '_to' no arquivo '_file'
    """
    sed_cmd = ["sed", "-i", "0,/" + _from + "/{s/" + _from + "/" + _to + "/}", _file]
    sp.run(sed_cmd)


def rename_last_log(net, mode, batch_size):
    """
    Renomeia o ultimo log gerado pelo caffe e move para a pasta tcc_logs
    """
    shutil.copy(
        os.path.realpath('logs/caffe.INFO'),
        './tcc_logs/train_{net}_{mode}_{batch_size}.log'.format(
            net = net, mode = mode, batch_size = str(batch_size).rjust(3, '0')
        )
    )


def remove_states(net):
    """
    Remove informacoes da rede pos-execucao, a fim de evitar que a rede resuma o
    treinamento anterior
    """
    path = os.getcwd() + '/models/bvlc_{network}/'.format(network=net)
    for file in os.listdir(path):
        if file.endswith('.caffemodel') or file.endswith('.solverstate'):
            os.remove(path + file)

```

```
56
57  def not_runned(net, mode, batch_size):
58      """
59      Determina se determinada metrica ja foi avaliada ou nao.
60      """
61      filepath = './tcc_logs/train_{net}_{mode}_{batch_size}.log'.format(
62          net=net, mode=mode, batch_size=str(batch_size).rjust(3, '0')
63      )
64      return (not os.path.exists(filepath))
65
66
67  def check_dir(dirname):
68      """
69      Confere se diretorio ja existe, caso contrario o cria.
70      """
71      if not os.path.exists(dirname):
72          os.mkdir(dirname)
73
74
75  os.environ['LD_LIBRARY_PATH'] = 'lib'
76
77  BATCH_SIZES = (1, 2, 4, 8, 16, 32, 64, 128, 256, 512)
78  EPOCHS = 10
79  MODES = ('CPU', 'GPU')
80  NETWORKS = ('alexnet', 'googlenet')
81  NUM_IMAGES = 1024
82
83  CAFFE = 'caffe'
84  PARAM_CMD = 'train'
85  PARAM_LOG_DIR = '-log_dir=logs'
86  PARAM_SOLVER = '-solver={solver}'
87  PARAM_MODEL = '-model={model}'
88  PARAM_REDIR = '> /dev/null'
89  SOLVER = os.getcwd() + '/models/bvlc_{network}/solver.prototxt'
90  MODEL = os.getcwd() + '/models/bvlc_{network}/train_val.prototxt'
91
92  check_dir('logs')
93  check_dir('tcc_logs')
94
95  for net in NETWORKS:
96      model = MODEL.format(network=net)
97      solver = SOLVER.format(network=net)
98      param_solver = PARAM_SOLVER.format(solver=solver)
99
100     for mode in MODES:
101         replace_in_file('mode: [A-Z]\\+', 'mode: ' + mode, solver)
102
103         for batch_size in BATCH_SIZES:
104             if not_runned(net, mode, batch_size):
105                 num_iterations = (NUM_IMAGES//batch_size) * EPOCHS
106                 replace_in_file(
107                     'max_iter: [0-9]\\+', 'max_iter: ' + str(num_iterations),
108                     solver
109                 )
110                 replace_in_file(
111                     'batch_size: [0-9]\\+', 'batch_size: ' + str(batch_size),
112                     model
113                 )
114
```

```
115                 print("Rodando rede {net} no modo {mode} com tamanho de lote {batch}...".
                        format(net=net, mode=mode, batch=batch_size))
116                 cmd = [CAFFE, PARAM_CMD, PARAM_LOG_DIR, param_solver]
117                 cp = sp.run(cmd, stdout=sp.DEVNULL, stderr=sp.DEVNULL)
118                 if cp.returncode == 0:
119                     rename_last_log(net, mode, batch_size)
120                     remove_states(net)
121             else:
122                 print("Rede {net} no modo {mode} com tamanho de batch {batch} ja testada.
                        ".format(net=net, mode=mode, batch=batch_size))
```

ANEXO B – SCRIPT DE EXECUÇÃO DA BIBLIOTECA TENSORFLOW

```
"""
Script para rodar todos os treinamentos necessarios para o trabalho em CPU e GPU,
para a biblioteca TensorFlow

Para sua correta execucao, aconselha-se que o script esteja localizado no diretorio
research/slim do diretorio git de modelos da TensorFlow (confira em
http://github.com/tensorflow/models), no mesmo nivel do script train_image_classifier.py
Alem disso, a base de dados lmdb do conjunto de entrada deve ter sido previamente criada.
Para tal, confira instrucoes no arquivo README.md localizado no mesmo diretorio. Para
criacao de bases de dados personalizadas, uma sugestao e estudar e modificar os scripts
existentes para a base de dados flowers.

Nota: As configuracoes deste script se destinam especificamente ao ambiente de execucao
utilizado neste trabalho denominado lsc5. Para ambientes diferentes e versoes do
TensorFlow diferentes, modifique as constantes utilizadas.

Uso: python3 <este_script>.py

Autor:      Rafael Gauna Trindade
Linguagem:  Python 3
Data:       22/12/2017
"""

import subprocess as sp
import os, shutil

def not_runned(net, mode, batch_size):
    """
    Determina se determinada metrica ja foi avaliada ou nao.
    """
    filepath = './tcc_logs/train_{net}_{mode}_{batch_size}.log'.format(
        net=net, mode=mode, batch_size=str(batch_size).rjust(3, '0')
    )
    return (not os.path.exists(filepath))

BATCH_SIZES = (1, 2, 4, 8, 16, 32, 64, 128, 256, 512)
EPOCHS = 10
MODES = {'CPU':'--clone_on_cpu=True', 'GPU':'--clone_on_cpu=False'}
NETWORKS = {'alexnet':'alexnet_v2', 'googlenet':'inception_v1'}
WEIGHTS = {'alexnet':'0.0005', 'googlenet':'0.0002'}
NUM_IMAGES = 1024

PYTHON = 'python3'
PARAM_SCRIPT = 'train_image_classifier.py'
PARAM_TRAIN_DIR = '--train_dir={dir}'
PARAM_MODEL = '--model_name={model}'
PARAM_DATASET_DIR = '--dataset_dir=flowers'
PARAM_WEIGHT_DECAY = '--weight_decay={decay}'

for net_name, net_nick in NETWORKS.items():
    param_model = PARAM_MODEL.format(model=net_nick)
    param_weight_decay = PARAM_WEIGHT_DECAY.format(decay=WEIGHTS[net_name])

    for mode_name, mode_param in MODES.items():
        for batch_size in BATCH_SIZES:
```

```
56              if not_runned(net_name, mode_name, batch_size):
57                  num_steps = (NUM_IMAGES//batch_size) * EPOCHS
58                  param_steps = '--max_number_of_steps={steps}'.format(steps=num_steps)
59                  param_batch = '--batch_size={batch}'.format(batch=batch_size)
60
61                  log_name = 'train_{net}_{mode}_{batch_size}'.format(
62                      net = net_name,
63                      mode = mode_name,
64                      batch_size = str(batch_size).rjust(3, '0')
65                  )
66
67                  train_dir = '/tmp/' + log_name
68                  os.mkdir(train_dir)
69                  param_train_dir = PARAM_TRAIN_DIR.format(dir=train_dir)
70
71                  print("Rodando rede {net} no modo {mode} com tamanho de batch {batch}..."
                        .format(net=net_name, mode=mode_name, batch=batch_size))
72                  cmd = [
73                      PYTHON, PARAM_SCRIPT, param_train_dir, param_model, mode_param,
74                      PARAM_DATASET_DIR, param_weight_decay
75                  ]
76                  with open('./tcc_logs/' + log_name + '.log', 'w') as logfile:
77                      cp = sp.run(cmd, stdout=logfile, stderr=sp.DEVNULL)
78                      if cp.returncode == 0:
79                          shutil.move(train_dir, './tcc_logs/' + log_name)
80                      else:
81                          shutil.rmtree(train_dir)
82              else:
83                  print("Rede {net} no modo {mode} com tamanho de batch {batch} ja testada.
                        ".format(net=net, mode=mode, batch=batch_size))
```

ANEXO C – SCRIPT PARA GERAÇÃO DE ARQUIVOS .CSV

```
"""
Script para extrair informacoes de perda e tempo dos logs emitidos pelas
bibliotecas Caffe e Tensorflow Os valores sao reunidos em arquivos .csv e
exportados para uma pasta 'csv'

Os arquivos de log devem ter nomes no formato
    train_{nome rede}_{CPU ou GPU}_{tamanho lote}.log
E estar contidos na pasta especificada via linha de comando

Uso: python3 <este_script>.py <pasta_logs> <nome_ambiente> <nome_biblioteca>

Autor:     Rafael Gauna Trindade
Linguagem: Python 3
Data:      22/12/2017
"""

import sys, re, os, csv

networks    = ['alexnet', 'googlenet']
modes       = ['CPU', 'GPU']
batch_sizes = [str(2**i).rjust(3, '0') for i in range(0, 10)]
folder, machine, lib = None, None, None

try:
    folder, machine, lib = sys.argv[1:4]
except:
    print("""usage: python3 {script} <log_dir> <machine_name> <lib_name>
arguments:
log_folder   : directory where library logs are located
machine_name : name of machine where the training was runned
lib_name     : name of library. Options are 'caffe' and 'tensorflow'
    """.format(script=sys.argv[0]))
    exit(1)


def log_filename(net, mode, batch):
    """
    Funcao que formata o nome de um arquivo de log
    """
    return '{folder}/train_{net}_{mode}_{batch}.log'.format(
        folder = folder,
        net    = net,
        mode   = mode,
        batch  = batch
    )


def try_open(net, mode, batch):
    """
    Funcao que tenta abrir um arquivo de log, caso ele exista.
    """
    filename = log_filename(net, mode, batch)
    if os.path.exists(filename):
        return open(filename)
    return None
```

```
56
57  # Define expressao regular a ser procurada nos arquivos de log
58  pattern = None
59  if lib == 'caffe':
60      pattern = re.compile(r' Iteration (?P<step>\d+) \(-?\d+(.\d+)? (e(-|\+)\d+)? iter/s,
            (?P<time>\d+(.\d+)?)s/1 iters\), loss = (?P<loss>(-)?\d+.\d+(e-\d+)?)')
61  elif lib == 'tensorflow':
62      pattern = re.compile(r'INFO:tensorflow:global step (?P<step>\d+): loss = (?P<loss>-?\
            d+(.\d+)?) \((?P<time>\d+(.\d+)?) sec/step\)')
63  else:
64      print("Error: Library name must be 'caffe' or 'tensorflow'")
65      exit(1)
66
67  data_times = []
68  data_loss = []
69
70  # Itera por rede, modo, tamanho de lote e linha de arquivo de log
71  for net in networks:
72      for mode in modes:
73          for batch in batch_sizes:
74              file = try_open(net, mode, batch)
75              if file:
76                  times = []
77                  last_size = len(data_loss)
78                  for line in file:
79                      values = None
80                      # Aplica a busca pela expressao regular
81                      if lib == 'caffe':
82                          line_parsed = line.replace('\n','').split(']')
83                          if (len(line_parsed) == 2) and line_parsed[1].count('Iteration'):
84                              values = pattern.match(line_parsed[1])
85                      elif lib == 'tensorflow':
86                          values = pattern.match(line)
87
88                      # Armazena valores se encontrar um resultado
89                      if values:
90                          v_dict = values.groupdict()
91                          data_loss.append((
92                              net, mode, batch,
93                              v_dict['step'],
94                              v_dict['loss'],
95                              v_dict['time']
96                          ))
97                          times.append(float(v_dict['time']))
98
99                  # Apresenta um resumo da quantidade de informacoes encontrada no
100                 # arquivo de log atual
101                 print(
102                     'Computed', net, 'on', mode, 'with', batch,
103                     str(len(data_loss) - last_size), 'iterations found'
104                 )
105                 if len(times) > 0:
106                     avg_time = sum(times)/len(times)
107                     data_times.append((net, mode, batch, avg_time))
108             else:
109                 print('File', log_filename(net, mode, batch), 'not found')
110
111 # Cria pasta csv caso nao exista
112 if not os.path.exists('csv'):
```

```
113     os.mkdir('csv')
114 
115 # Escreve csv com as informacoes de perda
116 with open('csv/loss_{mach}_{lib}.csv'.format(mach=machine,lib=lib), 'w') as csvfile:
117     fieldnames = ['#network', 'mode', 'batch', 'step', 'loss', 'time']
118     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
119     writer.writeheader()
120     for data in data_loss:
121         dic = {}
122         for i, value in enumerate(fieldnames):
123             dic[value] = data[i]
124         writer.writerow(dic)
125 
126 # Escreve csv com as informacoes de tempo de execucao
127 with open('csv/times_{mach}_{lib}.csv'.format(mach=machine,lib=lib), 'w') as csvfile:
128     fieldnames = ['#network', 'mode', 'batch', 'avg_time']
129     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
130     writer.writeheader()
131     for data in data_times:
132         dic = {}
133         for i, value in enumerate(fieldnames):
134             dic[value] = data[i]
135         writer.writerow(dic)
```