

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UMA ABORDAGEM PARA GERAÇÃO DE
IMAGENS BASEADA NO USO DE GPU E
REDES NEURAIS ARTIFICIAIS

HÁLAMO GIULIAN REIS DE ANDRADE

JOÃO PESSOA-PB
Dezembro-2018

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Uma abordagem para geração de imagens baseada no uso de
GPU e redes neurais artificiais

Hálamo Giulian Reis de Andrade

Dissertação submetida à Coordenação do Curso de Pós-Graduação em
Informática da Universidade Federal da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sinais, Sistemas Digitais e Gráficos

Alisson Vasconcelos de Brito
(Orientador)

João Pessoa, Paraíba, Brasil

©Hálamo Giulian Reis de Andrade, 14 de Dezembro de 2018

Catálogo na publicação
Seção de Catalogação e Classificação

A553a Andrade, Hálamó Giulian Reis de.

Uma abordagem para geração de imagens baseada no uso de GPU e redes neurais artificiais / Hálamó Giulian Reis de Andrade. - João Pessoa, 2018.
68 f. : il.

Orientação: Alisson V Brito.
Dissertação (Mestrado) - UFPB/CI.

1. Geração de imagens. 2. Computação paralela. 3. Redes neurais artificiais. I. Brito, Alisson V. II. Título.

UFPB/BC



UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA



Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Hálamo Giulian Reis de Andrade, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 14 de dezembro de 2018.

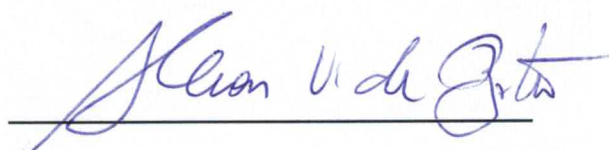
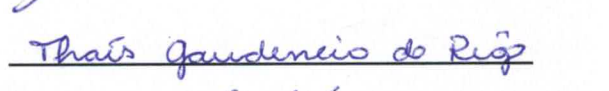
1 Aos quatorze dias do mês de dezembro, do ano de dois mil e dezoito, às dezesseis horas e
2 trinta minutos, no Centro de Informática da Universidade Federal da Paraíba, em
3 Mangabeira, reuniram-se os membros da Banca Examinadora constituída para julgar o
4 Trabalho Final do Sr. Hálamo Giulian Reis de Andrade, vinculado a esta Universidade sob a
5 matrícula nº 20161020309, candidato ao grau de Mestre em Informática, na área de
6 "Sistemas de Computação", na linha de pesquisa "Sinais, Sistemas Digitais e Gráficos", do
7 Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A
8 comissão examinadora foi composta pelos professores: Alisson Vasconcelos de Brito (PPGI-
9 UFPB) Orientador e Presidente da Banca, Thais Gaudêncio do Rego (PPGI-UFPB),
10 Examinadora Interna, Márcio Emanuel Ugulino de Araújo Júnior (IFPB), Examinador Externo
11 à Instituição. Dando início aos trabalhos, o Presidente da Banca, cumprimentou os
12 presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato
13 para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado: "Uma
14 abordagem para geração de imagens baseada no uso de GPU e redes neurais artificiais".
15 Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o
16 seguinte parecer: "**aprovado**". Do ocorrido, eu, Claurton de Albuquerque Siebra,
17 Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai
18 assinada por mim e pelos membros da banca examinadora. João Pessoa, 14 de dezembro
19 de 2018.


Prof. Dr. Claurton de Albuquerque Siebra

Prof. Dr. Alisson Vasconcelos de Brito
Orientador (PPGI-UFPB)

Prof. Dr. Thais Gaudêncio do Rego
Examinadora Interna (PPGI-UFPB)

Prof. Dr. Márcio Emanuel Ugulino de Araújo Júnior
Examinador Externo ao Programa (IFPB)


Resumo

Diante da crescente demanda por soluções de visão computacional embarcadas, surge cada vez mais a necessidade de efetuar uma verificação funcional estratégica. Nesse âmbito, o presente trabalho objetiva a especificação e implementação de um gerador de imagens sintéticas que produz imagens derivadas a partir de bases de imagens iniciais. O estudo compreende uma pesquisa bibliográfica com os trabalhos da literatura científica pertinentes à temática, detalhes de uma implementação e também uma avaliação experimental para expor mais informações sobre a presente proposta. O processo de geração das imagens derivadas foi concebido através de componentes que trabalham com métodos de geração por deformação e geração por redes neurais artificiais. Os componentes desenvolvidos foram projetados com computação paralela, utilizando a plataforma CUDA, além de também utilizar TensorFlow para implementações das redes neurais envolvidas. Foram implementadas as redes neurais artificiais do tipo *Convolutional Neural Network* (CNN) e *Generative Adversarial Networks* (GAN). Os resultados acerca do componente implementado corroboram com a viabilidade de sua utilização no aumento de dados, em verificações funcionais distribuídas e para o treinamento de redes neurais artificiais.

Palavras-chave: geração de imagens, computação paralela, redes neurais artificiais.

Abstract

With the crescent demand for embedded computer vision solutions, a strategic functional verification is increasingly needed. In this context, the present work aims at the specification and implementation of a synthetic image generator that produces images derived from initial image datasets. This work includes a bibliographical research in the works of the pertinent scientific literature, details of an implementation and also an experimental evaluation to present more information about the present proposal. The process of generation of the derived images was conceived through components that work with methods of generation by deformation and generation by artificial neural networks. The developed components were designed with parallel computing, using the CUDA platform, as well as using TensorFlow for implementations of the neural networks involved. There were implemented Convolutional Neural Network (CNN) and Generative Adversarial Networks (GAN) in one of the methods of image generation. The results about the implemented component corroborate the feasibility of its use in the field of data augmentation, in functional distributed verifications and in the training of artificial neural networks.

Keywords: image generation, parallel computing, artificial neural networks.

Agradecimentos

Agradeço ao meu pai, Vicente Andrade, e principalmente à minha mãe, Kátia Reis, que me incentiva a priorizar os estudos desde que sou pequeno, nunca titubeando em puxar minha orelha quando necessário, sempre fazendo o possível para me apoiar nessas aventuras acadêmicas que a vida me reserva.

Ao Professor Dr. Alisson Brito, meu orientador, que me acolheu no LaSER e me norteou durante o caminho das pedras trilhado no mestrado. Também ao Professor Dr. Álvaro Medeiros que tanto me incentivou a concluir essa importante etapa, além de todos os professores que se envolveram direta ou indiretamente.

Um grande agradecimento aos amigos e companheiros de pesquisa, em especial: Júlio Maia, Renato Abreu, Daniel Moraes, Thiago Werlley e Joanacelle Melo.

À CAPES que financiou essa pesquisa através de bolsa e também à NVIDIA, que cedeu um kit de desenvolvimento da plataforma para sistemas embarcados Jetson TX1 para que fossem rodados os experimentos.

Acrônimos

API: Interface de Programação de Aplicação, do inglês *Application Programming Interface*.

ARM: originalmente do inglês *Acorn RISC Machine* e depois do inglês, *Advanced RISC Machine*.

CAPTCHA: Teste de *Turing* Público Completamente Automatizado para Distinção entre Computadores e Humanos, do inglês *Completely Automated Public Turing test to tell Computers and Humans Apart*.

CPU: Unidade Central de Processamento, em inglês *Central Processing Unit*.

CNN: Rede Neural Convolucional, do inglês *Convolutional Neural Network*.

CUDA: Arquitetura de Dispositivo de Computação Unificada, do inglês *Compute Unified Device Architecture*.

DEN: Rede Dinamicamente Expansível, do inglês *Dynamically Expandable Network*.

DSP: Processador Digital de Sinal, do inglês *Digital Signal Processor*.

DUT: Componente Em Teste, do inglês *Design Under Test*.

FLOPS: Operações de Ponto Flutuante Por Segundo, do inglês *Floating Point Operations Per Second*.

FPGA: Matriz de Portas Programáveis em Campo, do inglês *Field Programmable Gate Array*.

GAN: Redes Generativas Adversárias, do inglês *Generative Adversarial Networks*.

GPGPU: Unidade de Processamento Gráfico de Propósito Geral, do inglês *General Purpose on Graphics Processing Unit*.

GPU: Unidade de Processamento Gráfico, do inglês *Graphics Processing Unit*.

HLA: Arquitetura de Alto Nível, do inglês *High Level Architecture*.

LaSER: Laboratório de Sistemas Embarcados e Robótica da UFPB.

OpenCL: Linguagem de Computação Aberta, do inglês *Open Computing Language*.

OpenMP: MultiProcessamento Aberto, do inglês *Open Multi-Processing*.

RISC: do inglês *Reduced Instruction Set Computer*.

RTI: Infraestrutura de Tempo de Execução, do inglês *Run Time Infrastructure*.

SoC: Sistema Em um Chip, do inglês *System-On-a-Chip*.

ULA: Unidade Lógica Aritmética.

Sumário

1	Introdução	1
1.1	Contextualização Inicial e Motivação	1
1.2	Objetivo	5
1.2.1	Objetivos Específicos	5
1.3	Metodologia	6
1.4	Contribuições	6
2	Fundamentação Teórica	9
2.1	Computação Paralela e Programação em GPU	9
2.2	Redes Neurais Artificiais e <i>Deep Learning</i>	13
2.3	Considerações Finais	18
3	Trabalhos Relacionados	19
3.1	Literatura Visitada	19
3.2	Considerações Finais	24
4	Proposta de Geração de Imagens	25
4.1	Visão Geral: descrevendo o ambiente	25
4.2	Geração de Imagens por Deformação	27
4.2.1	Processo de Calibragem	30
4.3	Geração de Imagens por Rede Neural	33
4.3.1	Treinamento da Rede Neural	36
4.4	Considerações Finais	37

5	Avaliação Experimental	39
5.1	Plano dos Experimentos	39
5.2	Experimento 1: calibragem	43
5.2.1	Resultados Obtidos	45
5.3	Experimento 2: análise por tipo de deformações	46
5.3.1	Resultados Obtidos	47
5.4	Experimento 3: geração de imagens para treinamento de redes neurais . . .	48
5.4.1	Resultados Obtidos	50
5.5	Experimento 4: qualidade por inspeção ocular	51
5.5.1	Resultados Obtidos	52
5.6	Experimento 5: CPU vs GPU	54
5.6.1	Resultados Obtidos	55
5.7	Resultados Adicionais	56
5.8	Considerações Finais	57
6	Conclusão	59
	Referências Bibliográficas	66
A	Amostra de dígitos utilizados	67
B	Apêndice B	68

Lista de Figuras

1.1	Contribuições e componentes em contexto de verificação funcional.	7
2.1	Gráfico comparativo de desempenho por operações de ponto flutuante. . . .	10
2.2	Arquitetura CPU e GPU. Mostrando a organização geral das Unidades de Controle (UC), Unidade Lógica e Aritmética (ULA) e blocos de memórias (Cache e DRAM).	11
2.3	Velocidade de acesso à memória CPU vs GPU.	11
2.4	Linha histórica: Inteligência Artificial, <i>Machine Learning</i> , <i>Deep Learning</i> . .	14
2.5	Exemplo de estrutura básica de uma rede neural artificial. Valores de entrada representados por "a", pesos (do inglês <i>weight</i>) representados por "w", <i>bias</i> indicado pelo "b" e "y" representa o valor de saída.	15
2.6	Extração de características por níveis de camada.	16
3.1	Imagens geradas para CAPTCHA pelos autores Kaur e Behal.	20
3.2	Sprites gerados utilizando técnicas de redes neurais pelos autores Horsley e Perez-Liebana.	21
3.3	Amostras de imagens de celebridades falsas geradas por GANs.	23
4.1	Visão geral do ambiente.	26
4.2	Geração por deformação.	28
4.3	Fluxograma do processo de calibragem do gerador por deformação.	31
4.4	Busca de limiares durante processo de calibragem.	32
4.5	Geração por GAN.	34
4.6	Fluxo alternativo para geração de imagens.	36
4.7	Operação do componente gerador de imagens.	37

5.1	Foto dos dois <i>hosts</i> utilizados nos experimentos executados nas dependências do laboratório LASER. À esquerda a plataforma de sistemas embarcados Jetson TX1 e à direita o PC.	43
5.2	Fluxo de execução do Experimento 1: calibragem.	44
5.3	Porcentagem de precisão nos diferentes estados durante a calibragem, no Experimento 1.	45
5.4	Fluxo de execução do Experimento 2: análise das deformações e quantidade de imagens.	47
5.5	Percentual de acerto no Experimento 2, durante a geração de bases de imagens com diferentes quantidades de imagens.	48
5.6	Fluxo de execução do Experimento 3: uso para treinamento de redes neurais.	49
5.7	Fluxo de execução do Experimento 3: qualidade por inspeção ocular.	52
5.8	Quantidade de imagens geradas e classificadas durante Experimento 3.	53
5.9	Amostras das imagens de dígitos escritos à mão. a) do dataset inicial, sem alterações; b) geradas por deformação; c) geradas pela GAN.	53
5.10	Amostras das imagens geradas pelo processo de geração por rede neural.	54
5.11	Gráfico do tempo de geração por número de imagens produzidas. CPU: curva vermelha. GPU: curva azul.	55
5.12	Desempenho dos algoritmos KNN e SVM durante a expansão de um <i>dataset</i> de imagens no Experimento 2.	56
5.13	Amostra da composição de background com imagens de dígitos escritos à mão gerados pela ferramenta desenvolvida.	57
A.1	Arquivo "digits.png" contendo amostras de dígitos escritos à mão. Fonte: diretório <i>samples</i> da instalação do OpenCV.	67
B.1	Amostras de imagens geradas pela GAN.	68

Lista de Tabelas

3.1	Comparação entre diferentes estratégias para geração de imagens.	22
4.1	Tecnologias utilizadas e suas aplicações em cada componente de geração de imagens simples.	27
5.1	Uso do Virtual Bus e itens de <i>hardware</i> nos experimentos realizados.	42
5.2	Experimento treinamento redes neurais.	50

Capítulo 1

Introdução

Nesse capítulo está presente uma discussão introdutória com a contextualização da problemática abordada, também expondo a motivação desse trabalho. Nas seções seguintes serão apresentados os objetivos de pesquisa, seguidos da metodologia adotada e as contribuições alcançadas.

1.1 Contextualização Inicial e Motivação

Durante décadas, com a crescente demanda por sistemas embarcados, a indústria de *hardware* passou a manufaturar produtos em forma de *System-On-a-Chip* (SoC). Esse, pode ser entendido como um produto que reúne os diversos componentes computacionais de um sistema eletrônico específico agrupados em um só chip.

Durante o processo de projeto de um SoC muito esforço é gasto com a verificação funcional. Isso se deve principalmente por conta desse processo tentar submeter o projeto em questão ao maior número de possibilidades de testes possíveis, para com isso simular o comportamento do processamento do chip em detrimento de estímulos recebidos em situações do mundo real.

Com isso, dependendo da natureza do processamento e da diversidade das entradas, será necessário uma quantidade crescente de dados de estímulos para alimentar o processo de verificação funcional, podendo inclusive tornar o processo muito dispendioso [Delorme 2014]. Em compensação, um projeto com uma verificação mal gerenciada pode resultar em uma verificação funcional com baixa cobertura, ou seja, que potencialmente não abrange ao menos

os pontos críticos do projeto. Além disso, pode levar a uma rápida degradação da robustez do projeto, já que o produto pode ter comportamentos inesperados não verificados [IEEE Standard for the Functional Verification Language e 2017].

Assim, é necessário equilibrar o dilema com o balanceamento entre empregar mais recursos na verificação funcional para assegurar o projeto, ou aliviar um pouco na verificação para economizar recursos monetários, tempo e esforços adicionais [Bedoya, Perez e Marin 2016]. Pode ser tomado como exemplo um projeto de um circuito de um *Digital Signal Processor* (DSP) focado no processamento de imagens de alta resolução. Nesse caso, considerando imagens de 720p, temos em média entradas de no mínimo 2,7MB em cada *frame* sem tratamento (*raw*). Incluir na verificação funcional todas as possibilidades possíveis de processamento para imagens com essa quantidade de *bytes* pode tornar o projeto inviável.

No exemplo descrito, já temos um ostensivo uso de processamento de imagens, algo comum nas interações em ambientes automatizados, utilizando de técnicas de detecção de face, reconhecimento facial, rastreamento de objetos, entre outros [Bhatia, Patel e Chauhan 2016]. O uso de técnicas de visão computacional com a utilização de algoritmos paralelos é outra das tendências que podem ser observadas. Estudos como [Oh, Yi e Yi 2015] e [Bhatia, Patel e Chauhan 2016] nos dão mais informações acerca de melhorias obtidas através de computação paralela em relação às versões seriais. Nesses cenários, existem muitas possibilidades levando em conta computação paralela juntamente com a computação distribuída, conforme também mostram estudos experimentais em [Otterness et al. 2017], o que leva à necessidade de verificação de componentes distribuídos.

De modo geral, para não incorrer em gastos desnecessários, é comum a utilização de um conjunto de imagens selecionadas estrategicamente para que seja efetuada a verificação da forma mais efetiva possível e utilizando o mínimo de processamento. Assim, uma alternativa viável para muitos casos é a utilização desses conjuntos de imagens disponíveis para download na Internet, como no caso do: NIST [Garris 1992]; MNIST e EMNIST [Cohen et al. 2017]; CIFAR-10 e CIFAR-100 [Krizhevsky 2009]; ImageNet [ImageNet 2016]; AWA e AWA-2 [Lampert, Pucher e Dostal 2018]. Esses *datasets* consistem em bases de dados de imagens organizadas por categoria (pessoas, carros, caracteres, etc) a fim de serem utilizadas para os fins específicos, e são inclusive muito utilizadas para outros fins, como o treinamento de redes neurais [Kumar e Garg 2017]. A utilização de *datasets* de imagens

também simplifica comparações de desempenho entre possíveis algoritmos já existentes. É muito utilizada em artigos científicos comparativos e até competições [ImageNet 2016], já que dessa forma é possível estabelecer um ponto de partida comum entre os algoritmos.

Acontece que, muitas vezes existe a base de imagens, mas essa pode não conter a diversidade de imagens suficiente para todas as aplicações, levando à necessidade de estender a base de imagens existente, como por exemplo aconteceu com alguns *datasets* NIST que deram origem a outros [Cohen et al. 2017]. Ou mesmo, inclusive, também pode acontecer de não existir uma base de imagens de objetos específicos disponível para download, levando assim à necessidade de empreender esforços adicionais na captura de imagens e criação de uma base de imagens. Nesse caso, geralmente critérios são estabelecidos para a captura das imagens, conforme no caso das bases de dados CIFAR [Krizhevsky 2009].

Além do mencionado, muitas bases de dados contêm apenas um único objeto em cada imagem, mas muitas vezes o cenário em que os projetos devem atuar em produção demandam vários objetos por imagem. Outro problema é que muitas vezes é necessário encontrar vários objetos específicos em uma mesma imagem e *datasets* de combinações específicas.

Sobretudo é destacado nesse trabalho as bases de imagens para uso em verificação funcional, mas, a partir das informações apresentadas é possível perceber uma demanda por bancos de imagens para diversas aplicações.

Ademais, existe uma crescente demanda por soluções que englobam ambientes cada vez maiores e com interações mais complexas. Para isso, é comum encontrar sistemas com arquiteturas distribuídas. Ao distribuir o sistema ao longo de componentes dispersos em uma rede, algumas vantagens podem ser adquiridas. Essa arquitetura trará ao dispor dos usuários uma flexibilidade em relação à capacidade total de processamento, com a expansão do sistema em termos de memória total e carga de trabalho, como também pode possibilitar, inclusive, a integração com outros sistemas responsáveis por funcionalidades complementares. Isso também traz a necessidade de efetuar, de forma distribuída, a verificação de um componente.

Diante do exposto, é visado atender à necessidade de verificação de componentes, sejam em suas formas distribuídas ou mesmo em componentes dedicados, como também o projeto de SoC específicos para visão computacional, pois, muitas vezes não é viável transferir as informações de entrada para outro nó em uma rede de computadores. Com isso, processar

a tomada de decisão torna-se crítico, principalmente quando a demanda acontece em tempo real. Isso é perceptível no caso do uso em massa de processamento de imagens onde a demanda crescente por dispositivos de *hardware* e também *softwares* específicos para esse fim, como sistemas de câmeras integradas (*smart cameras*), componentes de sistemas *cyber-físicos*, sistemas de visão computacional embarcados etc.

Em contrapartida com o exposto, esse trabalho objetiva a especificação e construção de um componente gerador de imagens, de forma que torne possível criar novas imagens a fim de estender uma base de imagens inicial. Também é visada a criação de imagens compostas por uma combinação de elementos pré-especificados e com variação de aspectos visuais pré-estabelecidos, para com isso efetivar a geração automatizada de bases de imagens personalizados para verificação funcional de algoritmos de visão computacional. Isso de forma que seja construído como um componente reusável para que seja possível utilizá-lo em projetos modularizados.

A especificação aqui presente conta com duas principais aplicações: a primeira é a de gerar um *dataset* a partir de poucas imagens iniciais, tentando assim minimizar os esforços para criar um *dataset* inexistente; a segunda é a de estender um *dataset* já existente. Em ambas as aplicações o componente gerador produz imagens diversificando as entradas através da aplicação de deformações e também redes neurais, mais especificamente o uso de Generative Adversarial Networks (GAN) [Creswell et al. 2018]. Esse processo de geração deve buscar a aplicação de métricas de variação de aspectos gráficos, com limiares configuráveis de acordo com o contexto da aplicação. Assim, serão controladas as distorções que as imagens devem sofrer.

Para tentar assegurar uma maior eficiência no processo da geração das imagens será proposto o uso de tecnologias que sejam assistidas por GPU, pois o componente desenvolvido também deve estar apto a gerar imagens com uma performance considerável para uso em tempo de execução. Nesse caso, após configurado devidamente, o próprio componente tem a possibilidade de atuar gerando as imagens sob demanda em um ambiente de verificação funcional. Para isso já existem algumas possibilidades de *frameworks* que podem ser acelerados com o emprego de GPU, como o TensorFlow. Além disso, para executar determinados procedimentos, como a combinação das imagens sintetizadas, também é possível fazer uso de programação paralela, maximizando o processo a partir da plataforma paralela da NVIDIA,

Compute Unified Device Architecture (CUDA).

Assim, essa pesquisa foi iniciada a partir da seguinte hipótese: é possível implementar de forma paralela um componente gerador de imagens para composição de bases de imagens com possibilidade de atuação de forma distribuída?

Esse trabalho foi organizado da seguinte forma, nas próximas seções do capítulo corrente, serão apresentados os objetivos, a metodologia de pesquisa e as contribuições do trabalho. Os capítulos seguintes da dissertação incluem, Capítulo 2, contendo a fundamentação teórica do trabalho; Capítulo 3, com os trabalhos relacionados presentes na literatura científica; Capítulo 4, contendo tanto detalhes de implementação como a apresentação do ambiente desenvolvido; Capítulo 5, que apresenta descrições dos experimentos bem como resultados obtidos nessa pesquisa; e finalmente o Capítulo 6, onde são feitas as considerações finais.

1.2 Objetivo

Conceber um mecanismo para geração de imagens artificiais de forma eficiente para uso em verificação funcional distribuída de algoritmos de processamento de imagens.

1.2.1 Objetivos Específicos

- Desenvolver um algoritmo para geração de imagens que, a partir de imagens iniciais, cria imagens derivadas que são visualmente similares e com as mesmas características das imagens iniciais;
- Implementar otimizações do gerador utilizando de computação paralela, predominantemente em GPU;
- Implementar um componente que inclui o gerador de imagens e o utiliza para intercomunicar, de forma distribuída, e fornecer imagens geradas para outros possíveis componentes;
- Executar experimentos para avaliação dos componentes desenvolvidos.

1.3 Metodologia

A metodologia desse trabalho inicialmente é baseada em uma pesquisa bibliográfica com trabalhos que discutem geração de imagens sintéticas e implementações em computação paralela. Com isso, ao longo da pesquisa foi implementado o projeto de um gerador de imagens de acordo com o relacionamento das informações obtidas na literatura científica. A partir daí ocorreu de fato a implementação do ambiente completo. Por conseguinte, a execução dos experimentos que forneceram os resultados para discussões e demais considerações. Portanto, essa pesquisa também teve metodologia baseada em experimentos e estudo de caso.

Para execução dos experimentos foram utilizados dois *hosts* conectados via rede *wireless*. A seguir, a listagem do hardware:

- Plataforma para sistemas embarcados da NVIDIA, Jetson TX1 equipada com a GPU NVIDIA Maxwell, de 256 *cores* CUDA, CPU Quad ARM® A57/2 MB L2 e 4 GB de memória RAM compartilhada;
- Computador com CPU Intel i5-5200U 2.20 GHz, GPU NVIDIA Geforce 920 com 2 GB de vídeo e 8 GB de RAM;

Os experimentos foram todos executados no ambiente do Laboratório de Sistemas Embarcados e Robótica da UFPB (LaSER).

Acerca dos resultados, sempre que possível, a metodologia visará avaliá-los com base em dados provenientes da literatura científica. Onde se referir a implementações em GPU, será apresentado um comparativo com uma versão correspondente em CPU.

1.4 Contribuições

O trabalho procurou trazer como contribuições a especificação e implementação de um componente gerador de imagens, de forma que seja possível integrá-lo facilmente com possíveis ambientes de verificação funcional ou mesmo efetuar uma verificação de forma distribuída e, quando possível, fazendo uso de dispositivos heterogêneos capazes de computação paralela, como *General Purpose Graphics Processing Unit* (GPGPU).

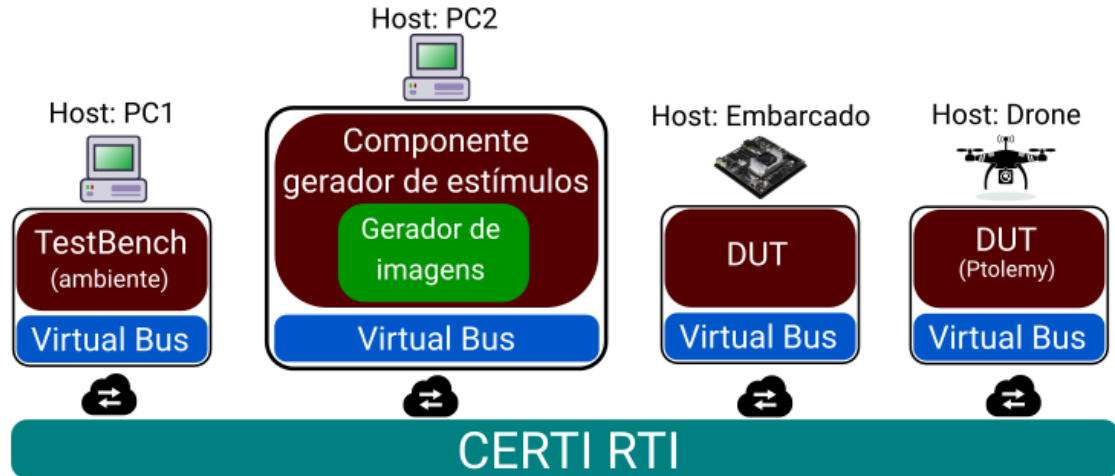
Também foram alcançados resultados acerca de novas possíveis formas de geração de

imagens para estímulos em verificação funcional, assim como traz informações sobre o desempenho obtido nas soluções encontradas e implementadas.

Os estudos deste trabalho resultaram em publicações no SBESC 2016 (VI Simpósio Brasileiro de Engenharia de Sistemas Computacionais) [Andrade et al. 2016], um artigo na revista JISA (*Journal of Internet Services and Applications*) [Silva et al. 2018], uma publicação no SBCCI 2018 (*31st Symposium on Integrated Circuits and Systems Design*) [Silva et al. 2018], além de um artigo que está em processo de finalização da escrita para submissão.

Em um sentido mais específico à linha de pesquisa desenvolvida no Laboratório LaSER da UFPB, temos uma ilustração demonstrando a contextualização da contribuição do presente trabalho. Na Figura 1.1 é exibido um diagrama da infraestrutura de componentes em um ambiente de verificação funcional desenvolvido no laboratório LASER. Conforme ilustrado, o Certi [CERTI 2018], que é uma implementação de código aberto do padrão *High Level Architecture* (HLA) [Noulard, Rousselot e Siron 2009], trabalha como base para o VirtualBus.

Figura 1.1: Contribuições e componentes em contexto de verificação funcional.



Fonte: O autor.

O chamado Certi RTI, do inglês *Run-Time Infrastructure*, provê funcionalidades básicas para o VirtualBus trabalhar como *middleware* de intercomunicação entre os *hosts* em rede [Silva et al. 2018]. O TestBench é o ambiente que provê as funcionalidades para verificação funcional, podendo ter componentes dispersos e com processamento em GPU. Esses últimos são resultado dos trabalhos [Silva et al. 2018] e [Andrade et al. 2016]. Os *hosts* com

componentes chamados de DUT, do inglês *Design Under Test*, são os que serão efetivamente verificados. E finalmente, em um dos *hosts*, acoplado ao componente Gerador de Estímulos do TestBench, está o componente Gerador de Imagens, principal fruto do presente trabalho.

Capítulo 2

Fundamentação Teórica

Nesse capítulo serão abordados temas fundamentais para a melhor compreensão do trabalho. Serão discutidos os conceitos essenciais relacionados ao desenvolvimento da proposta de geração de imagens.

Tanto o projeto desenvolvido, quanto os experimentos dessa pesquisa, utilizam como base duas tecnologias a serem descritas nesse capítulo. Essas são: redes neurais generativas adversárias, mais popularmente conhecido na comunidade científica como GAN; e tecnologias de programação paralela para GPU de uso geral, também conhecida como GPGPU.

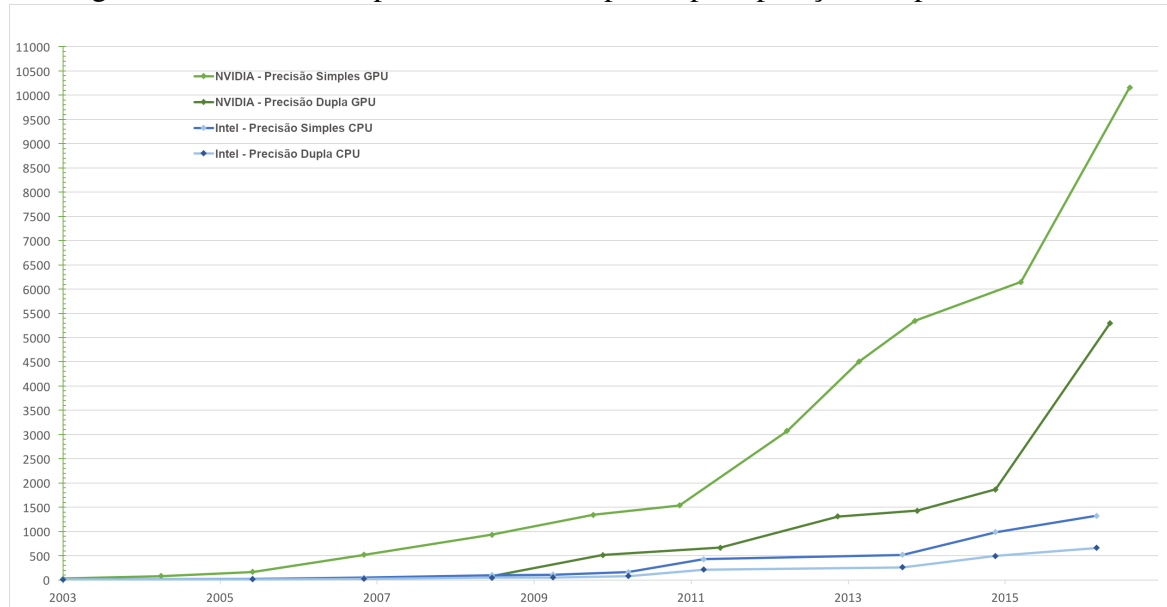
2.1 Computação Paralela e Programação em GPU

O uso de programação paralela já se tornou bastante popular. Principalmente com o advento dos multiprocessadores e sua programação *multithreading* a partir de APIs como *Open Multi-Processing* (OpenMP) [Dagum e Menon 1998] temos hoje um crescente uso de computação *multicore*. Porém, foi com a programação para GPU que o desempenho alavancou realmente. Já contamos também com tecnologias paralelas genéricas em relação ao tipo de *hardware*. Podemos citar a *Open Computing Language* (OpenCL) [Khronos 2018], que é uma plataforma com suporte para alguns tipos de multiprocessadores, como CPU, GPU, FPGA, entre outros.

Além de outros tipos de multiprocessadores, temos muito bem popularizada a GPU. A plataforma para programação de GPU que mais vem trazendo inovações e melhorias é a CUDA da NVIDIA [NVIDIA 2018]. Na Figura 2.1 está presente a evolução do poder de

processamento ao longo dos anos com informações do pico teórico de operações de ponto flutuante, tanto de GPUs da NVIDIA quanto CPUs da Intel.

Figura 2.1: Gráfico comparativo de desempenho por operações de ponto flutuante.



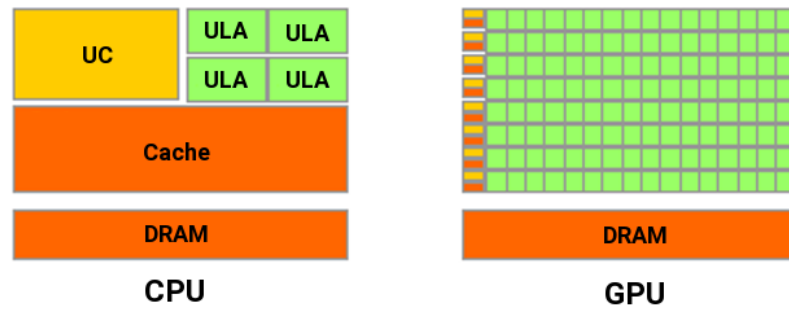
Fonte: Adaptado do Guia de programação paralela da NVIDIA [NVIDIA 2018].

O pico teórico em Operações de Ponto Flutuante por Segundo (FLOPS) refere-se ao cálculo da quantidade de operações de ponto flutuante por segundo de acordo com as diversas plataformas da NVIDIA ao decorrer dos anos, bem como em comparação com processadores da Intel da mesma época. É visível a superioridade das GPUs quando necessárias muitas operações de ponto flutuante. Isso se deve às características particulares de sua estrutura. Conforme mostra Figura 2.2, a estrutura de uma GPU comporta muitos núcleos, porém esses tem uma organização diferenciada, em geral com um aumento na quantidade das Unidades Lógicas Aritméticas (ULA).

Principalmente pela grande quantidade de unidades lógicas aritméticas é possível obter altas taxas de operações em ponto flutuante, exibidas em Giga-Flops na Figura 2.1. Outro fator crucial é sua largura de banda de memória. Além do mais, existem técnicas em que é possível otimizar a velocidade de acesso à memória em aplicações paralelas a partir do uso adequado dos tipos de memória disponíveis [NVIDIA 2018], melhorando ainda mais os resultados.

Conforme exibido na Figura 2.3, as GPUs há muito já vinham sendo equipadas com uma

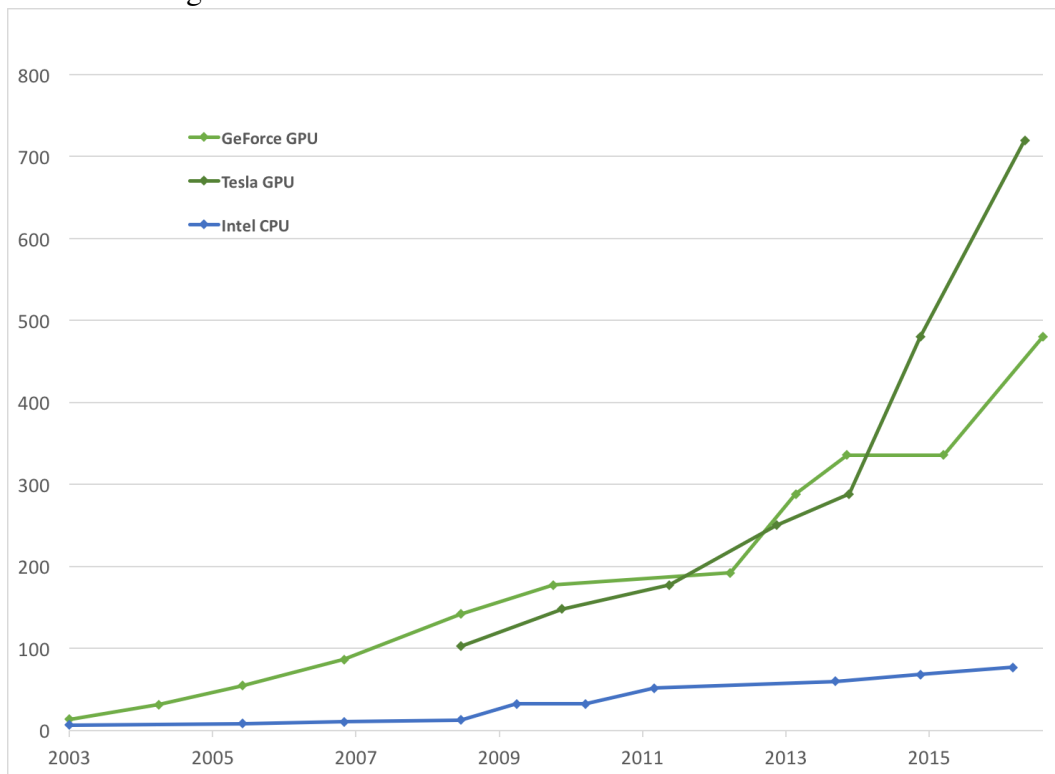
Figura 2.2: Arquitetura CPU e GPU. Mostrando a organização geral das Unidades de Controle (UC), Unidade Lógica e Aritmética (ULA) e blocos de memórias (Cache e DRAM).



Fonte: Adaptado do Guia de programação paralela da NVIDIA [NVIDIA 2018].

largura de banda mais veloz. Esse gráfico que exibe essas informações contém os picos teóricos de largura máxima de banda de memória em GB/s, comparando ao decorrer dos anos as especificações de GPUs GeForce, GPUs Tesla e CPUs da Intel.

Figura 2.3: Velocidade de acesso à memória CPU vs GPU.



Fonte: Adaptado do Guia de programação paralela da NVIDIA [NVIDIA 2018].

Em resumo, a especificação de uma implementação de programa que é executado em uma plataforma paralela, como CUDA, é chamado de *kernel* (esse termo será utilizado algumas

vezes ao decorrer desse trabalho). Para que seja possível rodar um *kernel*, é necessário configurar alguns parâmetros. Por exemplo, o número de *threads* que irão rodar o programa, e então a configuração de como essas *threads* estarão organizadas em grupos. Cada grupo de *threads* em CUDA é chamado de *block* e cada bloco é agrupado em um *grid*.

Essa configuração é necessária, pois, à medida que o hardware onde será executado pode ser diferente, o número de núcleos tende a variar bastante, conforme já indicava o estudo em [Stone, Gohara e Shi 2010]. Além do mais, via de regra, existem diversas formas de paralelizar um mesmo programa a ser implementado.

Existem várias maneiras de alcançar o paralelismo, bem como diferentes técnicas, mas, em geral, executar uma aplicação com tecnologia paralela consiste basicamente em realizar as seguintes etapas:

- Receber dados de entrada;
- Alocar memória necessária para executar operações no dispositivo (GPU ou outro multiprocessador em questão);
- Copiar dados da memória principal para a memória do dispositivo;
- Invocar o programa chamado *kernel* (que será executado nos núcleos do dispositivo em questão);
- Ao final da execução no dispositivo, copiar para a memória principal os dados de saída computados.

Existe uma exceção para essa etapa de copiar os dados da memória principal da CPU para a GPU, pois alguns dispositivos, em especial plataformas de computação embarcada, tem a memória compartilhada. Ou seja, a mesma memória é compartilhada entre CPU e GPU. Esse é o caso das plataformas Jetson da NVIDIA.

Conforme exposto, GPUs lidam muito bem com tarefas de computação intensiva, inclusive, já são bem utilizadas para uso em verificação funcional de sistemas e circuitos integrados de larga escala, conforme trabalhos [Perotti, Garcez e Boella 2014], [Bertacco e Chatterjee 2011], [Deng 2010] e também [Peng et al. 2018].

Em outros trabalhos, como Macri et al [Macri et al. 2015], é demonstrada a utilização da estrutura OpenCL em relação à execução de projetos de simulação, assim como também

é largamente utilizado de CUDA. A maior vantagem em usar o OpenCL para a proposta desse trabalho, seria a vasta compatibilidade com hardware heterogêneo, visto que OpenCL é independente das GPUs da NVIDIA, mesmo tendo suporte às mesmas.

Em alguns experimentos de simulações, o foco é conectar componentes diferentes, como alguns trabalhos que integram o processamento paralelo massivo em componentes de simulação por FPGA ou GPU [Macri et al. 2015], [Weber et al. 2011]. No entanto, nos casos que usam GPU, há um custo computacional no uso do OpenCL quando comparado com a tecnologia CUDA [Weber et al. 2011]. Essa é a desvantagem em contrapartida com a flexibilidade quanto ao *hardware* em que se pode rodar o OpenCL.

2.2 Redes Neurais Artificiais e Deep Learning

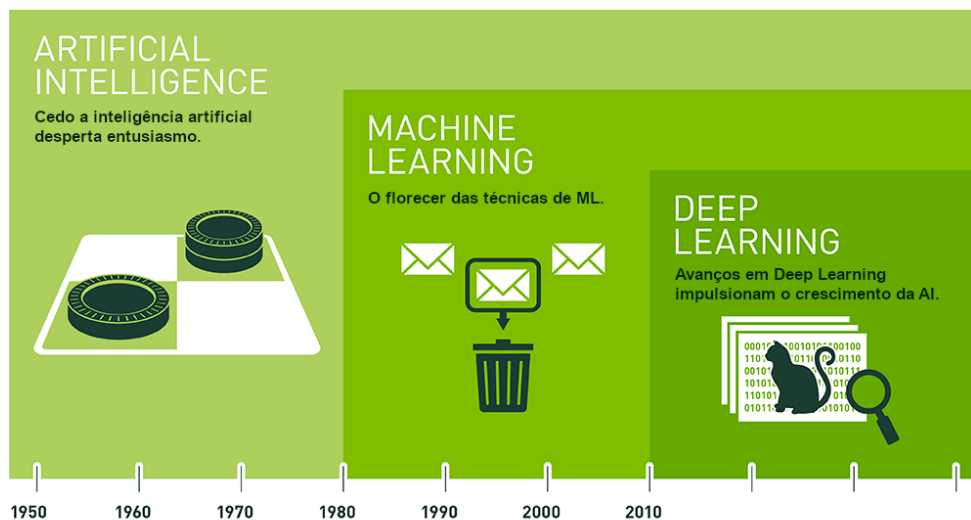
Uma questão que pode surgir ao olhar para essa seção é: por que usar *Deep Learning* (DL)? A resposta tem alguns pontos importantes, sendo o primeiro que DL consegue tratar com efetividade alguns problemas complexos que nós seres humanos há tempos levamos vantagem sobre os computadores. Um simples exemplo é a identificação visual de objetos, conforme demonstrado ao longo da visão geral no trabalho [Schmidhuber 2014]. A partir de DL, tarefas como reconhecimento de padrões chegaram a implementações em um novo patamar que os algoritmos tradicionais não alcançaram. Isso se deve à forma que as estruturas de DL conseguem trabalhar.

Outro ponto importante é que, basicamente, a ideia por trás de Aprendizado de Máquina, ou *Machine Learning* (ML), em geral é aprender padrões para repeti-los [Perotti, Garcez e Boella 2014]. Com isso, a partir de um mesmo tipo de rede neural é possível resolver uma vasta gama de problemas que sejam tarefas similares, dependendo apenas do modelo a ser treinado. Ainda que, em adicional, já existem estudos em novos tipos de redes neurais, como a *Dynamically Expandable Network* (DEN), que pode alternar dinamicamente entre capacidades para resolver tarefas distintas. Conforme nos mostra a pesquisa em [With 2018] isso é possível a partir do treinamento de uma estrutura que fica sobreposta, e ao mesmo tempo, compartilhada entre as tarefas. Resultados já foram obtidos com as bases de imagens MNIST, CIFAR-100 e AWA.

Esclarecendo mais os conceitos, *Deep Learning* pode ser entendido como uma parte de

Machine Learning (ML), que por sua vez é uma área da Inteligência Artificial (IA), do original inglês: *Artificial Intelligence*. Conforme exibido na Figura 2.4, a linha do tempo da IA já vem de antes dos anos 50 e do advento dos computadores eletrônicos, com muitas descobertas matemáticas que só puderam ser aplicadas na computação em décadas posteriores ao seu registro. O florescer do *Machine Learning* veio em seguida já com algumas aplicações práticas, mas pode-se dizer que avanços mais notórios e que quebraram mais barreiras vieram com aplicações de DL, conforme demonstrado no trabalho de [LeCun, Bengio e Hinton 2015].

Figura 2.4: Linha histórica: Inteligência Artificial, *Machine Learning*, *Deep Learning*.



Fonte: Adaptado da página de apresentação de ferramentas para Deep Learning da NVIDIA [NVIDIA 2018].

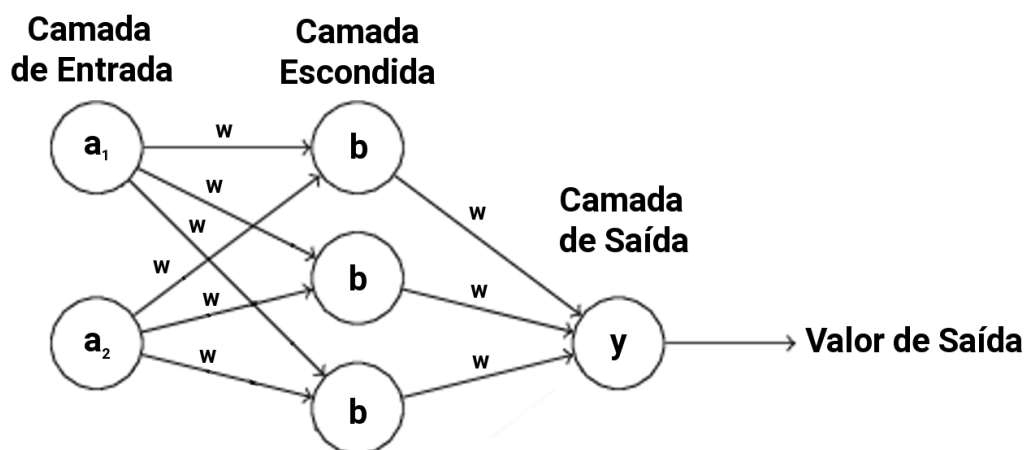
Grandes avanços e resultados impressionantes foram alcançados em trabalhos dos anos recentes, como por exemplo: na área de geração de imagens em [Karras et al. 2017] e [Horsley e Perez-Liebana 2017]; sintetização de fala [Arik et al. 2017]; transferência de estilos gráficos [Luan et al. 2017]; reconhecimento de objetos [Redmon e Farhadi 2016]; agentes de diálogos (em inglês: *Conversational Agents*) [Li et al. 2016]; entre outros. Porém, houveram descobertas impactantes para DL datadas no século passado.

Parte da inspiração dessa tecnologia veio de descobertas sobre o córtex visual em meados dos anos 60, mais especificamente dos trabalhos [Hubel e Wiesel 1959] e [Hubel e Wiesel 1962], conforme conta [Schmidhuber 2014]. Basicamente os trabalhos mostram que

foram observadas, no córtex visual dos gatos, células simples e células complexas, e essas disparavam estímulos em resposta a certas propriedades da entrada, como orientações de bordas.

Esse pano de fundo foi suficiente para criação de diversos tipos de redes neurais. Em analogia, temos as redes neurais artificiais com os neurônios recebendo entradas a serem processadas podendo disparar diversas saídas, conforme exibe a Figura 2.5. Uma **unidade** (*unit*) consiste em um dos nós de uma rede neural. As unidades recebem valores de entrada e processam com outros parâmetros, que geralmente são **pesos** e **bias**, para em seguida passarem por uma função que fornecerá a saída. Essa última função é chamada de **função de ativação**. Cada grupo vertical de unidades é chamado de **camada** (*layer*).

Figura 2.5: Exemplo de estrutura básica de uma rede neural artificial. Valores de entrada representados por "a", pesos (do inglês *weight*) representados por "w", *bias* indicado pelo "b" e "y" representa o valor de saída.

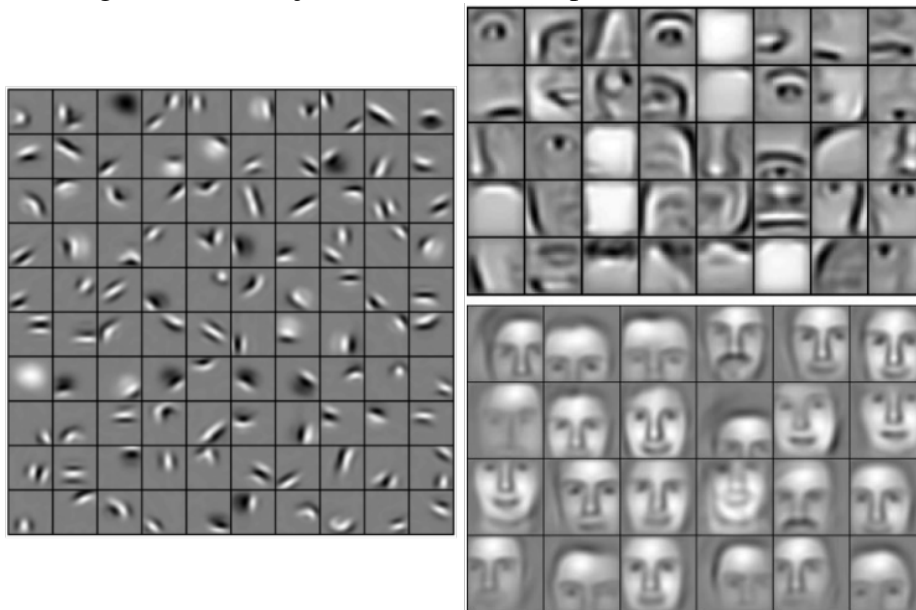


Fonte: Adaptado de imagem presente no trabalho [Nielsen 2018].

O chamado aprendizado ocorre basicamente a partir da atualização e correção desses parâmetros em cada unidade da rede neural à medida em que a mesma é treinada. DL leva vantagem sobre as outras técnicas porque à medida que são adicionadas mais camadas é aumentada a capacidade de aprendizado por características mais abstratas. Por exemplo, no caso de uma rede neural para reconhecimento de faces, a primeira camada pode reconhecer bordas, a seguinte pode reconhecer nariz, boca e olhos. Logo, a cada camada é aumentado o nível de abstração das características até finalmente termos filtros para faces, conforme

indicado na Figura 2.6.

Figura 2.6: Extração de características por níveis de camada.



Fonte: Adaptado de [Lee et al. 2011].

Cada característica pode ser associada com um filtro. O processo pode ser entendido da seguinte forma, quando um filtro é aplicado diretamente a aquela característica específica, por exemplo a imagem de um olho, dependendo da função de ativação configurada as unidades envolvidas devem gerar valores mais representativos para tal característica. Com esses bons indicadores são disparadas as próximas camadas que aplicarão outros filtros cada vez mais específicos, e assim é possível detectar com sucesso características de objetos [Dettmers 2015].

Esse é o princípio básico que a maioria das redes neurais seguem atualmente, pois o aprendizado conceitual adquirido pode ser entendido como os parâmetros que são atualizados à medida em que a rede neural é treinada. Conforme indica [Schmidhuber 2014], o ponto chave em DL é que os parâmetros das camadas que detectam características não são projetadas por engenheiros humanos: são consolidadas a partir de dados usando um procedimento de aprendizado de propósito geral. Esse procedimento em si é o que chamamos na prática de treinamento. Assim, *Deep Learning* permite a composição de modelos computacionais em múltiplas camadas de processamento, baseados em estruturas de redes neurais, para assim aprender características de representações de dados com múltiplos níveis de abstração

[LeCun, Bengio e Hinton 2015].

Apesar de, a princípio, a ação de adicionar mais camadas em uma rede neural poder adicionar maior capacidade de abstração na extração de características, o aumento de camadas é um dos fatores cruciais para caracterizar um problema conhecido como *vanishing gradient*. Esse problema é caracterizado por, durante a etapa de treinamento onde são calculados gradientes, é chegado um ponto em que não é possível atualizar os valores dos pesos (*weights*) devido ao valor proporcional da derivada parcial ser muito pequeno, impedindo assim que os valores dos pesos sejam atualizados. Esse problema é exposto com mais detalhes no trabalho [Squartini, Hussain e Piazza 2003].

Em DL temos dois principais tipos de aprendizado, ou treinamento, o aprendizado supervisionado (*Supervised Learning*) e o aprendizado não supervisionado (*Unsupervised Learning*). O treinamento não supervisionado trabalha apenas com dados não rotulados. Basicamente funciona como um aprendizado da estrutura dos dados, categorizando dados similares [Kumar e Garg 2017].

Diferentemente, o aprendizado supervisionado acontece quando a rede neural é treinada com dados rotulados. Ou seja, existem os dados a serem processado e um rótulo com a denominação do que são aqueles dados. De modo geral, dessa forma a rede neural vai aprendendo o que é cada representação, de acordo com os dados e os rótulos respectivos [Schmidhuber 2014]. Essa forma de aprendizado **supervisionado** foi a utilizada nesse trabalho, já que existem informações a cerca dos dados iniciais que devem alimentar as redes neurais utilizadas.

Interligando os dois principais temas desse capítulo, é possível falar sobre processamento em GPU de redes neurais com técnicas de DL. O uso de GPGPU associado com DL foi um dos impulsionadores na expansão das técnicas de DL, conforme também exposto no histórico presente no trabalho [Schmidhuber 2014]. Além do já presente uso de DL em técnicas de verificação funcional [Perotti, Garcez e Boella 2014].

Pode-se observar a forte presença de tecnologias DL em um dos maiores eventos de reconhecimento visual, o evento ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky et al. 2015]. É possível conferir as tecnologias utilizadas pelas equipes na páginas de resultados do evento (<http://image-net.org/challenges/LSVRC/2017/results>) e diversos tipos de redes neurais marcam presença forte no decorrer dos vários anos de realização do evento.

Com isso, DL tem se tornado uma tendência e possui uma comunidade crescente, com também muitos *frameworks* dando suporte, inclusive com fácil portabilidade para executar boa parte do processamento em GPU.

Dentre os vários tipos de redes neurais existentes, a rede neural utilizada nesse trabalho foi a do tipo *Generative Adversarial Network* (GAN) [Goodfellow et al. 2014]. A mesma consiste basicamente em duas redes neurais trabalhando de forma adversária, como o nome já sugere. Essas duas redes neurais são executadas em paralelo acirrando uma disputa, que consistem em um jogo de soma-zero (*zero-sum game*) [Raghavan 1994], onde uma rede gera dados como candidatos válidos e a outra tem o papel de avaliar se esses candidatos realmente são válidos. No caso da rede utilizada nesse trabalho, uma rede neural gera imagens de dígitos e a outra tenta descobrir se a imagem contém um dígito válido ou não. Com a repetição consecutiva dessa disputa, ambas as redes vão sendo treinadas com as características identificadas, otimizando os resultados ao decorrer do processo [Creswell et al. 2018].

2.3 Considerações Finais

Conforme exposto nesse capítulo, as principais tecnologias utilizadas nesse trabalho, apesar de relativamente novas e ainda apresentarem melhorias significantes anos após anos, já estão bem maduras. O processamento com GPGPU e computação paralela se ajusta muito bem para o trabalho com imagens, visto que é muito mais fácil paralelizar problemas que trabalham com matrizes de dados. Também é possível visualizar que o uso de DL é adequado para o processamento de características de imagens, viabilizando implementações de reconhecimento e implementações para geração de novas imagens. A partir desse embasamento, será exposto no capítulo seguinte alguns trabalhos que demonstram essa maturidade das tecnologias abordadas.

Capítulo 3

Trabalhos Relacionados

Nesse capítulo serão expostos para discussão alguns dos trabalhos relacionados mais relevantes para contextualização com o presente estudo. Durante a pesquisa à literatura, que buscou por estudos similares, foram encontrados muitos trabalhos que tratam da geração de imagens para testes de Turing público completamente automatizado para diferenciar computadores de humanos, conhecido originalmente como "*Completely Automated Public Turing test to tell Computers and Humans Apart*" (CAPTCHA). Aqui serão apresentados os mais relevantes para o contexto abordado.

O produto final gerado nesses processos de CAPTCHA tem, basicamente, a finalidade específica de dificultarem o reconhecimento das imagens geradas, ou seja, são geradas imagens mais difíceis de serem identificadas visualmente. Apesar de muitas técnicas acrescentarem ruídos, o processo termina por gerar figuras similares, mas com as mesmas características principais. Com isso, temos uma convergência que aproxima essa proposta com a proposta desse presente trabalho.

Além dos estudos mencionados também será apresentada uma revisão bibliográfica de trabalhos atuais, que demonstram o estado da arte relativos a avanços em tecnologias relevantes para esse estudo.

3.1 Literatura Visitada

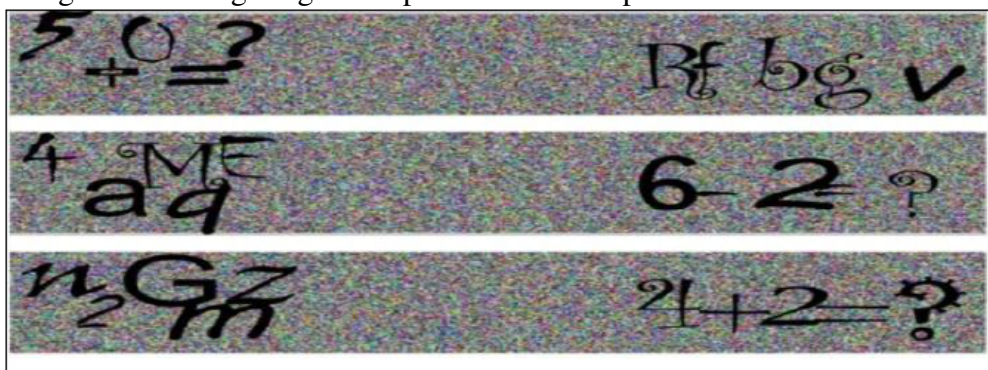
Durante a formulação do projeto de *software* que foi desenvolvido com esse trabalho, foram consultadas muitos artigos e alguns em especial nortearam para técnicas e tecnologias

específicas de destaque no contexto aplicado. Conforme mencionado, alguns trabalhos se relacionam por terem como objetivo gerar imagens similares a partir de distorções e alterações. Isso ocorre principalmente no caso dos artigos que tratam de CAPTCHA baseados em reconhecimento de caracteres.

Basicamente, foi para tentar evitar alguns problemas de segurança envolvendo robôs acessando aplicações *web* que os validadores CAPTCHA foram projetados [Obimbo, Halligan e De Freitas 2013]. Como o próprio acrônimo CAPTCHA sugere, essa validação diferencia humanos de computadores (robôs) para controlar a autorização de acesso a sistemas. No caso dos CAPTCHA baseados em texto essa validação ocorre através de uma imagem apresentada ao usuário, de forma que o mesmo precisa identificar quais os caracteres presentes na imagem. A grande parte das implementações desse tipo visam distorções e degradações das imagens originais dos caracteres. Além disso, existem outras técnicas complementares, que incluem a inserção de formas geométricas, linhas e adição de *background*.

Na Tabela 3.1 podemos ter uma ideia geral dos trabalhos relacionados, evidenciando as principais contribuições e aspectos de diversas áreas relevantes para o presente trabalho. Nessa listagem está presente o trabalho de [Kaur e Behal 2015] que trata de técnicas para geração de imagens com fins de testes CAPTCHA. Esse artigo aborda as possíveis melhorias para tornar um modelo de imagens com, supostamente, caracteres mais difíceis de serem reconhecidos. Além das outras técnicas já citadas, esse trabalho utiliza de técnicas para alterar o estilo da fonte, alinhamento e também altera a posição dos caracteres apresentados na imagem gerada. Na Figura 3.1 temos um exemplo das imagens geradas pela proposta do autor.

Figura 3.1: Imagens geradas para CAPTCHA pelos autores Kaur e Behal.



Fonte: Adaptado de [Kaur e Behal 2015].

Esse artigo conta com um comparativo de resultados obtidos pela proposta do autor com outros trabalhos, além de conter referências a outros estudos relevantes. Também existem exemplos como o trabalho [Obimbo, Halligan e De Freitas 2013] e [Vimina e Areekal 2009] que abordam CAPTCHA mais diversos. Os trabalhos de referência na área de geração de CAPTCHA influenciaram a implementação que será detalhada no Capítulo 4, em especial a geração de imagem por deformação, que está descrita na Seção 4.2.

Dando prosseguimento aos trabalhos listados na Tabela 3.1, temos o segundo item que utiliza de uma variação de GAN, mesclando redes convolucionais, com algumas alterações propostas pelos autores [Horsley e Perez-Liebana 2017], como mecanismo de rede neural para ambos componentes, Gerador e Discriminador da GAN utilizada nos experimentos. É uma abordagem que traz bons resultados com uma arquitetura relativamente simples. Nesse trabalho temos como característica alguns ruídos que permanecem em muitas das imagens geradas, dependendo da quantidade de épocas em que a rede neural foi treinada. Porém, para nossa aplicação, o ruído não seria bem um problema, algo que será abordado mais adiante no Capítulo 4.

Na Figura 3.2 temos uma pequena amostra dos resultados obtidos a partir das técnicas aplicadas pelos autores.

Figura 3.2: Sprites gerados utilizando técnicas de redes neurais pelos autores Horsley e Perez-Liebana.



Fonte: Adaptado de [Horsley e Perez-Liebana 2017].

Tabela 3.1: Comparação entre diferentes estratégias para geração de imagens.

Referência	Tecnologia	Aplicação	Pros.	Cons.
[Kaur e Behal 2015]	Independe de tecnologia	Geração de imagens para CAPTCHA	Discute técnicas de pequenas distorções na geração dos CAPTCHA.	Gera ruídos excessivos.
[Horsley e Perez-Liebana 2017]	DCGAN	Geração de sprites para jogos 2D.	Resultados positivos em relação à qualidade visual para a resolução proposta.	Ruídos ainda permanecem nas imagens geradas.
[Kanere, Mhatre e Jaiswal 2016]	Programação Paralela com linguagem R	Agrupamento de imagens visualmente semelhantes	Potencialmente uma boa fonte para novas bases de imagens.	O modelo pode agrupar imagens de contexto diverso.
[Karras et al. 2017]	GAN e metodologia de treinamento progressivo	Geração de imagens de celebridades falsas em alta resolução.	Alta qualidade nos resultados finais trabalhando com imagens HD.	O processo de treinamento da rede neural é bastante lento e consome muitos recursos.
[Osadchy et al. 2017]	Convolutional Neural Network (CNN)	Geração de imagens para CAPTCHA difíceis de quebrar.	É possível mensurar a quantidade de ruído aplicado.	A maioria das técnicas é limitada à aplicação em caracteres.

O terceiro artigo [Kanere, Mhatre e Jaiswal 2016] é um trabalho que utiliza de um algoritmo para agrupar em *datasets* as imagens visualmente semelhantes. O projeto efetua de forma paralela uma extração de características utilizando histogramas de luminosidade, para com isso trabalhar com matriz de distância e assim agrupar as imagens semelhantes. É um trabalho que não gera imagens, mas deixa uma ideia útil para um ponto de partida de como é possível selecionar imagens similares para servirem de ponto de partida. Também se relaciona com o trabalho por tratar de processamento paralelo com imagens.

O próximo da Tabela 3.1 é o trabalho de pesquisa de alguns pesquisadores da NVIDIA [Karras et al. 2017], onde foram geradas cerca de 30.000 imagens de falsas celebridades na resolução 1024 x 1024. A metodologia, basicamente, consiste na utilização de GAN tendo seu treinamento gradual em relação à quantidade de pixels das imagens geradas. O treinamento inicia com imagens de baixa resolução e gradualmente aumenta para imagens maiores. Assim, foi possível gerar imagens realmente convincentes conforme visível na Figura 3.3.

Figura 3.3: Amostras de imagens de celebridades falsas geradas por GANs.



Fonte: Adaptado de [Karras et al. 2017].

Apesar de toda essa qualidade nas imagens geradas, o custo de processamento para conseguir treinar e atingir esse desempenho ainda é um tanto alto.

O último trabalho [Osadchy et al. 2017] é uma pesquisa que tem a proposta de gerar imagens para CAPTCHA de forma que essas sejam difíceis de serem detectadas por computadores. Para isso os autores propuseram a adição de um ruído adverso que é difícil de ser removido pelas tecnologias atuais e não prejudica a visualização e reconhecimento das imagens pelos seres humanos (*immutable adversarial noise*). Com esse trabalho, obtemos uma possibilidade para adicionar à nossa metodologia, que é a inclusão de ruído. Porém, não foi possível aplicar ao gerador desenvolvido e exposto nos capítulos seguintes.

Trabalhos sobre a criação de *datasets* poderiam também ser citados nessa seção, porém a maioria deles tem em comum a captura manual de imagens. Como por exemplo o trabalho [Krizhevsky 2009] que inclusive expõe a metodologia utilizada durante o processo, bem como os padrões de especificações exigidos. Outros trabalhos, como [Cohen et al. 2017], fizeram tratamento e seleção de imagens em uma base de imagens existente para criar uma base de imagens mais específica.

3.2 Considerações Finais

Cada um dos trabalhos apresentados tem conhecimentos e visões distintas que se aplicam diretamente ao presente estudo. Cada referência trouxe uma inspiração em um aspecto diferente. De modo geral, o presente trabalho se diferencia por tentar mesclar técnicas de deformações e geração de imagens com linha de pesquisa no contexto de aumento de dados (*data augmentation*). A partir desses trabalhos apresentados é possível ter um panorama geral das tecnologias utilizadas no contexto do processo de desenvolvimento do gerador de imagens proposto. Essa proposta será apresentada em detalhes no capítulo seguinte.

Capítulo 4

Proposta de Geração de Imagens

Nesse capítulo será apresentada a proposta do autor para a geração de imagens, passando por duas principais visualizações: a conceitual, com uma descrição geral do ambiente; e a específica, para cada tipo de geração de imagem utilizado.

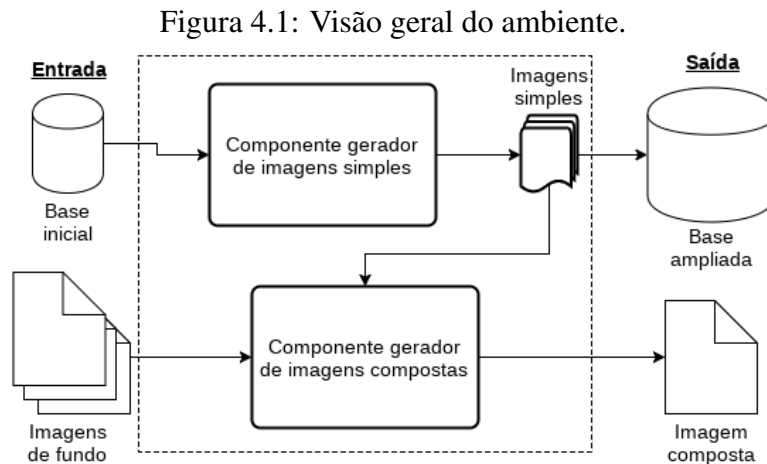
4.1 Visão Geral: descrevendo o ambiente

Em linhas gerais, a geração das imagens consiste em produzir novas imagens sintéticas a partir de um conjunto inicial, de forma que a metodologia prevê a produção de dois tipos de produto final, ambos derivados das imagens iniciais: a ampliação da base de **imagens com objetos simples** e a geração de **imagens compostas**.

As **imagens de objetos simples**, como o nome já induz, consistem em imagens contendo apenas uma unidade de objeto ou desenho. Por exemplo, um dos conjuntos de imagens utilizada como base inicial para os experimentos realizados nesse trabalho foi o conjunto com 5.000 amostras de imagens de dígitos escritos à mão, contidos como parte do pacote de amostras do OpenCV (localizado no diretório "*samples*" da instalação do OpenCV, no arquivo *digits.png*, vide Apêndice A). Nesse arquivo, cada imagem contém um dígito, de resolução 20x20 *pixels*. A ideia é partir de um conjunto de imagens simples, como o exemplo citado, produzindo novas imagens simples através das técnicas que serão explanadas mais a seguir.

Por outro lado, cada unidade das **imagens compostas** é constituída por uma determinada quantidade de objetos simples sobre um plano de fundo. De modo geral, a geração de ima-

gens compostas consiste, primeiramente, na produção das imagens de objetos simples para em seguida copiar esses objetos gerados em uma imagem de plano de fundo, formando assim a **imagem composta**. Ou seja, a base desse trabalho é o processo de geração das imagens com objetos simples e em adicional a produção de imagens compostas, conforme exibido na Figura 4.1.



Fonte: o autor.

Para criar as imagens compostas é necessário ter como entrada a imagem de *background*, além das imagens menores dos objetos que serão espalhados ao longo do *background*. O processo é bem simples, pois consiste apenas em copiar as imagens dos objetos, colando-as na imagem de fundo em coordenadas predefinidas ou aleatórias por meio de diversas *threads* através da GPU. Na prática, é feito a partir de um *kernel* CUDA que recebe como parâmetro as imagens simples, uma imagem de *background* e os *offsets*, gerados aleatoriamente ou não, para cada posição em que as imagens serão inseridas. Dessa forma, cada *thread block* CUDA é responsável por copiar uma imagem de objeto.

Conforme exposto na Figura 4.1 é possível atingir os dois tipos de produto final ao completar todo o processo. Temos uma base de imagens simples ampliada e a geração de imagens compostas, para utilização em verificações mais amplas.

O processo de geração de imagens de objetos simples ocorre de duas formas. A primeira, denominada *geração por deformação*, está ilustrada ao decorrer da Seção 4.2. A segunda forma é denominada *geração por rede neural* e tem suas particularidades descritas ao longo da Seção 4.3. Durante as seções de geração das imagens simples serão descritos também os estágios em que as imagens passam ao decorrer do processo, bem como as etapas executadas

ao decorrer desse processo.

Para implementar o ambiente proposto foram cogitadas algumas tecnologias. A decisão final ficou basicamente em torno da necessidade de um *framework* para manipulação de imagens, um *framework* para construção de redes neurais artificiais e também a utilização de uma plataforma para computação *multicore*. A escolha final foi OpenCV, TensorFlow e CUDA, respectivamente. Porém, um impasse ficou na escolha da linguagem de programação, pois o *framework* TensorFlow é nativo para linguagem Python, mas tem *binding* para programar em C++. Já no caso de CUDA é o contrário, nativo para C/C++ e com *binding* para Python, através do PyCUDA. A escolha final foi pela linguagem de programação Python por uma questão de facilidade de configuração do ambiente com as integrações entre as diferentes bibliotecas.

Tabela 4.1: Tecnologias utilizadas e suas aplicações em cada componente de geração de imagens simples.

–	Gerador por Deformação	Gerador Rede Neural	Suporte GPU
OpenCV	X	X	X
PyCUDA	X		X
TensorFlow		X	X

Conforme visível na Tabela 4.1, em ambos os componentes foi possível utilizar de computação paralela através do suporte para GPU provido pelos *frameworks* selecionados para o projeto.

4.2 Geração de Imagens por Deformação

Dos dois métodos de geração de imagens utilizados, o primeiro a ser tratado é chamado de Geração por Deformação. Ele foi assim denominado pois o processo consiste em deformar as imagens de entrada e assim produzir imagens de saída com características similares, porém com conteúdo diferenciado pelas deformações aplicadas.

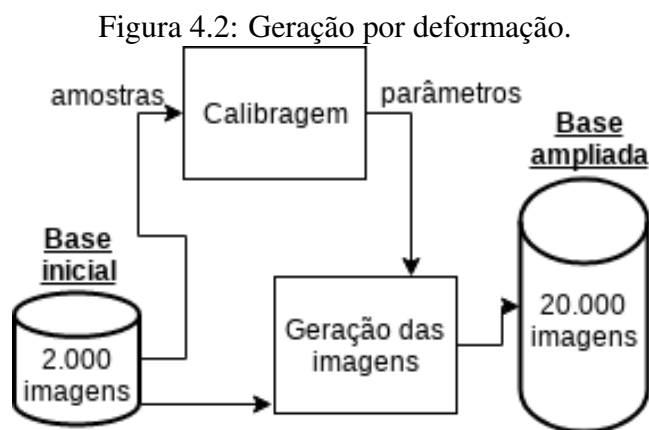
A motivação para criação desse método foi usar como ponto de partida algumas imagens já consideradas positivas, no caso de uma verificação. Ou seja, são imagens que realmente contém o que o algoritmo deve buscar. A linha de raciocínio base é criar versões alternativas

dessas imagens, deformando-as gradativamente para detectar até que ponto os algoritmos reconhecerão a imagem. Conforme já mencionado, a inspiração para essa implementação veio basicamente de trabalhos com geração de CAPTCHA.

O processo como um todo pode ser entendido passando pelas seguinte etapas:

- **Parametrização inicial**, onde são estabelecidos parâmetros como: imagens de entrada, aspectos de deformação e fator de ampliação;
- **Calibragem**, que visa obter automaticamente alguns parâmetros sensíveis dependendo das imagens iniciais e está descrito detalhadamente na Subseção 4.2.1;
- **Execução** das deformações, que é o processo em si de deformar as imagens de entrada, gerando novas imagens a partir das métricas estabelecidas nas etapas anteriores.

A *parametrização inicial* é quando são configurados os elementos indispensáveis para a geração. Conforme exemplo ilustrado na Figura 4.2, uma base inicial que possui 2.000 imagens de amostras de dígitos manuscritos foi configurada como entrada. Cada arquivo desse conjunto de imagens contém apenas um dígito escrito à mão. Nesse caso é possível configurar, por exemplo, *deformações de aspecto* como um ângulo de rotação, redimensionamento de escala e controle de brilho e contraste. Além do mais, as variações podem ser diversas e customizadas, bem como é possível a adição de ruídos arbitrários.



Fonte: o autor.

Também é possível perceber qual o *fator de ampliação* que foi aplicado nesse exemplo, pois 2.000 imagens foram fornecidas como entrada, e ao final o processo forneceu como

saída uma base ampliada com 20.000 imagens. Ou seja, nesse caso, foi configurado para gerar imagens com um fator de ampliação de 10x.

Na prática esse fator de ampliação pode ser configurado por aspecto de deformação. Por exemplo, pode ser configurado para gerar 2x a quantidade original de imagens apenas deformando por redimensionamento. Isso significa dizer que o componente estará configurado para produzir 4.000 imagens, onde ele deve gerar 4.000 números aleatórios que correspondem a uma escala. Essa escala podendo ser um número positivo ou um número negativo, que irá aumentar ou diminuir as imagens, respectivamente.

Para compor essa ampliação em um fator de 10x, é possível configurar diferentes proporções para deformações de aspectos específicos. É possível configurar um fator de 4x para o aspecto rotacionar, 2x para combinar os aspectos redimensionar e rotacionar e mais 4x combinando redimensionamento, alterações de contraste e brilho. Nesse caso, levando em consideração a base inicial de 2.000 imagens, teremos 8.000 imagens alteradas por rotacionamento, 4.000 por redimensionamento e rotacionamento e mais 8.000 com alterações de contraste, brilho e redimensionamento combinados, totalizando 20.000 imagens geradas.

Seguindo o fluxo da Figura 4.2, com os parâmetros iniciais configurados, primeiramente são selecionadas apenas algumas *amostras de calibragem*. Com essas amostras selecionadas, se dá então a execução de um *processo de calibragem* do componente deformador que deverá efetuar a geração da base ampliada. A partir desse processo são obtidos os *parâmetros* calibrados para que a geração de imagens seja mais eficaz no sentido de gerar imagens com uma qualidade satisfatória no quesito reconhecimento dos objetos na base ampliada.

Conforme mostrado na Figura 4.2, o primeiro passo é obter amostras da base inicial para realizar o processo de calibragem. Com o processo de calibração completo, os parâmetros estão disponíveis para *executar a geração* das imagens, deformando as imagens de base inicial integralmente.

Outras configurações também são possíveis nesta fase, como a delimitação do tamanho e aspecto da base a ser gerada, assim como o tamanho das imagens a serem geradas, para padronizar a saída do sistema e também configurar um componente para verificação. Porém essas configurações também vão sendo inferidas a partir das entradas do processo.

Dessa forma, é executada a geração de imagens por deformação. As imagens passam por três estados: estado inicial; estado de calibragem; e estado final. Para esclarecer como foi

executado o processo de calibragem, o mesmo será descrito mais detalhadamente a seguir.

4.2.1 Processo de Calibragem

Em linhas gerais, o chamado processo de calibragem busca melhorar a taxa de acerto do algoritmo de referência usado sobre as imagens geradas como produto final. Isso é possível porque ele trabalha diretamente com os parâmetros que controlam as deformações das imagens. Durante esse processo parâmetros são alterados arbitrariamente para gerar algumas imagens e submeter essas ao algoritmo de referência. Isso acontece para em seguida analisar as saídas do algoritmo de referência em detrimento dos parâmetros utilizados para produzir a imagem distorcida. Na Figura 4.3 é apresentado um fluxograma descrevendo o processo de calibragem que acontece na geração de imagens por deformação.

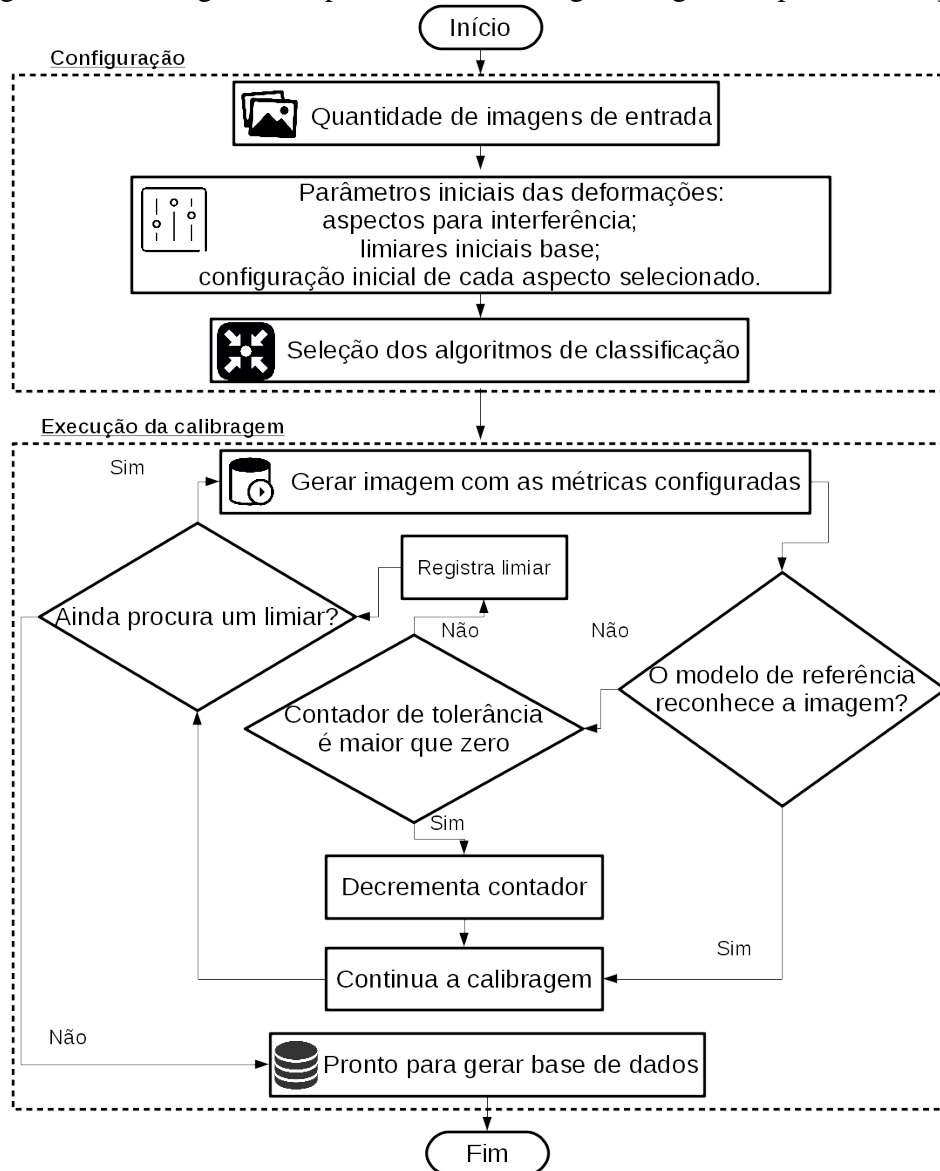
O processo pode ser dividido, basicamente, em duas etapas: **Configuração** e **Execução**. A configuração consiste principalmente na atribuição de informações, como o número de imagens que serão selecionadas como entrada para a execução da calibragem; seleção de quais parâmetros de deformação serão calibrados e também a configuração de um ou mais algoritmos de referência (os algoritmos que reconhecerão ou classificarão os objetos).

Por exemplo, nos experimentos executados foram usadas imagens de dígitos manuscritos, foram utilizados os algoritmos de referência K-Nearest Neighbor (KNN) [Bhatia e Ashev 2010] e Support Vector Machine (SVM) [Ma e Guo 2014]. Com essas configurações definidas, ele pode executar a calibragem, e assim o componente de deformação começa a gerar imagens a partir das métricas definidas.

As imagens produzidas são encaminhadas para o algoritmo de referência configurado, por exemplo, o SVM. O algoritmo verifica se reconhece o objeto na imagem gerada. Se sim, continua a calibragem com outros valores de métricas para deformações nas imagens geradas. Através desse processo de geração e verificação das imagens pelo algoritmo de referência, em algum momento, o gerador começa a produzir imagens que não passam no algoritmo de referência. Assim, as métricas mais recentes vão sendo registradas como limites de reconhecimento aceitáveis.

Durante esta etapa, uma variável de controle de tolerância também é usada para imagens não reconhecidas. Como indicado no fluxo de Execução de Calibragem, exibido na Figura 4.3, consiste em um contador que vai sendo decrementado. Desta forma, é possível compen-

Figura 4.3: Fluxograma do processo de calibragem do gerador por deformação.



Fonte: o autor.

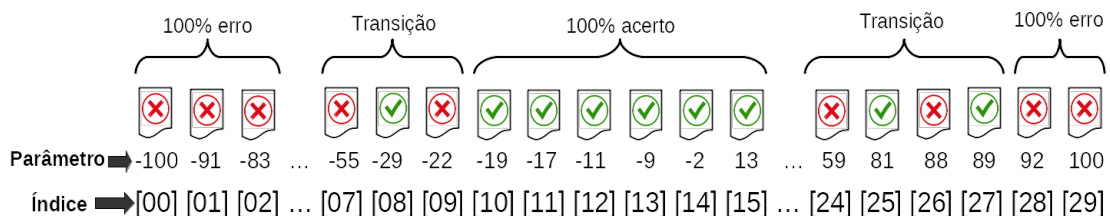
sar nos casos em que a geração defeituosa ocorre gradualmente e uma possível imprecisão do algoritmo utilizado.

Inicialmente é atribuído a esse contador de tolerância um valor baseado na quantidade de amostras selecionadas. Durante a calibragem, essa variável vai sendo decrementada à medida em que imagens geradas não são reconhecidas. Este valor é definido arbitrariamente como 10 % do número de amostras selecionadas para calibragem. Essa escolha pelo valor de 10 % é devido aos resultados iniciais com os algoritmos utilizados nos experimentos. Verificou-se que existia uma média de erro de 10 a 3 % e esse valor como parâmetro funcionou bem para corrigir essa margem. O contador funciona especialmente para regular quando uma calibragem está sendo executada com muitas imagens.

Por exemplo, na calibragem do tipo de deformação redimensionamento, com limiar inicial -100 (redimensionando para 100% menor que o original) até 100 (100% maior) utilizando 30 amostras. Nesse caso, são gerados 30 números inteiros aleatórios de -100 até 100, que serão os parâmetros para produção das 30 imagens de calibragem. Em seguida é aplicado o algoritmo de referência para verificar os limiares, observando os valores dos parâmetros gerados.

Conforme exibe a Figura 4.4, temos uma representação das imagens geradas nesse exemplo, sendo as com um "x" indicando erro (não reconhecidas pelo algoritmo) e as com "v" indicando acerto (reconhecidas pelo algoritmo). É perceptível que existe uma faixa de transição, em que são alternados acertos e erros além das faixas de 100% de erro e 100% de acerto.

Figura 4.4: Busca de limiares durante processo de calibragem.



Fonte: o autor.

Dessa forma, a calibragem vai descobrindo os limiares dos parâmetros para cada tipo de deformação. Por exemplo, na prática, quando se trabalha com dígitos, é possível que ao

rotacionar a imagem de um certo dígito o algoritmo de referência não possa mais reconhecê-lo, ou mesmo pode reconhecê-lo como sendo outro dígito. Na calibragem desse parâmetro, o gerador gira gradualmente até atingir o ponto onde isso ocorre, registrando assim o limiar de reconhecimento para o aspecto de rotação. O contador de tolerância nesse caso funciona dando uma margem de erro.

Em seguida, com essas métricas definidas é iniciado o processo de execução e as imagens que farão parte do banco de dados expandido são produzidas, completando assim todo o processo de geração de imagens por deformação.

4.3 Geração de Imagens por Rede Neural

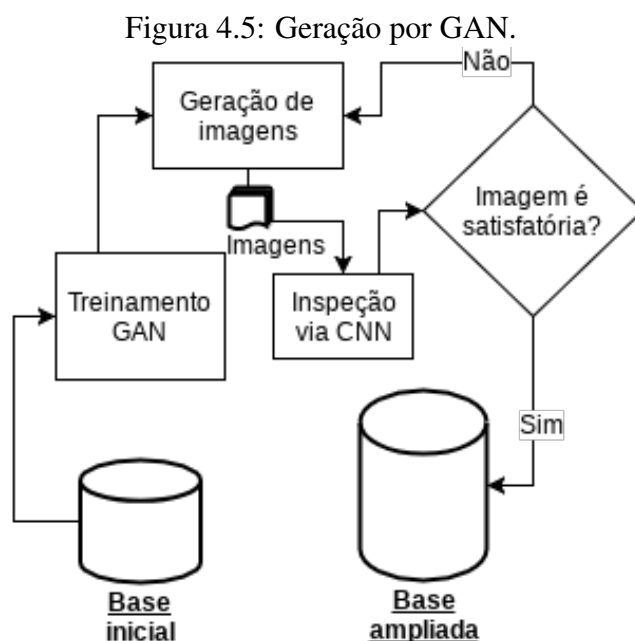
Essa outra forma de geração de imagens consiste em utilizar uma base de imagens já existente para treinar redes neurais e em seguida criar novas imagens que reproduzem as mesmas características gerais da base inicial. As imagens geradas nesse processo são produto de uma rede neural generativa, treinada para reconhecer os padrões da entrada e assim gerar uma saída com imagens similares.

Após a implementação do método de geração de imagens por deformação, explicado na seção anterior, surgiram alguns questionamentos. Por exemplo, foi alcançada uma variabilidade controlada das imagens geradas, mas essa era limitada aos tipos de aspectos de deformação. Além do mais, dependendo da categoria de imagens em que será aplicado, o método pode ter essa limitação ainda mais agravada.

Com essa motivação, foram levantados alguns trabalhos acadêmicos que utilizavam o estado da arte de outro tipo de tecnologia aplicada à geração de imagens. Trabalhos com redes neurais, como os comentados no Capítulo 3, serviram de grande influência para a aplicação dessas técnicas na implementação desse método de geração de imagens. Conforme poderá ser observado, esse método tem a principal vantagem de criar uma maior variabilidade nas imagens geradas, isso de forma não tão previsível como a variabilidade atingida pelas deformações efetuadas pelo método anterior.

É possível ter uma ideia geral do funcionamento dessa geração de imagens a partir do resumo ilustrativo contido na Figura 4.5. Basicamente o processo conta com duas redes neurais: uma do tipo GAN, onde após seu treinamento é utilizada a parte da rede generativa

para produzir as imagens; e também uma do tipo *Convolutional Neural Network* (CNN), que é utilizada apenas para controlar a qualidade das imagens geradas.



Fonte: o autor.

Conforme indicado na figura, a geração de imagens nesse processo começa a partir de uma base inicial sendo passada para o treinamento de uma rede neural do tipo GAN. Essa etapa é crucial, pois a qualidade das imagens que serão produzidas depende principalmente do treinamento dessa rede neural. Essa etapa também é a que requer um pouco mais de tempo de execução, porém, só precisa ser executada uma vez, pois o modelo do treinamento efetuado com todos os parâmetros atualizados pode ser salvo em disco.

Com a GAN treinada, já é possível começar a gerar as imagens. Porém, durante os primeiros experimentos foi verificado que é gerada uma grande quantidade de imagens defeituosas, mesmo após 500 épocas de treinamentos conforme será apresentado nos resultados. Ou seja, muitas vezes a rede neural generativa produz muitos ruídos. Para amenizar esse tipo de problema de forma mais prática, foi adicionada uma outra rede neural do tipo CNN.

Tentando controlar a entrega dessas imagens defeituosas, essa CNN analisa o que foi produzido antes de adicionar ao grupo de imagens da base ampliada, conforme exibido na Figura 4.5. Dessa forma, é possível decidir se a imagem é aceitável ou se será descartada para dar espaço para outra de qualidade superior.

Na prática, essa qualidade é mensurada da seguinte forma, a CNN tenta classificar a

imagem que foi gerada, também atribuindo a ela um valor como uma forma de pontuação. Pode-se dizer que esse valor serve para expressar numericamente quanto o algoritmo confia que as características da imagem correspondem às de um item válido em sua classificação. Assim, essa decisão de descartar ou não cada item ocorre baseando-se nessa pontuação atribuída pela CNN em detrimento de uma pontuação mínima exigida. Em resumo, a imagem é gerada pela GAN, em seguida é aplicada a função da CNN para obter uma pontuação. Caso não seja alcançado o valor mínimo nessa pontuação a imagem é descartada.

Para estabelecer esse valor mínimo, é feito um cálculo baseando-se nos resultados obtidos com uma quantidade arbitrária de amostras das imagens de entrada ("Base inicial" indicada na Figura 4.5). Lembrando que os dados de entrada estão todos rotulados e por isso devem servir como referência de qualidade.

Assim, após selecionada a quantidade de amostras, é aplicada a função da CNN, obtendo os valores que estão representados como x na Fórmula 4.1, bem como n representa o número de amostras. De acordo com o exibido, com os resultados da CNN aplicada às amostras selecionadas, é calculada a média aritmética e depois multiplicada por 0,9 para dar um desconto de 10% sobre a média obtida.

$$V_{min} = \left(\frac{x_1 + x_2 + x_3 \dots x_n}{n} \right) \cdot 0,9 \quad (4.1)$$

Com esse valor encontrado, V_{min} , é seguido o fluxo indicado na Figura 4.5. Se o valor obtido ao passar a imagem pela CNN for menor que o V_{min} encontrado, logo a imagem será descartada.

Esse processo de geração por rede neural pode ser usado apenas para estender uma base de imagens que já possuem um número considerável de itens. Isso é devido à necessidade de treinar as redes neurais envolvidas neste processo. Contudo, apesar do custo inicial para treinar as redes neurais, esse processo pode gerar imagens com as características da base inicial, mas com formatos até inéditos em comparação às imagens de entrada. Isso permite a produção, opcionalmente, de uma maior variedade de imagens para a base estendida.

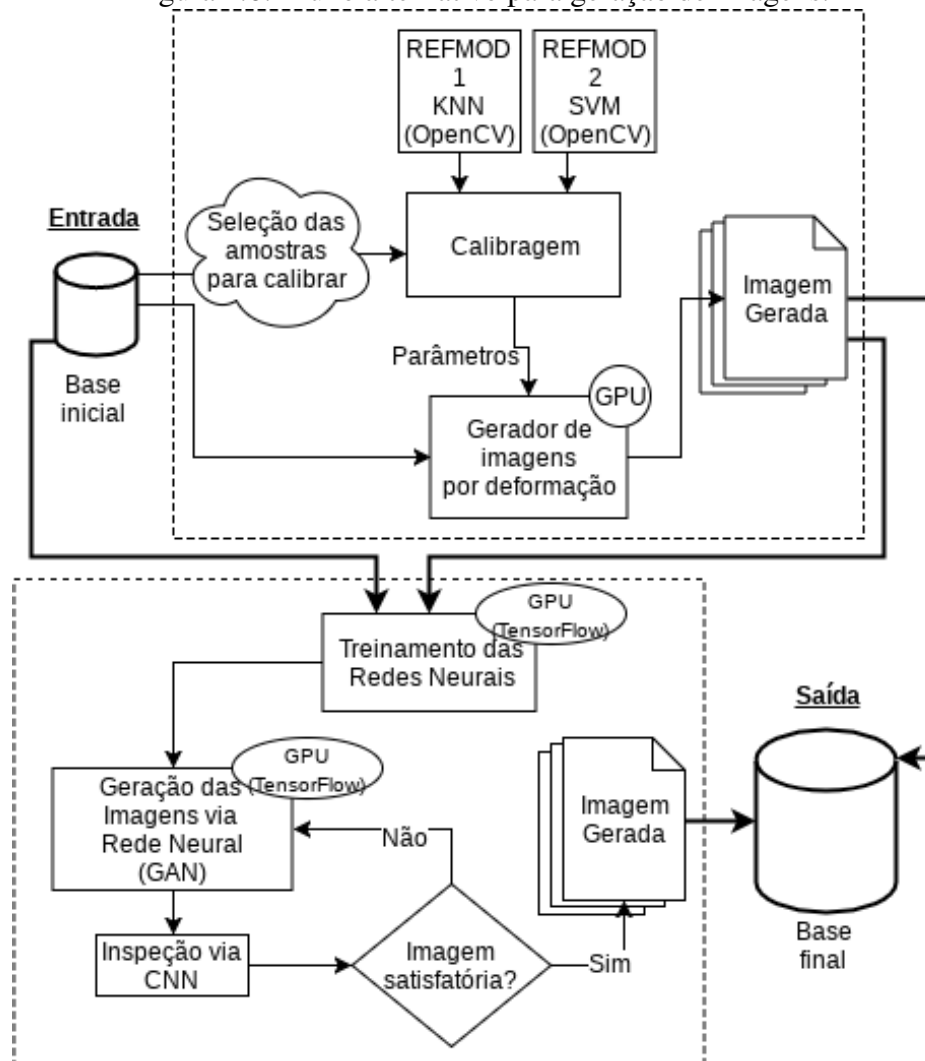
A seguir, algumas considerações sobre o treinamento das redes neurais envolvidas nesse processo de produção de imagens.

4.3.1 Treinamento da Rede Neural

A principal causa do surgimento de ruídos nas imagens é a execução da rede neural generativa com um treinamento insuficiente. A necessidade por grande quantidade de dados para treinar satisfatoriamente uma rede neural ainda é uma de suas características.

A execução desse processo de treinamento pode ser considerada como uma desvantagem desse método, pois esse processo demanda um tempo adicional. Claro que, dependendo principalmente do *hardware*, esse processo pode ser muito mais dispendioso ou não.

Figura 4.6: Fluxo alternativo para geração de imagens.



Fonte: o autor.

Um fluxo alternativo de geração de imagens pode ser traçado a partir do encadeamento dos dois processos aqui descritos. É possível gerar imagens por deformação (conforme Seção

4.2) e a partir dessas primeiras imagens, juntamente com as imagens de entrada, iniciar o processo de treinamento para geração pela GAN (Seção 4.3).

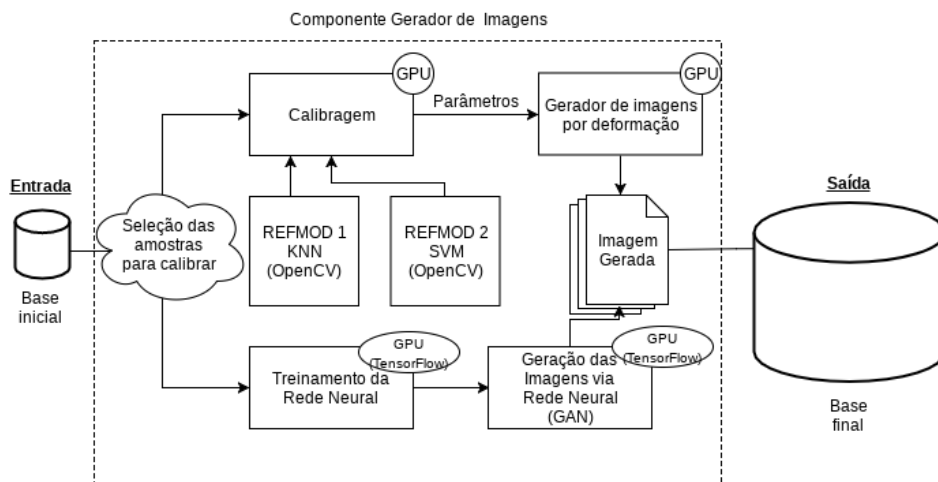
O treinamento ocorre de forma que a saída de um processo alimenta também a entrada do outro, conforme exibido na Figura 4.6. Dessa forma, é possível otimizar o treinamento das redes neurais principalmente nos casos em que há pouca quantidade de imagens de entrada.

Essa solução adotada funciona para mitigar o treinamento insatisfatório e também evita o problema de superajustamento dos parâmetros (*overfitting*) ao aumentar o número de épocas de treinamento das redes neurais, conforme discutido em [Hawkins 2004].

4.4 Considerações Finais

Na Figura 4.7 está o processo completo executado pelo componente gerador de imagens. Conforme exibido, é possível produzir imagens de ambos os processos de geração descritos.

Figura 4.7: Operação do componente gerador de imagens.



Fonte: o autor.

No fluxo normal, as imagens de entrada alimentam o treinamento da GAN, bem como servem para o processo de calibragem e para a geração por deformação. Assim, as imagens são geradas pela GAN e pelo processo de deformação, respectivamente. Em seguida, as imagens produzidas pelos dois processos são agrupadas para formar, como saída de todo o processo, a base de imagens estendidas. Na implementação utilizada, como pode ser visto na Figura 4.7, dois algoritmos de referência foram configurados para o processo de calibra-

gem. Ambos são algoritmos amplamente utilizados para tarefas de classificação e possuem implementações que estão presentes no OpenCV versão 3.x.

Também é possível perceber que a GPU foi usada para acelerar o processo de geração por deformação, o treinamento de GAN e para gerar as imagens através da rede neural generativa.

Capítulo 5

Avaliação Experimental

Nesse capítulo será apresentada inicialmente a construção do ambiente para que fossem executados os experimentos em si. Logo em seguida, apresenta as informações específicas sobre cada experimento executado e os resultados obtidos, finalizando com as considerações finais do capítulo.

5.1 Plano dos Experimentos

Após conceber a metodologia para geração de imagens, conforme exposto no Capítulo 4, foram implementados em um projeto os componentes do gerador de imagem. Com isso, serão descritos os experimentos que demonstram de forma prática a contribuição desse trabalho. Mas antes, nessa seção serão apresentados detalhes técnicos para tentar manter uma reprodutibilidade dos experimentos, além de uma visão geral de como foram planejados.

As outras seções desse capítulo incluem descrições de como foram executados os cinco principais experimentos. Alguns experimentos são focados em demonstrar pontos de otimização do próprio componente gerador implementado, como: diferentes resultados de acordo com a parametrização e as imagens de entrada, ou mesmo a otimização com versão em GPU. Demais experimentos focam em mostrar aspectos do produto final produzido, como: taxa média de reconhecimento das imagens geradas; qualidade das imagens produzidas e eficácia para utilização das imagens para treinamento de redes neurais.

Primeiramente, fica aqui esclarecido que os experimentos foram implementados, preferencialmente, com as versões das seguinte tecnologias:

- Linguagem de programação Python 2.7;
- OpenCV 3.1;
- PyCUDA v2017.1.1 com o CUDA 8.0;
- TensorFlow 1.4;
- Numpy 1.13.3.

Obviamente, os experimentos que visaram calcular o tempo de execução apenas em CPU não utilizaram da tecnologia paralela CUDA. Outro detalhe é que, para obter algumas das métricas citadas, as imagens geradas foram submetidas a um algoritmo de referência. Esse algoritmo é o que faz o reconhecimento das imagens. O algoritmo utilizado como padrão foi o *Support Vector Machine* (SVM), implementado pelo *framework* OpenCV. O SVM utilizado durante os experimentos foi preparado de acordo com o tutorial oficial disponibilizado pela equipe do OpenCV [Team 2018].

As implementações que visaram experimentos com a metodologia de geração de imagens por deformação (Seção 4.2) consistem, basicamente, de códigos Python manipulando imagens de entrada com OpenCV, Numpy e também kernels CUDA executados através do PyCUDA. Para demonstrar de forma acadêmica, foi suficiente utilizar apenas as deformações dos tipos: *redimensionamento*, *rotação*, controle de *brilho* e *contraste*. No caso das versões em CPU, as funções que implementam os citados tipos de deformações são executados também em OpenCV, ao invés de serem *kernels* CUDA.

Já os códigos para os experimentos relativos à geração por rede neural, conforme Seção 4.3, implementam redes neurais com o *framework* TensorFlow, além de utilizarem OpenCV e Numpy para manipulação das imagens. A implementação da GAN utilizada nos experimentos pode ser descrita da seguinte forma:

- Uma **rede neural generativa**, que foi construída com 4 camadas: a camada de entrada, contando com 100 unidades; 2 camadas escondidas, também chamadas de *hidden dense fully connected layers*, cada uma com 128 neurônios e ambas utilizando função de ativação *Leaky ReLU*; e por fim uma camada de saída, com 784 unidades e contando com tangente hiperbólica como função de ativação.

- Uma **rede neural discriminativa**, similar à implementação da generativa, também conta com 4 camadas no total. Tem entrada de 784, que é o tamanho da imagem trabalhada (28x28 *pixels*); mais duas camadas escondidas contando com 128 neurônios cada e também com função de ativação *Leaky ReLU*; já a camada de saída, conta com apenas uma unidade que passa por uma função *sigmoid*. Nesse caso a saída é binária, correspondendo à confirmação se a imagem processada pela rede neural contém o que é procurado ou não.

Já a rede neural CNN que foi utilizada em um dos experimentos, juntamente com a GAN, conta com uma camada de entrada, uma convolucional, depois uma de *pooling*, novamente mais uma convolucional seguida por outro *pooling* e finalizando com uma camada normal. Sendo essas camadas conforme descritas a seguir:

- Camada de entrada com 784 unidades;
- Camadas convolucionais contando com filtros de 5x5 *pixels*, uma processando 32 características (ou também chamadas de *features*) e outra com 64, com função de ativação ReLU.
- As camadas de *pooling* filtram 2x2, utilizando preenchimento com zero nas bordas.
- Camadas normais com 1024 unidades e também utilizando função de ativação ReLU.

Ambas as redes neurais que compõem a GAN, quanto a rede neural CNN utilizada, trabalham com valores *float64*. Isso porque, são obtidos melhores resultados com esse tipo de dado, em contrapartida com o tipo *int32* que até pode parecer ser mais óbvio por ocupar menos memória. Com isso, todas as imagens trabalhadas foram tratadas com o tipo *float64*.

Conforme planejado, os experimentos foram executados nas dependências do Laboratório de Sistemas Embarcados e Robótica (Laser [LASER 2018]), no Centro de Informática da UFPB. O hardware completo utilizado nos experimentos que foram executados inclui os itens a seguir:

1. Computador Dell com processador Core i5-5200U a 2.20GHz, RAM DDR3 de 8GB de 1600MHz, com placa de vídeo NVIDIA GeForce 920M com 384 núcleos CUDA a 954MHz e 2GB de memória de vídeo dedicada à frequência de 1800MHz;

2. Plataforma para sistemas embarcados da NVIDIA, Jetson TX1 equipada com a GPU NVIDIA Maxwell, de 256 cores CUDA, CPU Quad ARM® A57/2 MB L2 e 4 GB de memória RAM compartilhada;

Durante a execução de alguns experimentos, a fim de simular uma verificação de forma distribuída, foi utilizado o Virtual Bus [Silva et al. 2018], integrando os componentes de *hardware* listados, também com o conhecimento adquirido em [Andrade et al. 2016]. Esse Virtual Bus é baseado no padrão HLA (IEEE *standard* 1516) e foi utilizado para intercomunicar a Jetson TX1 com o PC, especificados na lista de *hardware* anteriormente. Para tanto, foi utilizada a implementação CERTI, versão 3.5.1 do RTI. Mais detalhes sobre CERTI e sua utilização podem ser encontrados no trabalho [Noulard, Rousselot e Siron 2009]. Com isso, temos a seguir na Tabela 5.1 os quadros em que foi utilizado cada item de *hardware* e também informando sobre o Virtual Bus.

Tabela 5.1: Uso do Virtual Bus e itens de *hardware* nos experimentos realizados.

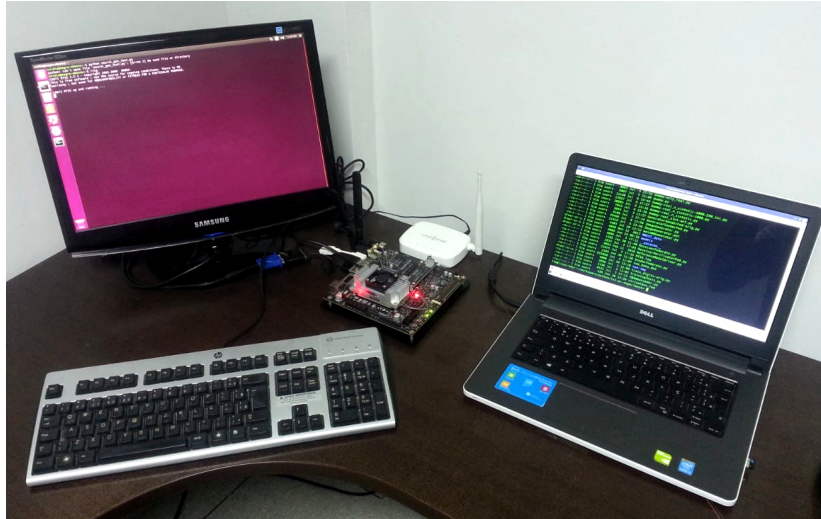
–	PC	Jetson TX1	Virtual Bus
Experimento 1	X	X	X
Experimento 2	X	X	X
Experimento 3	X	X	X
Experimento 4	X	X	X
Experimento 5	X		

Conforme é possível checar na Tabela 5.1, enquanto o Experimento 5 foi executado localmente apenas no PC, os outros experimentos utilizaram ambos os *hosts* disponíveis. Os experimentos de 1 a 4 foram executados da seguinte forma: o componente gerador, localizado na Jetson TX1, produzia as imagens e as enviava para o PC através do Virtual Bus. Ao receber as imagens, essas eram submetidas ao algoritmo de referência e assim incrementalmente computado o resultado de acordo com o experimento em específico.

Para ilustrar como foram executados os experimentos, temos na Figura 5.1 o ambiente real em que os experimentos foram executados, dando uma ideia de como é visualmente a execução.

A seguir, nas próximas seções descritos os pontos particulares de cada experimento.

Figura 5.1: Foto dos dois *hosts* utilizados nos experimentos executados nas dependências do laboratório LASER. À esquerda a plataforma de sistemas embarcados Jetson TX1 e à direita o PC.



Fonte: o autor.

5.2 Experimento 1: calibragem

A fim de observar o comportamento do gerador de imagens por deformação, esse experimento examinou algumas possibilidades de parametrização da calibragem. Foi implementado a fim de identificar a configuração, que posteriormente, proporcionou a geração de imagens com uma taxa de acerto mais alta.

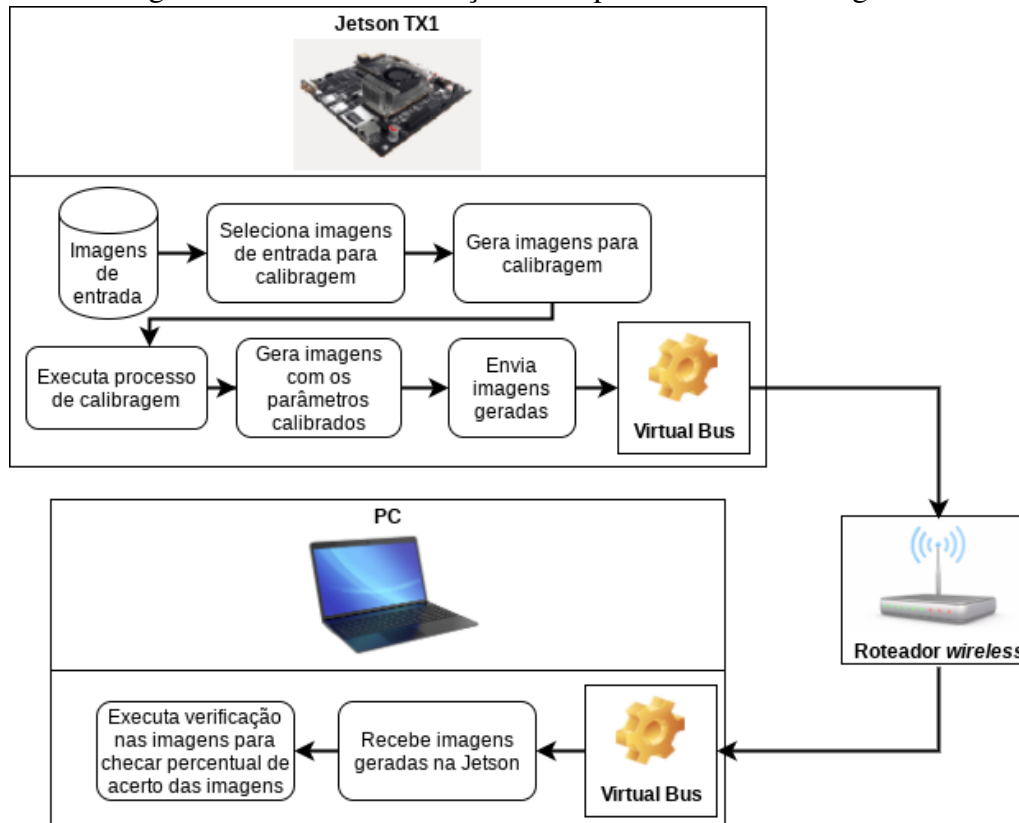
A citada parametrização refere-se à quantidade de imagens utilizadas para o processo de calibragem. A taxa de acerto, refere-se à média em que o algoritmo reconhece as imagens finais geradas. Isso quer dizer que, dependendo dessa configuração na calibragem, ao final as imagens geradas por deformação podem ser mais ou menos reconhecidas pelo algoritmo de referência.

Conforme descrito anteriormente (Subseção 4.2.1), o processo de calibragem necessita de algumas imagens de entrada. A ideia geral desse experimento foi realizar algumas execuções do processo de calibragem, variando a quantidade de imagens usadas como entrada, para em seguida observar o resultado obtido.

O fluxo geral desse experimento está exposto na forma de um diagrama na Figura 5.2. O processo de calibragem bem como a geração das imagens, para verificar os diferentes resultados, foi efetuada na Jetson TX1 e enviada para um PC verificar com o algoritmo de

reconhecimento dos caracteres. Ao efetuar a calibragem, foram geradas 2.000 imagens em cada execução, em seguida enviando todas as imagens via Virtual Bus para o PC computar os resultados de acordo com o reconhecimento das imagens pelo algoritmo de referência.

Figura 5.2: Fluxo de execução do Experimento 1: calibragem.



Fonte: o autor.

Destas execuções, seguem aqui nesse trabalho as três mais relevantes para análise:

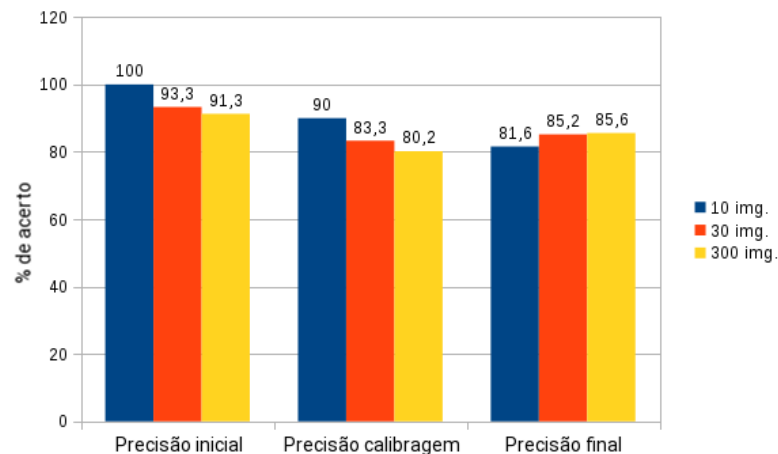
- Usando 10 imagens como entrada;
- 30 imagens como entrada;
- 300 imagens como entrada.

Levando em conta a taxa de reconhecimento das imagens geradas como produto final, será possível observar os diferentes resultados obtidos de acordo com cada configuração inicial da calibragem.

5.2.1 Resultados Obtidos

Como resultados desse experimento foi possível produzir a Figura 5.3, onde são apresentados os percentuais de acerto por fase do processo. Dessa forma, a ideia principal é tornar possível observar o impacto das mudanças em cada etapa, incluindo as imagens finais geradas por cada processo executado.

Figura 5.3: Porcentagem de precisão nos diferentes estados durante a calibragem, no Experimento 1.



Fonte: o autor.

Os resultados estão agrupado por: Precisão Inicial, com resultados do algoritmo de reconhecimento de dígitos aplicado nas imagens originais e sem qualquer processamento; Precisão calibragem, que se refere às imagens geradas durante a calibragem a fim de obter os valores de limiar (conforme Seção 4.2.1); e finalmente a Precisão final, que são os resultados do algoritmo de referência aplicado nas imagens geradas como produto final.

O primeiro grupo serve para dar uma referência do estado das imagens reais. Como esperado, as imagens finais de todos os três processos executados (10, 30 ou 300 imagens) tem uma precisão menor que o estado inicial, visto que a deformação das imagens também adiciona uma certa dificuldade para o algoritmo reconhecer as imagens. Ou seja, tomando como exemplo a execução com 30 imagens iniciais (exibida na cor laranja no gráfico da Figura 5.3), dessas 30 foram geradas mais 30 imagens sintéticas durante a calibragem, onde apenas 25 (85,3%) foram reconhecidas pelo algoritmo. Ao final foram geradas 2.000 imagens a partir desse gerador calibrado e dessas, apenas 1.704 (85,2%) foram reconhecidas.

Assim, observando a diferença entre o processo de calibragem com 30 e 300, não encon-

tramos uma diferença muito significativa no que diz respeito ao reconhecimento das imagens finais geradas (exibido no gráfico no grupo Precisão final). Podemos concluir assim que não é necessária uma quantidade muito grande de imagens para efetuar a calibragem, ou seja, não é gasto muito tempo com esse processo de preparação.

Este resultado vem dos três principais passos realizados no Experimento 1: o primeiro, calibrando com apenas 10 imagens; o segundo contando com 30 imagens e o terceiro e último com 300 imagens para calibragem. A partir desses dados, é possível perceber que não são necessárias muitas imagens para fazer uma calibragem satisfatória, mas com apenas 10 imagens os resultados não são os melhores. Pois, neste caso, temos 10 classes para reconhecimento (os 10 dígitos 0 a 9). Também é visível que, em geral, quando você tem uma alta porcentagem de correspondências durante o processo de calibração, não é possível explorar muito bem os pontos em que o algoritmo começa a falhar. Isso pode levar à geração de uma base de imagens final com baixa porcentagem de acerto pelos algoritmos utilizados, ou seja, imagens mais difíceis de serem reconhecidas.

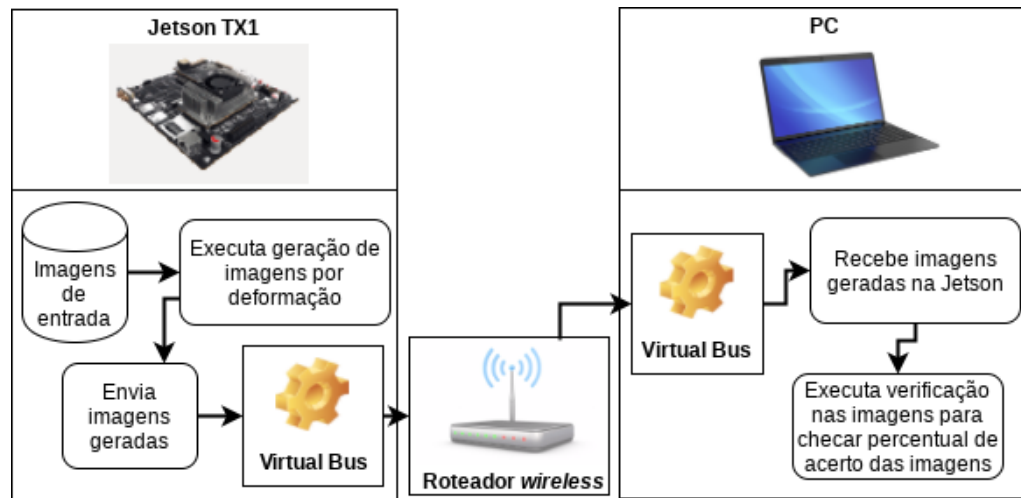
5.3 Experimento 2: análise por tipo de deformações

Ao replicar muito uma mesma imagem, sem repeti-la idêntica, é preciso aplicar vários tipos de deformações para se obter uma maior variedade. Porém, quando deformada em excesso, pode acarretar na má formação dos objetos nas imagens. Para verificar isso, esse experimento visou executar o processo de geração de imagens em diferentes quantidades e separando tipos de deformações. Em seguida, as imagens foram submetidas ao algoritmo de reconhecimento.

Conforme Figura 5.4, as imagens geradas na Jetson TX1 foram enviadas para o PC executar o reconhecimento dos caracteres com o algoritmo de referência, computando assim os resultados.

Na prática, inicialmente partiu de uma base de 2.500 imagens reais, para começar gerando 2.500 imagens sintéticas, para em outra execução gerar 5.000 e em seguida 10.000 imagens. Cada uma dessas execuções tinha como base as mesmas 2.500 imagens reais. Assim, é possível observar as taxas de reconhecimento presentes nessas bases de imagens geradas, bem como verificar qual deformação tem o maior impacto na taxa de acerto, com-

Figura 5.4: Fluxo de execução do Experimento 2: análise das deformações e quantidade de imagens.



Fonte: o autor.

parando os resultados obtidos.

5.3.1 Resultados Obtidos

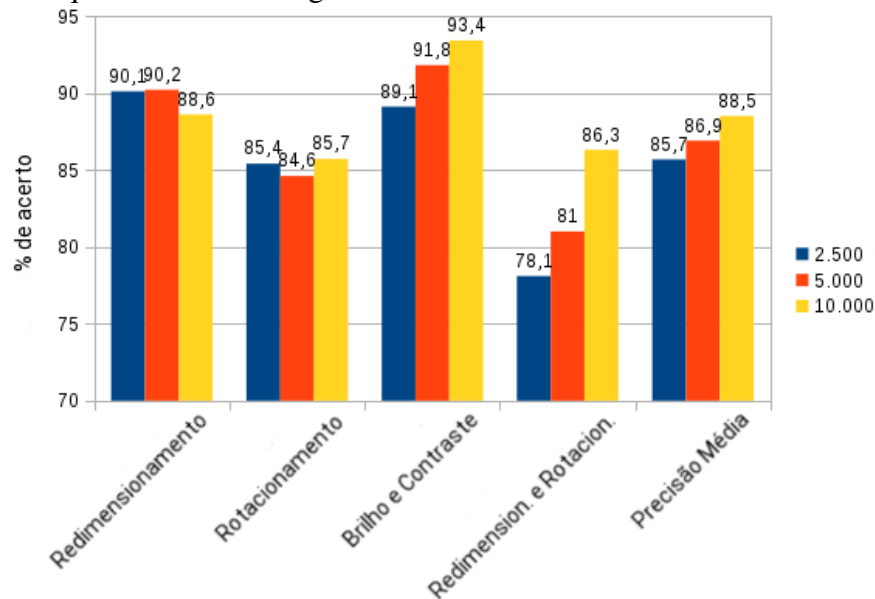
Na Figura 5.5 há resultados dos experimentos durante as execuções de gerações de imagens, no Experimento 2.

Esse gráfico apresenta a porcentagem de acertos por base de imagens gerada, agrupadas por método de deformação: redimensionamento, rotacionamento, brilho e contraste, além de redimensionamento e rotacionamento simultâneo.

Claramente no gráfico, os aspectos de deformação que mais dificultam o algoritmo de classificação são o de rotação e o redimensionamento. Isso acontece porque esses aspectos, especialmente quando combinados, podem alterar muito os recursos das imagens, o que pode tornar o reconhecimento mais desafiador.

O percentual de acerto mais baixo, na geração de 2.500 imagens (na cor azul no gráfico da Figura 5.5), nesse caso pode ter sido por conta da menor diversidade de imagens geradas. Isso ocorre porque ao gerar mais imagens, o gerador termina produzindo imagens com mais diversidade mas com pouca diferença umas das outras. Assim, por mais que a geração dos parâmetros de deformação seja randômica, é menor a chance serem geradas imagens nos extremos dos limiares.

Figura 5.5: Percentual de acerto no Experimento 2, durante a geração de bases de imagens com diferentes quantidades de imagens.



Fonte: o autor.

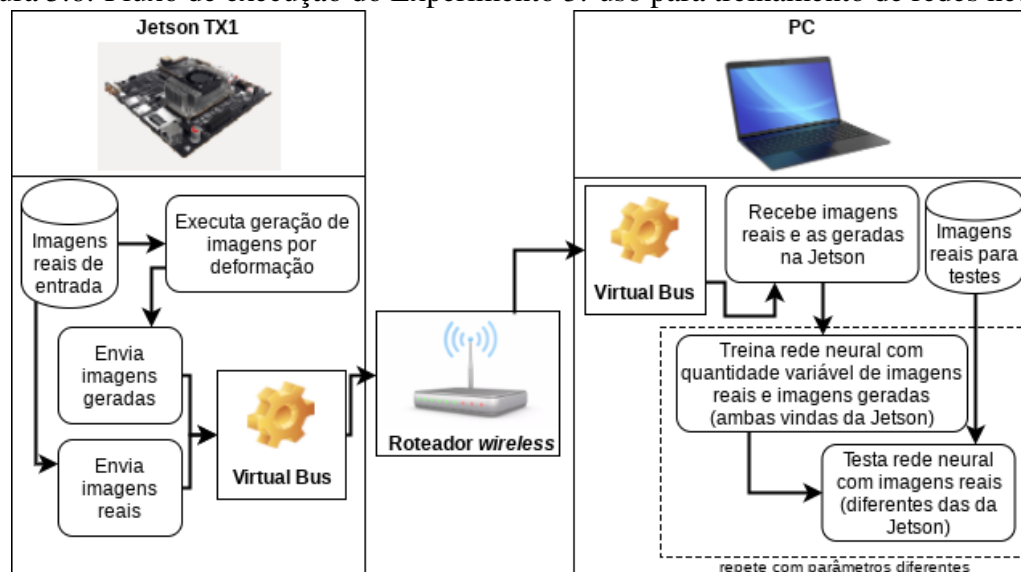
Também é possível verificar que em alguns casos as imagens foram melhoradas, no que diz respeito ao seu reconhecimento pelo algoritmo de referência. Conforme temos o resultado da geração de imagens por deformação "brilho e contraste" na Figura 5.5 (93,4%). Mas no geral, conforme esperado, as imagens adicionam um pouco mais de dificuldade para o algoritmo reconhecer os caracteres. Também é possível perceber que é viável utilizar essa geração de imagens tanto para gerar poucas imagens quanto uma proporção maior, duplicando a quantidade inicial ou até mais.

5.4 Experimento 3: geração de imagens para treinamento de redes neurais

Esse experimento tenta mostrar a eficácia das imagens geradas, a partir do método de deformação (Seção 4.2), em relação à sua utilização para o treinamento de redes neurais. A ideia principal aqui é comparar os resultados obtidos ao treinar uma rede neural com imagens reais, de uma base inicial de imagens, em contrapartida com imagens sintéticas, obtidas pelo método de geração de imagens por deformação.

Para execução desse experimento, visando uma boa reprodutibilidade, foram utilizadas como imagens reais iniciais as amostras contidas no diretório *samples* do OpenCV. Essas amostras consistem em 5.000 imagens de dígitos. Dessas imagens reais, metade são para serem utilizadas no treinamento e a outra metade exclusivamente para realizar os testes da rede neural treinada, seja com imagens reais, sintética ou uma combinação delas. A rede neural utilizada para rodar esses experimentos também foi a SVM implementada pelo OpenCV.

Figura 5.6: Fluxo de execução do Experimento 3: uso para treinamento de redes neurais.



Fonte: o autor.

Esse experimento executou o treinamento da rede neural com uma quantidade inicial de imagens reais e foi adicionando imagens sintéticas ao treinamento da rede neural nas execuções seguintes. Essas imagens sintéticas foram geradas a partir do conjunto inicial. A quantidade de imagens reais também foi variada em alguns casos. De modo geral, a comparação de resultados seguiu o roteiro:

- Treinar uma rede neural de reconhecimento de caracteres com um conjunto de imagens reais;
- Testar a rede neural que foi treinada, mas utilizando outras imagens reais diferentes das anteriores, exclusivas para testes;
- Gerar imagens sintéticas, utilizando como entrada do processo de geração as imagens iniciais reais;

- Executar mais treinamentos e testes, variando as quantidades e combinações de imagens reais e sintéticas;
- Comparar os resultados.

Na Figura 5.6 é possível ter uma ideia geral de como foi executado o experimento. As imagens geradas na Jetson TX1 foram recebidas no PC, juntamente com as imagens reais, que serviram para as execuções dos treinamentos na rede neural SVM.

5.4.1 Resultados Obtidos

Na Tabela 5.2 estão os resultados relativos ao Experimento 3, onde foram testados cenários diversos para o treinamento de uma rede neural SVM.

Tabela 5.2: Experimento treinamento redes neurais.

# imagens reais	# img. sintéticas	# total treinam.	# img. teste	Precisão
250	0	250	250	89,4%
250	750	1.000	250	92,4%
1.000	0	1.000	250	93,1%
2.500	0	2.500	2.500	93,7%
2.500	2.500	5.000	2.500	94,9%

A primeira coluna traz a quantidade de imagens reais, advindas de um *dataset* de dígitos escritos à mão. A segunda coluna, **# img. sintéticas**, representa a quantidade de imagens geradas a partir da quantidade de imagens presente na coluna anterior. A terceira coluna, **# total treinamento**, representa ao total quantas imagens foram utilizadas para treinar a rede neural. Ou seja, é a soma de imagens reais e imagens sintéticas geradas a partir das reais. A quarta coluna, **#img. teste**, é a quantidade de imagens reais (diferentes das informadas na primeira coluna) utilizadas para testar a precisão da rede neural que foi treinada com as imagens informadas. Ao final, na quinta e última coluna, temos o percentual de precisão alcançado com a rede neural treinada de acordo com os dados informados.

Basicamente, temos dois grupos de execução. O primeiro, composto pelas três primeiras linhas da tabela, e o segundo, contando com as três últimas linhas. O primeiro grupo trata da comparação entre treinar a rede neural com apenas 250 imagens ou 1.000 imagens, com

a variante de 1.000 imagens mesclando com imagens sintéticas. O segundo grupo compara o desempenho da rede neural treinada com 2.500 e também com 5.000, sendo essas últimas mescladas com imagens sintéticas.

Os resultados são animadores, pois no primeiro grupo, onde foram geradas 750 imagens sintéticas a partir de 250 reais, chegamos a 1.000 imagens para o treinamento da rede neural. Tendo assim um resultado de 92,4%, melhor que os 89,4% iniciais apenas com 250 imagens. Claro que, não é o mesmo se comparado ao treinamento com 1.000 imagens reais, onde foi alcançado um percentual de precisão de 93,1%. Porém, é uma alternativa viável para os casos em que há uma escassez de imagens.

No segundo grupo, com 2.500 imagens reais, também foi possível melhorar a taxa de de precisão de 93,7% para 94,9%. Lembrando que, quanto mais se aproxima dos 100% de precisão, mais difícil vai ficando de se melhorar essa taxa. É importante lembrar também que o percentual de precisão alcançado com as imagens sintéticas é variável e essas taxas na tabela são uma média de várias execuções. Isso ocorre pois o processo de geração de imagens por deformação tem parâmetros que são gerados aleatoriamente.

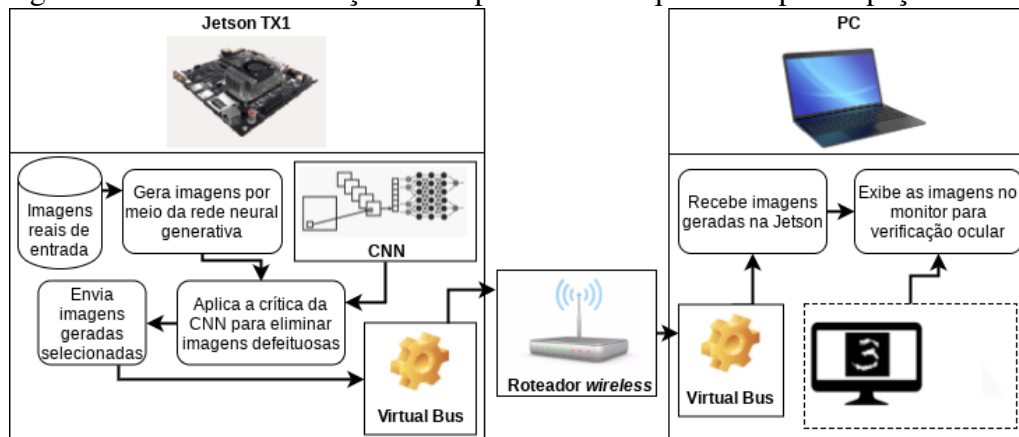
5.5 Experimento 4: qualidade por inspeção ocular

Com um foco um pouco diferente dos experimentos anteriores, esse experimento trata da geração de imagens por rede neural, descrito na Seção 4.3 e tenta mensurar a qualidade das imagens produzidas nesse processo.

Um outro indicador será apresentado para calcular a precisão da geração de imagens via GAN, que será a classificação humana. Ou seja, consiste simplesmente em uma pessoa analisando a imagem e discriminando o objeto que ela vê. Esse indicador será usado para tentar mensurar a qualidade das imagens produzidas.

O Experimento 4 consistiu em gerar 500 imagens através da GAN treinada, aplicando automaticamente as críticas da CNN para eliminar as imagens consideradas de baixa qualidade (conforme descrito na Seção 4.3) e finalmente exibir as imagens para um humano realizar uma análise visual a fim de classificar cada imagem gerada. Com isso, é possível verificar a porcentagem aproximada de imagens rejeitadas pela CNN e o percentual de confiança das imagens geradas pela versão final da GAN implementadas e treinadas.

Figura 5.7: Fluxo de execução do Experimento 3: qualidade por inspeção ocular.



Fonte: o autor.

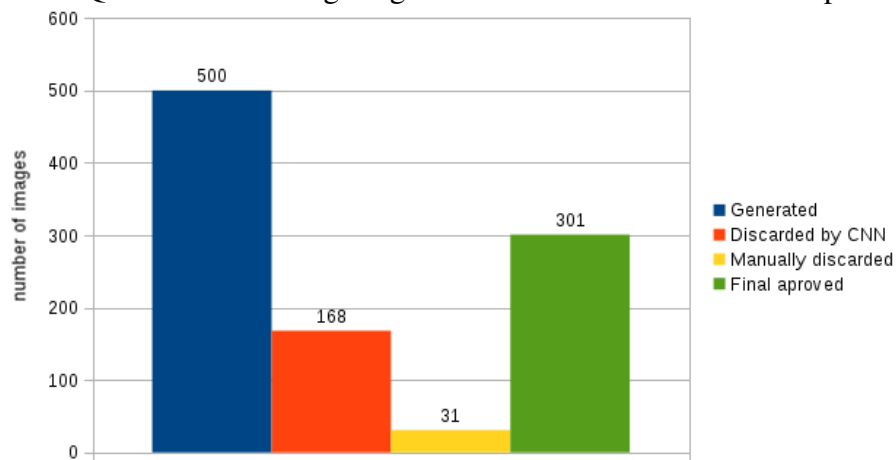
Na Figura 5.7 temos um diagrama exibindo o fluxo geral desse experimento. Após as imagens serem geradas pela rede neural generativa na Jetson TX1, foi aplicada a crítica da CNN para eliminar imagens consideradas como defeituosas. Em seguida as imagens sintéticas que passaram pela CNN são enviadas via Virtual Bus para o PC exibi-las na tela para um usuário visualizá-las e as classificar como válidas ou não.

5.5.1 Resultados Obtidos

A partir do Experimento 4, foi gerado o gráfico na Figura 5.8, indicando que a partir de cerca de 500 imagens produzidas pelo componente gerador GAN, 168 imagens (cerca de 33 %) foram descartadas pela rede CNN. Das 332 imagens restantes, quando examinadas manualmente por um humano, outras 31 imagens foram declaradas inválidas. Isso nos dá uma porcentagem de erro de 9 % das imagens entregues pelo componente gerador por rede neural.

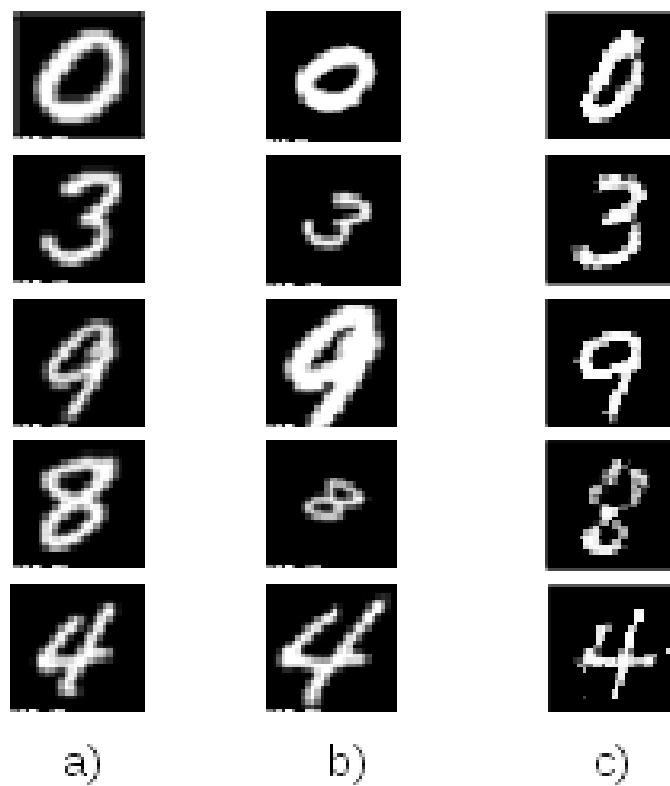
Finalmente, temos algumas amostras das imagens manipuladas durante os experimentos de acordo com a Figura 5.9. Na Figura 5.9-a, estão agrupadas amostras de imagens da Base Inicial de dígitos manuscritos. Em seguida, na Figura 5.9-b existem amostras de imagens geradas pela Experimento 2, pelo componente de geração por deformação. Então, na Figura 5.9-c são amostras que consistem das imagens geradas pela rede neural GAN durante o Experimento 3. É possível conferir mais imagens geradas pela GAN no Anexo B, na Figura B.1.

Figura 5.8: Quantidade de imagens geradas e classificadas durante Experimento 3.



Fonte: o autor.

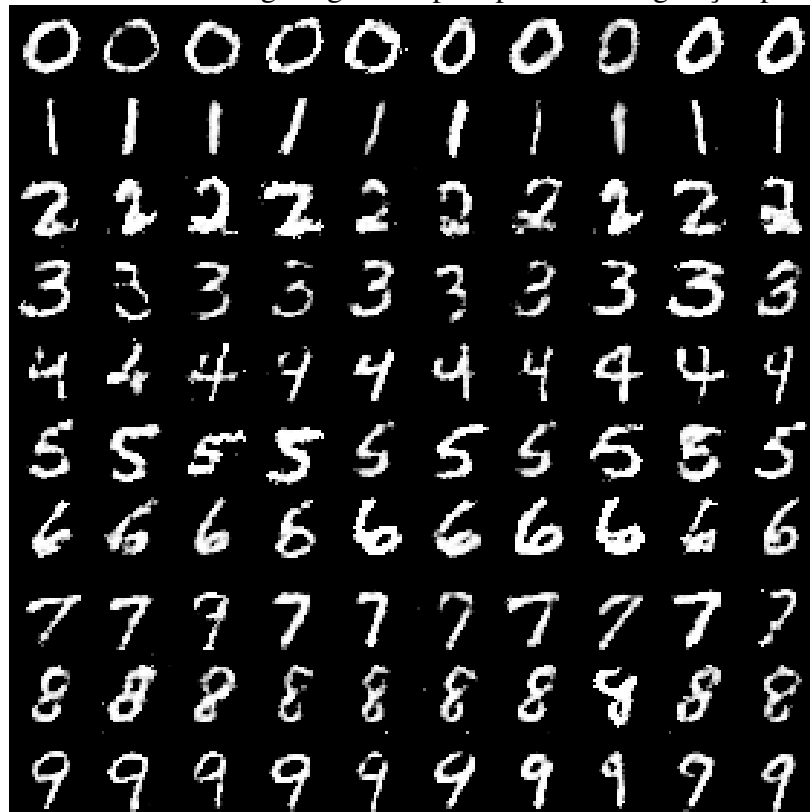
Figura 5.9: Amostras das imagens de dígitos escritos à mão. a) do dataset inicial, sem alterações; b) geradas por deformação; c) geradas pela GAN.



Fonte: o autor.

É possível perceber que a maioria das imagens geradas por rede neural são de um estilo um pouco diferente das imagens reais, com alguns ruídos como *pixels* aleatórios. Porém, além disso elas preservam um pouco da consistência e do alinhamento presente nas imagens reais. Na Figura 5.10 temos mais imagens produzidas pelo processo de geração por rede neural.

Figura 5.10: Amostras das imagens geradas pelo processo de geração por rede neural.



Fonte: o autor.

5.6 Experimento 5: CPU vs GPU

O quinto e último experimento, visou demonstrar a eficiência da geração de imagens através da GPU. Teve como foco específico a geração por deformação (Seção 4.2), já que a geração por rede neural foi implementada em TensorFlow e existem trabalhos que já fazem comparativos entre suas versões em CPU e em GPU como [Lawrence et al. 2017], [Balaban 2018].

No caso da geração por deformação, foram executadas algumas gerações de imagens em implementações tanto em CPU como em GPU, incrementando gradualmente a quantidade

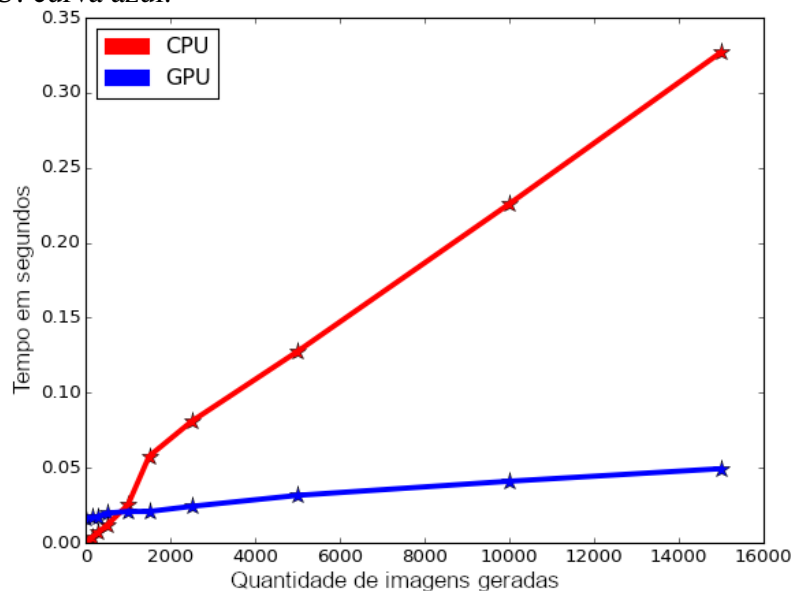
de imagens geradas a cada execução. Dessa forma foi possível comparar os tempos de processamento em diversos casos a fim de construir um gráfico demonstrando uma diferença progressiva.

Esse experimento foi executado apenas no PC, visto que ficava melhor para colher e comparar os dados, já que o mesmo é equipado com uma GPU com suporte CUDA, conforme descrito na Seção 5.1.

5.6.1 Resultados Obtidos

Na Figura 5.11 reunimos dados da execução do Experimento 5 e algumas execuções adicionais com o objetivo de calcular o tempo de processamento tanto na GPU quanto na CPU. Como resultado, temos este gráfico demonstrando uma maior estabilidade no tempo de execução da GPU à medida que a carga de trabalho aumenta. O tempo de execução pela CPU cresce linearmente, enquanto na GPU é quase constante. Isso demonstra que essa solução pode gerar grande quantidade de imagens sem degenerar o desempenho.

Figura 5.11: Gráfico do tempo de geração por número de imagens produzidas. CPU: curva vermelha. GPU: curva azul.



Fonte: o autor.

Também pode ser notado que o tempo de execução da CPU compensa apenas a geração de algumas centenas de imagens, porque a GPU mantém um desempenho bem equilibrado à medida que a carga útil da imagem aumenta o tamanho. Esses tempos de execução da GPU

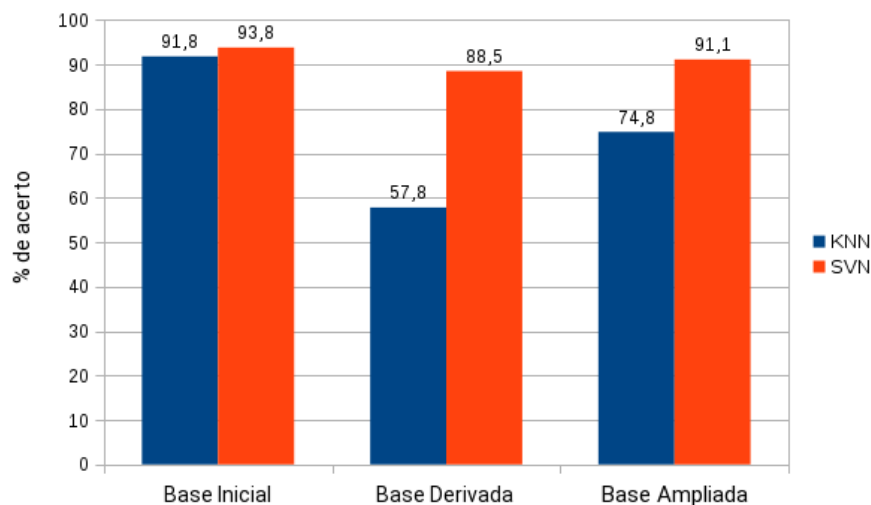
já incluíram o tempo de transferência da memória do host para o dispositivo e vice-versa, bem como algumas conversões de tipo necessárias.

5.7 Resultados Adicionais

Aqui serão apresentados dados adicionais coletados durante execuções dos desenvolvidos e também descritos nas seções anteriores.

Os experimentos foram realizados com os dois algoritmos de referência mencionados anteriormente, KNN e SVM, ambos configurados com parâmetros padrão da implementação OpenCV. No entanto, foram adotados os resultados do algoritmo SVM como padrão para a maioria dos gráficos. Além de ficar mais fácil interpretar os resultados de um algoritmo só, isso deveu-se ao fato de que esta implementação teve os melhores resultados, como mostrado na Figura 5.12.

Figura 5.12: Desempenho dos algoritmos KNN e SVM durante a expansão de um *dataset* de imagens no Experimento 2.



Fonte: o autor.

Na Figura 5.12, estão a porcentagem de acertos, que são as respostas de classificações corretas dadas pelos dois algoritmos. A comparação foi feita durante os três principais estágios das imagens. O estado inicial, isto é, os dados de entrada sem qualquer tratamento, indicado no gráfico como *Base Inicial* e também chamado de imagens reais. Seguido pelas imagens produzidas pelo componente gerador, identificadas como *Base Derivada*. E por fim,

indicado no gráfico como *Base Extendida*, que é o produto final do processo, consistindo da base inicial juntamente com a base derivada.

Para que fique mais fácil de reproduzir os resultados aqui obtidos, tanto o KNN quanto o SVM foram treinados apenas com amostras padrão do OpenCV. Portanto, os resultados, quando se referem à precisão, referem-se à implementação OpenCV do algoritmo SVM.

A Figura 5.13 dessa seção exibe uma composição de um *background* com a aplicação das imagens simples geradas. Assim, essa figura demonstra que também é possível gerar as imagens compostas e como elas podem ser visualmente, citadas como mais uma ideia de verificação no início do Capítulo 4.

Figura 5.13: Amostra da composição de background com imagens de dígitos escritos à mão gerados pela ferramenta desenvolvida.



Fonte: o autor.

5.8 Considerações Finais

Durante esse processo de desenvolvimento e testes, foi observado que alguns tipos de alterações de aspecto são mais sensíveis às parametrizações. Também podem ser observados diferentes resultados de acordo com a quantidade e o tipo de imagens utilizado. Isso para

citar os principais possíveis pontos sensíveis.

A execução do experimento de forma distribuída através da rede foi bem sucedida, porém, é importante observar que as imagens devem ser enviadas em um número considerável de *bytes*, conforme também já discutido em [Andrade et al. 2016].

Com os resultados apresentados, é possível interpretá-los como indicadores importantes de que essa metodologia de geração de imagens pode ser bem sucedida para estender um *dataset* existente. Em adicional, também tivemos bons resultados no que diz respeito à sua utilização para melhorar treinamento de redes neurais, principalmente nos casos em que não se tem disponíveis muitas imagens em bases de imagens existentes.

Capítulo 6

Conclusão

A partir dos resultados obtidos, é possível verificar que, com a metodologia aplicada, foi possível estender bases de imagens, agregando mais variedade e tornando o processo de classificação um pouco mais desafiador. Embora os experimentos tenham funcionado com imagens muito pequenas, o desempenho permaneceu em um nível viável, já que foi usada a implementação paralela. Assim, foi possível estender bastante um conjunto de dados inicial sem incorrer em grandes latências. A metodologia exposta também é genérica o suficiente para ajustar várias categorias de imagens.

Os resultados da Seção 5.6 demonstram que a presente proposta consegue atingir o objetivo de geração de imagens com eficiência utilizando de computação paralela em GPU. Já os demais experimentos do Capítulo 5, executados de forma distribuída em rede, corroboram para o objetivo de o componente gerador ser capaz de fornecer imagens para outro componente diverso.

É importante observar que, embora seja possível obter bons resultados com a geração de imagens através de GAN, para uso prático, os experimentos demonstraram que é necessária uma melhoria na tecnologia de treinamento. Tanto a redução do tempo de treinamento quanto a consequente diminuição das imagens de baixa qualidade geradas. No entanto, como a funcionalidade principal já foi aqui demonstrada, essas tarefas foram deixadas para trabalhos futuros.

Referências Bibliográficas

- [Andrade et al. 2016]ANDRADE, H. G. R. et al. The integration of gpu-based and heterogeneous devices using hla. In: *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2016. p. 162–167.
- [Arik et al. 2017]ARIK, S. Ö. et al. Deep voice: Real-time neural text-to-speech. *CoRR*, abs/1702.07825, 2017. Disponível em: <<http://arxiv.org/abs/1702.07825>>.
- [Balaban 2018]BALABAN, M. *Titan RTX Deep Learning Benchmarks*. 2018. <https://lambdalabs.com/blog/titan-rtx-tensorflow-benchmarks/>. [Online; Acessado em 11 de janeiro de 2019].
- [Bedoya, Perez e Marin 2016]BEDOYA, A. E.; PEREZ, Y. M.; MARIN, H. A. P. A review on verification and validation for embedded software. *IEEE Latin America Transactions*, v. 14, n. 5, p. 2339–2347, May 2016. ISSN 1548-0992.
- [Bertacco e Chatterjee 2011]BERTACCO, V.; CHATTERJEE, D. High performance gate-level simulation with gp-gpu computing. In: *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*. [S.l.: s.n.], 2011. p. 1–3.
- [Bhatia, Patel e Chauhan 2016]BHATIA, A. R.; PATEL, N. M.; CHAUHAN, N. C. Parallel implementation of face detection algorithm on gpu. In: *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*. [S.l.: s.n.], 2016. p. 674–677.
- [Bhatia e Ashev 2010]BHATIA, N.; ASHEV, V. Survey of nearest-neighbor techniques. *International Journal of Computer Science and Information Security*, 2010. ISSN 1098-6596.
- [CERTI 2018]CERTI. jul. 2018. [Online; Acessado em 19 Jul. 2018]. Disponível em: <<http://savannah.nongnu.org/projects/certi>>.

- [Cohen et al. 2017]COHEN, G. et al. Emnist: Extending mnist to handwritten letters. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2017. p. 2921–2926.
- [Creswell et al. 2018]CRESWELL, A. et al. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, v. 35, n. 1, p. 53–65, Jan 2018. ISSN 1053-5888.
- [Dagum e Menon 1998]DAGUM, L.; MENON, R. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, v. 5, n. 1, p. 46–55, Jan 1998. ISSN 1070-9924.
- [Delorme 2014]DELORME, N. Mixed-signal verification challenges. In: *2014 10th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*. [S.l.: s.n.], 2014. p. 1–1.
- [Deng 2010]DENG, Y. Gpu accelerated vlsi design verification. In: *2010 10th IEEE International Conference on Computer and Information Technology*. [S.l.: s.n.], 2010. p. 1213–1218.
- [Dettmers 2015]DETTMERS, T. *Deep Learning in a Nutshell: Core Concepts*. 2015. <https://devblogs.nvidia.com/deep-learning-nutshell-core-concepts/>. [Online; Acessado em 08 Jun. 2018].
- [Garris 1992]GARRIS, M. D. Design and collection of a handwriting sample image database. *Social Science Computer Review*, v. 10, n. 2, p. 196–214, 1992. Disponível em: <<https://doi.org/10.1177/089443939201000205>>.
- [Goodfellow et al. 2014]Goodfellow, I. J. et al. Generative Adversarial Networks. *ArXiv e-prints*, jun. 2014.
- [Hawkins 2004]HAWKINS, D. The Problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, 2004. ISSN 00952338.
- [Horsley e Perez-Liebana 2017]HORSLEY, L.; PEREZ-LIEBANA, D. Building an Automatic Sprite Generator with Deep Convolutional Generative Adversa-

- rial Networks. p. 134–141, 2017. Disponível em: <http://www.cig2017.com/wp-content/uploads/2017/08/paper_50.pdf>.
- [Hubel e Wiesel 1959]HUBEL, D. H.; WIESEL, T. N. *Receptive fields of single neurones in the cat's striate cortex*. oct 1959. 574–591 p.
- [Hubel e Wiesel 1962]HUBEL, D. H.; WIESEL, T. N. *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*. jan 1962. 106–154.2 p.
- [IEEE Standard for the Functional Verification Language e 2017]IEEE Standard for the Functional Verification Language e. *IEEE Std 1647-2016 (Revision of IEEE Std 1647-2011)*, p. 1–558, Jan 2017.
- [ImageNet 2016]IMAGENET, S. V. L. *ImageNet - Stanford Vision Lab*. 2016. <http://www.image-net.org/index>. [Online; Acessado em 19 Abr. 2018].
- [Kanere, Mhatre e Jaiswal 2016]KANERE, K.; MHATRE, H.; JAISWAL, A. High performance parallel processing to cluster visually similar image data sets. In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. [S.l.: s.n.], 2016. p. 2690–2692.
- [Karras et al. 2017]KARRAS, T. et al. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. Disponível em: <<http://arxiv.org/abs/1710.10196>>.
- [Kaur e Behal 2015]KAUR, K.; BEHAL, S. Designing a Secure Text-based CAPTCHA. In: *Procedia Computer Science*. [S.l.: s.n.], 2015. ISSN 18770509.
- [Khronos 2018]KHRONOS, G. I. *OpenCL Overview - The Khronos Group Inc*. 2018. <https://www.khronos.org/opencl/>. [Online; Acessado em 12 Abr. 2018].
- [Krizhevsky 2009]KRIZHEVSKY, A. Learning Multiple Layers of Features from Tiny Images. ... *Science Department, University of Toronto, Tech. ...*, p. 1–60, 2009. ISSN 1098-6596.

- [Kumar e Garg 2017]KUMAR, V.; GARG, M. L. Deep learning in predictive analytics: A survey. In: *2017 International Conference on Emerging Trends in Computing and Communication Technologies (ICETCCT)*. [S.l.: s.n.], 2017. p. 1–6.
- [Lampert, Pucher e Dostal 2018]LAMPERT, C. H.; PUCHER, D.; DOSTAL, J. *Animals with Attributes 2*. 2018. <https://cvml.ist.ac.at/AwA2/>. [Online; Acessado em 19 Jul. 2018].
- [LASER 2018]LASER, U. *Laboratório de Sistemas Embarcados e Robótica (LaSER)*. 2018. <https://sites.google.com/a/ci.ufpb.br/laser/>. [Online; Acessado em 12 Abr. 2018].
- [Lawrence et al. 2017]LAWRENCE, J. et al. Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud. *Student-Faculty Research Day, CSIS, Pace University, Pleasantville, New York*, 2017.
- [LeCun, Bengio e Hinton 2015]LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 521, p. 436, may 2015. Disponível em: <<http://dx.doi.org/10.1038/nature14539>
<http://10.0.4.14/nature14539>>.
- [Lee et al. 2011]LEE, H. et al. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Commun. ACM*, ACM, New York, NY, USA, v. 54, n. 10, p. 95–103, out. 2011. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/2001269.2001295>>.
- [Li et al. 2016]LI, J. et al. A persona-based neural conversation model. *CoRR*, abs/1603.06155, 2016. Disponível em: <<http://arxiv.org/abs/1603.06155>>.
- [Luan et al. 2017]LUAN, F. et al. Deep photo style transfer. *CoRR*, abs/1703.07511, 2017. Disponível em: <<http://arxiv.org/abs/1703.07511>>.
- [Ma e Guo 2014]MA, Y.; GUO, G. *Support vector machines applications*. [S.l.: s.n.], 2014. ISSN 19395108. ISBN 9783319023007.

- [Macri et al. 2015]MACRI, M. et al. Efficient Lava Flows Simulations with OpenCL: A Preliminary Application for Civil Defence Purposes. In: *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. [S.l.]: IEEE, 2015. p. 328–335. ISBN 978-1-4673-9473-4.
- [Nielsen 2018]NIELSEN, M. A. *Neural Networks and Deep Learning*. Determination Press, 2018. [Online; Acessado em 19 Jul. 2018]. Disponível em: <<http://neuralnetworksanddeeplearning.com/>>.
- [Noulard, Rousselot e Siron 2009]NOULARD, E.; ROUSSELOT, J.-Y.; SIRON, P. CERTI, an Open Source RTI, why and how. 2009.
- [NVIDIA 2018]NVIDIA. *CUDA C Programming Guide*. 2018. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. [Online; Acessado em 12 Jul. 2018].
- [NVIDIA 2018]NVIDIA, C. *CUDA Zone | NVIDIA Developer*. 2018. <https://developer.nvidia.com/cuda-zone>. [Online; Acessado em 11 Jul. 2018].
- [Obimbo, Halligan e De Freitas 2013]OBIMBO, C.; HALLIGAN, A.; De Freitas, P. CaptchAll: An improvement on the modern text-based CAPTCHA. *Procedia Computer Science*, Elsevier Masson SAS, v. 20, p. 496–501, 2013. ISSN 18770509. Disponível em: <<http://dx.doi.org/10.1016/j.procs.2013.09.309>>.
- [Oh, Yi e Yi 2015]OH, C.; YI, S.; YI, Y. Real-time face detection in full hd images exploiting both embedded cpu and gpu. In: *2015 IEEE International Conference on Multimedia and Expo (ICME)*. [S.l.: s.n.], 2015. p. 1–6. ISSN 1945-7871.
- [Osadchy et al. 2017]OSADCHY, M. et al. No bot expects the deepcaptcha! introducing immutable adversarial examples, with applications to captcha generation. *IEEE Transactions on Information Forensics and Security*, v. 12, n. 11, p. 2640–2653, Nov 2017. ISSN 1556-6013.
- [Otterness et al. 2017]OTTERNESS, N. et al. An evaluation of the nvidia tx1 for supporting real-time computer-vision workloads. In: *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. [S.l.: s.n.], 2017. p. 353–364.

- [Peng et al. 2018]PENG, S. et al. Framework for efficient sca resistance verification of iot devices. In: *2018 IEEE International Conference on Applied System Invention (ICASI)*. [S.l.: s.n.], 2018. p. 468–471.
- [Perotti, Garcez e Boella 2014]PEROTTI, A.; GARCEZ, A. d’Avila; BOELLA, G. Neural networks for runtime verification. In: *2014 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2014. p. 2637–2644. ISSN 2161-4407.
- [Raghavan 1994]RAGHAVAN, T. Zero-sum two-person games. In: *Handbook of Game Theory with Economic Applications*. [S.l.: s.n.], 1994. ISBN 9780444894274.
- [Redmon e Farhadi 2016]REDMON, J.; FARHADI, A. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. Disponível em: <<http://arxiv.org/abs/1612.08242>>.
- [Russakovsky et al. 2015]RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, v. 115, n. 3, p. 211–252, 2015. ISSN 1573-1405. Disponível em: <<https://doi.org/10.1007/s11263-015-0816-y>>.
- [Schmidhuber 2014]SCHMIDHUBER, J. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. Disponível em: <<http://arxiv.org/abs/1404.7828>>.
- [Silva et al. 2018]SILVA, T. W. B. et al. A distributed functional verification environment for the design of system-on-chip in heterogeneous architectures. In: *2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI)*. [S.l.: s.n.], 2018. p. 4849–4854.
- [Silva et al. 2018]SILVA, T. W. B. et al. Environment for integration of distributed heterogeneous computing systems. *Journal of Internet Services and Applications*, v. 9, n. 1, p. 4, jan 2018. ISSN 1869-0238. Disponível em: <<https://doi.org/10.1186/s13174-017-0072-1>>.
- [Squartini, Hussain e Piazza 2003]SQUARTINI, S.; HUSSAIN, A.; PIAZZA, F. Preprocessing based solution for the vanishing gradient problem in recurrent neural networks. In: *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03*. [S.l.: s.n.], 2003. v. 5, p. V–V.
- [Stone, Gohara e Shi 2010]STONE, J. E.; GOHARA, D.; SHI, G. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science and Engineering*, v. 12, n. 3, p. 66–72, 2010. ISSN 15219615.

- [Team 2018]TEAM, O. D. *OpenCV: OCR of Hand-written Data using SVM*. 2018. https://docs.opencv.org/trunk/dd/d3b/tutorial_py_svm_opencv.html. [Online; Acessado em 12 Abr. 2018].
- [Vimina e Areekal 2009]VIMINA, E. R.; AREEKAL, A. U. Telling computers and humans apart automatically using activity recognition. In: *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2009. ISBN 9781424427949. ISSN 1062922X.
- [Weber et al. 2011]WEBER, R. et al. Comparing hardware accelerators in scientific applications: A case study. *IEEE Transactions on Parallel and Distributed Systems*, v. 23, n. 1, p. 58–68, 2011. ISSN 10459219.
- [With 2018]WITH, E. Lifelong Learning With Dynamically Expandable Networks. *Iclr2018*, 2018.

Apêndice A

Amostra de dígitos utilizados



Figura A.1: Arquivo "digits.png" contendo amostras de dígitos escritos à mão. Fonte: diretório *samples* da instalação do OpenCV.

Apêndice B

Apêndice B

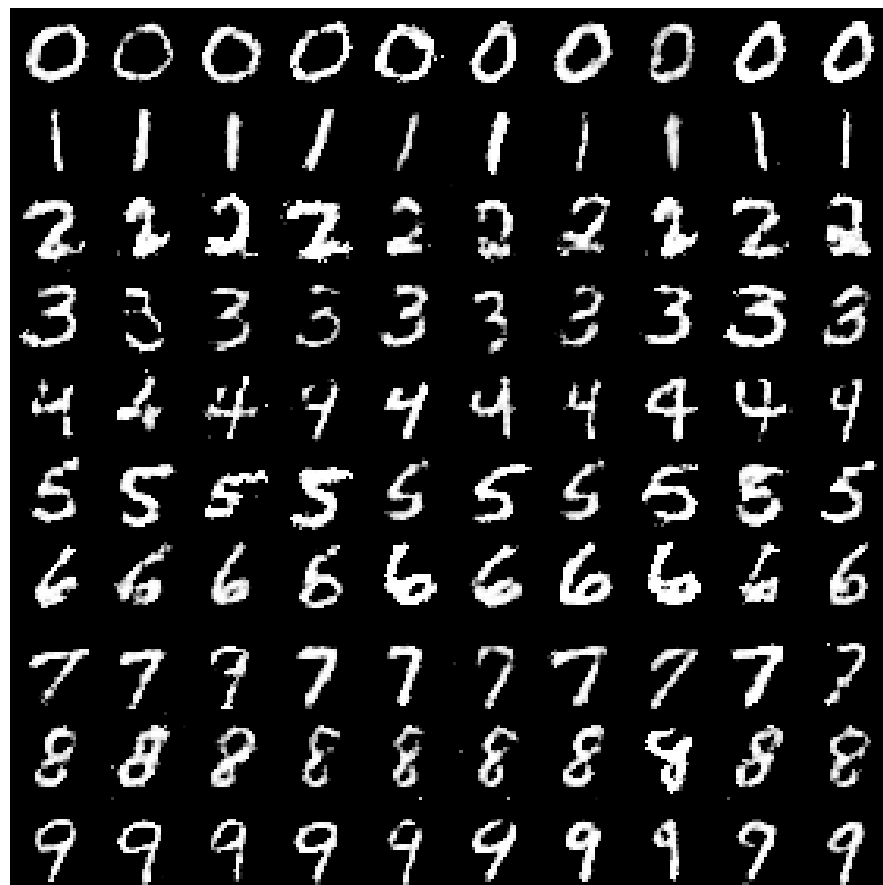


Figura B.1: Amostras de imagens geradas pela GAN.