UNIVERSIDADE ESTADUAL DE CAMPINAS - UNICAMP
Faculdade de Tecnologia

PAULO LUIS ZAMPIERI

AVALIAÇÃO DE DESEMPENHO DE REDES NEURAIS CONVOLUCIONAIS UTILIZANDO IMAGENS SINTÉTICAS

LIMEIRA – SP
2021

PAULO LUIS ZAMPIERI

AVALIAÇÃO DE DESEMPENHO DE REDES NEURAIS CONVOLUCIONAIS UTILIZANDO IMAGENS SINTÉTICAS

Dissertação apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Tecnologia, na Área de Sistemas de Informação e Comunicação.

Orientador: Prof. Dr. André Franceschi de Angelis.

Este exemplar corresponde a versão final da Dissertação defendida por Paulo Luis Zampieri orientada pelo prof. Dr. André Franceschi de Angelis.

LIMEIRA – SP

2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Faculdade de Tecnologia
Felipe de Souza Bueno - CRB 8/8577

Z14a

Zampieri, Paulo Luis, 1970-
Avaliação de desempenho de redes neurais convolucionais utilizando imagens sintéticas / Paulo Luis Zampieri. – Limeira, SP : [s.n.], 2021.

Orientador: Andre Franceschi de Angelis.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Tecnologia.

1. Redes neurais convolucionais. 2. Redes neurais (Computação). 3. Classificação de imagem. 4. Aprendizado profundo. I. Angelis, Andre Franceschi de, 1969-. II. Universidade Estadual de Campinas. Faculdade de Tecnologia. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Performance assessment of convolutional neural networks using synthetic images
Palavras-chave em inglês:
Convolutional neural networks
Neural networks (Computer science)
Image classification
Deep learning
Área de concentração: Sistemas de Informação e Comunicação
Titulação: Mestre em Tecnologia
Banca examinadora:
André Franceschi de Angelis
Marco Antonio Garcia de Carvalho
Jurandir Zullo Junior
Data de defesa: 22-11-2021
Programa de Pós-Graduação: Tecnologia

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-5745-9013
- Currículo Lattes do autor: http://lattes.cnpq.br/7614521121625959

FOLHA DE APROVAÇÃO

Abaixo se apresentam os membros da comissão julgadora da sessão pública de defesa de dissertação para o Título de Mestre em Tecnologia na área de concentração Sistemas de Informação e Comunicação, a que se submeteu o aluno Paulo Luís Zampieri, em 22 de novembro de 2021 na Faculdade de Tecnologia – FT/UNICAMP, em Limeira/SP.

Prof. Dr. André Franceschi de Angelis
Presidente da Comissão Julgadora

Prof. Dr. Marco Antonio Garcia de Carvalho
Universidade Estadual de Campinas – Faculdade de Tecnologia (FT)

Prof. Dr. Jurandir Zullo Junior
Universidade Estadual de Campinas - CEPAGRI

ATA da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria de Pós-graduação da FT.

AGRADECIMENTOS

Agradeço a Deus pela vida, conhecimento, sabedoria e capacidade. Foram alguns anos de muitos esforços, estudos, aprendizados e bastante dedicação para realização desta pesquisa.

Agradecimento especial ao meu orientador, prof. Dr. André F. de Angelis por esta oportunidade desafiadora e que sempre me apoiou pacientemente e deu todo suporte para finalização deste trabalho.

À toda minha família, minha mãe Maria A. Barbosa Zampieri e meu pai, Victorino Zampieri por sempre me apoiaram e incentivarem a nunca desistir e acreditar nos sonhos.

Gostaria também de expressar meus sinceros agradecimentos à aluna de doutorado, Thaís Rocha, pelos conhecimentos transmitidos, pré-disposição, ajuda, paciência e apoio quando precisei.

Agradeço também ao Bruno Amadio Caires, do departamento de informática da Faculdade de Tecnologia da Unicamp, que sempre me ajudou com problemas técnicos com os servidores.

Um especial agradecimento também à NVIDIA pelo fornecimento da ferramenta (GPU e DIGITS) utilizada neste trabalho.

À empresa a qual trabalho atualmente, Buckman Laboratórios Ltda., por ter apoiado, desafiado, e me incentivado a buscar este conhecimento. Meu especial agradecimento.

E, finalmente, um especial agradecimento à CAPES:

O presente trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Código de financiamento 001. Processo nº 23038.013648/2018-51.

RESUMO

A utilização de classificadores de imagens através de *Deep Learning* (DL) ou aprendizado profundo é uma das tendências tecnológicas que vem sendo amplamente utilizada. Esta técnica tem evoluído e, através das Redes Neurais Convolucionais (CNN), utilizando aprendizagem supervisionada, é capaz de extrair atributos, identificar padrões e reconhecer imagens. Os objetivos principais deste trabalho foram avaliar a acurácia dos classificadores de imagens utilizando as redes GoogLeNet e AlexNet, combinados com os *frameworks*, TensorFlow, Caffe e Torch, e determinar quais as melhores configurações (hiperpârametros) a serem utilizados nas CNN. Em relação à literatura em geral, o diferencial desta pesquisa é o uso de sequências de imagens com padrões correlacionados por componentes estáticos (forma) e dinâmicos (movimentação). No que diz respeito a trabalhos específicos com sequências de imagens, as novidades são o uso de conjuntos diversos de padrões e a comparação de *frameworks*. Foram utilizadas 50.000 imagens sintéticas rotuladas (geradas por software), distribuídas em 25 conjuntos distintos, para a execução dos testes, com padrões simples, perceptíveis por pessoas, como a distância entre pontos, ângulos em vértices, cobertura de áreas e densidade de pontos. A escolha dos parâmetros para utilização na CNN se deu através de testes preliminares. A avaliação de desempenho baseou-se nos indicadores de acurácia de nível 1 e 5. Os resultados demonstraram que a ferramenta utilizada neste trabalho, o DIGITS da Nvidia, consegue fazer classificações com percentual de acerto similar às encontradas na literatura em outros domínios de aplicação. A rede AlexNet com o TensorFlow foi que apresentou melhor resultado, 88,84%, seguida pelo Torch, 79,37% e o Caffe com 61,61%. Já a rede GoogLeNet apresentou resultados inferiores, com 44,92% com o Torch e 55,49% com o Caffe. Diversos contextos podem se beneficiar dos resultados aqui obtidos e da metodologia empregada, inclusive em termos de comparação de desempenho (*benchmark*).

Palavras-chaves: Redes Neurais Convolucionais (RNC); Imagens sintéticas; DIGITS-Nvidia; GoogLeNet; AlexNet; Caffe; Torch; TensorFlow

ABSTRACT

The application of image classifiers through Deep Learning (DL) has become a widely used technological trend. Through the Convolutional Neural Networks (CNN), this technique has evolved and, using supervised learning, it is able to extract features, identify patterns, and recognize images. The main aims of this work were to evaluate the accuracy of the image classifiers using GoogLeNet and AlexNet networks, combined with the frameworks, TensorFlow, Caffe and Torch and determine the best configurations (hiperparameters) to be used in the CNNs. In relation to the literature in general, the differential of this research is the use of image sequences with patterns correlated by static (shape) and dynamic (movement) components. Regarding specific works with image sequences, the novelties are the use of different sets of patterns and the comparison of frameworks. Distributed in 25 different datasets, approximately 50,000 labeled synthetic images (generated by software) were used to perform the tests. The images had simple patterns perceptible by people, such as the distance between points, angles at vertices, and density of points. To choose the parameters to be used on the CNNs, some tests were carried out to determine the best values. The performance evaluation was based on level 1 and 5 accuracy indicators.

The results showed that the tool used in this work, Nvidia's DIGITS, can classify images with the accuracy like those found in the literature in other application domains. AlexNet network with TensorFlow showed the best results, 88.84%, followed by Torch, 79.37% and Caffe with 61.61%. GoogLeNet network presented lower results, with 44.92% with Torch and 55.49% with Caffe. Other application domains can benefit from the results of this work and the methodology used, including in terms of performance comparison (benchmark).

Keywords: Convolutional Neural Networks (CNN); Synthetic images; DIGITS-Nvidia; GoogLeNet; AlexNet; Caffe; Torch; TensorFlow

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIAÇÕES E SIGLAS

ANN	*Artificial Neural Network*
AUROC	*Area Under the Receiver Operating Characteristic*
CNN	*Convolutional Neural Network*
DCNN	*Deep Convolutional Neural Network*
DL	*Deep Learning*
DIGITS	*Deep Learning GPU Training Systems*
EPE	Erro Padrão
FISGEN	*Fast Image Sequence Generator*
FC	*Fully Connected* / Totalmente Conectada / *Dense Layer*
GOES	*Geostationary Operational Environmental Satellite*
ML	*Machine Learning*
ReLU	*Rectified Linear Units*
RGB	*Read, Green, and Blue*
RNA	Redes Neurais Artificiais
RNC	Redes Neurais Convolucionais
SGD	*Stochastic Gradient Descent*
SSH	*Secure Socket Shell*
VPN	*Virtual Private Network*

SUMÁRIO

1. INTRODUÇÃO

As Redes Neurais Artificiais (*Artificial Neural Networks* – ANNs) são parte da área da computação conhecida como Inteligência Artificial. Elas despertam interesse por suas capacidades de reconhecimento de padrões e aprendizagem a partir de dados. As Redes Neurais Convolucionais (*Convolutional Neural Networks* – CNNs) são um tipo de ANN que se beneficia de mecanismos extras para, por exemplo, melhor processamento de informações não-lineares e redução de dimensionalidade do problema, característica especialmente útil no tratamento de imagens. Uma CNN é capaz de abstrair atributos de fotografias para fazer o reconhecimento de objetos e detalhes da cena.

Ainda na área de Inteligência Artificial, aprendizado profundo ou *deep learning* (DL) refere-se de maneira geral ao processo de aprendizado de alguma ANN constituída de camadas ocultas de neurônios artificiais, isto é, aquelas que não são especificamente camadas de entrada ou de saída. É possível construir CNNs com arquitetura DL, buscando a soma dos benefícios de ambas as abordagens. Neste trabalho, todas as redes usadas são convolucionais e de aprendizado profundo; logo, são referenciadas apenas como CNNs de agora em diante.

No entanto, apesar do potencial que apresenta, o emprego das CNNs está longe de ser um procedimento simples e padronizado, aplicável a qualquer problema. Pelo contrário, é necessário um preparo cuidadoso dos dados de treinamento, das redes e das suas configurações, tendo um foco específico e claro, para que resultados aceitáveis sejam alcançados. O uso de programas e *frameworks* prontos facilita a utilização de CNNs, mas não dispensa atenção a preparativos, nem garante fidelidade de solução. Como em alguns sistemas não-lineares, pequenas variações das condições iniciais implicam alta variabilidade nas saídas.

Assim, é muito complicado avaliar “genericamente” uma CNN qualquer. Ela pode ser ótima em algumas situações e péssimas em outras, mesmo se aplicada a um único problema, a depender das configurações feitas. Uma conduta razoável é definir um domínio de aplicação restrito e fazer nele as avaliações de interesse, ainda que isso traga dificuldades para análises comparativas (*benchmarks*), dado que a literatura científica vai oferecer, na melhor das hipóteses, poucos estudos similares no mesmo domínio.

Este trabalho se iniciou com uma motivação, um problema concreto, cujos detalhes são apresentados mais adiante, enfrentado por pesquisadores da Faculdade de Tecnologia: a melhoria da previsão de vazões de rios para auxílio ao planejamento da operação de sistemas elétricos de grande porte, posto que as técnicas atuais ainda apresentam grandes variações entre os valores previstos e os efetivamente observados. Há inúmeros métodos, técnicas e abordagens para este problema, alguns dos quais já atingem seus limites de precisão atualmente.

No entanto, o uso simultâneo de CNNs, DL e imagens obtidas por satélites (de agora em diante, “imagens de satélites”) só agora começa a ser considerado nesse problema, ainda num estágio incipiente, apresentando uma oportunidade de pesquisa e definindo um claro domínio de aplicação. No entanto, o nível de complexidade desta combinação é alto demais para que ela possa inicialmente ser aplicada diretamente, tendo em conta que é um desenvolvimento pioneiro e muitas das variáveis influentes no processo sequer são conhecidas.

Esta oportunidade traz logo de início a questão de viabilidade do uso combinado dos três elementos para atacar o problema. Supõe-se, razoavelmente, que a combinação tenha potencial na área, mas uma resposta definitiva sobre sua viabilidade e seu desempenho efetivo não existe.

A busca simplista por essa resposta, com a escolha de uma CNN e a sua alimentação com imagens de satélites e parâmetros padronizados (*default*) claramente não seria satisfatória, dadas inclusive as considerações sobre os necessários cuidados de preparação de dados e configurações. Em particular, há muitos detalhes nas imagens de satélite que podem comprometer os esforços que façam o seu uso diretamente, como, por exemplo, resolução, padrão e banda de imagem, nível de ruído, cobertura de terreno etc. A exata dimensão do pré-processamento exigido é desconhecida, mas seguramente ele é complexo, exigindo investimentos de tempo e outros recursos.

Antes de fazer tais investimentos, uma ação interessante seria fazer um estudo exploratório, com viés eminentemente empírico, procurando listar CNNs que pudessem ser aplicadas ao problema de previsão de vazões. Seria também desejável ter alguma ideia sobre o nível de previsibilidade das CNNs, configurações e ferramentas escolhidas quando confrontadas com conjuntos de imagens rotuladas bem conhecidos, já que um fator que influencia o resultado de uma CNN é o próprio conteúdo das imagens a ela submetidas.

Bancos de imagens para *benchmark* desse domínio não estão disponíveis, obrigando a busca por alternativas, como, por exemplo, a geração sintética de imagens que, inclusive, podem ser inicialmente simplificações das obtidas por satélites, de maneira a manter o escopo do trabalho em níveis gerenciáveis.

Todos os pontos considerados, este trabalho teve como proposta realizar uma investigação exploratória sobre o desempenho de CNNs com arquitetura DL em condições estritamente controladas, quando as redes são aplicadas a imagens sinteticamente geradas, avaliando-as pelos indicadores de acurácia de nível 1 e de nível 5 (*Top-1 Accuracy e Top-5 Accuracy*). A abordagem consistiu em criar imagens agrupadas em conjuntos distintos (classes) e verificar os valores dos indicadores após o processo de classificação das entradas de teste submetidas às diferentes configurações experimentais.

Dois aspectos devem ser destacados: i) foram usados padrões de baixa ou média complexidade nas figuras geradas, prevendo que futuros trabalhos possam aumentá-los e aproximá-los tanto quanto possível das imagens de satélites, fornecendo assim uma trilha incremental para o seu realismo; b) a motivação inicial da pesquisa foi um problema de predição e a abordagem aqui adotada investiga um processo de classificação via CNNs, que exige discretização das vazões a serem previstas, mas, por outro lado, pode ser mais simples na operação. Esta pesquisa não faz avaliações de desempenho relacionadas à predição.

Para concretizar este trabalho, foram selecionadas 2 CNNs supervisionadas[1], 3 *frameworks* e conjuntos de imagens sintéticas automaticamente rotuladas (*datasets*) com base em diferentes padrões embebidos nas figuras por um software gerador. A avaliação das redes foi feita primariamente pela média dos indicadores para cada par (CNN, *framework*).

O método usado se mostrou adequado, rápido e de baixo custo. A análise de desempenho, apesar da alta variabilidade das saídas, indicou potencial para futura aplicação das CNNs a imagens de satélite, tendo em conta os bons valores dos indicadores justamente em figuras mais assemelhadas à visão superior das nuvens. Foram determinados empiricamente as configurações dos hiperparâmetros para as CNNs, reveladas algumas limitações operacionais e levantadas necessidades de pré-

[1] É um tipo de aprendizagem onde são apresentados os dados já rotulados e que as saídas já são conhecidas, por exemplo, a classificação de números manuscritos, cujas saídas seriam de 0 a 9.

processamento e preparação de dados.

A seção 1.1 tece breves comentários, para fins de contextualização, sobre o problema da previsão de vazões, oferecendo ao leitor subsídios no tema. Os objetivos do trabalho são descritos em texto próprio, na seção 1.2 e a organização geral deste documento é descrita em 1.3.

1.1. A Energia Hidroelétrica e a Previsão de Vazões dos Rios

A geração de energia elétrica no Brasil utiliza previsões de vazão de rios, que são realizadas através de modelos matemáticos, estocásticos e hidrológicos e com auxílio de sistemas computacionais. Há uma diferença considerável entre as previsões e os valores observados (ROCHA, 2015).

Da matriz elétrica brasileira, 64,9% provêm da energia hidráulica (ENERGÉTICA, 2020). O parque elétrico brasileiro tem uma dependência muito grande deste tipo de energia, devido à disponibilidade de recursos hídricos e por serem economicamente viáveis (FLORES, FERREIRA e ZAMBOTI, 2017).

Apesar da previsão de vazão ser uma tarefa complexa, ela é indispensável para um melhor dimensionamento da produção de energia. É notório que, melhorando-se a acurácia deste índice, melhora-se o apoio às decisões estratégicas, otimizam-se os recursos e reduzem-se os custos adicionais com produção de energia provenientes de outras fontes para atender as demandas (ROCHA, 2015). Previsões mais precisas dão suporte a uma operação mais eficiente, com reflexos econômicos, logísticos e ambientais diretos, influindo na formulação de políticas públicas para o setor elétrico e no valor da conta do consumidor final.

Para um bom planejamento e gestão de recursos hídricos, são usados modelos hidrológicos, determinísticos, estocásticos e/ou baseados em ANNs. Tais modelos são complexos e exigem ferramentas matemáticas sofisticadas, vasto conhecimento, habilidade de parametrização dos modelos, entre outras (GOMES e MONTENEGRO, 2010).

Modelos baseados em ANNs têm sido mais desenvolvidos e utilizados, devido principalmente à evolução do poder computacional e das técnicas aprimoradas ao longo do tempo. Estas ANNs conseguem abstrair características e atributos importantes, tais como sazonalidade, periodicidade, tendências, entre outras, e

conseguem aprender com os dados, tornando-as mais eficientes e precisas. Por exemplo, GOMES et al. (2010) propuseram um modelo baseado em ANNs para previsão de vazões na Bacia do Rio São Francisco e FLORES et al. (2017) propuseram um modelo para previsão da vazão diária do afluente para a bacia do rio Preto. Ambos utilizaram ANNs, porém empregaram apenas dados históricos (medições) para tais previsões.

O uso combinado de CNNs de aprendizado profundo, dados históricos e imagens de satélites ainda é assunto em aberto e, ainda que tenha motivado este trabalho, não é parte de seu escopo. A próxima seção indica os objetivos desta pesquisa.

1.2. Objetivos do Trabalho

O objetivo geral deste trabalho foi realizar uma investigação exploratória sobre o desempenho da classificação de imagens sintéticas automaticamente rotuladas com uso de CNNs padronizadas, de arquitetura DL, em condições estritamente controladas. Buscou-se responder:

a) Qual o desempenho das CNNs no processo de classificação discreta de imagens nas quais estão incrustrados padrões precisos de métricas contínuas;
b) Quais configurações das CNNs (hiperparâmetros) são mais favoráveis ao seu desempenho nesse processo;
c) Quais percepções se pode coletar sobre como as características dos dados de entrada afetam o processo;
d) Comparativamente, qual configuração de CNN e *framework*, dentre as ensaiadas, oferece maior desempenho.

Os objetivos específicos foram:

a) Selecionar CNNs, ferramentas e configurações iniciais para estabelecimento de um ambiente experimental;
b) Construir conjuntos de dados de imagens sintéticas automaticamente rotuladas, geradas segundo diferentes modelos e métricas, construindo base de dados de experimentação e *benchmark*;

c) Determinar uma configuração dos hiperparâmetros das redes favorável ao conjunto de experimentos previsto, coletando percepções sobre a influência de diferentes ajustes de valores;
d) Avaliar comparativamente o desempenho das CNNs ensaiadas de acordo com os indicadores disponibilizados pelas ferramentas de software.

1.3. Organização do Texto

O capítulo 2 apresenta fundamentos sobre redes neurais artificiais e convolucionais e uma introdução sobre as redes e os *frameworks* utilizados. No capítulo 3 são apresentados os trabalhos relacionados. Já no capítulo 4 está descrita a metodologia empregada neste trabalho. No capítulo 5 são apresentados os Resultados e Discussão e o capítulo 6 apresenta a Conclusão. No final, o leitor encontrará as Referências e Apêndices.

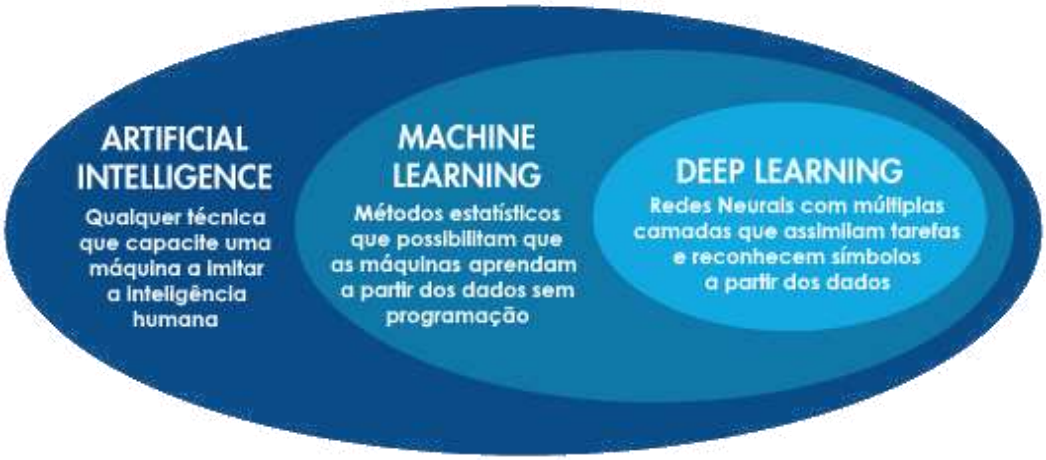
2. FUNDAMENTOS

As redes neurais encontram aplicações em diversos campos, como análise de séries temporais, reconhecimento de voz e padrões, processamento digital de imagens e sinais e controle, devido à sua habilidade de aprender a partir de dados de entrada, em processos supervisionados ou não (FLORES, FERREIRA e ZAMBOTI, 2017). Este capítulo é dedicado à parte teórica do trabalho e está dividido em subseções para descrever: 2.1) Fundamentos de redes neurais artificiais; 2.2) Redes Neurais Convolucionais; 2.3) A CNN GoogLeNet; 2.4) A CNN AlexNet; 2.5) Aspectos comparativos entre as duas CNNs mencionadas; 2.6) *Frameworks* usados.

2.1. Fundamentos de redes neurais artificiais

DL é uma subárea de *Machine Learning* (ML), que, por sua vez, é uma área de Inteligência Artificial. É um dos principais temas atuais da área de visão computacional e largamente difundido, obtendo resultados extraordinários no estado da arte (VARGAS, PAES e VASCONCELOS, 2016). A Figura 1 ilustra uma definição breve destes conceitos.

Figura 1 - Conceitos sobre Inteligência Artificial e suas subáreas



Fonte: (LAVAGNOLI, 2020)

A capacidade computacional tem evoluído constantemente, principalmente depois que pesquisadores descobriram que, a utilização de GPUs[2], que até então eram utilizadas principalmente para jogos virtuais, passou a ser essencial para utilização em DL. Essas GPUs têm muitos núcleos, como a utilizada neste trabalho, a Titan X Pascal da NVIDIA que possui cerca de 3.000 CUDA *cores*. Isso facilita cálculos matemáticos, como por exemplo, multiplicação de vetores e matrizes, em paralelo e com uma velocidade muito superior quando comparado a uma CPU, viabilizando a utilização de DL na resolução dos mais variados tipos de problemas.

Segundo (HALEV-SHWARTZ e BEN-DAVID, 2014), *ML* é empregado em questões de detecção automatizada de padrões significativos, como por exemplo, mecanismos de buscas aprendendo como trazer os melhores resultados, software de AntiSpam que aprende como filtrar e-mails, software de detecção de fraudes em cartões de créditos, carros equipados com sistemas de prevenção de acidentes e celulares com recursos de reconhecimento de rostos e voz. *ML* também é muito utilizado em aplicações científicas como bioinformática, medicina, astronomia, e muitas outras áreas.

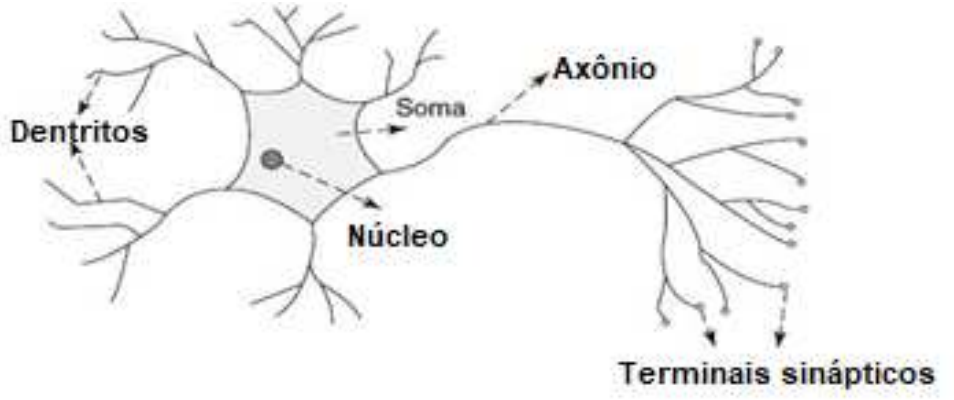
DL são técnicas que, através de algoritmos, abstraem atributos de conjuntos de dados, representado assim por um modelo matemático. Segundo (PONTI e COSTA, 2017), em *DL* existem métodos que aprendem a determinar uma função arbitrária, por meio de composições de funções. A profundidade (camadas) permite à rede aprender, a partir dos dados, sequências de funções que transformam vetores mapeando-os de uma camada para outra até que atinja o resultado, ou bem próximo ao desejado.

Em seu artigo, MAKANTASIS (2020) cita que técnicas baseadas em *DL* têm demonstrado resultados promissores tanto para detecção de objetos, como veículos, faces, quanto em classificação de dados hiper espectrais. CHEN et al. (2014), citam em seu artigo que uma técnica comum nessas aplicações é a classificação de cada pixel da imagem em dados hiper espectrais e está se tornando uma ferramenta valiosa para monitorar a superfície terrestre. Tem sido muito utilizado em diversos segmentos como mineralogia, física, agricultura, astronomia, química, meio ambiente entre outras áreas.

[2] *Graphics Processing Units*

Há várias técnicas desenvolvidas pela comunidade de pesquisadores, porém uma que tem sido muito utilizada é ANN. Estas foram inspiradas na biologia humana, conforme pode-se ver na Figura 2. DONGARE et al. (2012) descrevem que “Neural” é um adjetivo para neurônio, e “Network” é uma estrutura semelhante a um grafo. ANNs são sistemas de computação e foram baseados na analogia das redes neurais biológicas, que segundo REDDY et al. (2002), “são sistemas de informação não lineares e paralelos, que se caracterizam pela robustez, tolerância a falhas e capacidade de aprender”.

Figura 2 - Neurônio Biológico

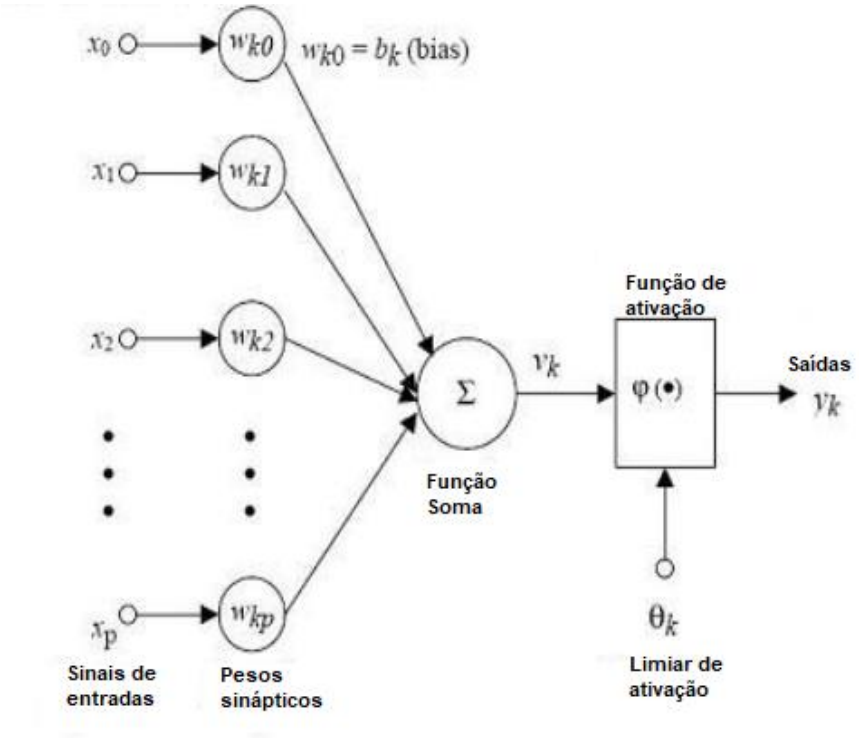


Fonte: Adaptado de SHARMA et al. (2012)

ANNs são codificadas em uma linguagem de computador, por exemplo Python, Java, entre outras, que simulam o comportamento de como o cérebro humano processa as informações. Elas acumulam os conhecimentos, através da detecção de padrões e relacionamento entre os dados, e o processo de treinamento é onde elas aprendem com os dados. DONGARE et al. (2012) descrevem que são muito utilizadas no mapeamento de funções não lineares complexas, processamento digital de imagens, reconhecimento de padrões e classificação, entre outras aplicações. Elas consistem em muitos neurônios interligados, cada um com seu peso sináptico (w) e são formadas por camadas (AGATONOVIC-KUSTRIN e BERESFORD, 1999).

O primeiro modelo de ANN criado foi o *Perceptron*, que utiliza uma única camada de neurônios e que consegue classificar dados em apenas duas classes. Este modelo possui pesos (w) que são atualizados e a saída é baseada em um limiar (*threshold*). É um modelo bem simples, porém é a base para muitos outros modelos mais avançados. Ele consegue resolver funções lógicas como AND, OR e NOT (WALLISCH, LUSIGNAN, *et al.*, 2009). Na Figura 3, está ilustrado um modelo *Perceptron*, com várias entradas, representada por X e uma saída, representada por Y.

Figura 3 - Arquitetura básica do Perceptron



Fonte: Adaptado de DONGARE et al. (2012)

A expressão matemática é dada por Fonte: DONGARE et al. (2012):

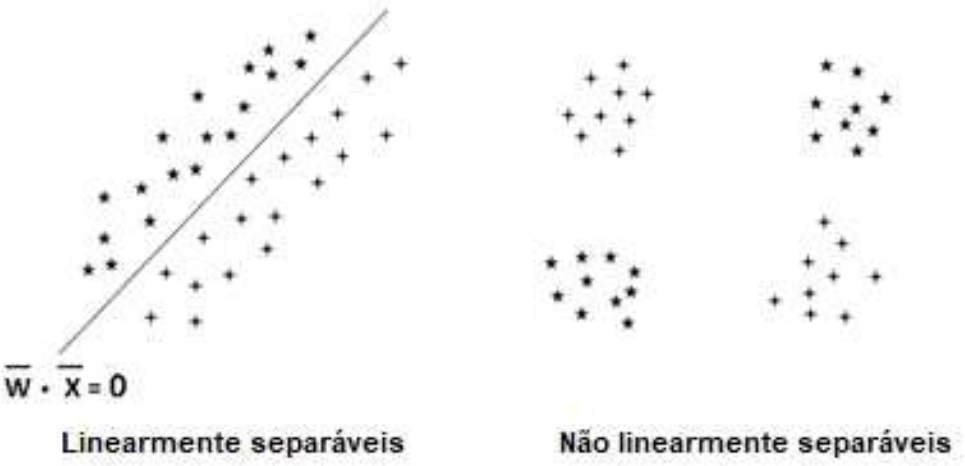
$$V_k = \sum_{j=1}^{p} W_{kj} X_j \qquad (1)$$

Na década de 1970, percebeu-se que o *Perceptron* não era capaz de aprender a função lógica do tipo XOR que é uma função não linear. Tempos após, novos modelos foram desenvolvidos, “ADALINE” e “MADALINE”, sendo o último a

primeira ANN utilizada para uma situação real, como por exemplo em FLORES et al. (2017) que propôs um modelo chuva-vazão para previsão de vazão de afluente.

Em 1986, foi criada a ANN de múltiplas camadas, *Multi-layer Perceptron* (MLP) (*Backpropagation*). Este modelo consegue aprender com o erro, é realimentado e utiliza várias iterações para o aprendizado. Foi, então, possível implementar problemas não lineares. Este tipo de *Perceptron* foi chamado de *Multi-layer Perceptron* (MLP), criado por Geoffrey*. A Figura 4 ilustra o que são dados linearmente e não linearmente separáveis.*

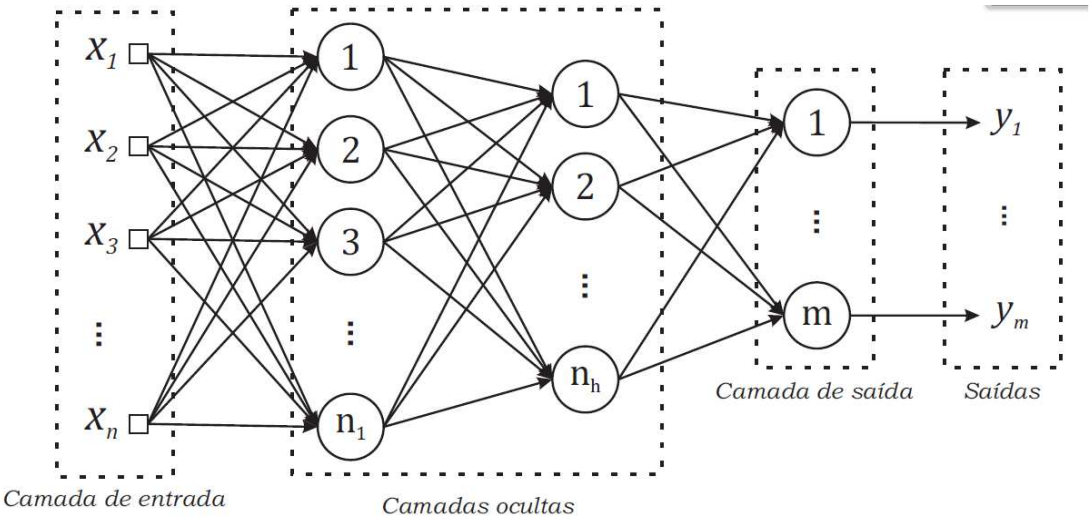
Figura 4 - Ilustração de dados linearmente e não linearmente separáveis



Fonte: (AGGARWAL, 2018)

Uma ANN pode ser dividida em três camadas: entrada, intermediária ou oculta e saída. e na Figura 5 representa uma ANN e suas camadas.

Figura 5 - Identificação das partes de uma rede neural.



Fonte: (BELOTTI, 2019)

- Camada de entrada: É a responsável por receber os dados de entrada. Esta camada não possui neurônios e os dados são geralmente normalizados.
- Camada oculta, intermediária ou escondida: Esta possui neurônios e a maior parte do processamento é realizado dentro desta camada e é nela que os atributos e padrões são extraídos.
- Camada de saída: Responsável pela apresentação dos resultados, de acordo com os cálculos das camadas anteriores. Esta camada é composta por tantos neurônios quantos forem a saída. Por exemplo, se a saída fosse a identificação de dígitos, esta camada teria 10 neurônios, um para cada dígito (BELOTTI, 2019).

SANTOS et al. (2016) definem que uma rede com aprendizado supervisionado é aquela onde existe uma resposta para o problema apresentado e que seja possível compará-la com a resposta obtida. No aprendizado não supervisionado não existe o agente externo que indica a resposta para os padrões de entrada. Este modelo procura encontrar similaridades entre os dados/objetos de entrada.

ANNs têm sido utilizadas em inúmeras aplicações e têm apresentado resultados excelentes, sendo que uma de suas aplicações é previsão de vazão de bacias hidrográficas (GOMES e MONTENEGRO, 2010).

Um dos maiores desafios ao se utilizá-las para previsão de vazão é a super parametrização de entradas segundo (ADEYEMO, OYEBODE e STRETCH, 2018). Em seu trabalho, propuseram a utilização de ANN para previsão de vazão do rio Mkomazi, Sul da África e avaliaram dois métodos distintos sendo que, em seu primeiro método, enfrentaram problemas de *overfitting*, quando o sistema já conhece muito bem os dados que estão sendo submetidos a treinamento e em teoria, o sistema decorou e não aprende mais e o outro problema foi o de super parametrização. Dos 120 resultados obtidos, apenas os 24 melhores foram selecionados para análise. A melhor arquitetura foi a RC315L[3], selecionada dentre elas. O melhor resultado, se analisados os três parâmetros estatísticos: R^2 (Coeficiente de Correlação de Pearson), E (Erro Padrão) e EPE (Erro Padrão da Estimativa) foram com valores de 0,92, 77% e 8,29 respectivamente e pode ser utilizado para simular a previsão de vazões médias mensais para a bacia em questão.

Há várias arquiteturas de ANNs e uma delas é a CNN, que tem sido muito utilizada na resolução de problemas complexos, como reconhecimento e classificação de imagens utilizando uma sequência de camadas *feed-forward* (RAWAT e WANG, 2017).

As CNNs apareceram pela primeira vez no final da década de 1980 com o reconhecimento de código postal manuscrito como uma versão estendida de ANNs. Uma das camadas deste tipo de rede é a camada convolucional, que normalmente é seguida por uma camada de agrupamento (*pooling*), que tem o objetivo de reduzir a dimensionalidade dos mapas de características (*feature maps)*. Isso reduz o custo computacional associado ao treinamento da rede e diminui as chances de *overfitting* (BAYAR e STAMM, 2020).

As CNNs têm evoluído especialmente na classificação de imagens. Um dos requisitos básicos para sua utilização é a abundância de dados (imagens), pois no processo de treinamento a rede precisa de muitas imagens para poder abstrair os

[3] Nomenclatura adotada pelo autor. RNA com normalização (RC), 3 dados de entrada, 15 neurônios na camada oculta, função de ativação/transferência *Log-Sig*. (RC315L)

atributos e características para o aprendizado.

Está se tornando mais comum o uso de imagens de satélites para diversos tipos de aplicações. Estas imagens possuem alta resolução, confiabilidade e fácil acesso e, por conta disso, muitos pesquisadores que utilizam classificação e reconhecimento de imagens nos modelos tradicionais passaram a utilizar imagens de sensoriamento remoto (ABBAS e HAMZAH, 2020).

Com a popularidade do *DL*, vários *frameworks*[4] foram criados e ou melhorados para facilitar e aperfeiçoar o desenvolvimento e implementação destes métodos. Pode-se citar alguns deles: Caffe, DeepLearning4J, TensorFlow, Theano e Torch. Dentre estes vários *frameworks*, o Caffe, Theano e Torch são alguns dos mais utilizados atualmente na comunidade de DL e, finalmente, o TensorFlow tem recebido muita atenção desde sua primeira *release* e sua utilização tem crescido e sido aprimorado ao longo do tempo (BAHRAMPOUR, RAMAKRISHNAN, *et al.*, 2016).

A próxima seção é dedicada à parte teórica e apresenta as componentes de uma CNN, sua importância e utilização.

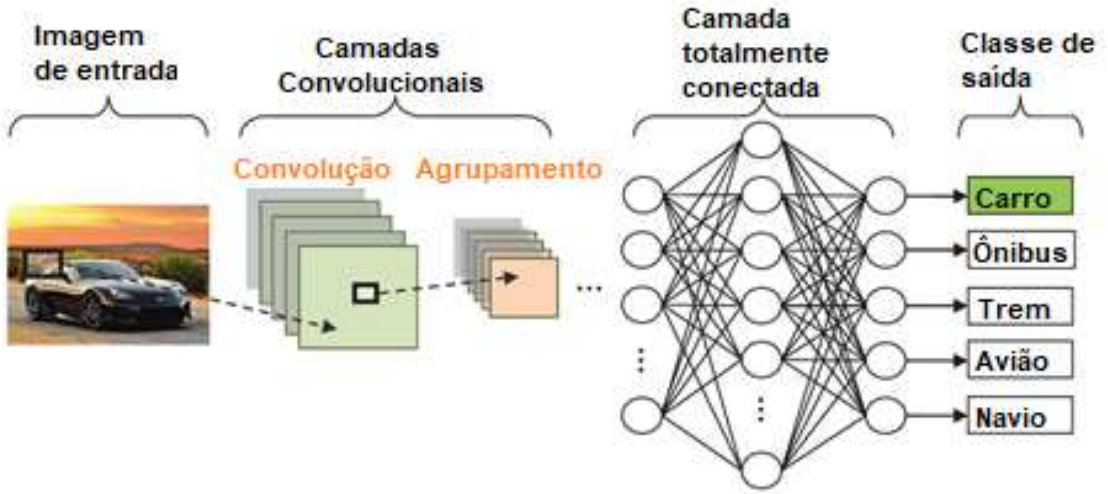
2.2. Redes Neurais Convolucionais (CNN)

Uma das principais aplicações de uma CNN é a utilização em processamento digital visual, em especial imagens, pois, com a camada de convolução, é possível a utilização de filtros considerando sua estrutura bidimensional (PONTI e COSTA, 2017). Uma CNN utiliza várias camadas para o processamento das informações não lineares e é capaz de abstrair atributos das imagens de forma que consiga fazer o reconhecimento de objetos e detalhes importantes.

[4] São projetos reusáveis de um programa ou parte dele, que são expressos como um conjunto de classes (RE, 2002).

A estrutura básica de uma CNN é composta por camadas convolucionais[5], camadas de *pooling*[6] seguidas por uma função de ativação e camada totalmente conectada (*fully connected (FC)*), que é a camada de saída (ABBAS e HAMZAH, 2020). A Figura 6 ilustra a arquitetura de uma CNN.

Figura 6 - Arquitetura básica de uma CNN



Fonte: Adaptado de (RAWAT e WANG, 2017)

Para construção de uma CNN, tem-se os seguintes componentes:

1. Camada de entrada
2. Camada de Convolução
3. Filtros convolucionais (*kernel)*
4. *Conceitos de filtragem de imagens*
5. Camadas de Agrupamento (*Pooling*)
6. Função de ativação
7. *Processo* de aprendizagem
8. Otimizador (Função de custo)
9. Regularização

[5] Camada composta por vários neurônios, que são responsáveis por aplicar o filtro uma parte da imagem e é aqui que se extrai os atributos da imagem (VARGAS, PAES e VASCONCELOS, 2016).
[6] Camada de *pooling* ou agrupamento tem a finalidade de reduzir a dimensionalidade dos dados na rede.

10. Camada de saída
11. Parâmetros e Hiperparâmetros

Nas próximas subseções, serão apresentados a definição de cada componente de uma CNN.

2.2.1. *Camada de entrada*

Esta é a camada que receberá como entrada os dados, usualmente imagens. Cada pixel da imagem corresponderá a um neurônio nesta camada. Esta camada será a responsável por transferir os dados à camada seguinte, que é a convolução.

2.2.2. *Camada de convolução*

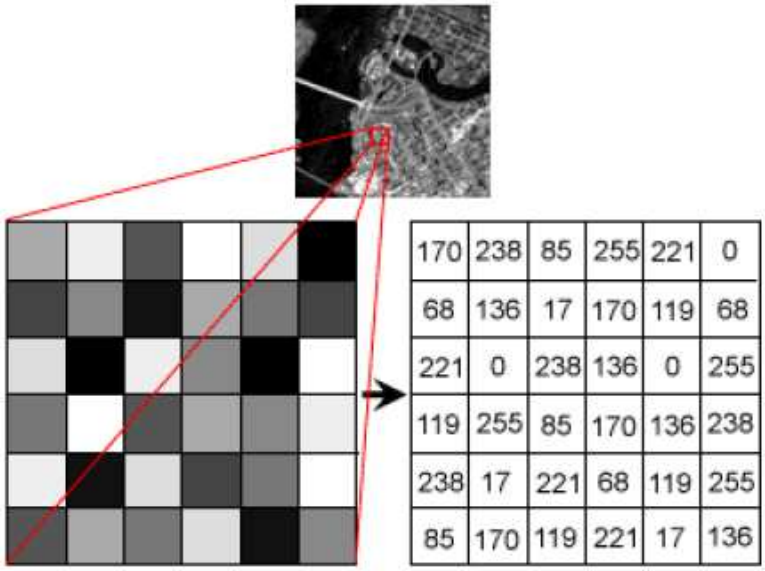
A camada convolucional é uma das mais importantes da CNN, pois é nesta camada que os atributos (*features)* são extraídos da imagem e é onde a rede aprende (RAWAT e WANG, 2017). Ela é composta por uma pilha de operações matemáticas, um tipo especializado de operação linear. Em imagens digitais, os pixels são armazenados em números inteiros e dispostos em uma matriz multidimensional e em seguida é aplicado o filtro, ou também conhecido como *kernel*, um extrator de características. Este recurso torna as CNNs muito eficientes para processamento de imagens digitais (YAMASHITA, NISHIO, *et al.*, 2019).

Cada neurônio da camada é um filtro aplicado à imagem, sendo que o filtro é uma matriz de pesos (w), definidos com a arquitetura K x K x D, onde ‘K’ é a dimensão espacial do filtro e ‘D’ representa a profundidade do filtro. Caso a imagem seja do tipo RGB, então teremos D = 3, pois representa os canais de cores. Os filtros mais utilizados são os de dimensões 5 x 5 x D, 3 x 3 x D e 1 x 1 x D (PONTI e COSTA, 2017).

A imagem é uma matriz de números inteiros, cada qual representando um componente da cor do pixel, que é a entrada da camada de convolução. Se a imagem for em escala de cinza, são 8 bits (2^8 => 0 a 255) e caso seja colorida, são 3 canais de cores, então, são 24 bits (2^{24} => 0 a 16.777.215). Na Figura 7 tem-se a

representação de uma imagem em uma matriz de número como ilustração.

Figura 7 - Exemplo de uma imagem, destacando o detalhe dos pixels e sua representação matricial



Fonte: (HOCHULI, 2020)

2.2.3. Filtros convolucionais ou *kernels*

O filtro é uma matriz com valores, que será aplicada sobre a imagem para obter algum efeito. Cada região da imagem que passou pelo filtro é denominada como campo receptivo local (*local receptive field*). Esta técnica é usada para determinar regiões de interesse da imagem. Os valores da matriz dependem do objetivo do filtro. A imagem é uma matriz que, quando multiplicada por este filtro, uma nova imagem será gerada, conforme o tipo de filtro escolhido.

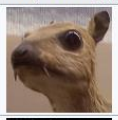
É necessário definir a arquitetura do filtro, como quantidade, tamanho e passo (*strides*[7]), porém não é necessário definir qual filtro utilizar, pois a própria rede determinará o melhor filtro a ser utilizado. É muito comum utilizar *stride* maior que 1, quando deseja-se reduzir o tempo de processamento, pulando pixels, o que reduz o tamanho da imagem (PONTI e COSTA, 2017).

[7] *Stride* ou passo: É a quantidade de pixels que o filtro irá se mover durante a leitura dos dados.

2.2.4. Conceitos de filtragem de imagens

Os mais conhecidos e utilizados são os filtros para detecção de bordas ou *edge detection*, nitidez ou *sharpen*, suavização ou *blur* (POWELL, 2020). A Figura 8 apresenta alguns filtros com sua matriz correspondente e o resultado da imagem de cada filtro como ilustração.

Figura 8 - Filtros convolucionais de diferentes tipos aplicados a uma imagem.

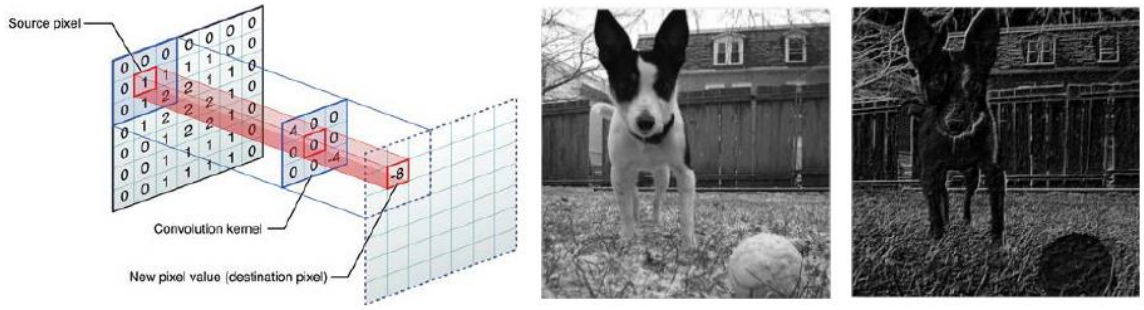
Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Operation	Kernel ω	Image result g(x,y)
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Fonte: (Kernel (image processing), 2020)

A Figura 9 apresenta a ação de um filtro de detecção de bordas que, aplicado a uma imagem, resulta em uma imagem escurecida, porém com as bordas salientadas.

Figura 9 - Imagem passando pela camada de convolução e como saída, salientando as bordas



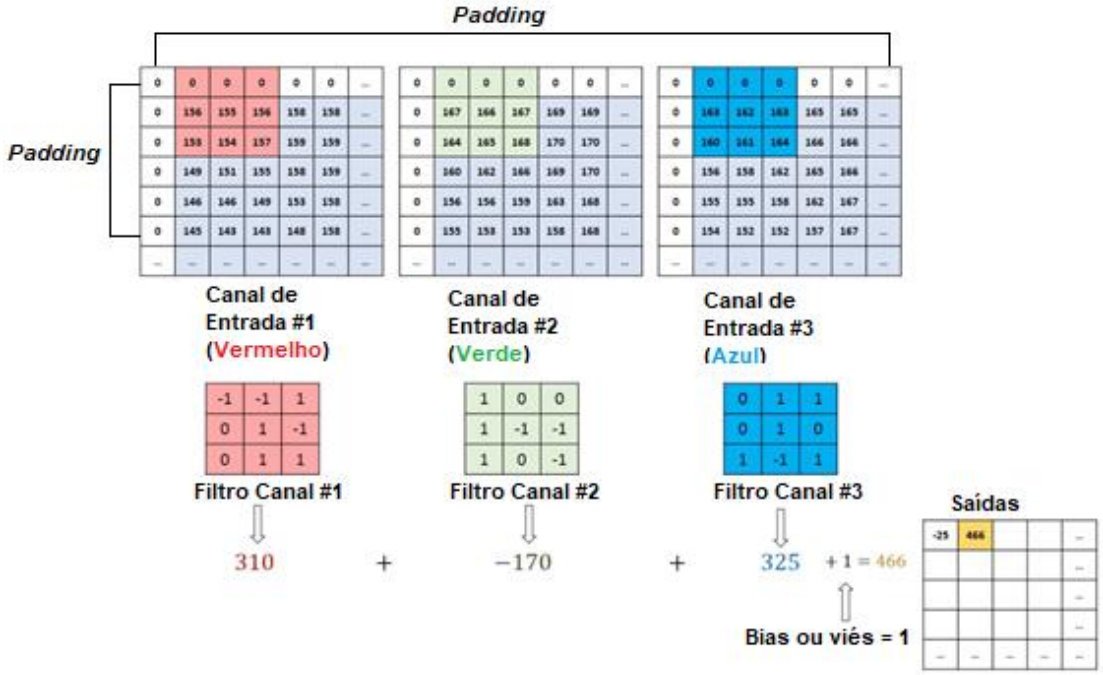
Fonte: (HOCHULI, 2020)

Um dos cuidados ao escolher o tamanho do filtro, é determinar o *padding*. Alguns *frameworks*, como o TensorFlow, se encarregam de acrescentá-lo à imagem. *Padding* é uma técnica que consiste em completar as colunas e linhas com zeros em cada lado do tensor[8] de entrada, de forma que, quando o filtro passar sobre a imagem, não haverá nenhum valor nulo e amplia-se a imagem (YAMASHITA, NISHIO, *et al.*, 2019).

Na Figura 10, uma imagem com 3 canais de cores (RGB) está representada por uma matriz de números inteiros. O *kernel* tem dimensão 3x3 e se desloca sobre a imagem com *stride* igual a 1. Para evitar valores nulos na matriz da imagem foi adicionado o *padding* nas colunas e linhas. A operação de cada canal de cores pelo seu filtro correspondente, adicionado o *bias*, resulta o pixel de saída.

[8] É uma matriz multidimensional com 3 ou mais dimensões.

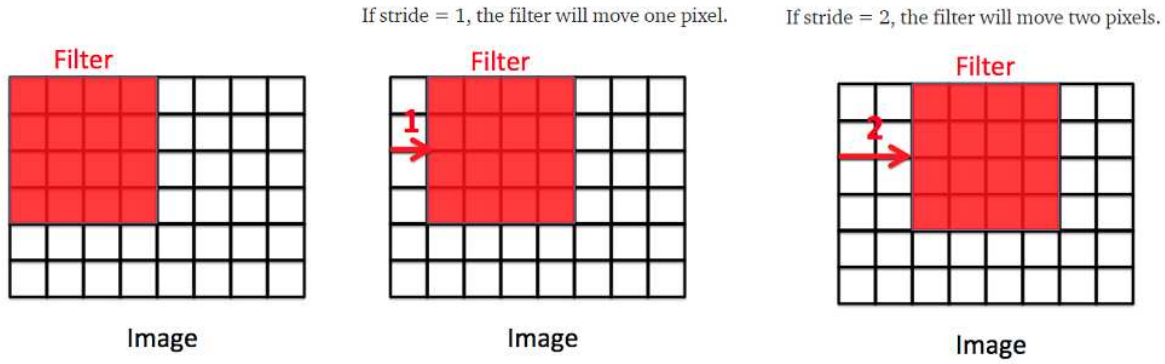
Figura 10 - Filtro 3x3 aplicado a cada um dos canais de cores uma imagem com *padding*.



Fonte: Adaptado de (WANGENHEIM, 2018)

Na Figura 11, tem-se a aplicação de um filtro com três diferentes *strides*, sendo que nos *strides* igual a dois e três, a imagem é reduzida. Seja uma imagem de 224 x 224 x 3, onde 3 representa os canais de cores RGB, e aplicamos um *stride* igual a dois, a imagem é reduzida para 112 x 112.

Figura 11 - Ilustração de um filtro convolucional com passo (*stride*) de 1 e 2

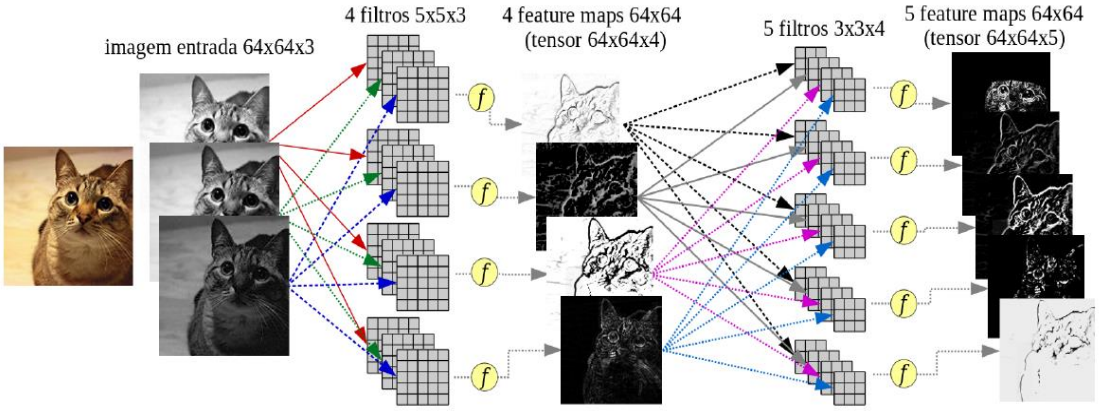


Fonte: (CHEN, 2017)

Quando o filtro passar pela imagem, dentro da camada de convolução, após ter sido definido os atributos dos filtros, serão gerados os mapas de características, mais conhecido como *features maps*. Cada região da imagem processada por este filtro, é chamada de campo receptivo local (*local receptive field*). Todos estes campos receptivos são filtrados utilizando os mesmos pesos para cada pixel.

Na Figura 12, está ilustrado como funciona a camada de convolução. Sobre a imagem do gato, cuja dimensão é de 64x64x3 (3 cores: RGB), são aplicados 4 filtros 5x5x3, e, após a aplicação da função de ativação (f), são gerados na camada seguinte 4 *features maps* (mapas de características), novas imagens, com dimensões de 64x64x4. Estes *features maps* são empilhados, formando tensores cuja profundidade é igual ao número de filtros, 4 (PONTI e COSTA, 2017). Estes tensores são as entradas para a próxima camada de convolução.

Figura 12 - Ilustração de aplicação de filtros convolucionais em uma imagem



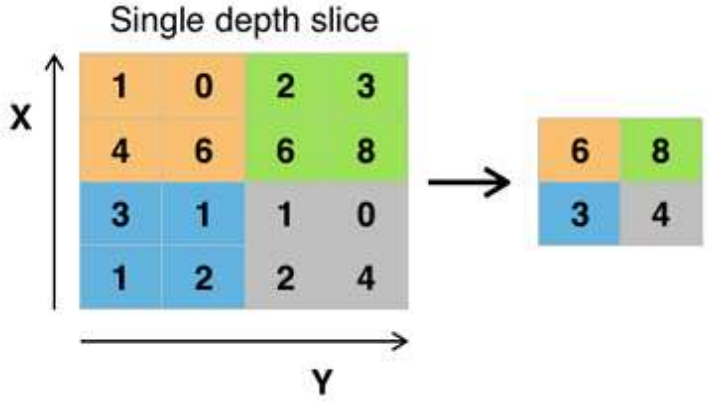
Fonte: (PONTI e COSTA, 2017)

Na segunda camada, cuja entrada são os tensores de 64x64x4, aplica-se 5 filtros de dimensão 3x3x4, que resultará 5 *features maps*, ou seja, tensores de 64x64x5. É possível notar que, ao passar o filtro convolucional sobre a imagem, é gerado uma sub amostragem da imagem, porém com algumas características, como por exemplo, destaque de bordas horizontais e bordas verticais.

2.2.5. Camada de Agrupamento ou *Pooling*

Outra camada muito importante, e sempre utilizada após a camada de convolução e ativação, é a camada de *pooling*, cuja funcionalidade é reduzir a dimensionalidade dos dados, objetivando agilidade no treinamento e agrupamento do conjunto de dados, que torna o processo de treinamento mais ágil e cria uma invariância espacial. Existem várias funções de *pooling*, porém a mais utilizada é a *Max Pooling,* representada na Figura 13 (GOMES, PAES e NADER, 2020). A redução é feita através da seleção do maior número da região demarcada, por exemplo, dentre os números 1, 0, 4, 6, o maior número é o 6, então ele é o escolhido. Com isso, reduz-se a dimensionalidade da imagem.

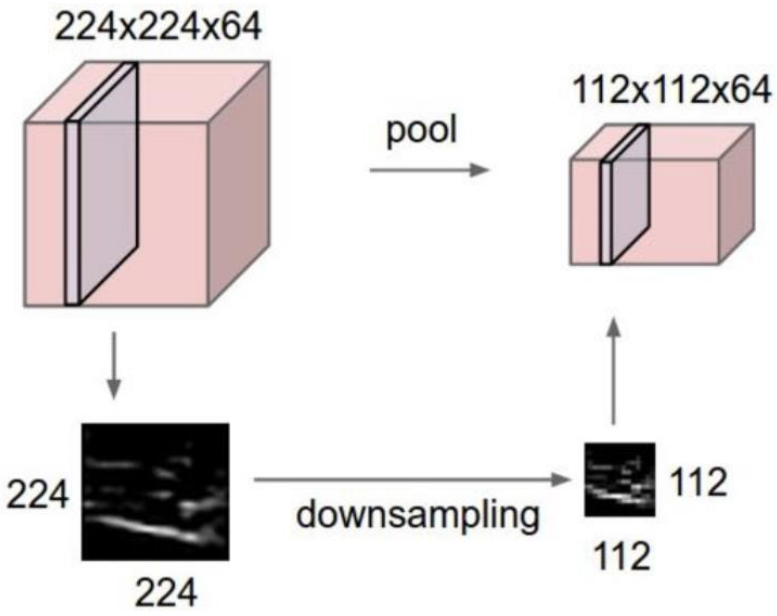
Figura 13 - Função de agrupamento - Max Pooling



Fonte: (FLORINDO, 2020)

No trabalho de (SPRINGENBERG, DOSOVITSKIY, *et al.*, 2015), sugerem que a camada de *pooling* possa ser substituída pelo aumento dos *strides* nas camadas de convolução, o que em termos, reproduz a mesma função, ou seja, a redução da dimensionalidade da imagem. Na Figura 14, temos a aplicação de *pooling* sobre uma imagem e a redução da sua dimensionalidade.

Figura 14 - Aplicação de *pooling* sobre a imagem com redução da dimensionalidade



Fonte: (BRAZ JUNIOR, 2020)

2.2.6. Função de Ativação

Um outro componente muito importante de uma CNN é a função de ativação, que é aplicada à saída do neurônio conforme o seu limiar (*threshold*). BELOTTI (2019) define que é o responsável por limitar a saída do neurônio dentro de um intervalo de valores razoáveis a serem assumidos pela sua própria imagem funcional.

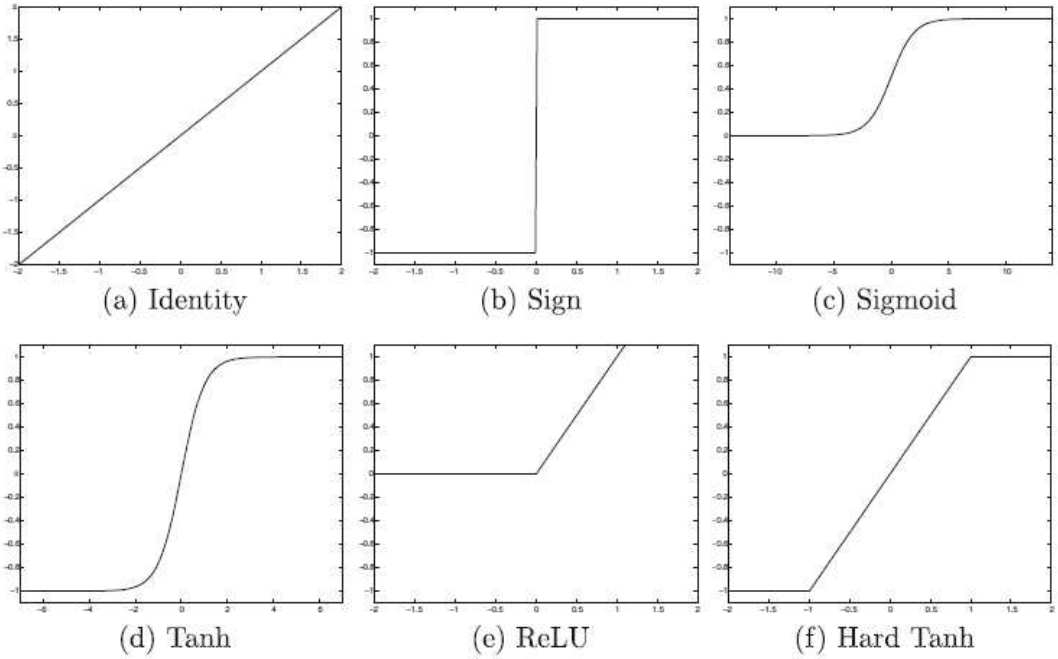
A função de ativação, presente na saída de cada neurônio é responsável por ativar ou não o neurônio seguinte. Essas funções de ativação têm geralmente algum grau de não-linearidade nas camadas intermediárias. A função de ativação ReLU é muito utilizada e trunca para zero os pixels negativos (PONTI e COSTA, 2017). Existem várias funções de ativação e cada uma com um propósito distinto. Uma das funções mais conhecidas é a Sigmóide (não linear) que é dada pela equação (DSA, 2020):

$$f(x) = \frac{1}{1+e^{-x}} \qquad (2)$$

As funções de ativação são classificadas em dois grupos: parcialmente diferenciáveis e totalmente diferenciáveis. O primeiro grupo compreende as funções em que pelo menos um ponto não possui derivada de primeira ordem definida e as três principais funções são: degrau, degrau bipolar e rampa simétrica. No segundo grupo, aquelas em que as derivadas de primeira ordem existem e são conhecidas para todos os pontos do seu domínio, sendo que as duas funções mais utilizadas são a logística e tangente hiperbólica (BELOTTI, 2019).

Dentre algumas das funções de ativação tem-se: Função Linear, Sigmóide, Tangente Hiperbólica (TanH), ReLu, Leaky ReLu e Softmax. A Figura 15, apresenta o gráfico de algumas destas funções de ativação.

Figura 15 - Exemplos de funções de ativação



Fonte: (AGGARWAL, 2018)

Na última camada, aplica-se a função de ativação correspondente, dependendo do objetivo da CNN. O Quadro 1 apresenta algumas funções de ativação que são aplicadas na última camada e seu propósito correspondente.

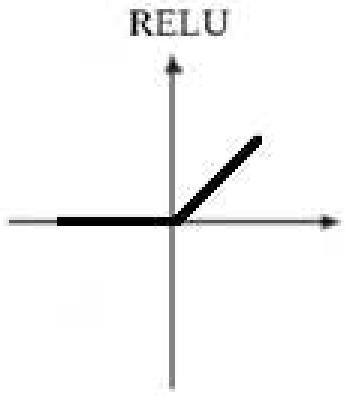
Quadro 1 - Lista das funções de ativação aplicadas à última camada para diferentes propósitos

Tipos de classificação	Função de ativação da última camada
Classificação binária	Sigmóide
Classificação multiclasses de classe única	Softmax
Classificação multiclasses em multiclasses	Sigmóide
Regressão para valores contínuos	Identidade (identity)

Fonte: (YAMASHITA, NISHIO, *et al.*, 2019).

A função RELU (*Rectified Linear Function*) é muito utilizada nas camadas ocultas por facilitar o treinamento, pois é uma função identidade para valores positivos, ou seja, ela cancela todos os valores negativos, enquanto é linear para os valores positivos (PONTI e COSTA, 2017). A Figura 16 apresenta uma representação da RELU e em seguida a função correspondente.

Figura 16 - Função de ativação RELU

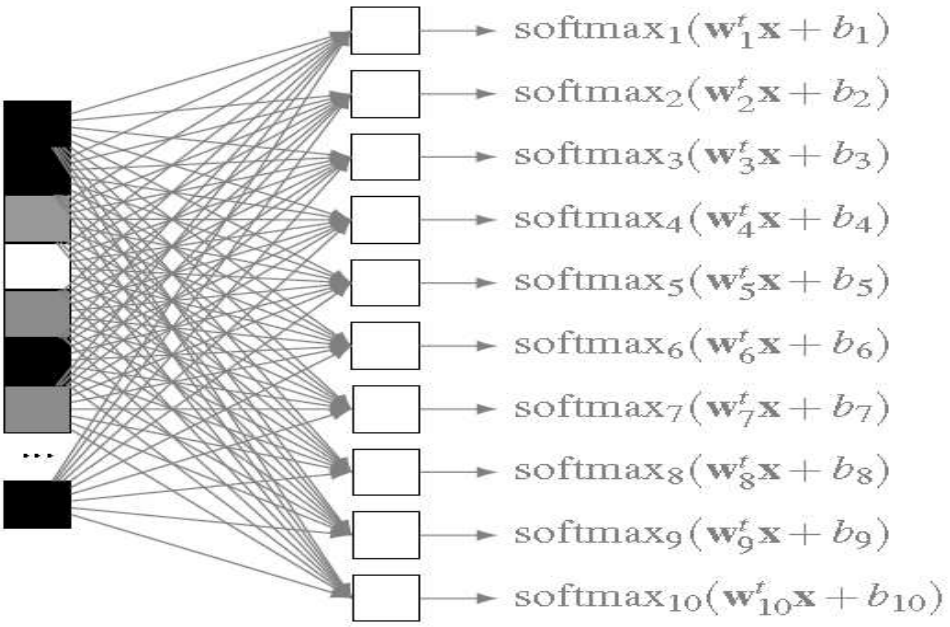


Fonte: Adaptado de (HOCHULI, 2020)

$$f(x) = \begin{cases} \mathbf{0}\ para\ x < 0 \\ \boldsymbol{x}\ para\ x\ \geq 0 \end{cases} \quad (3)$$

A Figura 17 apresenta a última camada, *fully connected*, e o emprego da função de ativação *SoftMax* para gerar a saída. Esta é uma função de classificação (PONTI e COSTA, 2017), e comumente utilizada na última camada da rede e cuja saída não se limita a duas classes ou menos, por exemplo: classificação de dígitos manuscritos, onde a saída compreende 10 neurônios, 1 para cada dígito.

Figura 17 - Emprego da função de ativação (SoftMax) na camada totalmente conectada para geração da probabilidade de saída de cada termo



Fonte: (PONTI e COSTA, 2017)

2.2.7. Processo de Aprendizagem

Após a criação das redes, com todas suas camadas, hiperparâmetros definidos, tem-se o processo de aprendizagem. Os *datasets*, que neste caso são as imagens, são geralmente divididos em três diferentes subconjuntos:

- Treinamento: Este conjunto de imagens é utilizado para treinar a rede. O erro é calculado em relação à entrada e os parâmetros aprendidos são atualizados por meio do algoritmo *backpropagation*.
- Validação: Este conjunto é utilizado para monitorar o desempenho da rede no processo de Treinamento e ajuste dos hiperparâmetros.
- Testes: Está é a última etapa do processo. O *dataset* passará por este processo em que será avaliado o desempenho do modelo.

Treinar uma rede é um processo de encontrar os *kernels* das camadas de convolução e os pesos das camadas FC de tal forma a minimizar o erro, que é a relação entre a entrada e a saída (YAMASHITA, NISHIO, *et al.*, 2019).

A fase de Treinamento é onde a rede aprende os pesos (w) a cada iteração, também chamado de época (*epoch*), e vai modificando-os ao longo deste processo, até que encontre os melhores coeficientes para os filtros (VARGAS, PAES e VASCONCELOS, 2016). É usual que nesta fase se escolha uma porcentagem de imagens muito maior, cerca de 70 e 80% e o restante são para Testes. O objetivo destes ajustes de pesos é reduzir ao mínimo o erro (FLORES e FERREIRA, 2017). Neste processo, a rede extrai informações importantes, como padrões, atributos e tendências do sistema.

A taxa de aprendizagem (*learning rate*) tem um papel importante neste processo. É um dos hiperparâmetros mais importantes a ser definido pelo usuário antes do início do treinamento da rede (YAMASHITA, NISHIO, *et al.*, 2019). É ela quem definirá o quão rápido ou lento o erro será ajustado. Se utilizarmos uma taxa de aprendizado (α) muito baixa, o método ficará muito lento e pode conseguir convergir para um mínimo local da função de erro. Por outro lado, uma taxa muito alta, o método poderá divergir (LEG/UFPR, 2020). Um valor de taxa muito utilizado em CNNs é de 10^{-3}.

O Bias ou Viés, é um dos hiperparâmetros utilizados em CNN. É um valor aleatório, invariante, escolhido pelo idealizador da rede cuja finalidade é somar à entrada do neurônio, impedindo assim que uma entrada seja nula. Ele é também chamado de limiar de ativação ou entrada de polarização (BELOTTI, 2019). Esta constante faz com que o modelo se adapte melhor aos dados fornecidos.

2.2.8. Otimizador (Função de custo)

Para minimizar o erro é necessário escolher a função de perda (*loss function)*, que também é conhecida como função de custo (*cost function)*. Normalmente, na classificação em multiclasses, a *loss function* utilizada é a entropia cruzada (*cross entropy*[9]*)*, enquanto o erro médio quadrátrico (mean *squared error*[10]*)* é tipicamente aplicado à regressão contínua de valores. A escolha desta função é um dos hiperparâmetros definidos pelo usuário e deve ser escolhida dependendo do

[9] Ou entropia cruzada, é usada frequentemente para quantificar a diferença entre duas distribuições de probabilidade. Ela determina o quão próximo está a distribuição prevista da verdadeira.
[10] Ou erro quadrático médio, que é a diferença entre o valor verdadeiro e o erro, que é dado pela média do quadrado de todos os erros.

objetivo da rede (YAMASHITA, NISHIO, *et al.*, 2019).

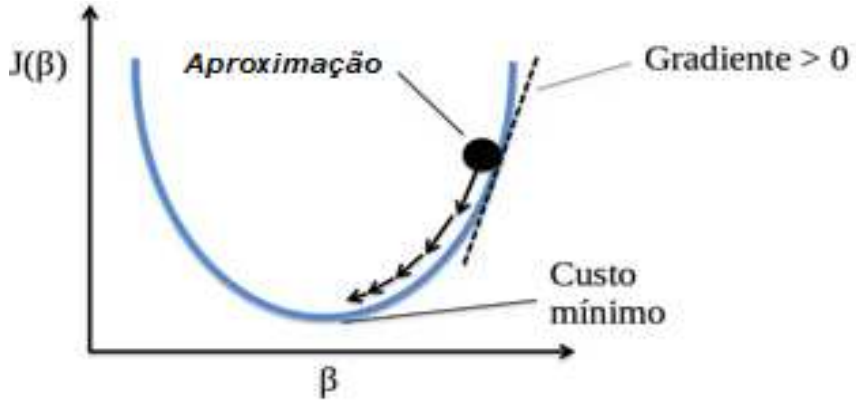
Através do gradiente descendente (explicação no parágrafo adiante), cujo objetivo é buscar o ponto mínimo, o erro é derivado em função da sua entrada, em sentido oposto ao gradiente, e é retro propagado (*backpropagation*) à rede e faz a modificação dos pesos de cada camada (FLORES, FERREIRA e ZAMBOTI, 2017).

O erro é reduzido a cada passada, época, utilizando de um algoritmo otimizador. Uma vez definida a *cost function*, os parâmetros são definidos de tal forma que consiga reduzir o erro ao mínimo possível. Muito comum a utilização do Gradiente Descendente Estocástico (Stochastic Gradiente Descent (*SGD)*) em combinação com o método de *backpropagation,* que obtém o gradiente utilizando a regra da cadeia. O *SGD* é eficiente, pois ele utiliza amostras aleatórias dos dados e não todas as instâncias existentes, o que deixa o processo mais ágil. Existem outros otimizadores como Momentum, AdaGrad, RMSPro e Adam (PONTI e COSTA, 2017).

Em outras palavras, o que o SGD faz é encontrar os pesos e o *bias* ideal, de forma que torne o custo o menor possível. Através do processo iterativo, ele toma a direção negativa do gradiente calculado. Ele tenta encontrar o mínimo da função erro (função erro, ou *loss function*) e vai apontar para onde a função está crescendo, porque isso toma a direção contrária. a Figura 18 ilustra a função.

O processo de atualização dos parâmetros é dado pela seguinte expressão: (considerando que (Xi,Yi) são amostras durante o processo de treinamento e um valor inicial $\beta^{(0)} \epsilon R^{p,}$ o algoritmo se repete até atingir a convergência (LEG/UFPR, 2020).

Figura 18 - Cálculo do erro mínimo utilizando o algoritmo Gradiente Descendente Estocástico (SGD)



Fonte: Adaptada de (LEG/UFPR, 2020)

$$\beta^{(k+1)} = \beta^{(k)} - \alpha_k \nabla J(\beta^{(k)}; \mathbf{x}_i, y_i),\ k = 0,1,\ldots \qquad (4)$$

O que o algoritmo faz é comparar a entrada à saída. Esta diferença é o que se denomina de erro. Este erro é retropropagado (*backpropagation*) às camadas anteriores, através da derivada gradiente e os pesos dos neurônios são ajustados e os dados de saída entram novamente na rede, porém, utilizando os pesos ajustados.

O processo do cálculo do erro é dado pela derivada parcial da perda em relação a cada parâmetro aprendível e o algoritmo retro propaga este erro às camadas anteriores, conforme a equação:

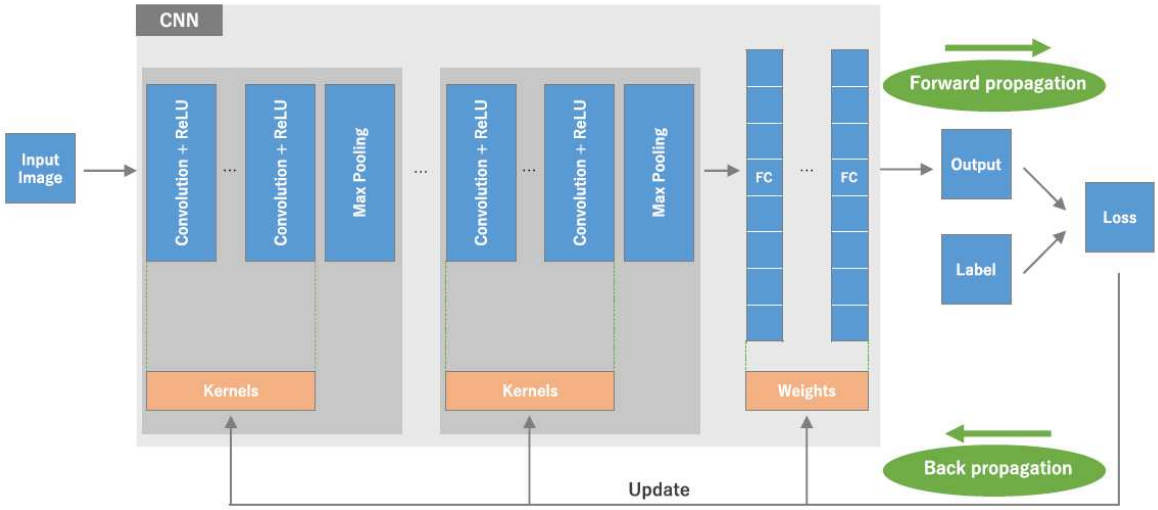
$$w := w - \alpha * \frac{\partial L}{\partial w} \qquad (5)$$

Onde:

- *w* é o parâmetro aprendível
- *α* é a taxa de aprendizagem
- **L** é a *Loss Function*

A Figura 19 ilustra a representação de uma CNN durante o processo de treinamento. Ela é composta por várias camadas de convolução, camadas de *pooling* e *FC*. O desempenho do modelo é calculado pela *loss function*, através do *forward propagation* no *dataset* do treinamento e os parâmetros aprendíveis, como pesos e *kernels*, são atualizados através do *backpropagation* com a otimização do gradiente descendente.

Figura 19 - Representação de uma CNN no processo de treinamento



Fonte: (YAMASHITA, NISHIO, *et al.*, 2019)

2.2.9. Regularização

O *dropout* é uma técnica utilizada em CNNs com o propósito de reduzir o *overfitting*, que é quando a rede aprendeu demais e não consegue mais aprender. Este fenômeno acontece na fase de treinamento e durante a propagação dos dados pela rede e é descrito como uma forma de regularização. A utilização é simples, ou seja, o descarte aleatório de neurônios de uma camada. Durante as iterações, esta técnica minimiza a variância e aumenta o viés das funções. Já na fase de testes, não se utiliza *dropout* (PONTI e COSTA, 2017).

Se um modelo tem um bom desempenho no processo de treinamento, contudo no processo de validação é pior, então há uma grande probabilidade de que o modelo esteja sofrendo de *overfitting* durante o treinamento. Uma possível solução para redução deste problema é o aumento do conjunto de dados para treinamento. Um modelo treinado em um conjunto de dados grande consegue generalizar melhor

(YAMASHITA, NISHIO, *et al.*, 2019).

Dropout foi introduzida por Geofrey Hinton em 2012, entretanto o Google detém a patente desta técnica (WIKIPEDIA, 2010). Existem outras técnicas de regularização, como por exemplo a L2, ou *weight decay* e *Batch normalization* (SRIVASTAVA, HINTON, *et al.*, 2014).

Data augmentation é também eficiente para a redução de *overfitting*, que é o processo de modificação dos dados de treinamento. No caso de imagens, elas são aleatoriamente modificadas durante o processo sofrendo uma transformação, como rotação, inversão, cortes, translação, entre outros. Desta forma, a imagem é considerada nova para a rede. Esta técnica também é utilizada para quando não se tem um *dataset* grande o suficiente para treinar uma rede. (YAMASHITA, NISHIO, *et al.*, 2019). No Quadro 2 tem-se uma lista dos métodos mais comuns atualmente para se mitigar o *overfitting*.

Quadro 2 - Lista dos métodos mais comuns para mitigação de *overfitting*

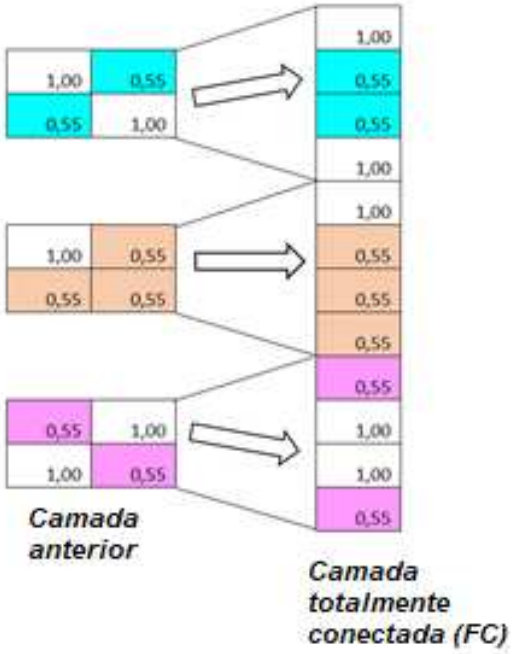
Técnicas para mitigar *overfitting*
Mais dados na fase de treinamento
Data augumentation
Regularização
Batch normalization
Redução da complexidade da arquitetura

Fonte: Adaptado de (YAMASHITA, NISHIO, *et al.*, 2019).

2.2.10. Camada de saída

Na penúltima camada, antes da camada de saída, tem-se a camada FC. Cada neurônio desta camada possui um peso associado a cada um dos elementos do vetor de entrada. Depois de várias camadas de convolução, a FC recebe os dados da camada anterior e faz a transição para a camada de saída. Esta camada de saída é um vetor, enquanto a anterior é um tensor, então obrigatoriamente, este tensor é convertido em um vetor, assim cada entrada é conectada a cada saída. Esta camada possui uma quantidade de neurônios na saída igual ao número de classes que a rede irá classificar (YAMASHITA, NISHIO, *et al.*, 2019). A Figura 20 ilustra essa conversão.

Figura 20 - Conversão de um tensor para um vetor na camada totalmente conectada



Fonte: Adaptado de (BRAZ JUNIOR, 2020)

Suponha-se que este tensor tenha as dimensões 4 x 4 x 40 então, no redimensionamento, os dados seriam convertidos para 1 x (4 x 4 x 40) = 1 x 640 elementos. Nesta camada, utiliza-se a função de ativação *Softmax,* quando a classificação for do tipo multiclasses e as saídas são maiores ou iguais a 3 elementos distintos. Na Figura 20, pode-se ver a camada FC imediatamente anterior à camada de saída.

2.2.11. *Hiperparâmetros e parâmetros*

Hiperparâmetros são os valores definidos pelo usuário que está modelando a rede neural, como por exemplo:

- Número de iterações (épocas)
- *Bias*
- Taxa de treinamento (*learning rate*)
- Quantidade de *kernels* e *strides*
- Quantidade de camadas convolucionais, *pooling*, entre outros.
- Função de custo

Por outro lado, os parâmetros não são escolhidos pelo usuário e sim calculados pela rede. “Eles estão relacionados basicamente aos valores aprendidos em todos os filtros nas camadas convolucionais, os pesos das camadas *FC* e os *bias*.” (PONTI e COSTA, 2017). Quanto maior é o número de parâmetros, mais complexa é a rede e, consequentemente, mais tempo para treinamento será exigido, além de maior poder computacional. O Quadro 3 apresenta uma lista dos parâmetros e hiperparâmetros utilizados em uma CNN.

Quadro 3 - Lista de parâmetros e hiperparâmetros de uma CNN

Tipo	Parâmetros	Hiperparâmetros
Camada de convolução	*Kernels* ou filtros	Tamanho e número de *kernels*, *stride*, *padding* e função de ativação.
Camada de *Pooling*	Nenhum	Método de *pooling*, tamanho do filtro, *stride* e *padding*.
Fully Connected	Pesos	Número de pesos e função de ativação.
Outros	-	Método de arquitetura, otimizadores, taxa de treinamento, função de perda, tamanho do mini-batch, época, regularização, inicialização dos pesos e divisão do dataset.

Fonte: (YAMASHITA, NISHIO, *et al.*, 2019).

Para ilustrar o cálculo destes parâmetros, considere uma arquitetura de rede que irá classificar imagens cuja dimensão seja de 64 x 64 x 3, onde este último valor refere-se ao tipo de imagem, RGB, representada pelas cores vermelha, verde e azul e a classificação será em 5 possíveis classes. Para esta arquitetura, serão três camadas de convolução, duas camadas de *pooling*, e duas camadas FC, assim, o cálculo do total de parâmetros se dá conforme ilustrado no Quadro 4.

Quadro 4 - Cálculo de parâmetros de uma CNN

Camada	Descrição	Saída
Convolução 1	10 filtros com tamanho 5 x 5 (RGB)	Tensor de dimensão 64 x 64 x 10 10 = Nº de filtros 64 x 64 = dimensão da imagem
Pooling	Subamostragem com fator 4 (janela de tamanho 2x2 e *stride* de 2)	Tensor de dimensão 16 x 16 x 10 16 = refere-se ao resultado da divisão de 64 x 4 (fator)
Convolução 2	Filtros de 3 x 3 x 10 20 = Nº de *feature* maps	Tensor de dimensão 16 x 16 x 20
Convolução 3	40 filtros de 1 x 1 x 20	Tensor de dimensão 16 x 16 x 40
Pooling 2	Subamostragem com fator 4 (janela de tamanho 2 x 2 e *stride* de 2)	Tensor de dimensão 4 x 4 x 40
Fully Connected 1	32 neurônios	32 valores
Fully Connected 2	5 neurônios	5 possíveis valores

Fonte: (PONTI e COSTA, 2017).

Considerando que cada um dos filtros das três camadas convolucionais, representado por uma matriz de L x C x D, além do Bias, e que nas camadas FC possuem os pesos (w) e o Bias associado a cada elemento do vetor recebido da camada imediatamente anterior, então o cálculo do número de parâmetros a serem aprendidos pela rede é dado pelo Quadro 5.

Quadro 5 - Cálculo de parâmetros

Dimensão	Nº de parâmetros	Camada
(10 x [5 x 5 x 3 + 1])	760	Convolução 1
+(20 x [3 x 3 x 10 + 1])	1820	Convolução 2
+ (40 x [1 x 1 x 20 + 1])	840	Convolução 3
+ (32 x [640 + 1])	20512	*Fully connected* 1
+ (5 x [32 + 1])	165	*Fully connected* 2
Total de parâmetros	24097	

Fonte: (PONTI e COSTA, 2017).

Definidos os conceitos básicos das componentes de uma CNN, a próxima seção apresenta conceitos específicos. Neste trabalho, foram utilizados alguns classificadores de imagens em *DL* já construídos e prontos para uso, como redes GoogLeNet e AlexNet, e combinados com três *frameworks:* Torch, Caffe e TensorFlow.

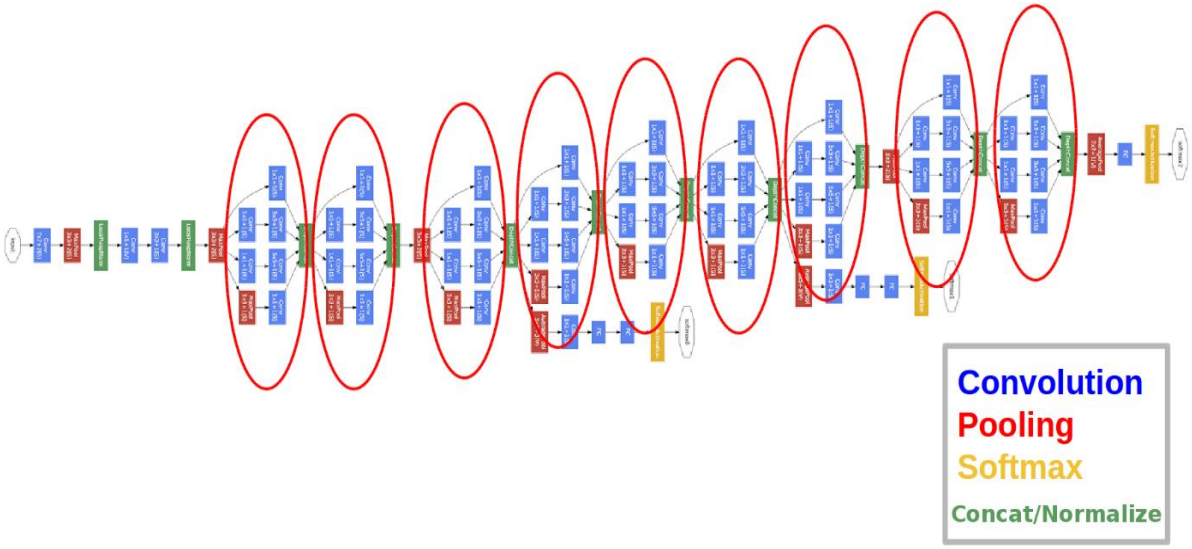
2.3. GoogLeNet

A GoogLeNet, criada pelos pesquisadores do Google, que propuseram uma nova arquitetura de CNN. Esta rede é mais complexa e com mais camadas que outras CNNs, sendo uma rede dentro de uma rede. Ela é chamada *Inception* e concatena filtros de diferentes tamanhos e dimensões dentro de único filtro novo. Possui 2 camadas convolucionais, 2 camadas de *pooling* e 9 módulos *Inceptions, sendo que cada módulo é composto de 6 camadas de convolução e 1 de pooling. Ao todo, somam-se 56 camadas convolucionais e 7 milhões de parâmetros* (YANG, YAN, *et al.*, 2018).

A função de ativação ReLU (*Rectified Linear Unit*) é aplicada a todas as camadas de convolução, incluindo as que estão dentro do módulo *Inception* (SZEGEDY, LIU, *et al.*, 2014) .

A GoogLeNet possui 22 camadas de profundidade e consegue utilizar melhor os recursos computacionais pela rede, ao mesmo tempo que aumenta seu tamanho, profundidade e largura. Apesar de uma estrutura bem robusta, a rede tem 12 vezes menos parâmetros que a AlexNet (definida em 3.4) (YANG, YAN, *et al.*, 2018). A Figura 21 ilustra a arquitetura desta rede.

Figura 21 - Arquitetura da rede GoogLeNet



Fonte: Adaptado de (FRANÇA e SOARES, 2016)

Ela possui uma camada de *Average Pool* de 5 x 5 com *stride* 3, uma camada de convolução de 1 x 1 com 128 filtros para redução da dimensionalidade e camada de ativação *ReLu*. Ainda conta com uma camada de *Dropout* e, como saída, utiliza a função linear *Softmax* para a classificação, que consegue classificar os elementos em até 1.000 classes diferentes (FRANÇA e SOARES, 2016).

No desafio do ILSVRC[11] de 2014, a rede alcançou 5,5% de erro na classificação do top-5, enquanto a AlexNet conseguiu 15,3% (YANG, YAN, *et al.*, 2018).

2.4. AlexNet

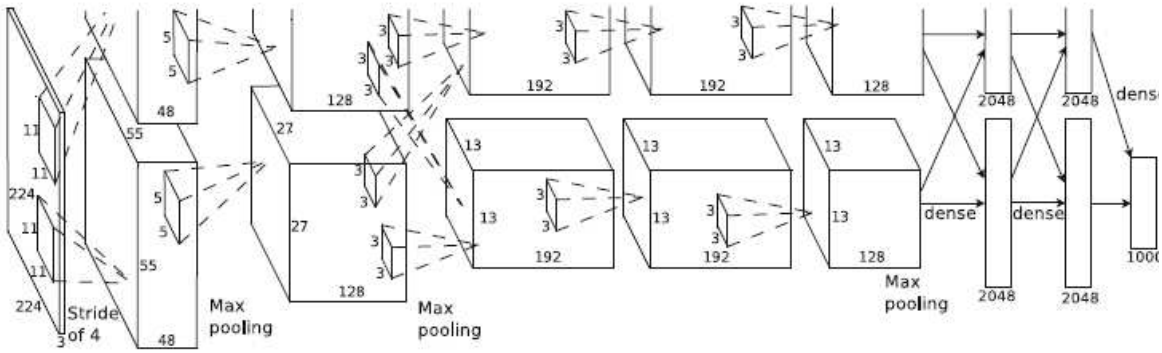
É uma CNN que foi projetada por Alex Krizhevsky em colaboração com Ilya Sustskever e Geofrey E. Hinton e foi treinada com 1.2 milhão de imagens de alta resolução e consegue classificar até 1000 classes diferentes (KRIZHEVSKY, SUTSKEVER e HINTON). Escrita com *Compute Unified Device Architecture* CUDA de forma que fosse possível a utilização da GPU para aumentar significativamente

[11] ImageNet Large Scale Visual Recognition Challenge

seu desempenho (BAHRAMPOUR, RAMAKRISHNAN, *et al*., 2016).

Esta rede conta com cinco camadas convolucionais, sendo três delas utilizadas para agrupamentos que se restringem a conectividade de filtros e duas camadas de normalização de resposta local. Também conta com três camadas de *pool*, ou também conhecidas como *Max-Pooling,* duas camadas FC com função de ativação ReLU e Softmax. A rede possui 60 milhões de parâmetros e 650.000 neurônios (XING, LV, *et al*., 2019). A Figura 22 apresenta a arquitetura da rede AlexNet.

Figura 22 - Arquitetura da rede AlexNet



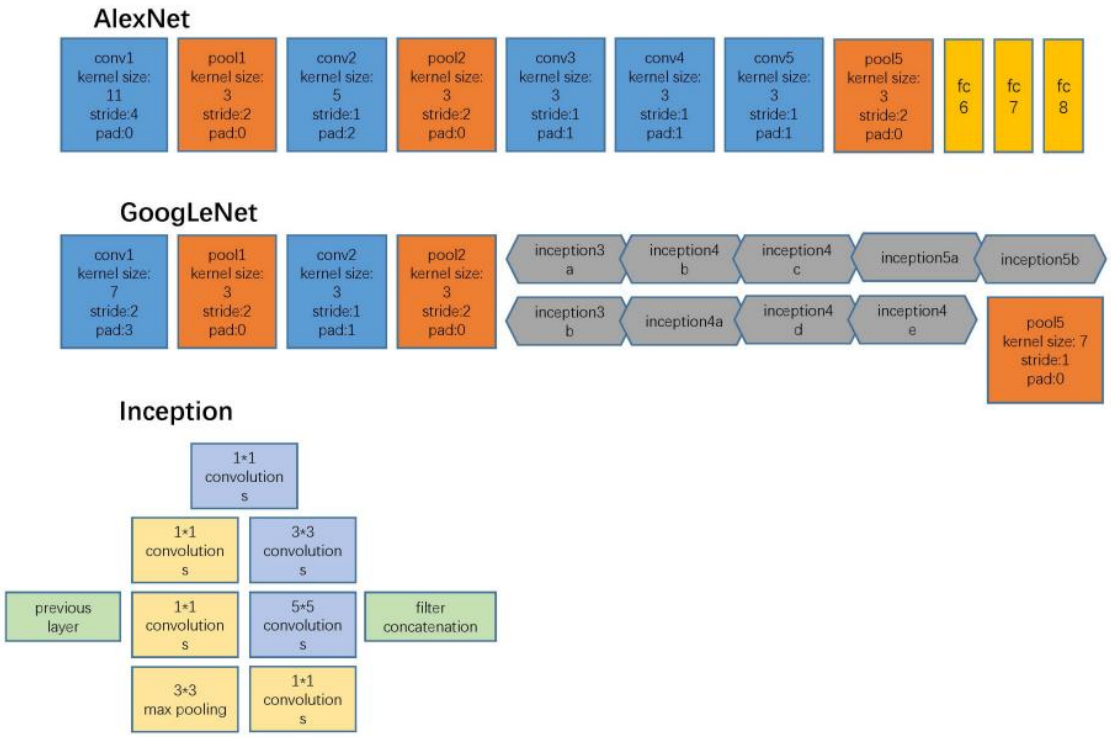
Fonte: (AGGARWAL, 2018)

A AlexNet se inicializa com imagens de 224 x 224 x 3 e utiliza 96 filtros de tamanho 11 x 11 x 3 na primeira camada com um *stride* de 4, resultando em uma imagem de 55 x 55 x 96. Após esta etapa, é executada a *Max-Pooling* na camada seguinte. Na segunda camada de convolução, utilizam-se 256 filtros de tamanho 5 x 5 x 96. O tamanho dos filtros da terceira, quarta e quinta camadas são de 3 x 3 x 256 (com 384 filtros), 3 x 3 x 384 (384 filtros) e 3 x 3 x 384 (256 filtros) respectivamente e todas as camadas de *Max-Pooling* utilizam *stride* de 2. A camada FC possui 4096 neurônios e, na camada de saída, é utilizada a função de ativação *Softmax* para fazer a classificação. (AGGARWAL, 2018).

2.5. Comparativo entre AlexNet e GoogLeNet

AlexNet e GoogLeNet podem ser treinadas desde o início das camadas. Um diagrama comparativo entre as duas arquiteturas está ilustrado na Figura 23 e no Quadro 6 informações gerais de cada uma das redes.

Figura 23 – Diagrama comparativo entre as redes AlexNet e GoogLeNet



Fonte: (YANG, YAN, *et al.*, 2018)

Quadro 6 - Informações gerais das redes AlexNet e GoogLeNet

Informações	AlexNet	GoogLeNet
Proposta no ano	2012	2014
% de erro top 5 no ILSVRC	16,40%	6,70%
Número de camadas FC	3	1
Total de parâmetros	20.176.258	5.975.602
Dropout	Sim	Sim
Data augumentation	Sim	Sim
Inception module	Não	Sim
Normalização	Sim	Sim

Fonte: (DONG, JIANG, *et al.*, 2020)

2.6. Frameworks

Neste trabalho foram utilizados *frameworks*, que são conjuntos de código prontos e que podem ser utilizados no desenvolvimento de software, aplicativos, website, entre outros, fornecendo ao desenvolvedor bibliotecas já prontas para uso, o que reduz o tempo de desenvolvimento destas ferramentas. Foram utilizados neste trabalho 3 frameworks: Caffe, Torch e TensorFlow, combinados com cada rede, AlexNet e GoogLeNet.

2.6.1. Caffe

Convolution Architecture For Feature Extraction CAFFE é um *framework* desenvolvido pela *Berkeley Vision and Learning Center (*BVLC) para *Deep Learning* e escrito em C++ com interface para Python. Suporta diferentes tipos de arquiteturas de DL e pode ser utilizado em classificação e segmentação de imagens e foi desenvolvido para ser executado em CPUs ou GPUs (Caffe, 2020). Suporta ainda várias camadas como, convolução, FC e *poolings* (BAHRAMPOUR, RAMAKRISHNAN, *et al.*, 2016).

As suas principais desvantagens são a parte documental, instalação não trivial e inflexibilidade (dificuldade com a criação de novas camadas).

2.6.2. Torch

BAHRAMPOUR et al. (2016) citaram em seu artigo que o Torch é um *framework* científico utilizado em *DL* cujo código é de domínio público. Ele foi construído utilizando uma linguagem de programação chamado Lua[12] e oferece flexibilidade e agilidade na construção de algoritmos e pode ser executada tanto em CPUs como em GPUs. Tem muitos algoritmos de DL desenvolvidos, orientado a objetos e é compatível com C.

É um *framework* com pacotes e bibliotecas muito eficientes e voltados ao *DL* para criação de aplicações em visão computacional. Como diferencial é a capacidade de ser programada utilizando paralelismo.

2.6.3. TensorFlow

É um *framework* desenvolvido pela equipe do Google Brain[13] e tem o propósito de implementar conceitos de *Machine Learning* (ML) e DL de uma maneira mais fácil. Foi escrita em linguagem C++ e fornece uma API[14] para Python para implementação de modelos de ML e DL facilmente.

Com esta biblioteca de métodos e classes pré-definidas, pode-se treinar e executar CNNs para classificação de dados, reconhecimento de imagens, processamento de linguagem natural, tradução e muito mais. Também tem implementação para utilização de GPUs, que alavanca o potencial computacional. Como principal vantagem, conta com a Google que investe muito no seu desenvolvimento e tem uma excelente documentação (TENSORFLOW, 2020).

2.6.4. Comparativo entre TensorFlow, Torch e Caffe

Os 3 frameworks foram desenvolvidos para utilização em inteligência artificial como ML e DL, porém cada um com alguma particularidade. A Figura 24 ilustra um comparativo entre eles.

---

[12] Linguagem criada em 1993 por uma equipe de desenvolvedores do Tecgran da PUC do Rio de Janeiro

[13] https://research.google/teams/brain/

[14] *Application Programming Interface* – conjunto de funções, métodos, classes, objetos e comandos que programadores podem utilizar no software e que possibilita a interação com sistemas externos (USEMOBILE, 2020).

Figura 24 - Comparativo entre *frameworks*: TensorFlow, Torch e Caffe

Características	TensorFlow	Torch	Caffe
Código + modelos	+ Código e modelos disponíveis incluindo o que o Google cria e disponibiliza, além de códigos abertos desenvolvido pela comunidade	Modelos criados e disponíveis para utilização	Muito códigos disponíveis online para utilização
Performance	Otimizado para grandes modelos e pode ser um consumir de memória, mas tão rápido quanto outros	Necessita do CUDA e como vantagem não necessita de compilação o que faz economizar bastante tempo durante a depuração do código.	Bem rápido e com boa desempenho
Deploy de modelos	TensorFlow como self-hosted web e a plataforma Google Cloud para utilização de serviços	Necessita do LuaJIT para rodar os modelos, o que pode ser um problema quando for fazer uma integração	Baseado em C++ e possui bibliotecas estáveis e pode ser compilado em diferentes dispositivos
Extra features	Escrito em C++ ; Interfaces com Python e Java; Programação distribuída; Multi-GPU; Suporta Windows e Android	Multi-GPU; Estável; Escrito em Lua e C/CUDA	Escrito em C++ com interfaces para Python e Matlab; Cross-plataform; Bem estável (incluindo Windows)

Fonte: Adaptado de (KALOGEITON, LATHUILIÈRE, *et al.*, 2017)

Definidos os conceitos das redes e *frameworks* utilizados nesta pesquisa, o próximo capítulo descreve a trabalhos relacionados.

3. TRABALHOS RELACIONADOS

Este capítulo apresenta trabalhos encontrados na literatura que investigam o desempenho de ANNs. A dificuldade de uma avaliação “genérica” das redes neurais implica que os achados na literatura sejam majoritariamente focados em algum domínio de aplicação específico. As citações foram organizadas cronologicamente.

As redes GoogLeNet e AlexNet foram avaliadas por Ballester et al. (2016) utilizando imagens abstratas como desenhos feitos à mão. Foram várias classes de imagens como aviões, pássaros, cachorros, entre outras e o *dataset* utilizado foi o TU-Berlin *sketch*. Os resultados mostraram que ambas as redes possuem coeficiente Pearson de 0,92 e que estas redes não são eficientes na classificação deste tipo de imagens e sofrem de *overfitting* neste tipo de situação.

SINGLA et al. (2016) utilizaram a GoogLeNet para fazer a classificação de comida/não comida. Eles utilizaram imagens públicas e os resultados mostram uma acurácia de 99,2% para classificação de comida e não comida e 83,6% no reconhecimento de categorias de comidas. Utilizaram 9866 imagens para Treinamento, 3430 para a Validação e 3347 para Avaliação/Testes.

SHIHADEH et al. (2018) apresentaram em seu trabalho classificação para diagnóstico de melanoma em 2 classes possíveis: Benigno e Maligno. Conseguiram uma acurácia de 74,59% com a rede AlexNet e 65,85% com a GoogLeNet. SUN et al. (2017) classificaram imagens de retina utilizando AlexNet, GoogLeNet, VGG-16 e ResNet-50 e conseguiram uma acurácia de 96,53%, 97,04%, 96,87% e 96,20% respectivamente. (DONG, JIANG, *et al.*, 2020) utilizaram as redes LeNet, AlexNet e GoogLeNet para fazer a identificação de células infectadas com malária e alcançaram acurácia de 96,18%, 95,79% e 98,13% respectivamente. BALAGOUROUCHETTY et al. (2020) também publicaram um artigo para diagnóstico de lesão hepática focal utilizando uma CNN chamada de FCNet, que foi uma derivação da GoogLeNet e conseguiram uma acurácia média de 97,37% e utilizaram *transfer learning*.

Na mesma linha, MUHAMMAD et al. (2018) fizeram o reconhecimento de nove tipos de frutas diferentes de um *dataset* público e conseguiram uma acurácia de 100% para as três redes.

SUSTIKA et al. (2018) elaboraram um comparativo de desempenho das CNNs AlexNet, MobileNet, GoogLeNet, VGGNet e Xception para avaliar a qualidade

de morangos a acurácia para a AlexNet foi de 87,37 e 85,26% para GoogLeNet utilizando classes de morangos. Relataram que a VGGNet foi a rede que conseguiu a melhor métrica, com 89,12% de acurácia.

ANGELIS *et al*. (2017) apresentaram uma prova de conceito demonstrando a possibilidade da geração de sequências arbitrárias de imagens automaticamente rotuladas a partir do desenvolvimento do software Fisgen (ver Apêndice A). Utilizaram-se de 4 padrões simples de imagem: deslocamentos lineares e circulares de pontos, preenchimento liso de áreas circulares e coberturas aleatórias de pontos. Um aspecto a destacar no trabalho é a proposição e implementação de um mecanismo de deslocamento temporal (defasamento) entre os rótulos e os conteúdos embebidos nas imagens individuais. Os autores indicaram que este mecanismo poderia ser utilizado na avaliação de processos de predição baseados em classificação, como o aqui ensaiado, mas não fizeram experimentações com redes neurais de nenhum tipo.

FRID-ADAR et al. (2018) propuseram a avaliação de CNN utilizando imagens médicas sintéticas com *data augumentation*. Foi proposto um método para geração destas imagens sinteticamente através da *Generative Adversarial Networks* (GAN). Os resultados com a utilização apenas de *data aumentation* foram de 78,6% *sensitivity* e 88,4% de *specificity*, porém, quando utilizados com *data augumentation* sintéticos, os resultados aumentaram para 85,7% e 92,4% respectivamente. Eles acreditam que esta abordagem pode generalizar melhor para outras aplicações na tarefa de classificação de imagens médicas e possivelmente apoiar radiologistas no processo de diagnóstico médico fornecendo melhores resultados.

A utilização de imagens para execução de alguma tarefa pode ser emprega em diferentes frentes como solução, seja na classificação, segmentação ou reconhecimento de objetos, como em (SATIYAH, RAHIM, *et al.*, 2018) que propuseram um trabalho de classificação de veículos utilizando três tipos diferentes: CNNs simples, AlexNet e GoogLeNet. Como resultados alcançaram 65,09% com a AlexNet, 64,22% com a GoogLeNet e 56,30% com uma CNN simples. A AlexNet foi a que produziu um resultado melhor. Ressaltaram que a baixa qualidade do *dataset* interfere diretamente na acurácia das redes e que muitas das imagens foram tiradas em condições não ideias como, falta de luz, tempo nublado ou muito ensolarado, chuva, entre outras.

AKMAL ABDUL HAMID et al. (2019) apresentaram em seu artigo um comparativo entre Bag of Features (BoF), CNN convencional e AlexNet pré treinada para reconhecimento de frutas e, uma das dificuldades neste tipo de problema é a similaridade e cor entre as frutas. Conseguiram uma acurácia de 98% para a BoF, 99,67% na CNN convencional e 57,08% utilizando LR = 0,001, 95,42% utilizando LR = 0,0005 e 100% com LR = 0.0001.

JEONG-HWAN et al. (2019) avaliaram a rede GoogLeNet classificando batimentos de eletrocardiogramas conseguindo uma acurácia média de 95%. OKSUZ (2019) também fez uma avaliação de CNNs para detecção automática de presença de artefatos relacionados ao movimento de imagens de ressonância magnética cardíaca (RMC) comparando 2 arquiteturas: 3D *spatio-temporal* (3D-CNN) e *Long-Term Recurrent Convolutional Network* (LRCN). Eles utilizaram *data augumentation* com geração de dados sintéticos.

ANGELIS *et al*. (2019) desenvolveram metodologia de avaliação das CNNs com o uso de sequências de imagens e as configurações *default* do DIGITS, utilizando 7 modelos e 3 variantes básicas. Seu enfoque foi estabelecer ambiente experimental funcional com as ferramentas disponíveis e demonstrar a possibilidade de algum nível de recuperação dos rótulos pelas CNNs.

Deve-se notar que aquela investigação foi realizada a partir do mesmo problema motivador, no mesmo grupo de pesquisa, com interação de pessoal e com recursos semelhantes, trazendo, ao mesmo tempo, sinergia e pontos de intersecção entre aquele trabalho e este.

Naquele estudo precursor, verificaram que dados automaticamente embutidos em sequências de imagens poderiam ser recuperados pelas CNNs, o que deu sustentação à proposta desta pesquisa e estabeleceu um diferencial em relação a investigações que empregaram imagens isoladamente obtidas, como é o caso dos concursos de avaliação de imagens

Utilizaram-se de conjuntos atualmente considerados maiores que o desejável, com 10.000 figuras cada, e coletaram os indicadores de acurácia de nível 1 e 5, além de tecerem considerações sobre a matriza de confusão. Suas conclusões em destaque são a alta sensibilidade do método em relação ao conteúdo da imagem, ao tipo de métrica embutida e à CNN utilizada. O estabelecimento de valores mais favoráveis aos experimentos foi sugerido como trabalho futuro.

No entanto, como tinham objetivos diferentes em relação ao presente estudo, não se ocuparam de algumas questões que são tratadas aqui, como a comparação de *frameworks*, a escolha de um otimizador, o registro do nível de balanceamento das classes, a redução do número de imagens, a definição de hiperparâmetros ou da proporção de imagens nas várias fases do processo e a adequação de outros indicadores de avaliação de CNNs.

Experimentalmente, este trabalho se diferencia daquele pela fase de calibração com testes preliminares, seleção de otimizador, taxa de aprendizado, número de épocas mais adequados a este trabalho, seleção do uso de um conjunto distinto, mais refinado e numeroso de modelos de geração (10 modelos e 8 variantes contra 7 modelos e 3), redução de 80% na quantidade de imagens por teste, software atualizado (Fisgen 2), plataforma DL estabilizada e coleta de dados relativos aos *frameworks*.

Em relação à literatura em geral, esta pesquisa se destaca pelo uso de sequência correlacionadas por elementos de conteúdo e evolução temporal e não de conjuntos de imagens isoladas (vide outros trabalhos citados neste capítulo), indicando potenciais de emprego das CNNs em séries temporais de imagens, com vistas a um possível uso no problema motivador.

ZHANG et al. (2020), que utilizaram uma CNN profunda para classificação de imagens em monitoramento de cristalização de açúcar. Em seu trabalho, utilizaram um total de 800 imagens para treinamento e 200 para validação. As redes utilizadas foram *Inception-V3, ResNet50, InceptionResNetV2* e uma simples CNN. Conseguiram acurácias de 87,8%, 84,3%, 90,1% e 67,7% respectivamente.

GOMES et al. (2020) utilizaram uma CNN para detecção de pedestres através de imagens. Fizeram uma comparação entre o modelo clássico com uma CNN. Foram testados três diferentes tipos de redes (Haar+AdaBoost, HOG+SVM e AlexNet), sendo que a que apresentou melhor resultado foi a AlexNet, cuja acurácia foi de 95,98%.

Um dos problemas muito pesquisado e que recebeu toda atenção da medicina entre 2020 e 2021 foi o COVID-19. (PHAM, 2020), através de raios-X de tórax, utilizando redes pré-treinadas, desenvolveu um trabalho para detecção da doença, a fim de resolver o problema dos departamentos médicos que estavam acima da capacidade no atendimento de urgências à população. As redes selecionadas

foram AlexNet, GoogLeNet e SqueezeNet. Ele utilizou várias métricas e concluiu que essas redes requerem menos tempo de treinamento, porém com os parâmetros bem ajustados e definidos, podem alcançar excelentes resultados como uma média de 98% para as três CNNs.

LI et al. (2021) criaram um *dataset* sintético baseado em três aspectos de imagens, textura, forma e escala de medição. As redes selecionadas foram redes populares entre os anos de 2010 e 2020 para fazerem a avaliação de desempenho. As imagens utilizadas têm um tamanho de 224 x 224 baseados nos três aspectos e com três categorias cada um, totalizando assim 27 tipos/grupos de imagens. No *dataset* para Treinamento foram utilizadas 32400 imagens. As imagens foram simples como círculo, retângulo e triângulo com e sem ruído gaussiano, tamanhos diferentes e com rotação. As redes selecionadas foram AlexNet, GoogLeNet, VGGNet, ResNet, SqueezeNet, MobileNet, DenseNet, MnasNet. Uma das métricas utilizadas foi a acurácia: obtiveram 96,03%, 87,37%, 87,12%, 98,71%, 75,88%, 51,18%, 55,74%, 81,55% e 3,70% respectivamente para cada rede.

4. METODOLOGIA

Consoante aos objetivos desta pesquisa, foram adotados procedimentos para selecionar CNNs, ferramentas e configurações, construir conjuntos de dados de imagens sintéticas, determinar uma configuração dos hiperparâmetros das redes e avaliar comparativamente o desempenho das CNNs ensaiadas.

Para viabilizar essa avaliação, foi necessária uma forte delimitação do escopo experimental, restringindo-se o número ferramentas computacionais de Inteligência Artificial (IA), a quantidade de opções e parâmetros das CNNs e seus respectivos *frameworks* e o tipo e a complexidade das imagens. Em função dos recursos disponíveis na FT, foi usado o software *Deep Learning GPU Training Systems* (DIGITS), produzido pela Nvidia Inc., que também forneceu em doação a placa de processamento (GPU). Essas escolhas determinaram aspectos operacionais e os indicadores de avaliação das redes.

Este capítulo está dividido em 2 seções, Recursos, onde estão listados equipamentos, imagens e software utilizados, e Métodos, onde estão indicados os procedimentos adotados ao longo da pesquisa.

4.1. Recursos

Os recursos utilizados foram os seguintes:

- Imagens sintéticas geradas pelo software FISGEN;
- Software DIGITS (*Deep Learning GPU Training Systems*) versão 6 da Nvidia;
- GPU NVIDIA Titan X Pascal;
- Microcomputador com sistema operacional Linux Ubuntu com a GPU e o DIGITS (plataforma DL) instalados;
- Software livre para estabelecer conexão SSH com o servidor, Putty.
- VPN quando o acesso ocorreu a partir de fora das dependências da Unicamp;
- Microcomputador com plataforma Windows e software para utilização de planilhas eletrônicas

A plataforma *DL* (DIGITS + GPU) está instalada e configurada nas dependências da FT/Unicamp com administração do setor de Tecnologia da Informação. Os acessos à plataforma foram através do protocolo SSH[15] e utilização do navegador Mozilla FireFox[16] (Detalhes no Apêndice B).

A próxima seção descreve o método utilizados nesta pesquisa

4.2. Métodos

A primeira etapa deste trabalho foi a revisão de literatura, feita a partir das bases de dados disponibilizadas pela CAPES. O foco das buscas iniciais foi a temática da energia elétrica e da previsão de vazões de rios, determinando que o problema motivador era atual e relevante. Posteriormente, foram adicionadas as buscas relativas ao desempenho de redes neurais artificiais, cujos achados estão citados no Capítulo 3.

A seleção de recursos foi feita dentro das condições da FT, determinando então o uso do software DIGITS, CNNs GoogLeNet e AlexNet, também utilizados por outros pesquisadores da instituição. A instalação básica do software no servidor de processamento foi feita e administrada pela equipe de Tecnologia de Informação da faculdade, que entregou para o trabalho as configurações *default* do sistema operacional e do software da Nvidia.

A construção do conjunto de imagens para experimentação foi feita com o uso do software Fisgen, (vide Apêndice A) também empregado em outros trabalhos da faculdade. Em linhas gerais, foram utilizados Modelos de Animação Gráfica (MAG) para criar conjuntos de imagens sintéticas com métricas embutidas, rotuladas automaticamente de acordo com essas mesmas métricas, para posterior submissão às CNNs.

O uso de imagens de satélite diretamente submetidas a ferramentas seria demasiadamente complexo, conforme já comentado, e não atenderia aos objetivos do trabalho. Portanto, foram utilizadas imagens sintéticas, com métricas relativamente simples e reconhecíveis por seres humanos, na mesma linha de raciocínio de LI et al. (2021)

---

[15] *Secure Socket Shell* – Conexão remota com a plataforma DL.
[16] Navegador de Internet livre, multi-plataforma desenvolvido pela Mozilla Foundation.

ANGELIS et al. (2017) demonstraram que o uso deste tipo de imagem é uma excelente opção para se treinar uma CNN e YAMASHITA et al. (2019) citaram em seu artigo que a obtenção de um *dataset* de alta qualidade com as classes identificadas pode ser custoso e demandaria muito tempo para rotulá-los, cuja tarefa é manual. Alternativa preferencial é o uso de imagens sintéticas, que são geradas por computador, em quantidades, formatos e tipos desejados, já rotuladas, tornando os dados confiáveis

O padrão estático embutido em cada imagem foi escolhido dentre linhas, ângulos, áreas preenchidas, áreas cobertas por pontos e nuvens de pontos. Além do padrão estático, uma sequência usando o mesmo MAG também desloca os elementos das sucessivas imagens, incluindo, portanto, um componente dinâmico de movimento nas figuras.

Uma sequência de imagens que usa o mesmo MAG é chamada, de agora em diante, *dataset*, de acordo com a nomenclatura dos programas de computador adotados. Cada MAG embute métricas relativas aos componentes estáticos e dinâmicos nas imagens e os referencia em seus rótulos, permitindo agrupamento de classes. Os *datasets* foram submetidos às CNNs e a avaliação de desempenho mediu o quanto elas foram capazes de recuperar as métricas embutidas nas imagens. O conjunto de todos os *datasets* gerados constituiu-se na base de dados de pesquisa.

MAGs são códigos Java que podem ser desenvolvidas livremente conforme o interesse do usuário, A ideia é que possam ser projetados MAGs progressivamente mais realistas, até que, num dado momento, eles sejam indistinguíveis das imagens de satélites reais. No entanto, este trabalho não tem como objetivo tal nível de verossimilhança, deixando esta questão para futuros desenvolvimentos.

O uso de um software fechado para execução das CNNs traz como vantagem a menor complexidade de operação, maior disponibilidade de suporte técnico e facilidade de aprendizado. Mesmo assim, o número de variáveis a configurar e as diversas faixas de parametrização tornam proibitiva a exploração de todas as possibilidades, pois trata-se de um problema de explosão combinatória. Há dois tipos de variáveis a definir: as configurações dos experimentos e os hiperparâmetros. As primeiras são relativas às propriedades dos *datasets* em si, como, por exemplo, tamanho do conjunto, resolução das imagens, modelo de geração etc. As segundas

dizem respeito aos ajustes da própria ferramenta de redes neurais como, por exemplo, número de épocas de treinamento, taxa de aprendizagem, proporção de imagens em cada fase etc.

A determinação das configurações dos *datasets* foi feita inicialmente com base na experiência dos componentes do grupo de pesquisa GPMaC. Foram ensaiados parâmetros que pudessem apresentar variabilidade quanto às métricas embutidas nas imagens. Os hiperparâmetros foram configurados após consultas a especialistas, pesquisa em literatura técnica e ajustes relativos aos valores *default* do DIGITS. No entanto, ainda assim foram necessários testes preliminares para refinamento e calibração dessas combinações de parâmetros.

Nesses testes empíricos, foram utilizadas 10.000 imagens sintéticas por *dataset* e diferentes percentuais de imagens para Treinamento, e Testes, 50x50 e 75x25 respectivamente. Com isso, foi possível ter familiaridade com o ambiente, a operação do software, a GPU, entre outros, além de determinar faixas aceitáveis dos hiperparâmetros.

O tempo de processamento de cada conjunto de aproximadamente 2.000 imagens flutuou entre 2 e 4 horas, em regime de submissão de lotes. Esses tempos, relativamente curtos, não foram considerados significativos para a avaliação pretendida das redes e, portanto, não foram anotados individualmente para cada *dataset*.

Há uma série de indicadores para avaliação do desempenho de redes neurais, como por exemplo, acurácia, precisão, *recall*, F1, composição da matriz de confusão, curvas ROC (*Receiver Operating Characteristic*) e AUC (*Area Under the Curve*). De acordo com o interesse e a metodologia deste trabalho, foi definido que seriam utilizados os indicadores de acurácia de nível 1 e 5 para a avaliação das redes quanto a sua capacidade de extração das métricas embutidas nas imagens, via processo de classificação.

Esta decisão tem suporte na literatura, pois grande parte dos trabalhos utiliza acurácia como indicador de desempenho das redes neurais, como pode ser visto no Capítulo Trabalhos Relacionados. Adicionalmente, esse é um indicador automaticamente fornecido pela ferramenta DIGITS, o que facilitou a coleta e a tabulação de dados dos resultados. Esse trabalho não se utilizou das técnicas de validação cruzada, também conhecidas como *cross-validation*.

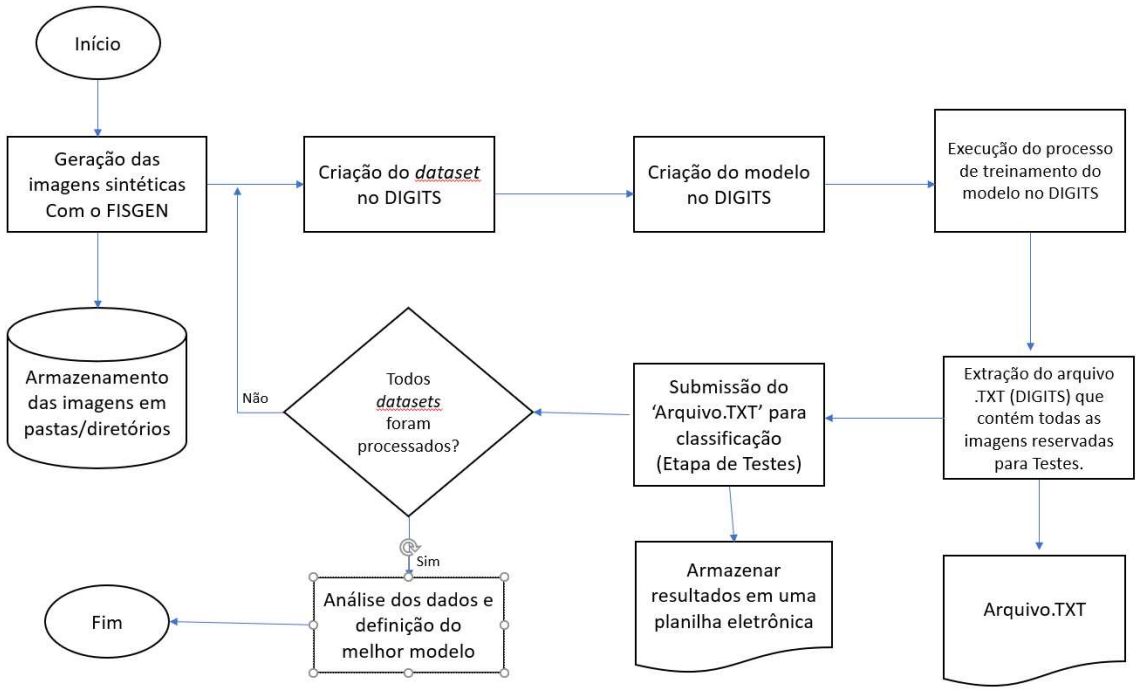
A partir daqui, esta seção apresenta os procedimentos utilizados nesta pesquisa. A sua execução necessita acesso à plataforma DIGITS (Apêndice B).

Detalhes de algumas etapas estão nos seguintes apêndices.

- Operação do DIGITS: Apêndice C;
- Escolha do otimizador e taxa de aprendizado: Apêndice D;
- Cálculo da acurácia: Apêndice E;
- Geração de Imagens Sintéticas: Apêndice F:

A Figura 25 apresenta o fluxograma do processo de geração, classificação das imagens, armazenamento dos resultados e análise. As etapas dos procedimentos ilustradas no fluxograma estão descritas nas subseções seguintes.

Figura 25 - Fluxograma das etapas para geração, classificação das imagens e análise dos resultados



Fonte: Autoria própria.

4.2.1. *Geração das imagens sintéticas com o FISGEN*

Assim, os conjuntos de imagens puderam ser criados sob estrito controle, reduzindo o número de variáveis influentes. No mesmo sentido, foram usados padrões simples e humanamente reconhecíveis, como distâncias entre pontos, ângulos formados por linhas retas, áreas preenchidas ou densidade de pontos contrastantes com o fundo, permitindo verificações rápidas da consistência dos conjuntos de imagens, metadados e avaliação do DL. O Quadro 7 ilustra cada modelo de imagens com suas métricas e o tipo de imagens que cada *dataset* foi criado.

Quadro 7 - Modelos de imagens e métricas de cada *Datasets*

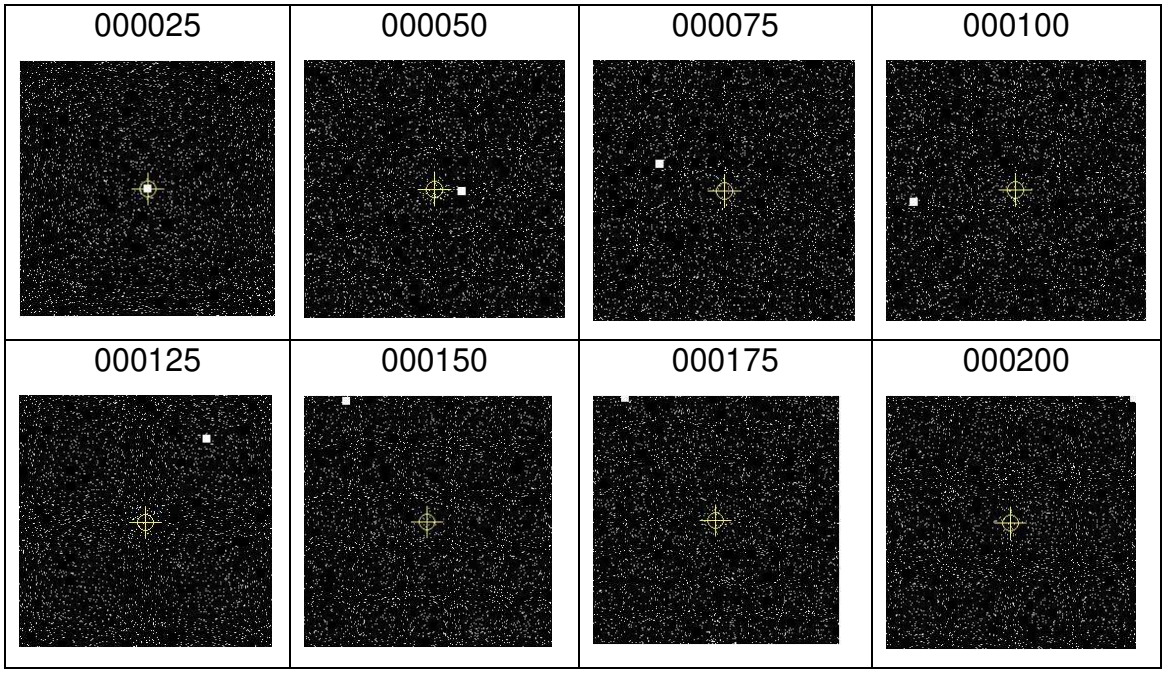
Modelo (classe Java)	**Cores**	**Representação ao longo da sequência**	**Métrica**	***Datasets***
Flooding	2 cores	Área circular crescente	Área coberta	T007, T017
GaussianCloudAndRiver	3 cores	Nuvem de pontos caminhando sobre terreno e rio	Espessura de linha	T021
GaussianCloudGrayScale	256 tons	Nuvem de pontos caminhando sobre terreno	Área coberta	T022
GaussianCloudMonochrome	2 cores	Nuvem de pontos caminhando sobre terreno	Área coberta	T008, T018, T023
GaussianCloudTripleColor	4 cores	Três nuvens de pontos caminhando sobre terreno	Área coberta	T010, T020, T024
MovingPointLinear	2 cores	Retângulo movendo-se linearmente	Distância ao centro	T001, T002, T011, T012
MovingPointRandom	2 cores	Retângulo movendo-se aleatoriamente	Distância ao centro	T005, T006, T015, T016
MovingPointRotating	2 cores	Retângulo movendo-se ciclicamente em torno do centro	Ângulo entre o retângulo e o vetor (1,0) do plano	T003, T004, T013, T014
SplashingPoints	2 cores	Nuvem de pontos aleatórios aumentando densidade	Área coberta	T009, T019
SplashingPointsGray	256 tons	Nuvem de pontos aleatórios aumentando densidade	Área coberta	T025

Fonte: Autoria própria

Foi utilizado um total de 25 *datasets* com 2.000 imagens cada um, com dimensão de 256 x 256 pixels, padrão .JPEG com 3 canais de 8 bits de cores. No Apêndice E está o procedimento para a geração das imagens sintéticas.

Cada *dataset* foi dividido entre vários grupos contendo conjuntos de imagens similares, segundo o indicador considerado. Para melhor ilustração, o *dataset* T015 foi utilizado como exemplo. Os grupos criados para este foram “000025”, “000050”, “000125”, “000150”, “000175” e “000200” e as imagens de cada grupo com similaridade entre si, de acordo com a distância Euclidiana entre o ponto, representado por um retângulo branco, e o centro. A Figura 26 ilustra cada um destes grupos com uma imagem.

Figura 26 - Amostra de imagem para cada grupo rotulado do dataset T015



Fonte: Autoria própria.

No grupo “00025”, a distância Euclidiana é praticamente zero, pois o retângulo está sobre o círculo. No “00050”, nota-se que o retângulo está se distanciando do centro, e assim acontece com os outros grupos. No “000175”, o ponto está bem distante do centro, e no “000200”, já não é possível visualizá-lo mais. Nota-se também que este ponto (retângulo), fica girando ao redor do centro em cada imagem.

O DIGITS processou cada imagem, levando em conta apenas a sua distância Euclidiana para classificá-la. Por exemplo, todas as imagens do grupo 000125, possuem uma pequena distância entre o centro e o retângulo. Já no grupo 000175, esta distância é bem maior. O DIGITS conseguiu diferenciar as imagens do

grupo 000125 e 000175.

4.2.2. *Criação do dataset no DIGITS*

Cada *datasets* foi criado individualmente através da interface gráfica do DIGITS, acessada remotamente como foi descrito no Apêndice C. Conforme item 4.2.1, foram ao todo 25 *datasets* utilizados nesta pesquisa. As imagens foram divididas, para Treinamento, Validação e Testes, conforme descrito na Tabela 1.

Tabela 1 - Distribuição das imagens para Treinamento, Validação e Testes

Fase	Porcentagem utilizada	Total de imagens
Treinamento	70%	1.400
Validação	20%	400
Testes	10%	200
Total	100%	2.000

Fonte: autoria própria.

Estes valores são escolhidos no momento de criação dos modelos, conforme descrito no Apêndice C.

Durante a etapa do Treinamento, ocorre a fase de Validação, em que o classificador já utiliza partes das imagens para validar o desempenho da rede. Esta é uma etapa muito importante, pois são apresentadas novas imagens ao classificador, e a rede já ajusta os pesos e filtros de forma que atinja o melhor desempenho.

A fase final é a de Testes. Nesta etapa, são apresentadas novas imagens ao classificador. Através da ferramenta DIGITS, calculou-se a acurácia de cada modelo criado, onde modelo é a combinação das redes e dos *frameworks*, conforme descrito no Quadro 4 mais adiante. Os dados foram armazenados em uma planilha eletrônica e estão descritos na sessão Resultados.

4.2.3. *Criação do modelo no DIGITS*

As redes selecionadas foram a AlexNet e a GoogLeNet. Os *frameworks* utilizados foram o Caffe, Torch e o TensorFlow.

Nesta etapa são criados os modelos de *Deep Learning*, escolhendo o tipo de rede, *dataset* desejado, os hiperparâmetros e a inicialização do processo de Treinamento do modelo.

Foram utilizadas duas redes em combinação com três diferentes *frameworks*, conforme ilustra o Quadro 8 e os hiperparâmetros definidos estão descritos no Quadro 9.

Quadro 8 - Redes e *frameworks* utilizados

Rede	*Frameworks*		
AlexNet	Caffe	Torch	TensorFlow
GoogLeNet	Caffe	Torch	TensorFlow

Fonte: autoria própria.

Quadro 9 - Hiperparâmetros escolhidos nesta pesquisa

Hiperparâmetro	Valor escolhido
Learning rate (taxa de aprendizagem)	0,001
Otimizador	SGD (Descida do Gradiente Estocástica)
Número de épocas	200

Fonte: autoria própria.

Após a definição destas propriedades dos modelos, a próxima etapa é criar o modelo.

4.2.4. *Execução do processo de treinamento do modelo no DIGITS*

O processo de Treinamento é iniciado automaticamente pelo DIGITS ao final da criação dos modelos e segue em regime de fila até o término dos trabalhos submetidos. O tempo de processamento é diretamente dependente do número de épocas, quantidade e tipo das imagens e da rede utilizada.

4.2.5. *Extração do arquivo .TXT (DIGITS) com imagens reservadas para Testes*

Nesta etapa, é necessário gravar o nome e a localização das imagens destinadas à fase de Testes em um arquivo .TXT. O nome é indiferente, podendo ser definido pelo usuário. Estas imagens foram separadas automaticamente pela ferramenta DIGITS no momento de criação do *dataset*, conforme descrito no Apêndice C e o total de imagens reservadas para Testes está na Tabela 1.

4.2.6. *Submissão do "Arquivo.TXT" para classificação (Testes)*

Esta etapa é automaticamente realizada pelo DIGITS a partir da ação mostrada em 4.2.4. Este processo é mais rápido do que a etapa de Treinamento, pois o número de imagens é menor, conforme descrito na Tabela 1, e os pesos e filtros já foram definidos na etapa de Treinamento.

Após a finalização da etapa 4.2.6, o DIGITS apresentará o resultado do modelo conforme descrito no Apêndice C. Pode-se armazenar os resultados em uma planilha eletrônica.

4.2.7. *Análise dos dados e definição do melhor modelo*

Com todos os modelos já executados e os resultados armazenados em uma planilha eletrônica, procede-se à avaliação do desempenho da DL. A partir da planilha eletrônica, com as acurácias Top 1 e Top 5 armazenadas, a avaliação de cada rede é dada pela média aritmética entre todos os *datasets* de cada modelo.

Top 1 – É a acurácia obtida na classificação, ou seja, quando uma imagem apresentada é classificada corretamente no grupo.

Top 5 – É a acurácia do modelo considerando as 5 respostas mais prováveis. Suponha-se que foi submetida a imagem de um cavalo à classificação dentre uma série de animais e o modelo apresentou uma matriz de possíveis respostas válidas: “Cachorro = 0.3; Gato = 0.3; Leão = 0.2; Elefante = 0.96; Cavalo = 0.03; Girafa = 0.01”, logo, a resposta estaria correta, pois o cavalo está dentre as 5 respostas mais prováveis.

$$Acurácia = \frac{Número\ de\ acertos}{Total\ de\ imagens} * 100 \quad (6)$$

A acurácia informada pela ferramenta DIGITS é dada pela equação (6) e o seu cálculo está descrito em detalhes no Apêndice E.

A próxima seção é dedicada à apresentação e discussão dos resultados obtidos no trabalho.

5. RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados da pesquisa feita seguindo a metodologia proposta.

Os modelos criados (rede+*framework*) passaram pelo processo de Treinamento e Validação simultaneamente, e o tempo de execução foi em torno de 2 a 4h em média cada um. Já para a fase de Testes o tempo médio foi em torno de 2 a 5 minutos.

A ferramenta utilizada foi o DIGITS, Nvidia e ela proporciona três redes diferentes, AlexNet, GoogLeNet e a LeNet, contudo foram utilizadas apenas as duas primeiras devido ao tamanho das imagens utilizadas. Em conjunto com as redes, a ferramenta proporciona três *frameworks*, Caffe, TensorFlow e Torch, que foram utilizados com cada uma das redes.

5.1. AlexNet

Utilizando a rede AlexNet, a Tabela 2 apresenta os resultados. À esquerda estão cada um dos *datasets*, e nas colunas ao lado estão cada *framework*. Nas colunas *Top* 1 e *Top* 5 representam a acurácia do modelo, sendo o primeiro indicador o mais relevante para este trabalho.

Calculando-se a média aritmética de todos os resultados, coluna *Top* 1, nota-se que o *TensorFlow* foi o que apresentou o melhor resultado, com 88,84% de acurácia e desvio padrão de 13,69. O Caffe e o Torch apresentaram uma 61,61% e 79,37% e 17,54 e 19,90 de desvio padrão respectivamente. No Apêndice E, encontra-se como é feito o cálculo geral da acurácia do modelo.

Tabela 2 - Resultados da classificação das imagens utilizando a rede AlexNet com Caffe, Torch e TensorFlow

Datasets	Top 1			Top 5		
	Caffe	Torch	TensorFlow	Caffe	Torch	TensorFlow
T001	50,00	40,50	99,00	100,00	79,00	100,00
T002	53,50	83,50	100,00	100,00	100,00	100,00
T003	64,50	64,50	62,50	100,00	100,00	100,00
T004	64,50	64,50	62,50	100,00	100,00	100,00
T005	56,50	82,59	82,00	100,00	100,00	100,00
T006	19,40	73,63	93,03	100,00	99,50	100,00
T007	55,17	90,64	100,00	100,00	100,00	100,00
T008	96,57	96,08	97,06	98,53	99,02	100,00
T009	51,78	12,69	97,97	96,95	63,45	100,00
T010	77,23	93,07	93,07	100,00	100,00	100,00
T011	59,00	86,00	100,00	100,00	100,00	100,00
T012	60,00	92,50	99,00	100,00	100,00	100,00
T013	64,50	64,50	58,00	100,00	100,00	100,00
T014	64,50	64,50	59,50	100,00	100,00	100,00
T015	47,24	77,39	83,42	100,00	100,00	100,00
T016	41,41	68,18	88,38	100,00	99,49	99,49
T017	56,65	90,64	100,00	97,54	100,00	100,00
T018	96,02	99,00	96,52	100,00	100,00	100,00
T019	52,79	93,91	97,46	99,49	100,00	100,00
T020	73,13	93,53	91,54	100,00	100,00	100,00
T021	99,50	100,00	100,00	100,00	100,00	100,00
T022	63,18	85,07	90,55	100,00	100,00	100,00
T023	62,81	88,94	84,92	100,00	100,00	100,00
T024	64,50	89,00	88,50	100,00	100,00	100,00
T025	45,96	89,39	95,96	97,47	100,00	100,00
Média	61,61	79,37	**88,84**	99,60	97,62	**99,98**
Desvio Padrão	17,54	19,90	13,69	0,92	8,26	0,10
Mediana	60,00	86,00	93,07	100,00	100,00	100,00
Mínimo	19,40	12,69	58,00	96,95	63,45	99,49
Máximo	99,50	100,00	100,00	100,00	100,00	100,00

Fonte: Autoria própria.

A Figura 27 apresenta um gráfico com o desempenho de cada *framework*, utilizando a rede AlexNet. A linha sólida (laranja) mostra o desempenho do TensorFlow para cada *dataset*. Nota-se que, além de apresentar a melhor média aritmética da acurácia do Top1 para os 25 *modelos*, este *framework* obteve melhores resultados na classificação de imagens na maioria dos *datasets, e se manteve* mais estável em comparação ao os outros dois *frameworks*.

Nos *datasets* T006 e T009, a acurácia do Torch e Caffe são muito baixas, enquanto a do TensorFlow está acima dos 90%, o que pode indicar ser mais consistente para este modelo.

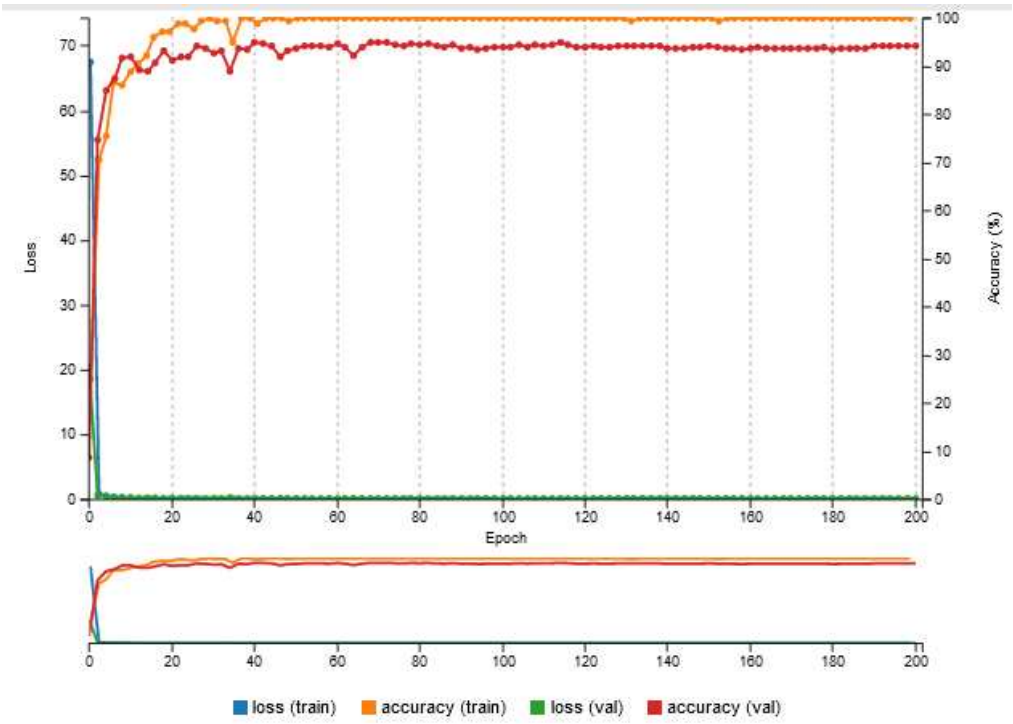
Figura 27 - Resultados da classificação das imagens com a rede AlexNet combinados com os frameworks: Caffe, TensorFlow e Torch

	T001	T002	T003	T004	T005	T006	T007	T008	T009	T010	T011	T012	T013	T014	T015	T016	T017	T018	T019	T020	T021	T022	T023	T024	T025
Caffe	50	53,5	64,5	64,5	56,5	19,4	55,17	96,57	51,78	77,23	59	60	64,5	64,5	47,24	41,41	56,65	96,02	52,79	73,13	99,5	63,18	62,81	64,5	45,9
TensorFlow	99	100	62,5	62,5	82	93,03	100	97,06	97,97	93,07	100	99	58	59,5	83,42	88,38	100	96,52	97,46	91,54	100	90,55	84,92	88,5	95,9
Torch	40,5	83,5	64,5	64,5	82,59	73,63	90,64	96,08	12,69	93,07	86	92,5	64,5	64,5	77,39	68,18	90,64	99	93,91	93,53	100	85,07	88,94	89	89,3

Fonte: Autoria própria.

A Figura 28 apresenta um gráfico representando o processo de treinamento do *dataset* T022 utilizando a rede AlexNet e o *framework* TensorFlow. Este gráfico é apenas para ilustrar como que o DIGITS vai aprendendo ao longo das épocas. É possível ver a acurácia nas fases ao longo das épocas. Nota-se que nas primeiras 20 épocas, a acurácia está abaixo dos 10%. À medida que se avança nas épocas, pode-se notar um aumento no desempenho. Isso significa que a rede está aprendendo com os dados, ajustando os pesos de forma a minimizar os erros. Nota-se que, partir da época 40, o DIGITS conseguiu atingir uma alta acurácia, tanto na fase de Treinamento, quanto na Validação. Ao mesmo tempo que o DIGITS faz o Treinamento, ele já faz o processo de Validação.

Figura 28 - Ilustração do processo de treinamento do *dataset* T023 utilizando a rede AlexNet e o framework TensorFlow



Fonte: Autoria própria.

5.2. GoogLeNet

Análise similar foi feita com a rede GoogLeNet e os resultados encontram-se na Tabela 3. Utilizando a mesma metodologia de cálculo, média aritmética, nota-se que o Caffe foi o que apresentou o melhor resultado, com 55,49% de acurácia.

Tabela 3 - Resultados da classificação das imagens utilizando a rede GoogLeNet com os *frameworks* Caffe e Torch

Datasets	Top 1		Top 5	
	Caffe	Torch	Caffe	Torch
T001	40,50	13,50	84,00	81,50
T002	49,00	14,00	100,00	69,50
T003	64,50	64,50	100,00	100,00
T004	64,50	64,50	100,00	100,00
T005	25,50	61,00	97,00	99,50
T006	18,91	27,36	99,50	91,54
T007	56,16	66,50	100,00	100,00
T008	85,29	96,08	92,16	98,53
T009	59,90	25,89	100,00	100,00
T010	81,19	25,74	100,00	96,04
T011	50,50	14,00	100,00	69,50
T012	33,00	14,00	100,00	69,50
T013	64,50	64,50	100,00	100,00
T014	64,50	64,50	100,00	100,00
T015	50,25	27,64	100,00	90,45
T016	34,85	25,25	100,00	89,90
T017	57,64	48,28	100,00	100,00
T018	96,02	96,52	100,00	98,51
T019	57,36	25,89	100,00	100,00
T020	66,17	27,86	100,00	89,05
T021	100,00	74,37	100,00	100,00
T022	25,87	49,75	86,57	73,63
T023	32,66	49,75	85,93	87,94
T024	53,50	58,50	94,50	95,00
T025	55,05	23,23	97,98	97,98
Média	**55,49**	44,92	**97,51**	91,92
Desvio Padrão	20,95	25,29	4,93	10,76
Mediana	56,16	48,28	100,00	97,98
Mínimo	18,91	13,50	84,00	69,50
Máximo	100,00	96,52	100,00	100,00

Fonte: Autoria própria.

A rede GoogLeNet apresentou resultados inferiores quando comparados à AlexNet. Os *frameworks* Caffe e Torch apresentaram uma acurácia média de 55,49% e 44,92% e desvio padrão de 20,95 e 25,29 respectivamente. O TensorFlow não funcionou com a GoogLeNet possivelmente por falta de memória.

5.3. Comparativo das redes

Comparando os resultados da AlexNet com a GoogLeNet nota-se que AlexNet obteve melhores resultados. A Quadro 10 ilustra os resultados comparativos.

Quadro 10 - Análise da acurácia média e desvio padrão por redes e *frameworks*

Rede	Acurácia média			Desvio Padrão		
	Caffe	Torch	TensorFlow	Caffe	Torch	TensorFlow
GoogLeNet	55,49	44,92	-	20,95	25,29	-
AlexNet	61,61	79,37	**88,84**	17,54	19,90	**13,69**

Fonte: Autoria própria

Nota-se que a AlexNet obteve melhor resultados em todos os *frameworks* e seu melhor resultado foi o TensorFlow com 88,84% e um desvio padrão de 13,69. Separando os resultados por tipo de imagem, o Quadro 11 ilustra as acurácias médias por rede e *framework*.

Quadro 11 - Acurácia média por tipo de imagem das redes e *frameworks*

Tipo de imagem	# classes	Acurácia média				
		GoogLeNet		AlexNet		
		Caffe	Torch	Caffe	Torch	TensorFlow
MovingPointLinear	25	43,25	13,88	55,63	75,63	99,50
MovingPointRotating	361	64,50	64,50	64,50	64,50	60,63
MovingPointRandom	25	32,38	35,31	41,14	75,45	86,71
Flooding	50	56,90	57,39	55,91	90,64	100,00
GaussianCloudMonochrome	5	71,32	80,78	85,13	94,67	92,83
SplashingPoints	61	58,63	25,89	52,29	53,30	97,72
GaussianCloudTripleColor	7	66,95	37,37	71,62	91,87	91,04

Fonte: Autoria própria

O grupo de imagem *MovingPointRotating (T003, T004, T013 e T014)* é o que possui um maior número de classes, 361 e, curiosamente, foi onde o TensorFlow obteve menor acurácia com a AlexNet, 60,63%. Excluindo-se estes *datasets*, a acurácia média de cada um seria acima dos 80% e a média seria de 94,21%.

5.4. Contextualização dos Resultados

O melhor resultado alcançado foi uma acurácia média de 88,84% com AlexNet e TensorFlow. Os resultados desta pesquisa, para os testes ensaiados, se comparam aos da literatura em outros domínios. O Quadro 12 ilustra o resultado dos trabalhos já publicados utilizando as redes AlexNet e GoogLeNet e os detalhes estão descritos no Capítulo 3. Trabalho Relacionados.

Quadro 12 - Resultados de trabalhos já publicados com as redes AlexNet e GoogLeNet

Autores (et al.)	Tipo trabalho	AlexNet	GoogLeNet	Ano
Singla	Classificação de imagens - comida/não comida	-	99,20%	2016
Sun	Classificação de imagens da retina	96,53%	97,04%	2017
Shihadeh	Diagnóstico de melanoma (Benigno e Maligno)	74,59%	65,85%	2018
Muhammad	Reconhecimento de nove tipo de frutas	100%	100%	2018
Sustika	Comparativo de desempenho avaliar qualidade morangos	87,37%	85,26%	2018
Satiyah	Classificação de veículos	64,22%	56,30%	2018
Jeong-Hwan	Classificação de batimentos de eletrocardiogramas	-	95%	2019
Akmal et al.	Comparativo BoF e AlexNet - reconhecimento de frutas	-	99,67%	2019
Dong	Identificação de células infectadas com malária	95,79%	98,13%	2020
Balagourouchetty	Lesão hepática focal	-	97,37%	2020
Pham	Covid-19 através de raios-X do torax	98%	98%	2020

Fonte: Autoria própria

Nota-se diferentes resultados utilizando estas duas redes e em diferentes domínios de aplicação.

Alguns trabalhos publicados que estão em domínios de aplicação próximo a este foi de LI et al. (2021) que utilizaram imagens simples como triângulos, retângulos e círculos, com diferentes níveis de ruídos e rotação. Utilizaram-se das redes mais populares entre os anos de 2010 e 2020 e dentre os resultados alcançados foram de 96,03% com a AlexNet e 87,37% com a GoogLeNet e ANGELIS (ANGELIS e ROCHA, 2019) que utilizaram classificação de imagens sintéticas utilizando as redes AlexNet e GoogleNet e conseguiram uma acurácia média de 53,6% e 50,4% respectivamente, ambos já referenciados no Capítulo 3. Trabalhos Relacionados.

Este trabalho utilizou-se de menos imagens quando comparados ao de ANGELIS et al. (2019), contudo conseguiu uma acurácia superior como pode ser visto no Quadro 10.

6. CONCLUSÃO

Este trabalho avaliou o desempenho de CNNs na classificação de imagens, utilizando a plataforma DIGITS da NVIDIA. As imagens utilizadas foram geradas sinteticamente com diferentes formatos e modelos e já rotuladas.

Para determinação dos hiperparâmetros vários ensaios foram feitos com as imagens selecionando diferentes taxas de aprendizado, otimizador, número de épocas e porcentagem de imagens para Treinamento, Validação e Testes. Com isso, foi possível determinar a melhor configuração.

A metodologia apresentou-se adequada para a avaliação do desempenho da ferramenta, deixando claro que este depende de uma série de fatores, como a parametrização das redes e o tipo das imagens submetidas ao classificador. Portanto, o uso da *DL* parece implicar escolhas específicas de redes neurais e *frameworks*, além de ajustes personalizados nos hiperparâmetros, para cada situação em que se pretenda o seu emprego.

Os resultados demonstraram que a ferramenta consegue classificações com uma taxa de acerto similar às encontradas na literatura, demonstrando bom potencial no reconhecimento de conjuntos sintéticos de imagens. A AlexNet apresentou melhores resultados quando comparados à GoogLeNet.

Outros domínios de aplicação podem se beneficiar dos resultados aqui obtidos e da metodologia empregada, inclusive em termos de comparação de desempenho (*benchmark*).

Sugestões de trabalhos futuros são: a) a investigação de possíveis otimizações de desempenho das redes e *frameworks* sobre os *datasets* avaliados, incluindo mecanismos de calibração automática dos hiperparâmetros e designação de classes; e b) a utilização de imagens de satélites de bacias hidrográficas para avaliação do desempenho das redes e *frameworks* comparativamente aos resultados aqui obtidos.

REFERÊNCIAS

ABBAS, M. K.; HAMZAH, M. A. Convolutional Neural Network for Satellite Image Classification. **Springer Nature Switzerland**, Diwaniyah, Iraq, p. 15, January 2020.

ADEYEMO, J.; OYEBODE, O.; STRETCH, D. River Flow Forecasting Using an Improved Artificial Neural Network, Durban - South Africa, 2018.

AGATONOVIC-KUSTRIN, S.; BERESFORD, R. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. **Journal of Phamaceutical and Biomedical Analysis**, New Zealand, 1999.

AGGARWAL, C. C. **Neural Networks and Deep Learning**. Gewerbestrasse, Switzerland: Springer International Publishing AG, 2018.

AKMAL ABDUL HAMID, N. N.; ADAWIYA RAZALI, R.; IBRAHIM, Z. Comparing bags of features, conventional convolutional neural network and alexnet for fruit recognition. **Indonesian Journal of Electrical Engineering and Computer Science**, Malasia, p. 8, 2019.

ANGELIS, A. F.; ROCHA, T. Evaluating the Deep Learning accuracy in data extraction from synthetic image sequences. **Pan-American Associacion of Computational Interdisciplinary Sciences**, Limeira, p. 7, November 2019.

ANGELIS, A. F.; ROCHA, T.; HIDALGO, L. G. Synthetic Images Generation for Deep Learning Assessment Towards the InFlow Forecast for Power Systems Operation. **DINCON - Conferência Brasileira de Dinâmica, Controle e Aplicações**, São José do Rio Preto/SP, p. 7, 30 Outubro 2017.

BAHRAMPOUR, S. et al. Comparative Study of Deep Learning Software Frameworks. **Research and Technology Center, Robert Bosch LLC**, 2016.

BALAGOUROUCHETTY, L. et al. GoogLeNet-Based Ensemble FCNet Classifier for Focal Liver Lesion Diagnosis. **IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS**, jun. 2020.

BALLESTER, P.; MATSUMURA ARAUJO, R. On the Performance of GoogLeNet and AlexNet Applied to Sketches. **Thirtieth AAAI Conference on Artificial Intelligence**, Pelotas - Brasil, p. 5, 2016.

BAYAR, B.; STAMM, M. C. A Deep Learning Approach To Universal Image Manipulation Detection Using A New Convolutional Layer. **Drexel University**, p. 6, 2020.

BELOTTI, J. T. **Previsão de vazões afluentes utilizando redes neurais artificias e ensembles.** Ponta Grossa: Universidade Tecnológica Federal do Paraná, 2019.

BRAZ JUNIOR, G. **Convolucional Neural Networks - CNN**. Universidade Federal Maranhão. São Luis-MA. 2020.

CAFFE. **Wikipidia**, 2020. Disponivel em: <https://en.wikipedia.org/wiki/Caffe_(software)>. Acesso em: 17 fev. 2020.

CHEN, T.-H. what-is-stride-in-convolutional-neural-network. **medium.com**, 7 Novembro 2017. Disponivel em: <https://medium.com/machine-learning-algorithms/what-is-stride-in-convolutional-neural-network-e3b4ae9baedb>.

CHEN, Y. et al. Deep Learning-Based Classification of Hyperspectral Data. **IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING**, China, v. 7, p. 14, jun. 2014.

DAS, S. CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more. **medium.com**, 2017. Disponivel em: <https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>. Acesso em: 4 dez. 2019.

DONG, Y. et al. Evaluations of Deep Convolutional Neural Networks for Automatic Identification of Malaria Infected Cells. **IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)**, p. 4, 2020.

DONG, Y. et al. Evaluations of Deep Convolutional Neural Networks for Automatic Identification of Malaria Infected Cells, p. 4.

DONGARE, A. D.; KHARDE, R. R.; KACHARE, A. D. Introduction to Artificial Neural Network. **International Journal of Engineering and Innovative Technology (IJEIT)**, 2012.

DSA. DeepLearningBook. **deeplearningbook**, Janeiro 2020. Disponivel em: <http://deeplearningbook.com.br/>. Acesso em: 26 nov. 2019.

ENERGÉTICA, -E. D. P. Matriz Energética e Elétrica. **epe.gov.br**, 2020. Disponivel em: <https://www.epe.gov.br/sites-pt/publicacoes-dados-abertos/publicacoes/PublicacoesArquivos/publicacao-479/topico-528/BEN2020_sp.pdf>. Acesso em: 02 dez. 2020.

EPE, E. D. P. E. **Balanço Energético Nacional**. Empresa de Pesquisa Energética. Rio de Janeiro. 2019.

FLORES, G. F.; FERREIRA, V. H.; ZAMBOTI, M. MODELO CHUVA-VAZÃO PARA PREVISÃO DE VAZÃO AFLUENTE DIÁRIA UTILIZANDO REDES NEURAIS ARTIFICIAIS. **XIII Simpósio Brasileiro de Automação Inteligente**, Porto Alegre, 2017.

FLORES, G.; FERREIRA, V. H. A Rain-Streamflow Model for Prediction of Limnimetric Behavior of Reservoirs Using Artificial Neural Networks. **Fluminense Federal University**, Niterói, 2017.

FLORINDO, J. B. **Redes Neurais Convolucionais - Deep Learning**. Universidade Estadual de Campinas. Campinas, p. 70. 2020.

FRANÇA, H. F. D. C.; SOARES, A. GoogLeNet - Going Deeper with Convolutions. **Instituto de Informática - Universidade Federal de Goiás**, Goiás, dez. 2016.

FRID-ADAR, M. et al. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. **Neurocomputing**, Israel, p. 11, 2018.

GOMES, A. C. V.; PAES, A.; NADER, C. V. Um Estudo sobre Redes Neurais Convolucionais e sua Aplicac¸ao em Deteccao de Pedestres. **Universidade Federal Fluminense**, Niterói-RJ Brasil, p. 4, 2020.

GOMES, L. F. C.; MONTENEGRO, S. M. G. L. Modelo Baseado na Técnica de Redes Neurais para Previsão de Vazões na Bacia do Rio São Francisco. **RBRH - Revista Brasileira de Recursos Hídricos**, Pernambuco, p. 11, 2010.

GUILHON, L. G. F.; ROCHA, V. F.; MOREIRA, J. C. Comparação de métodos de previsão de vazões naturais afluentes a aproveitamentos hidroelétricos. **RBRH - Revista Brasileira de Recursos Hídricos**, 2007.

HOCHULI, A. G. **Redes Neurais Convolucionais**. Universidade Federal Parana. Curitiba - PR. 2020.

JEONG-HWAN, K. et al. Assessment of Electrocardiogram Rhythms by GoogLeNet Deep Neural Network Architecture. **Hindawi - Journal of Healthcare Engineering**, Republic of Korea, p. 11, 2019.

KALOGEITON, V. et al. **Deep Learning frameworks**. [S.l.]: [s.n.], 2017.

KERNEL (image processing). **WikiPedia**, 21 out. 2020. Disponivel em: <https://en.wikipedia.org/wiki/Kernel_(image_processing)>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks, Toronto - Canada, p. 9.

LAVAGNOLI, S. Machine Learning ou Deep Learning. **OPENCADD**, 2020. Disponivel em: <https://opencadd.com.br/machine-learning-ou-deep-learning/>. Acesso em: 24 mar. 2020.

LEG/UFPR, L. D. E. E. G.-. Gradiente Descendente. **LEG/UFPR**, 5 nov. 2020. Disponivel em: <http://cursos.leg.ufpr.br/ML4all/apoio/Gradiente.html>.

LI, Q. et al. Performance Evaluation of Deep Learning Classification Network for Image Features. **IEEE**, China, p. 16, jan. 2021.

MAKANTASIS, K. et al. Deep Supervised Learning for Hyperspectral Data Classification Through Convolutional Neural Networks. **Technical University of Crete - Greece**, Grécia, p. 4, 2020.

MUHAMMAD, N. A. et al. Evaluation of CNN, Alexnet and GoogleNet for Fruit Recognition. **Indonesian Journal of Electrical Engineering and Computer Science**, Malasia, p. 8, 2018.

OKSUZ, I. et al. Automatic CNN-based detection of cardiac MR motion artefacts using k-space data augmentation and curriculum learning. **Medical Image Analysis**, London-UK, p. 12, 2019.

PHAM, T. D. Classification of COVID-19 chest X-rays with deep learning: new models or fine tuning? **Health Information Science and Systems**, Arabia Saudita, p. 11, 2020.

PONTI, M. A.; COSTA, G. B. P. D. **Como funciona o Deep Learning**. São Carlos/SP: ICMC - Universidade de São Paulo, 2017.

POWELL, V. Image Kernels. **Setosa**, 3 nov. 2020. Disponivel em: <https://setosa.io/ev/image-kernels/>.

RAWAT, W.; WANG, Z. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. **Massachusetts Institute of Technology**, South Africa, p. 118, 2017.

RE, R. **Um Processo para Construção de Frameworks a partir de Engenharia Reserva de Sistemas de Informação Baseados na Web:** Aplicação ao Domínio dos Leilões Virtuais. São Carlos/SP: ICMC/USP, 2002.

REDDY, K. S.; RANJAN, M. **Solar resource estimation using artificial neural networks and comparison with other correlation models**. India: Pergamon, 2002.

ROCHA, T. **Aplicação do SMAP para a Bacia do Rio Tietê**. [S.l.]: [s.n.], 2015.

SANG-GEOL, L.; YUNSICK, S.; EUI-YOUNG, C. Variations of AlexNet and GoogLeNet to Improve Korean Character Recognition Performance. **Journal of Information Processing Systems - JIPS**, Korea, p. 13, February 2018.

SANTOS, C. D. C. et al. Redes Neurais Artificiais aplicadas à previsão de vazões sazonais através da relação chuva-vazão. **XIV CEEL - ISSN 2178-8308**, Uberlândia-MG, 2016.

SATIYAH, R. D. et al. Performance Evaluation for Vision-Based Vehicle Classification Using Convolutional Neural Network. **International Journal of Engineering & Technology**, Malasia, p. 6, 2018.

SHAI, S.-S.; SHAI, B.-D. **Understanding Machine Learning:** From Theory to Algorithms. Cambridge: Cambridge University Press, 2014.

SHIHADEH, J.; ABSARI, A.; OGUNFUNMI, T. Deep Learning Based Image Classification for Remote Medical Diagnosis. **IEEE**, Santa Clara, California, p. 8, 2018.

SINGLA, A.; YUAN, L.; EBRAHIMI, T. Food/Non-food Image Classification and Food Categorization using Pre-Trained GoogLeNet Model. **Multimedia Signal Processing Group - Ecole Polytechnique Fédérale de Lausanne**, Lausanne - Switzerland, p. 9, 2016.

SOUSA, W. D. S.; SOUSA, F. D. A. S. **Rede neural artificial aplicada à previsão de vazão da Bacia Hidrográfica do Rio Piancó**. Campina Grande: Revista Brasileira de Engenharia Agrícola e Ambiental, 2009.

SPRINGENBERG, J. T. et al. **Striving for Simplicity:** The All convolutional Net. Freigurg, Germany: University of Freiburg, 2015.

SRIVASTAVA, N. et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. **Journal of Machine Learning Research**, Canada, p. 30, 2014.

SUN, J. et al. Retinal Image Quality Classification Using Fine-Tuned CNN. **Lecture Notes in Compuer Science, vol 10554, Springer, Cham.**, Singapura, p. 8, September 2017.

SUSTIKA, R. et al. Evaluation of Deep Convolutional Neural Network Architectures for Strawberry Quality Inspection. **International Journal of Engineering & Technology**, Indonesia, p. 7, 2018.

SZEGEDY, C. et al. Going Deeper with Convolutions. **IEEE Xplore**, Chapel Hill - Carolina do Norte (EUA), p. 9, 2014.

TENSORFLOW. **InfoWorld**, 2020. Disponivel em: <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html>. Acesso em: 17 fev. 2020.

USEMOBILE. Usemobile. **O que é uma API**, 2020. Disponivel em: <https://usemobile.com.br/o-que-e-uma-api/>. Acesso em: 21 fev. 2020.

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um Estudo sobre Redes Neurais Convolucionais e sua Aplicaçãoo em Detecção de Pedestres. **Instituto Federal Fluminense**, Niterói-RJ, 2016.

VIANA. O que é Overfitting e Underfitting em Machine Learning. **StackOverFlow**, 2019. Disponivel em: <https://pt.stackoverflow.com/questions/377643/o-que-%C3%A9-overfitting-e-underfitting-em-machine-learning>. Acesso em: 21 fev. 2020.

VIDUSHI, S.; SACHIN, R.; ANURAG, D. **A Comprehensive Study of Artificial Neural Networks**. India: International Journal of Advanced Research in Computer Science and Software Engineering, 2012.

WALLISCH, P. et al. **Matlab for Neurocientists**. Nova York - EUA: [s.n.], 2009.

WANGENHEIM, A. V. Deep Learning Glossário. **Lapix - UFSC**, 2018. Disponivel em: <http://www.lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/deep-learningglossario/>.

WIKIPEDIA. Dilution (neural networks). **WikiPedia**, 10 nov. 2010. Disponivel em: <https://en.wikipedia.org/wiki/Dilution_(neural_networks)>.

XING, Y. et al. Driver Activity Recognition for Intelligent Vehicles: A Deep Learning Approach. **IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY**, p. 12, 2019.

YAMASHITA, R. et al. Convolutional neural networks: an overview and application in radiology. **Insights into Imaging**, p. 19, 2019.

YANG, Y. et al. Glioma Grading on Conventional MR Images: A Deep Learning Study with Transfer Learning. **Frontiers in Nueroscience**, p. 10, 2018.

ZHANG, J. et al. Monitoring sugar crystallization with deep neural networks. **Journal of Food Engineering**, China e Australia, p. 9, fev. 2020.

APÊNDICE A – FISGEN (FAST IMAGE SEQUENCE GENERATOR)

FISGEN é um software desenvolvido pelo Prof. Dr. André F. de Angelis para suporte à pesquisa e tem como objetivo automatizar a geração de longas sequências de imagens rotuladas. Os rótulos das imagens individuais correspondem a uma ou mais métricas calculadas pelo programa em função da figura gerada. Os arquivos de saída podem ser, por padrão, no formato JPG ou GIF. A programação seguiu o paradigma da Orientação a Objetos e o código foi escrito em linguagem Java no ambiente Eclipse.

O software é um motor para geração de sequências de imagens e atua como um *framework* para esta tarefa a partir de um conjunto de classes fixo. O usuário deve fornecer um Modelo de Animação Gráfica (MAG), na forma de classe Java, para cada tipo de imagem que pretenda gerar, definindo nesse MAG a composição das figuras e as métricas de interesse.

A implementação do software foi feita para produzir imagens rotuladas para avaliação de usabilidade, funcionamento e desempenho de ferramentas de aprendizado de máquina profundo (Deep Learning) em estudos exploratórios de mestrados e doutorados da Faculdade de Tecnologia da Unicamp. Os requisitos iniciais foram levantados pela doutoranda Thaís Rocha.

O FISGEN tem interface de linha de comando (modo texto) e pode rodar em sistemas operacionais que ofereçam um *Java Runtime Environment (JRE)* compatível com Java 1.8 ou superiores. O software tem duas versões maiores (major releases) até o momento (dezembro/2021), como segue:

A versão 1 está estruturada em pacote único e tem 7 classes compondo o *framework* do motor. Sua codificação começou em 04/maio/2017 e sua última atualização (release 1.11) foi feita em 06/agosto/2020. Foram implementados para esta versão 14 MAGs usando os recursos de herança e polimorfismo do paradigma.

Um MAG deve necessariamente ser derivado (herança) da classe abstrata *AbstractGenerator* e implementar dois métodos abstratos, sendo livre para sobrescrever outros métodos que necessite ou criar métodos de seu uso privativo.

A classe *AbstractGenerator* é parte do motor e fornece métodos concretos de serviço como, por exemplo, a inclusão de ruído ou o posicionamento de marcas de referência na imagem. Ela possui o método *protected void doGeneration()*, acionado internamente pelo motor, implementado segundo o padrão *Template Method*, padronizando o funcionamento do programa. Os métodos abstratos a implementar são:

1. abstract protected void generateImage(Color foreground, Color background); - Esse método deve codificar a composição de uma figura por vez, tratando a questões de alocação de um *buffer* de memória para desenho, dimensões e posicionamento da imagem, utilização de cores e padrões e o preenchimento de um vetor com as métricas desejadas.
2. abstract protected int[][] generatePointTrack(); - Esse método carrega uma matriz com o posicionamento das figuras, usualmente seu centro, nas imagens a serem geradas, permitindo transição suave entre duas imagens da sequência.

Em um certo momento, houve a necessidade de adaptar o programa a uma especificidade do NVIDIA *Deep Learning GPU Training System* (DIGITS), ferramenta que passou a ser utilizada nas pesquisas. A adaptação, relativa ao agrupamento de imagens segundo as métricas dos modelos, alterou profundamente a arquitetura projetada, tornando-a complexa e motivando um novo projeto, que deu origem à versão seguinte.

A versão 2 do FISGEN foi resultado de um processo de reengenharia da versão anterior. Sua codificação se iniciou em 06/agosto/2020 e sua última atualização (*release* 2.0.00 (*Beta-test*)) foi liberada em 05/abril/2021. O software é composto de 3 pacotes (*system* e *utilities* para o motor; *imageGenerators* para os MAGs). Os pacotes do motor reúnem 7 classes no total.

A classe *AbstractGenerator* é parte do pacote de geradores e foi redesenhada para oferecer mais serviços prontos, ser mais flexível, usar menos memória e aperfeiçoar algumas rotinas internas. Em virtude da alteração da sua interface pública, os MAGs da versão anterior são incompatíveis com a nova. Foi introduzido como padrão no software um MAG nulo para facilitar tarefas de desenvolvimento e teste do programa. Até dezembro/2021 havia disponível um

conjunto de 20 MAGS para a versão 2 do FISGEN. O novo projeto dessa classe eliminou a matriz de posicionamento e redefinindo os métodos abstratos a implementar.

1. abstract protected void generateImage (Color foreground, Color background); - Mantém as funcionalidades básicas e princípios de funcionamento da versão anterior.
2. abstract protected int getGroupByLocalCriteria(int localData); - Permite que o MAG defina o critério de agrupamento de classes segundo sua métrica de interesse de forma personalizada e independente de posicionamento das figuras na composição das imagens.

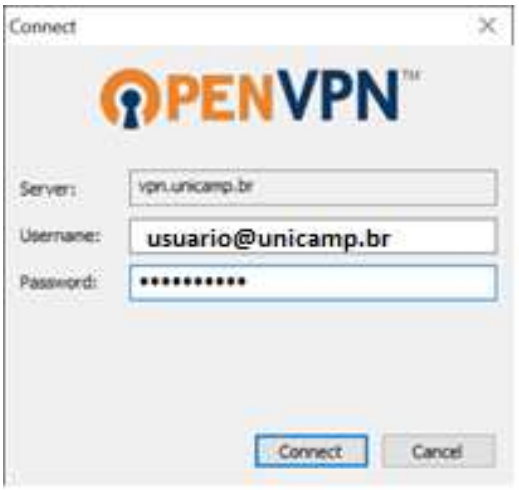
Este software está em registrado pelo INPI[17] e sua distribuição ficará sob a responsabilidade da agência INOVA[18].

[17] Instituto Nacional da Propriedade Industrial - https://www.gov.br/inpi/pt-br
[18] Dúvidas sobre a disponibilização, entrar em contato com o prof. Dr. André Franceschi de Angelis (mailto:andre@ft.unicamp.br).

APÊNDICE B – ACESSO AO SERVIDOR DE DEEP LEARNING

O acesso à ferramenta inicia-se através de uma sessão utilizando o software Putty. Caso o acesso seja de fora das dependências da Unicamp, é necessário estabelecer uma conexão VPN. O tutorial para instalação e configuração do OpenVPN está na página do Centro de Computação da Unicamp – CCUE, (https://www.ccuec.unicamp.br/ccuec/servicos/acesso_remoto_vpn). Os acessos são criados pelo departamento de Informática da própria Unicamp. Após instalado o software, o usuário poderá efetuar a conexão VPN, conforme Figura 29.

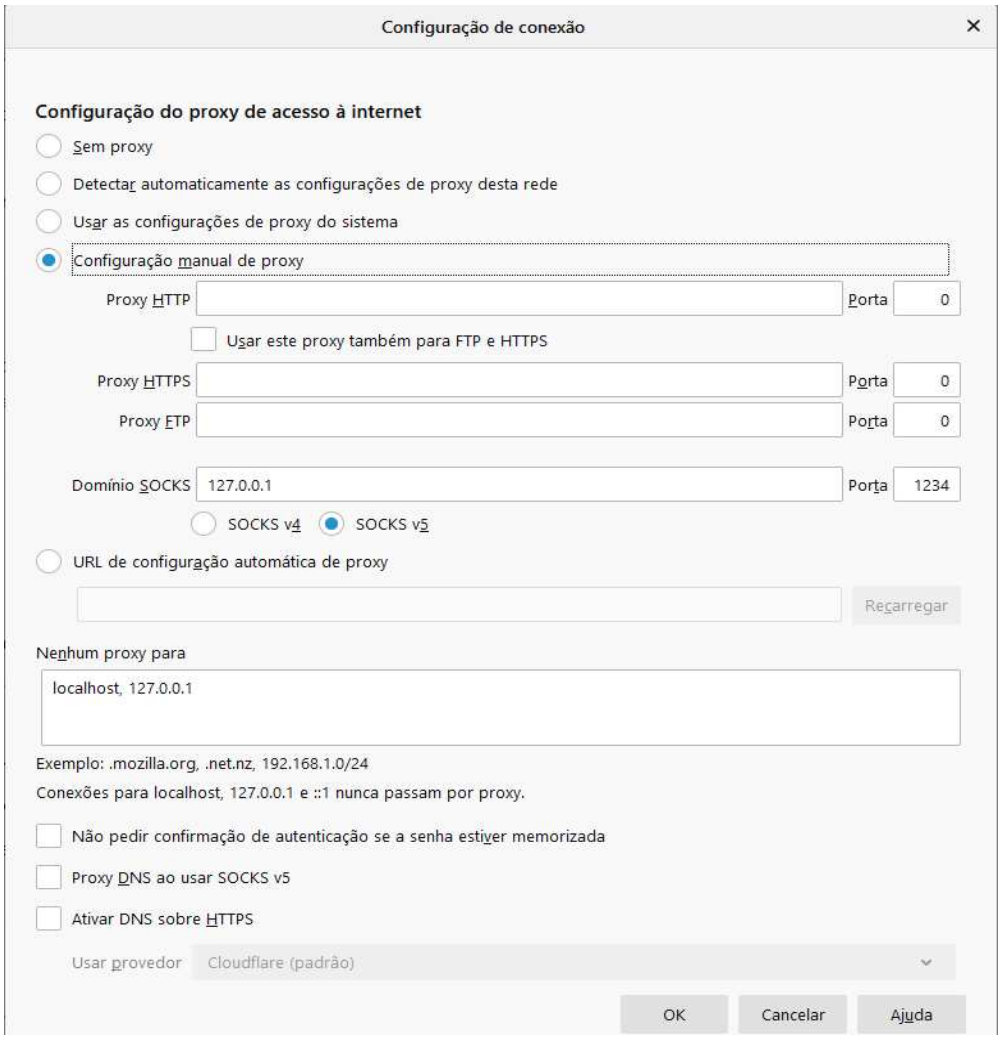
Figura 29 - Estabelecendo uma conexão VPN com o servidor da Unicamp



Fonte: Autoria própria.

A próxima etapa é estabelecer uma conexão SSH com o servidor LASCADO. Este acesso é através do software *Putty* e, através de uma linha de comando no MS-DOS, estabelece-se a conexão. A linha de comando para o acesso é "*Putty -D 1234 id_usuario@lascado.ft.unicamp.br"* onde o id_usuario corresponde ao usuário conforme ilustrado na Figura 30.

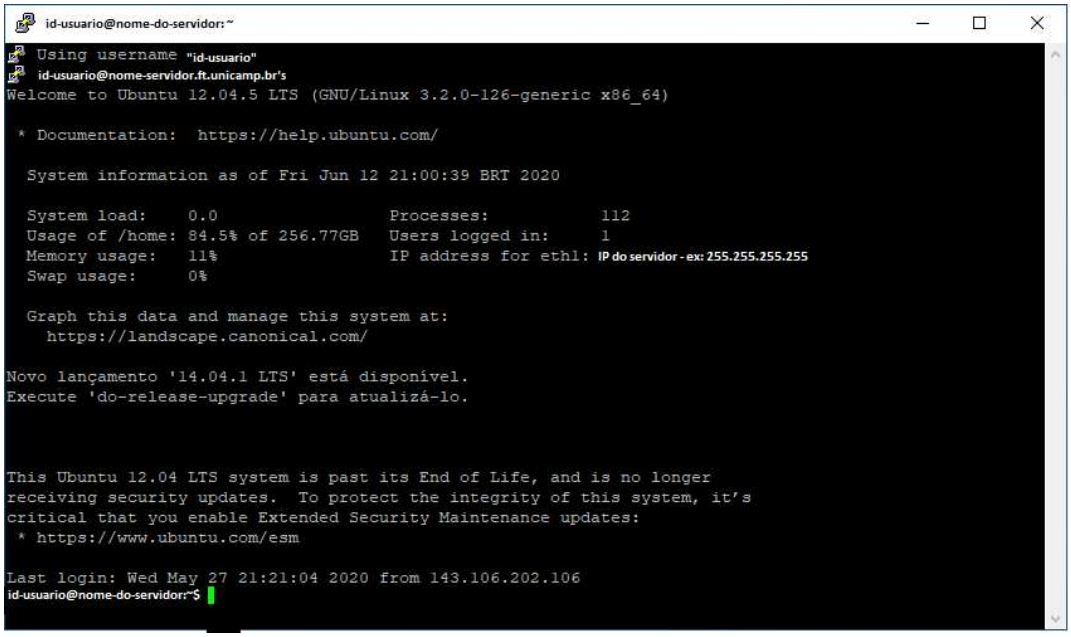
Figura 30 - Configuração do FireFox/Mozila para acesso ao DIGITS da NVIDIA instalado nas dependências da Unicamp



Fonte: Autoria própria.

Após estabelecer conexão com o servidor, a próxima etapa é se conectar ao servidor de *Deep Learning* via SSH. Através do terminal de comandos, executar “ssh IP_Servidor”. A Figura 31 apresenta um exemplo de conexão ao servidor utilizado nesta pesquisa.

Figura 31 - Conexão SSH com o servidor de Deep Learning da Faculdade de Tecnologia da Unicamp

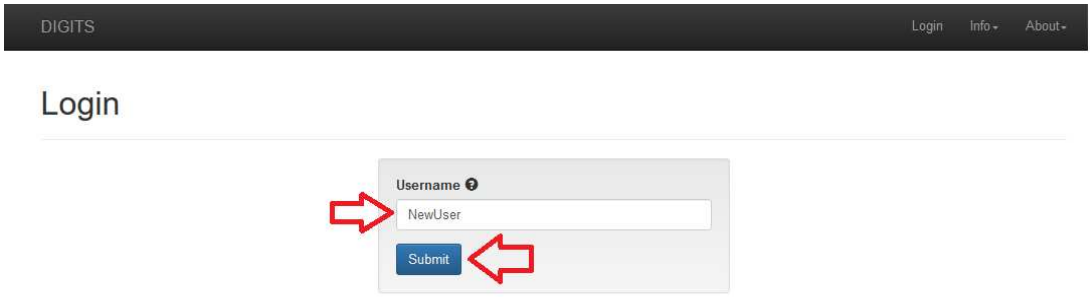


Fonte: Autoria própria.

E, por fim, para se ter acesso à interface gráfica, é necessário utilizar o navegador FireFox/Mozila, sendo que suas configurações deverão ser modificadas. Na sessão, Configuração de Rede, é necessário especificar o domínio SOCKS para 127.0.0.1 e a porta para 1234. Em seguida, deve ser selecionada a opção SOCKS v5 e configurado o proxy para localhost, 127.0.0.1. A Figura 31 apresenta a tela de configuração do navegador.

Estabelecidas as devidas conexões, e com o navegador devidamente configurado, abrir o navegador e, na barra de endereços, digitar http://<IP do servidor>:5000/. Caso haja múltiplos usuários, é conveniente criar uma área para cada um. Esta criação é simples e já na página inicial, indicar o '*Username'* e clicar no botão '*Submit'*. Caso não exista, será criado um. A Figura 32 ilustra como se cria um usuário e fazer o *login* na ferramenta DIGITS.

Figura 32 - Login à plataforma DIGITS e criação do novo usuário

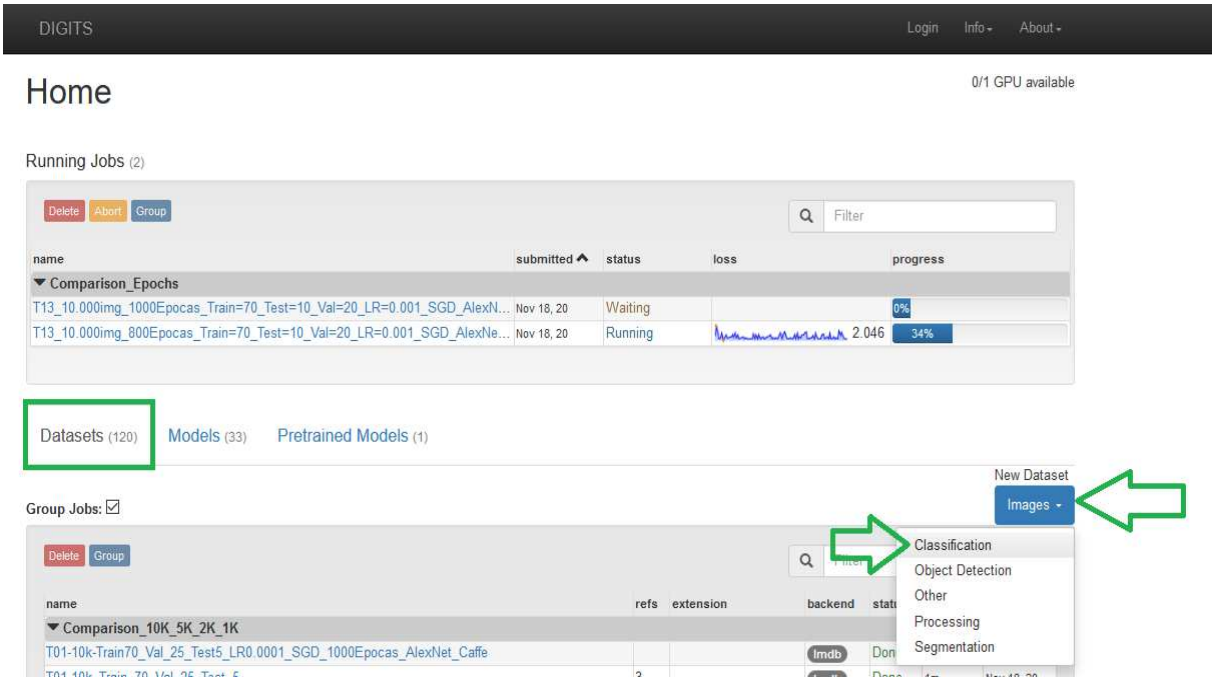


Fonte: Autoria própria.

APÊNDICE C– OPERAÇÃO DO DIGITS

A primeira etapa é criar os *datasets*. Conforme indicam as flechas em verde na Figura 33, selecionar a aba '*Datasets'*, em seguida clicando sobre a opção '*Images'* em '*New Dataset*' e escolhendo a opção '*Classification'*.

Figura 33 - Interface gráfica de acesso à plataforma DIGITS



Fonte: Autoria própria.

Nesta etapa, deverão ser criados todos os *datasets*. A Figura 34 ilustra a tela para criação com os parâmetros necessários para a criação:

(1) É localização das imagens. Indicar aqui a pasta/diretório onde se localizam as imagens.
(2) Indicar a porcentagem do total das imagens que serão utilizadas para Validação.
(3) Indicar a porcentagem do total das imagens que serão utilizadas para Teste.
(4) Indicar o formato da imagem que está sendo utilizado na classificação.
(5) Indicar o nome para o Grupo. Não há nenhum formato exigido e o

usuário pode escolher o nome que desejar.

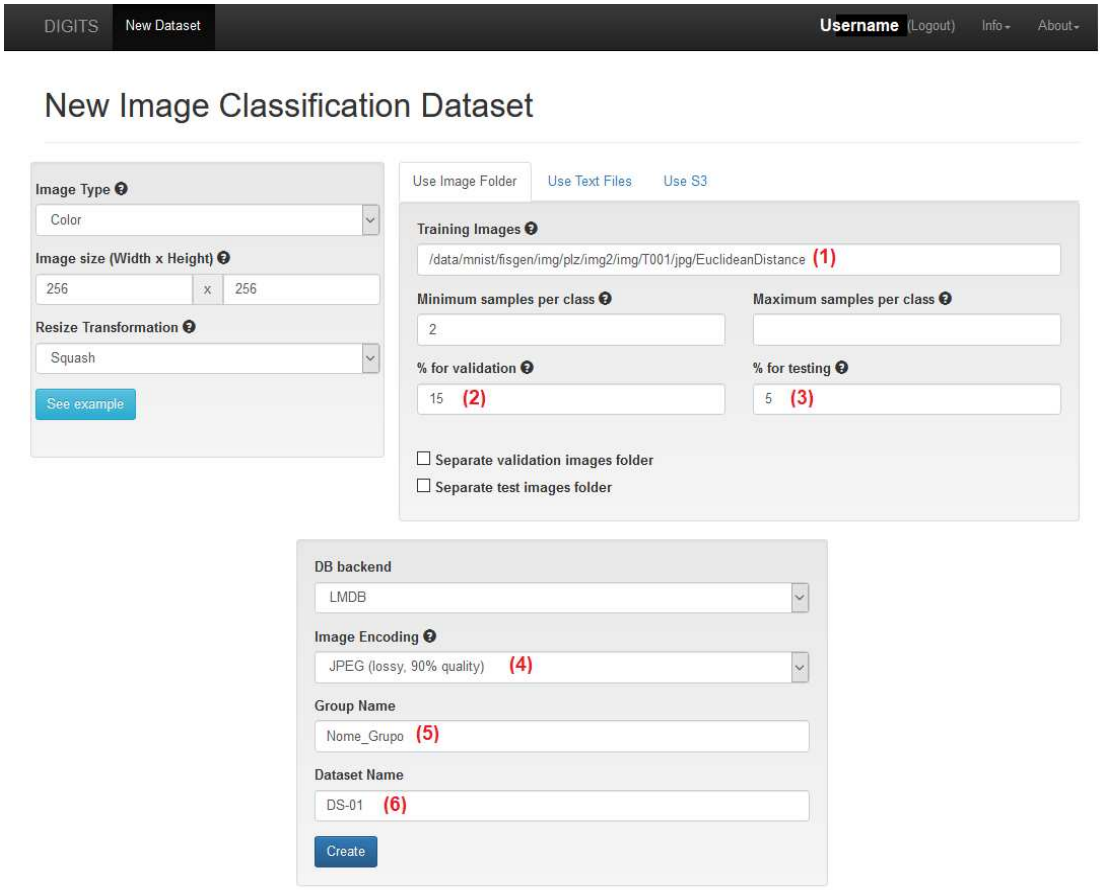
(6) Indicar o nome do *dataset*. Assim como o nome do grupo, o usuário pode escolher o nome que desejar.

Não foi necessário definir a porcentagem de imagens utilizadas para Treinamento. O DIGITS calcula automaticamente de acordo com a fórmula:

Imagens_Treinamento = 100 – (Imagens_Validação + Imagens_Teste)

O número de imagens utilizadas para Treinamento nesta pesquisa foi de 70% do total das imagens.

Figura 34 - Tela para criação dos datasets e os parâmetros necessários

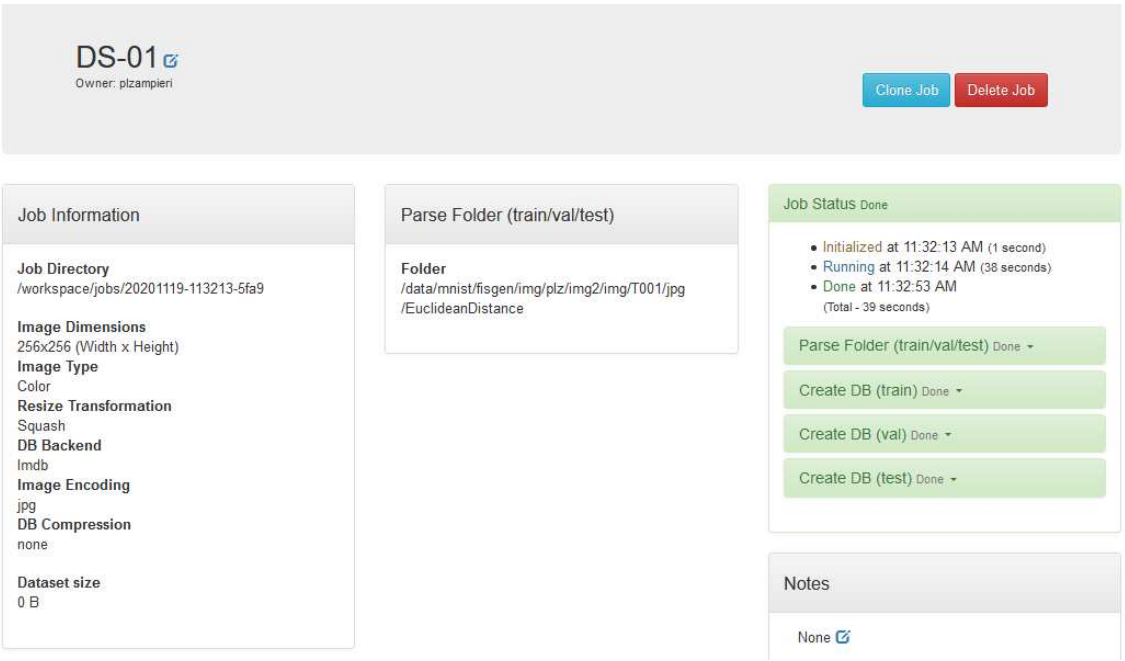


Fonte: Autoria própria.

Alguns parâmetros da Figura 34 não precisaram ser modificados, porém o usuário pode ainda escolher o tamanho da imagem e tipo. Neste trabalho foram utilizadas as dimensões de 256x256 e tipo ‘Color’.

Após clicado em ‘*create’,* o DIGITS criará o *dataset* e mostrará a seguinte tela, ilustrado pela Figura 35, indicando “*Job Status Done*”.

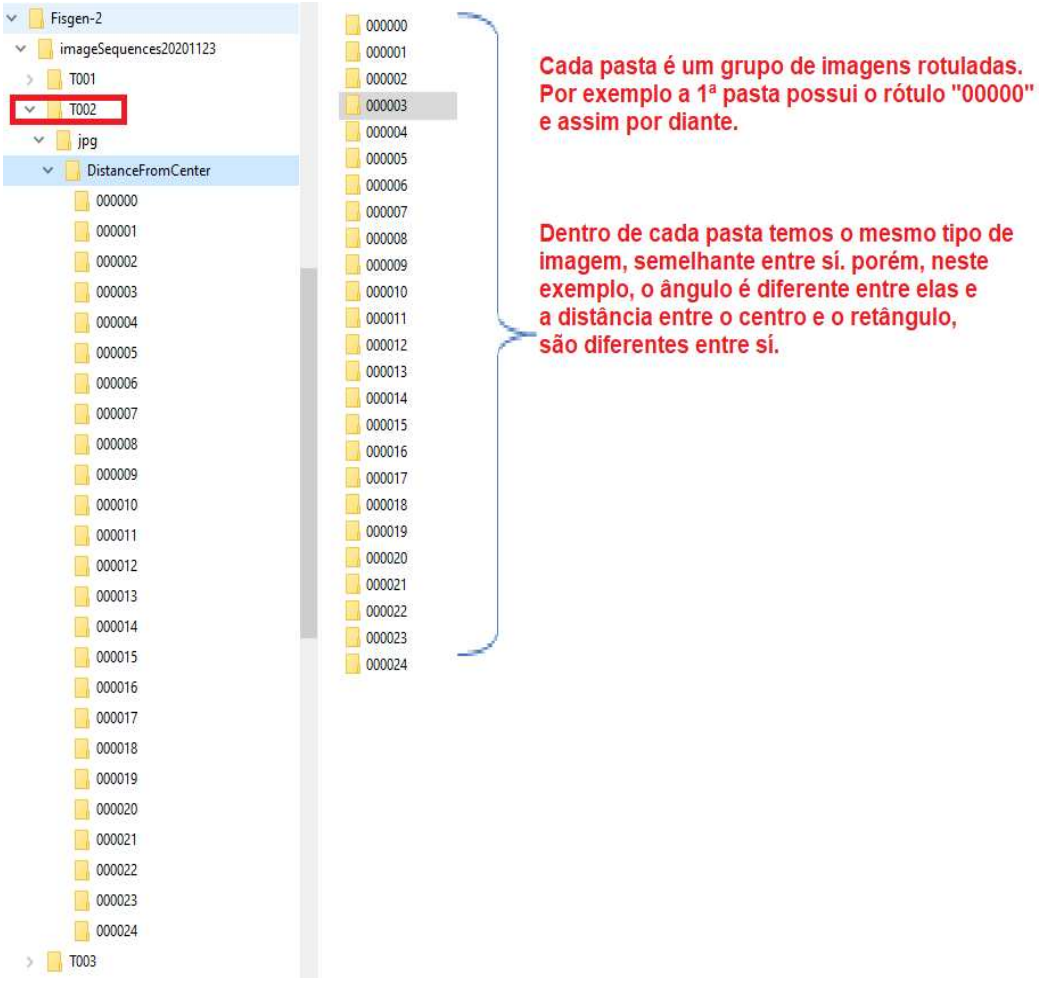
Figura 35 - Tela de indicação de que o *dataset* foi criado com sucesso



Fonte: Autoria própria.

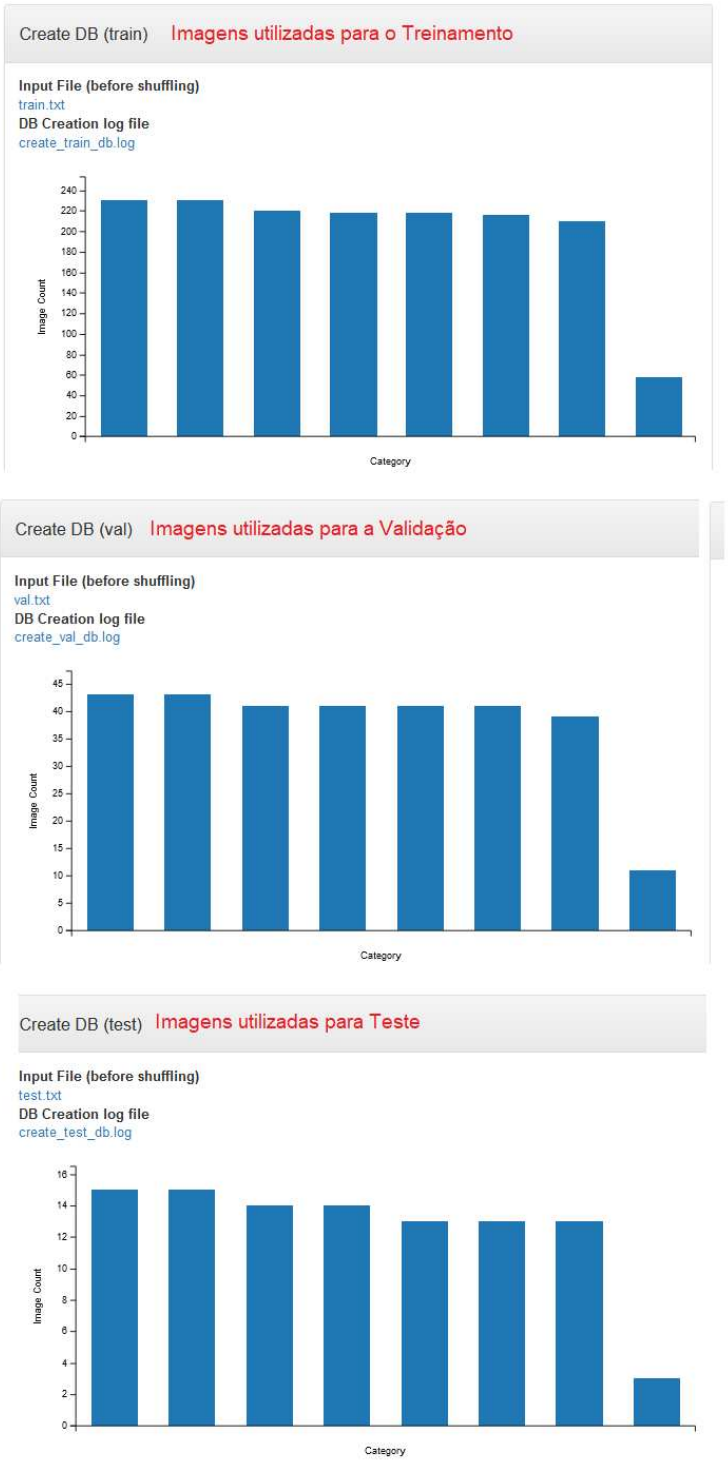
Rolando a tela da Figura 36 para baixo, é possível visualizar o total de imagens que foram separadas para o Treinamento, Validação e Testes e separadas por grupos. Estes grupos foram criados dentro da aplicação FISGEN, onde as imagens estão rotuladas, ou seja, separadas em diferentes pastas/diretórios indicando um rótulo para cada pasta. Na Figura 36, podemos ver estas pastas/diretórios para o *Dataset* T002 e na Figura 37, podemos ver o total de imagens separadas para Treinamento, Validação e Teste por grupo/categoria.

Figura 36 - Estrutura de criação de pastas/diretórios das imagens para o *dataset* T002



Fonte: Autoria própria.

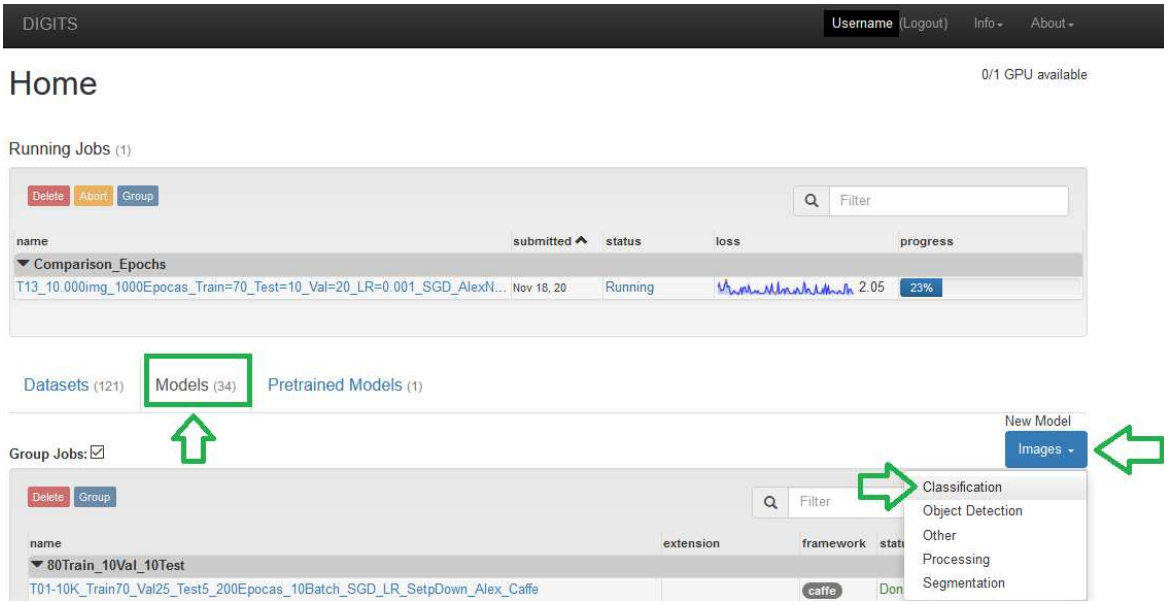
Figura 37 - Visualização do total de imagens criadas para Treinamento, Validação e Testes do *dataset* T002



Fonte: Autoria própria.

Após a criação do *dataset*, a próxima etapa é a criação do modelo que será submetido à classificação pela plataforma DIGITS. É nesta etapa que são definidos os hiperparâmetros, entre eles o tipo da rede, *framework*, taxa de aprendizagem, número de épocas e algoritmo para minimização do erro, que neste trabalho foi utilizado o *Stochastic Gradient Descent* (SGD). A Figura 38 ilustra a interface gráfica da plataforma DIGITS para a criação de cada modelo.

Figura 38 - Criação de modelos para classificação e imagens no DIGITS



Fonte: Autoria própria.

Para criação dos modelos, selecionar a aba '*Models*', e na opção '*New Model*', clicar em '*Images*' e selecionar '*Classification*' na lista de opções, conforme indica as flechas em verde na Figura 38.

A próxima etapa é selecionar escolher hiperparâmetros a seguir:

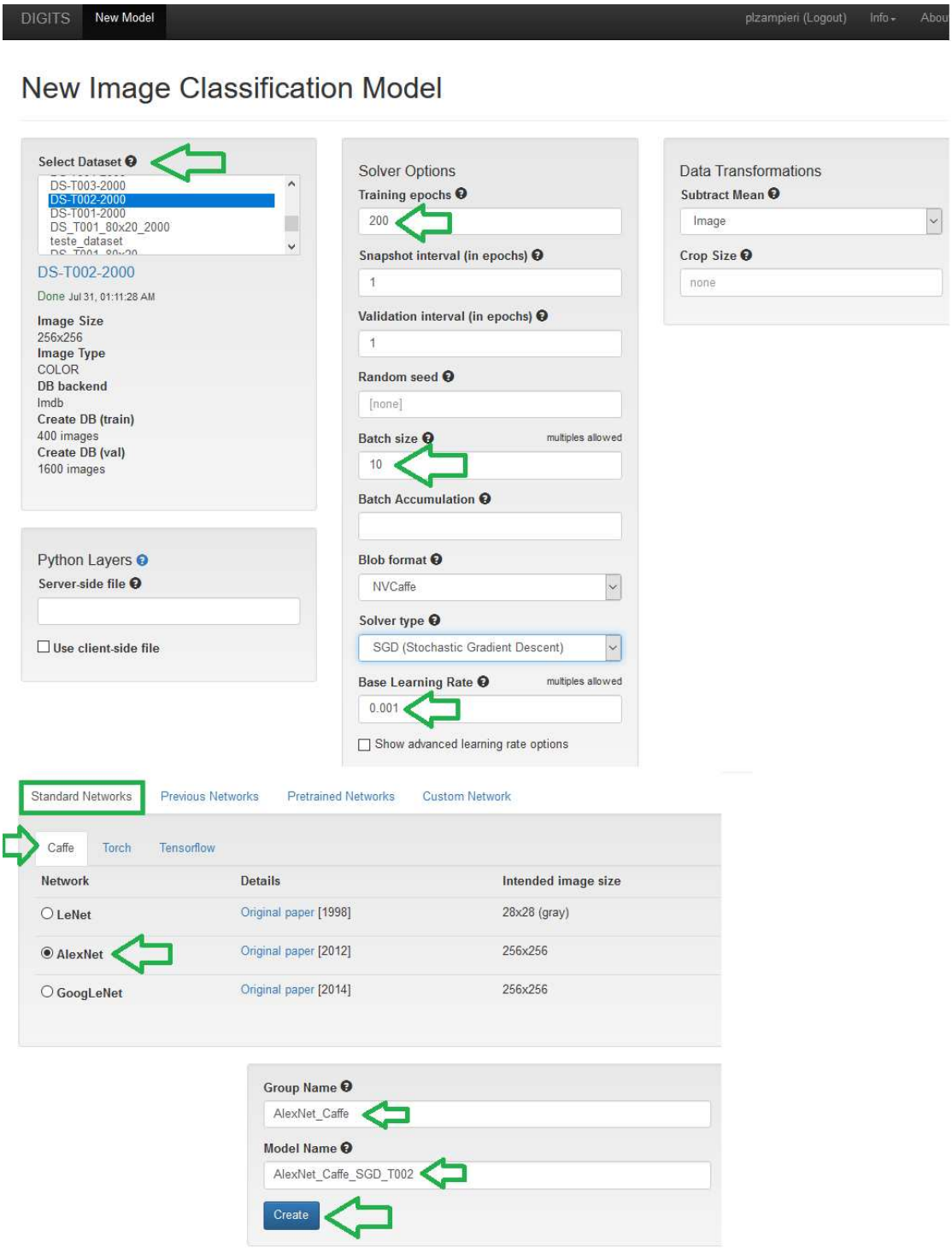
- *Select Dataset*: Nesta lista de opções, estão todos os *datasets* já criados. Selecionar aqui o *dataset* desejado.
- *Training Epochs*: É o número de épocas/iteração/vezes que a imagem percorrerá a rede, como se fosse um processo de loop. Neste processo, a rede tentará encontrar os pesos e filtros ideais de forma que o modelo atinja o melhor desempenho. No caso deste

trabalho, o número de épocas foi de 200.

- *Batch Accumulation*: É o número de imagens que serão processadas simultaneamente, ou lote de imagens. Isso agiliza o modelo na fase de Treinamento. Neste trabalho foi escolhido o número 10.
- *Solver Type*: Nesta opção, escolhe-se o algoritmo otimizador, que no caso deste trabalho foi o *SGD*.
- *Base Learning Rate*: Aqui define-se a taxa de aprendizagem, que neste trabalho foi de 0,001.
- *Standard Networks*: Nesta aba é onde pode-se escolher a rede que será utilizada, que nesta versão do DIGITS que foram utilizadas, a AlexNet e a GoogLeNet. A rede LeNet não foi utilizada neste trabalho. Em combinação com as redes, temos 3 *frameworks* diferentes disponíveis nesta versão: *Caffe*, *Torch* e *Tensor Flow*.
- *Group Name*: Escolha do nome do grupo. O usuário pode escolher o nome que desejar, pois não há validação neste campo.
- *Model Name*: Escolha do nome do modelo. Assim como Group Name, este campo não contém nenhuma validação e o usuário pode escolher o nome desejado. É conveniente escolher um nome descritivo, de forma que através dele, seja possível saber que rede, *framework*, *dataset*, etc está sendo utilizado.
- *Create*: Preenchido todos os campos, ao clicar neste botão, o modelo será criado e imediatamente enviado para Treinamento dentro da plataforma do DIGITS.

Na Figura 39 tem-se um modelo criado com os campos preenchidos com os hiperparâmetros utilizados neste trabalho. Os campos estão indicados por uma flecha verde.

Figura 39 - Criação de modelos para classificação de imagens no DIGITS



Fonte: Autoria própria.

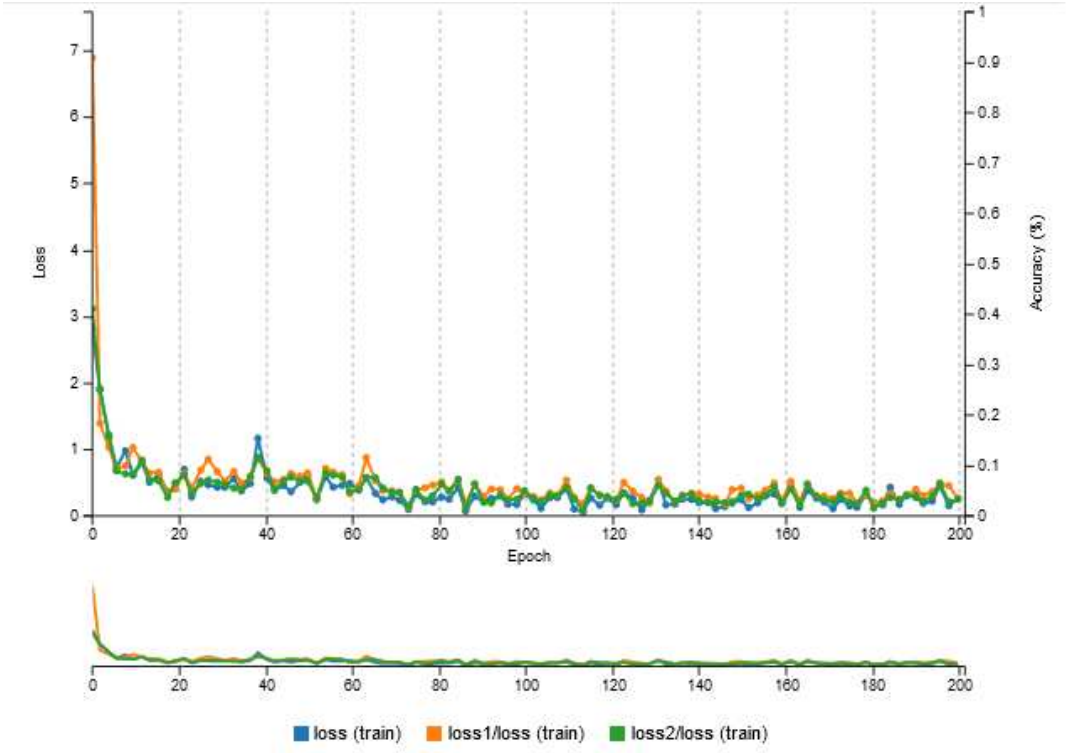
Neste momento, o modelo está passando pela fase de Treinamento e Validação ao mesmo tempo. Este processo é demorado, dependendo dos hiperparâmetros escolhidos e principalmente do número de imagens contidas no *dataset*. A Figura 40 apresenta um modelo sendo treinado através da plataforma DIGITS. Vale ressaltar que, neste mesmo processo, o modelo está passando pela fase de Validação.

Figura 40 - Ilustração de um modelo passando pelo processo de Treinamento e Validação no DIGITS

Fonte: Autoria própria.

Clicando sobre o modelo, indicado pela flecha verde na Figura 40, é possível ver detalhes desta fase de Treinamento e Validação. Na Figura 41, pode-se ver a função *Loss* (perda) sendo ajustada durante o treinamento. Neste exemplo, ela começa com sete e no final época, ela está quase próxima de zero. Isso significa que o modelo está aprendendo os pesos e filtros ideais de forma a reduzir o erro e, consequentemente, aumentando a acurácia do modelo.

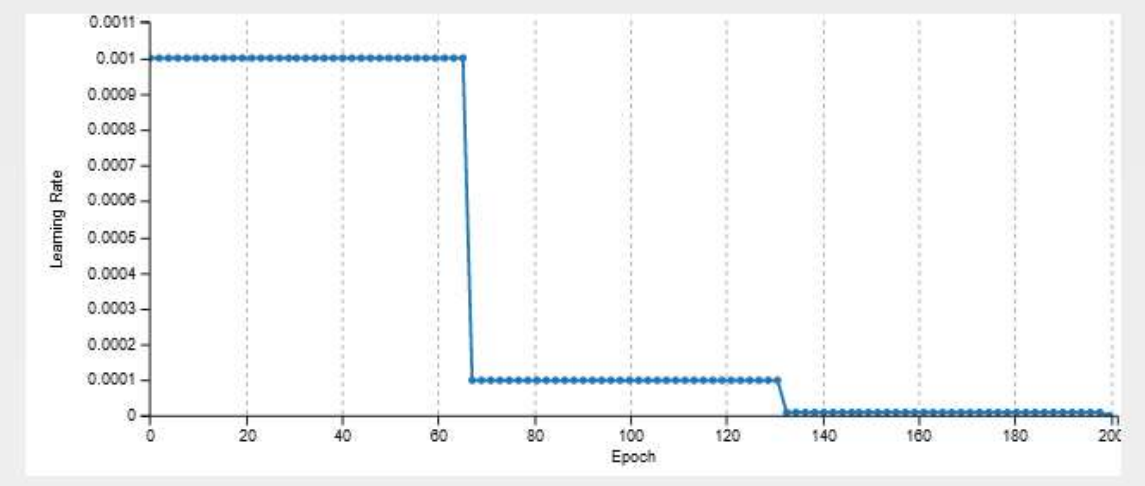
Figura 41 - Gráfico da função *Loss* sendo reduzida ao mínimo com o passar das épocas



Fonte: Autoria própria.

E na Figura 42 pode-se ver o gráfico da Taxa de Treinamento (*learning rate*) sendo ajustada à medida que passa pelas épocas. Dentro da plataforma do DIGITS, é possível escolher o tipo de curva desejada. No caso deste exemplo, é um gráfico tipo *Step Down*, porém, poderia ser escolhido *Exponential Decay*, que faria com que esta curva se tornasse uma exponencial.

Figura 42 - Gráfico da Taxa de Treinamento sendo ajustada à medida que percorre pelas épocas.

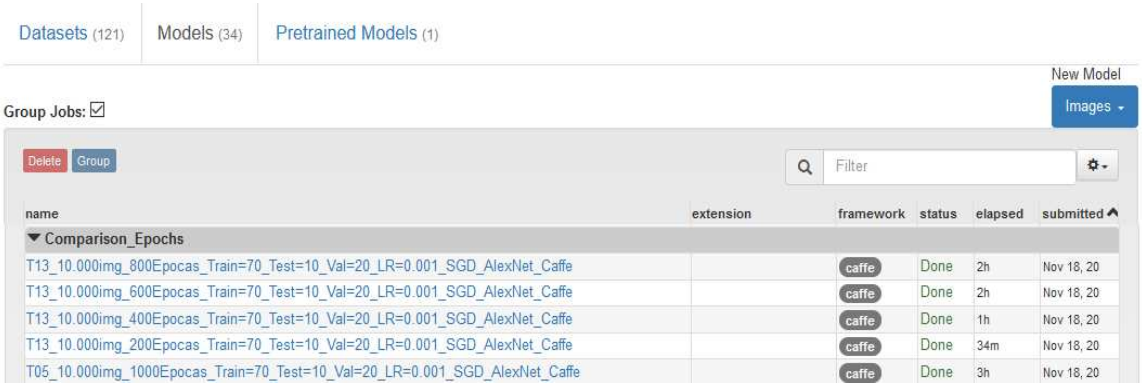


Fonte: Autoria própria.

Com o modelo Treinado e Validado, a próxima e última fase é a de Teste. Nesta etapa, serão submetidas todas as imagens reservadas para Teste à classificação. É nesta fase que veremos a acurácia do modelo criado.

Dentro da plataforma do DIGITS, selecionar a aba ‘Models’ e selecionar algum modelo já treinado e cujo status esteja em ‘*Done’*, significando que o modelo não apresentou nenhum erro. Na Figura 43 tem-se uma lista de modelos já treinados e prontos para Teste.

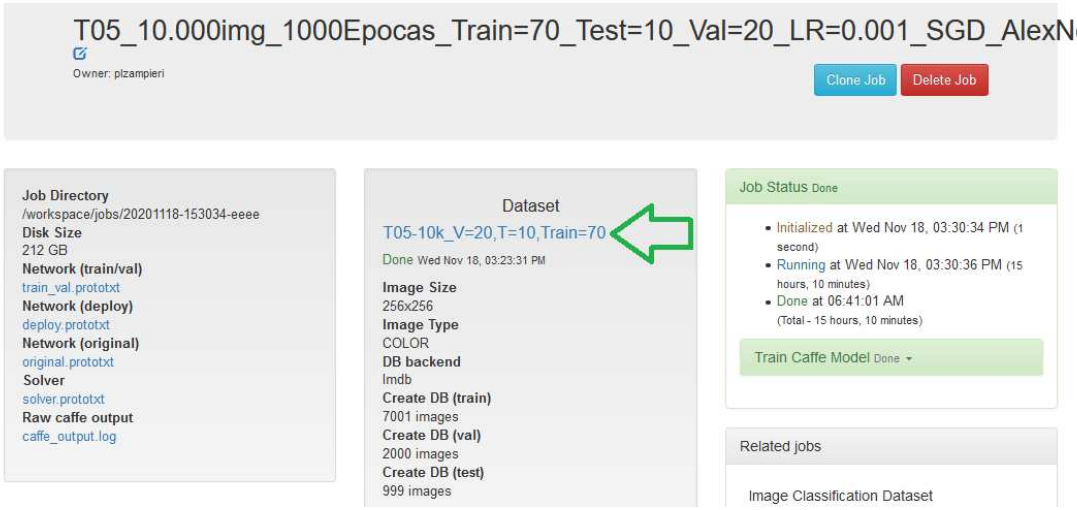
Figura 43 - Lista de modelos já treinados na plataforma DIGITS e prontos para a fase de Teste



Fonte: Autoria própria.

Nas colunas, temos algumas informações como o *framework* utilizado no modelo, o seu *status* e o tempo total utilizado na fase de Treinamento e Validação. Clicando sobre qualquer um dos modelos já treinados, será aberto a opção para testar o modelo. A primeira coisa a fazer é abrir o *dataset* e salvar a localização e nome das imagens separadas para testes. Neste caso, serão testados um conjunto de imagens, entretanto, no DIGITS, é possível testar imagens individuais. A Figura 44 apresenta um modelo já treinado e validado. É possível também ver algumas informações adicionais como, tamanho da imagem, tipo, quantidade de imagens selecionadas para a fase de Treinamento, Validação e Testes.

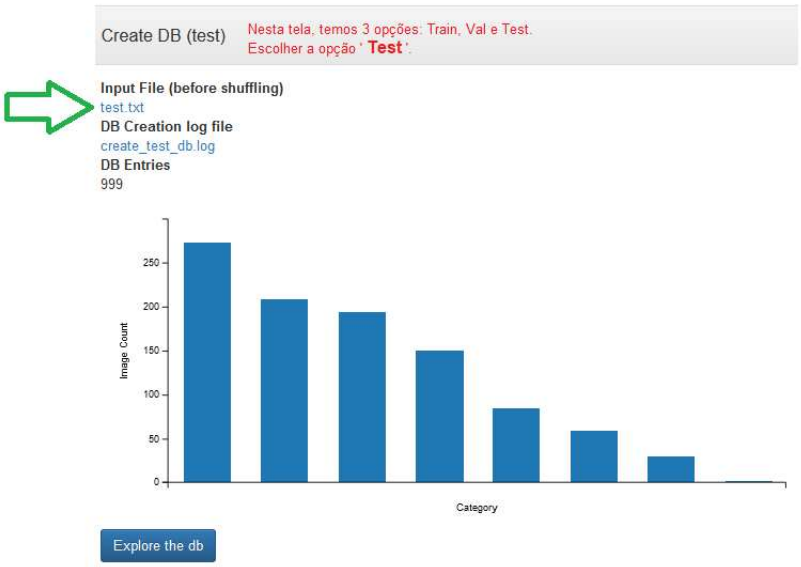
Figura 44 - Modelo já Treinado e Validado pela plataforma DIGITS e pronto para Testes



Fonte: Autoria própria.

Ao clicar sobre o nome do *dataset*, podemos ver os detalhes. A próxima etapa é rolar a tela para baixo até encontrar a opção '*Create DB (Test)*', abrir o arquivo texto '*Test.txt*', utilizado neste exemplo, que está indicado pela flecha verde na Figura 45.

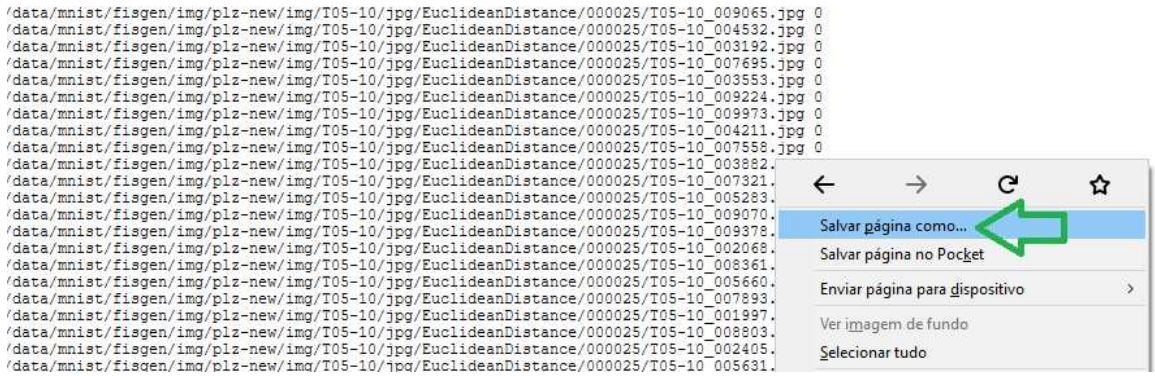
Figura 45 - Tela para seleção das imagens que serão testadas no modelo treinado



Fonte: Autoria própria.

Ao clicar sobre o arquivo ‘Test.txt’, será aberta a lista da localização e do nome de cada imagem separadas para a fase Teste. Com o botão direito do mouse sobre a lista, escolhe-se a opção ‘Salvar página como’, conforme ilustra a Figura 46.

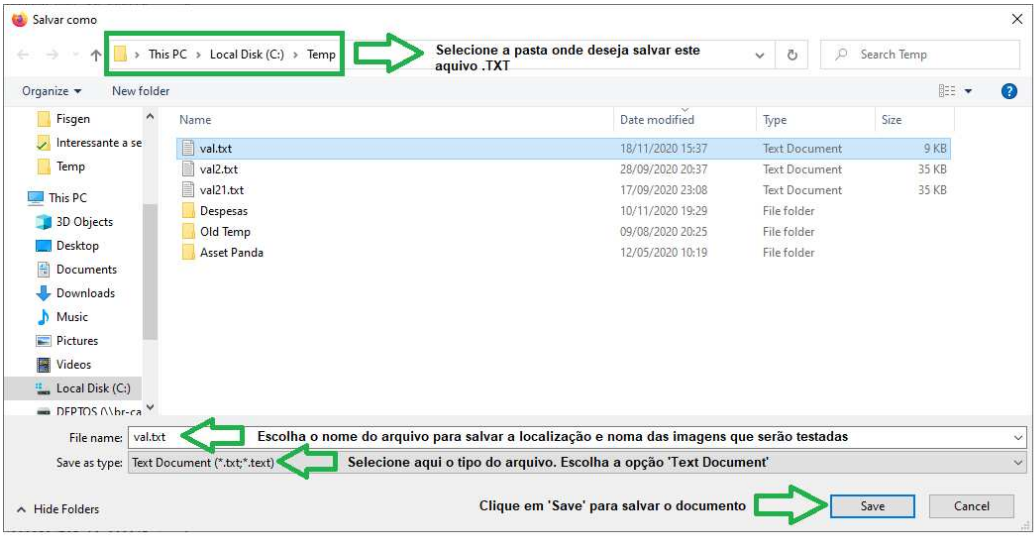
Figura 46 - Lista da localização e nome das imagens que serão testadas pelo DIGITS de um *dataset* qualquer



Fonte: Autoria própria.

Ao abrir a caixa de diálogo, escolhe-se a pasta onde deseja salvar o arquivo, define-se o nome e o tipo do arquivo, conforme ilustrado na Figura 47.

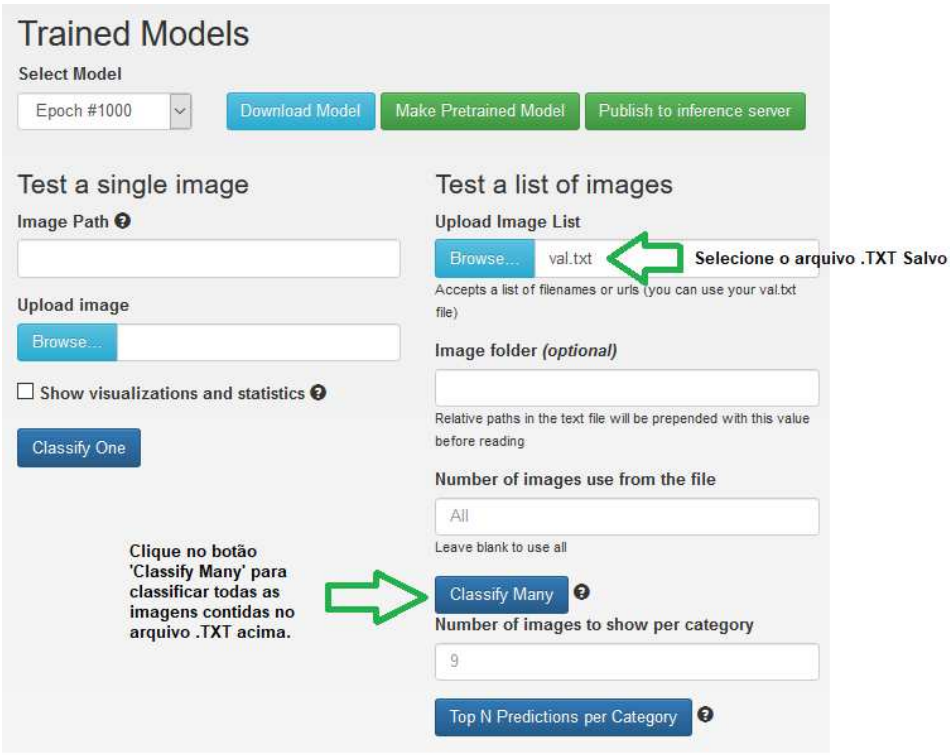
Figura 47 - Caixa de diálogo para salvar em arquivo .TXT as imagens que serão testadas no modelo



Fonte: Autoria própria.

Com o arquivo salvo, volta-se à tela anterior e rola-se para baixo até a opção '*Trained Models*' e ir até a opção '*Test a list of images*'. Com o botão '*browse*', selecionar o arquivo salvo ilustrado na Figura 47. Após selecionado o arquivo, clicar sobre o botão '*Classify Many*' conforme a Figura 48.

Figura 48 - Seleção do arquivo .TXT para classificação das imagens no DIGITS



Fonte: Autoria própria.

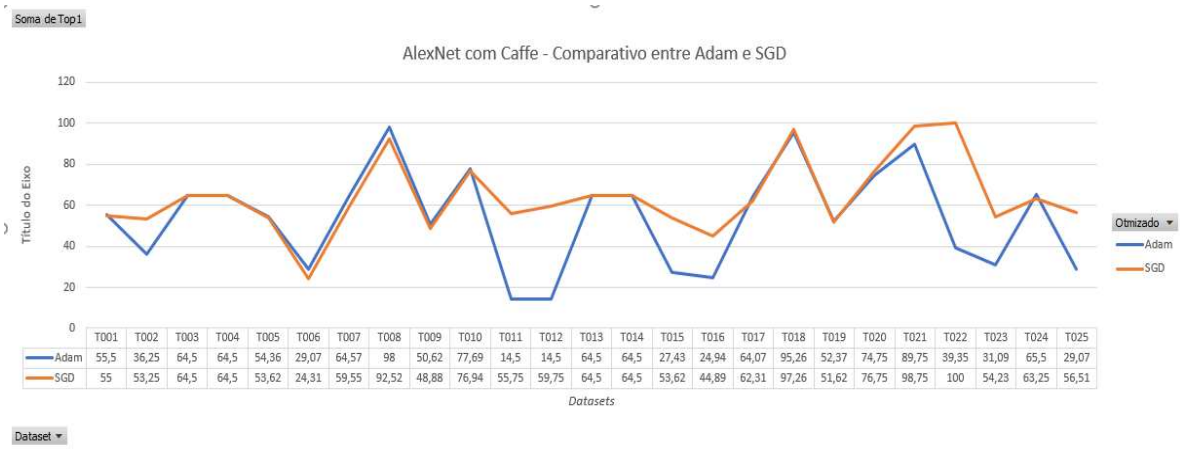
Em seguida, será aberta uma nova janela em branco. Aguarda-se até que o DIGITS classifique as imagens e os apresente no *browser*, Mozilla Firefox. O DIGITS apresentará dois resultados, Top1-Accuracy e o Top5-Accuracy e a matriz de confusão descritas no Apêndice E.

APÊNDICE D –ESCOLHA DO OTIMIZADOR E DA TAXA DE TREINAMENTO

Para a determinação do melhor otimizador, foram testados dois diferentes: Adam e SGD. Utilizando a rede AlexNet com os parâmetros, taxa de aprendizagem e número de épocas, com valores iguais aos apresentados no Quadro 8, o SGD foi o que apresentou melhor resultado utilizando os três *frameworks* descritos no Quadro 7.

A Figura 49 apresenta o desempenho do SGD utilizando a rede AlexNet com o *framework* Caffe. É possível notar que dos 25 *datasets* submetidos à classificação, o SGD obteve melhor desempenho em 13 deles.

Figura 49 - Comparativo entre Adam x SGD utilizando a rede AlexNet com o *framework* Caffe

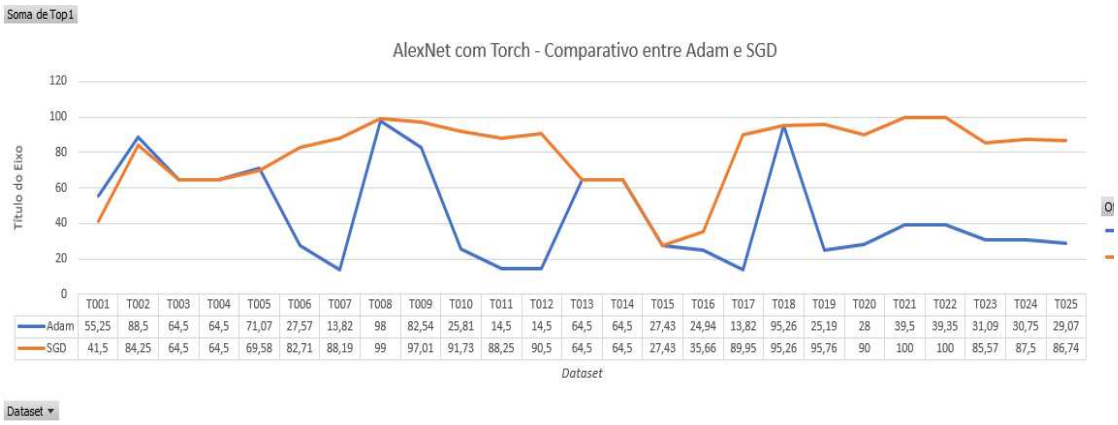


	T001	T002	T003	T004	T005	T006	T007	T008	T009	T010	T011	T012	T013	T014	T015	T016	T017	T018	T019	T020	T021	T022	T023	T024	T025
Adam	55,5	36,25	64,5	64,5	54,36	29,07	64,57	98	50,62	77,69	14,5	14,5	64,5	64,5	27,43	24,94	64,07	95,26	52,37	74,75	89,75	39,35	31,09	65,5	29,07
SGD	55	53,25	64,5	64,5	53,62	24,31	59,55	92,52	48,88	76,94	55,75	59,75	64,5	64,5	53,62	44,89	62,31	97,26	51,62	76,75	98,75	100	54,23	63,25	56,51

Fonte: Autoria própria.

Utilizando a rede AlexNet com o *framework* Torch, o SGD apresentou melhores resultados. Dos 25 *datasets*, 16 deles obtiveram melhores resultados com o SGD. A Figura 50 apresenta um comparativo entre o Adam e o SGD.

Figura 50 - Comparativo entre Adam x SGD utilizando a rede AlexNet com o *framework* Torch



	T001	T002	T003	T004	T005	T006	T007	T008	T009	T010	T011	T012	T013	T014	T015	T016	T017	T018	T019	T020	T021	T022	T023	T024	T025
Adam	55,25	88,5	64,5	64,5	71,07	27,57	13,82	98	82,54	25,81	14,5	14,5	64,5	64,5	27,43	24,94	13,82	95,26	25,19	28	39,5	39,35	31,09	30,75	29,07
SGD	41,5	84,25	64,5	64,5	69,58	82,71	88,19	99	97,01	91,73	88,25	90,5	64,5	64,5	27,43	35,66	89,95	95,26	95,76	90	100	100	85,57	87,5	86,74

Fonte: Autoria própria.

E o mesmo comparativo, porém utilizando o *framework* TensorFlow com a rede AlexNet, nota-se que o TensorFlow obteve melhor desempenho em 22 *datasets* do total de 25. A Figura 51 apresenta um comparativo entre o Adam e o SGD.

Figura 51 - Comparativo entre Adam x SGD utilizando a rede AlexNet com o TensorFlow

	T001	T002	T003	T004	T005	T006	T007	T008	T009	T010	T011	T012	T013	T014	T015	T016	T017	T018	T019	T020	T021	T022	T023	T024	T025
Adam	91,25	1,25	64,5	64,5	28,18	27,57	13,82	98	12,72	69,17	14,5	14,5	64,5	64,5	27,43	24,94	13,82	95,26	12,72	46,75	39,5	39,35	31,09	30,75	29,07
SGD	98	100	64,25	63,75	87,28	91,73	100	99,5	96,51	91,73	98	100	57,25	89,78	86,78	87,28	100	97,76	97,76	88,25	100	100	92,54	94	90,7

Fonte: Autoria própria.

Para a escolha da Taxa de Treinamento LR, foi feito um teste utilizando duas taxas de treinamento diferentes: 0,0001 e 0,001. Criado dois modelos com estas taxas e ambos utilizando a rede AlexNet com o *framework* Caffe. A Tabela 4 apresenta

os resultados para ambos os modelos e a Figura 52 apresenta um gráfico comparando as duas taxas de treinamento.

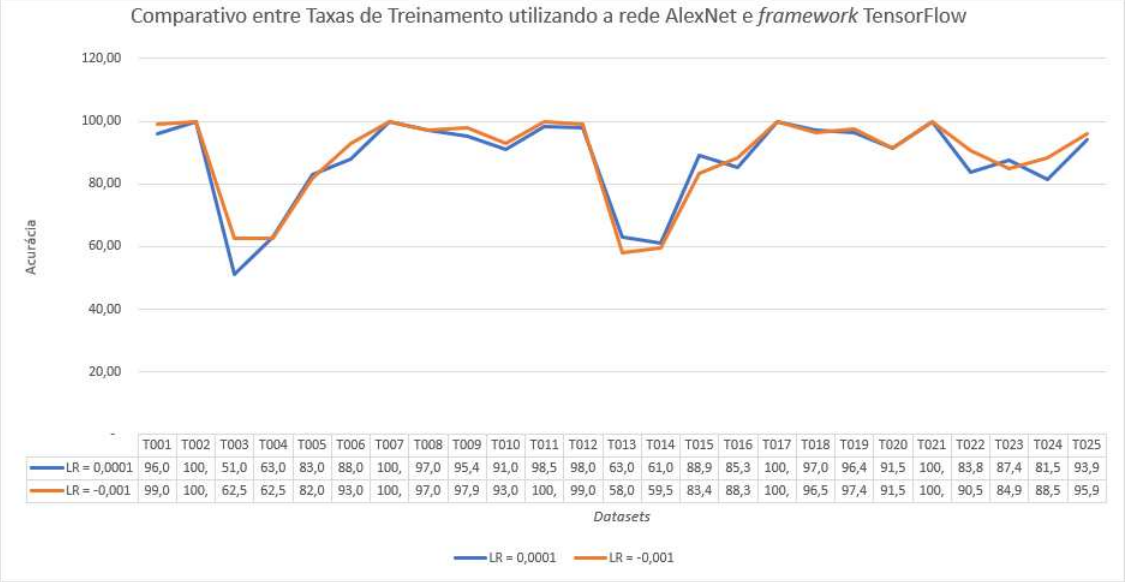
Tabela 4 - Comparativo entre Taxas de Treinamento de 0,001 x 0,0001 utilizando a rede AlexNet e o *framework* TensorFlow

Dataset	LR = 0,0001	LR = -0,001	Melhor Modelo	Rede	Framework
T001	96,00	99,00	LR = 0,001	AlexNet	TensorFlow
T002	100,00	100,00	LR = 0,001	AlexNet	TensorFlow
T003	51,01	62,50	LR = 0,001	AlexNet	TensorFlow
T004	63,00	62,50	LR = 0,0001	AlexNet	TensorFlow
T005	83,00	82,00	LR = 0,0001	AlexNet	TensorFlow
T006	88,06	93,03	LR = 0,001	AlexNet	TensorFlow
T007	100,00	100,00	LR = 0,001	AlexNet	TensorFlow
T008	97,06	97,06	LR = 0,001	AlexNet	TensorFlow
T009	95,43	97,97	LR = 0,001	AlexNet	TensorFlow
T010	91,09	93,07	LR = 0,001	AlexNet	TensorFlow
T011	98,50	100,00	LR = 0,001	AlexNet	TensorFlow
T012	98,00	99,00	LR = 0,001	AlexNet	TensorFlow
T013	63,00	58,00	LR = 0,0001	AlexNet	TensorFlow
T014	61,00	59,50	LR = 0,0001	AlexNet	TensorFlow
T015	88,94	83,42	LR = 0,0001	AlexNet	TensorFlow
T016	85,35	88,38	LR = 0,001	AlexNet	TensorFlow
T017	100,00	100,00	LR = 0,001	AlexNet	TensorFlow
T018	97,01	96,52	LR = 0,0001	AlexNet	TensorFlow
T019	96,45	97,46	LR = 0,001	AlexNet	TensorFlow
T020	91,54	91,54	LR = 0,001	AlexNet	TensorFlow
T021	100,00	100,00	LR = 0,001	AlexNet	TensorFlow
T022	83,80	90,55	LR = 0,001	AlexNet	TensorFlow
T023	87,44	84,92	LR = 0,0001	AlexNet	TensorFlow
T024	81,50	88,50	LR = 0,001	AlexNet	TensorFlow
T025	93,94	95,96	LR = 0,001	AlexNet	TensorFlow
MÉDIA	87,64	88,84			

Total de	LR = 0,0001	7
Total de	LR = 0,001	18

Fonte: Autoria própria.

Figura 52 - Gráfico comparativo entre as Taxas de Treinamento 0,0001 x 0,001 utilizando a rede AlexNet e o *framework* TensorFlow



	T001	T002	T003	T004	T005	T006	T007	T008	T009	T010	T011	T012	T013	T014	T015	T016	T017	T018	T019	T020	T021	T022	T023	T024	T025
LR = 0,0001	96,0	100,	51,0	63,0	83,0	88,0	100,	97,0	95,4	91,0	98,5	98,0	63,0	61,0	88,9	85,3	100,	97,0	96,4	91,5	100,	83,8	87,4	81,5	93,9
LR = -0,001	99,0	100,	62,5	62,5	82,0	93,0	100,	97,0	97,9	93,0	100,	99,0	58,0	59,5	83,4	88,3	100,	96,5	97,4	91,5	100,	90,5	84,9	88,5	95,9

Fonte: Autoria própria.

Nota-se através da Figura 52 que o modelo com a taxa de treinamento igual a 0,001 foi o que apresentou melhor resultado. A Tabela 4 mostra que a média deste modelo foi de 88,84 e que, de 25 *datasets* submetidos à classificação de imagens, 18 deles apresentaram melhores resultados utilizando a taxa de treinamento.

APÊNDICE E – CÁLCULO DA ACURÁCIA

Com o modelo treinado e testado com novas imagens, o DIGITS apresentará a acurácia. A Figura 53 ilustra a matriz de confusão.

Figura 53 - Matriz de confusão - resultado apresentado pelo DIGITS ao submeter um modelo à fase de Testes

Summary

Top-1 accuracy
73.57%

Top-5 accuracy
100.0%

Confusion matrix

	000025	000050	000075	000100	000125	000150	000175	000200	Per-class accuracy
000025	30	0	0	0	0	0	0	0	100.0%
000050	48	36	0	0	0	0	0	0	42.86%
000075	0	32	118	0	0	0	0	0	78.67%
000100	0	0	31	177	0	0	0	0	85.1%
000125	0	0	0	52	220	0	0	0	80.88%
000150	0	0	0	0	66	128	0	0	65.98%
000175	0	0	0	0	0	33	26	0	44.07%
000200	0	0	0	0	0	0	2	0	0.0%

Fonte: Autoria própria.

A reta em vermelho representa o número de imagens classificadas corretamente e à direita é a acurácia por cada grupo do modelo. A acurácia é dada pela razão do total de acertos de todos os grupos pelo total de imagens. A equação IV ilustra o cálculo.

$$Acurácia = \frac{Número\ de\ acertos}{Total\ de\ imagens} * 100 \quad (6)$$

Na Tabela 5, pode-se ver o cálculo da acurácia de cada grupo/categoria. Nota-se que algumas subclasses tiveram uma acurácia melhor, por exemplo a 000100, que obteve 85,10% de acurácia, enquanto na classe 000200, todas as imagens foram classificadas incorretamente. A média geral deste grupo foi de 73,57% de acurácia.

Tabela 5 - Ilustração do cálculo da acurácia de cada grupo/categoria.

Grupo	Acertos	Erros	Total Imagens	% acurácia
000025	30	0	30	100%
000050	36	48	84	42,86%
000075	118	32	150	78,67%
000100	177	31	208	85,10%
000125	220	52	272	80,88%
000150	128	66	194	65,98%
000175	26	33	59	44,07%
000200	0	2	2	0%
Total	735	264	999	73,57% (Top-1)

Fonte: autoria própria.

Para o modelo acima, o cálculo da acurácia é dado por:

$$Acurácia = \frac{735}{999} * 100 = 73{,}57\%$$

APÊNDICE F – GERAÇÃO DAS IMAGENS SINTÉTICAS

As imagens sintéticas foram geradas a partir de uma aplicação desenvolvida em Java, denominada como FISGEN - *Fast Image Sequence Generator* (Apêndice H). Esta aplicação foi desenvolvida pela Unicamp de Limeira-SP.

Para geração destas imagens, é necessário estabelecer conexão com o servidor de *Deep Learning*, que no caso desta pesquisa é um computador hospedado no Laboratório de Simulação e de Computação de Alto Desempenho (LaSCADo). Utilizando o terminal de comandos, selecione um diretório para as imagens. Cada conjunto de imagens terá um diretório específico, como no exemplo do Quadro 10, em que T022 contém 2000 imagens específicas que estão descritas no Quadro 13. Estas 2.000 imagens são distribuídas em grupos e cada grupo é um rótulo. No Quadro 9 tem-se os grupos “000000”,”000001”,”000002”, entre outros. Cada grupo contém um número de imagens do mesmo tipo, porém com alguma diferença, como por exemplo, distância euclidiana diferente. A proposta é que o DIGITS consiga perceber estas diferenças e as classifique corretamente em cada grupo.

Quadro 13 - Estrutura de diretórios dos datasets com *labels*

Dataset	Estrutura de pastas
“Nome do *dataset*” *“T022”*	

Fonte: Autoria própria.

O diretório “Nome do *dataset*” e o “Tipo da Imagem” são diferentes para cada *dataset*. No Quadro 12 encontra-se cada *dataset* e a respectiva classe de imagem.

Para a criação de cada *dataset*, executa-se o script no terminal de comandos. O script para geração de todos os *datasets* estão descritos no Quadro 14.

Quadro 14 - *Script* para geração dos *datasets* com as imagens sintéticas

```
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 25    -n T001 -g imageGenerators.MovingPointLinear
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl y  -dg 25    -n T002 -g imageGenerators.MovingPointLinear
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 900   -n T003 -g imageGenerators.MovingPointRotating
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl y  -dg 900   -n T004 -g imageGenerators.MovingPointRotating
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 25    -n T005 -g imageGenerators.MovingPointRandom
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl y  -dg 25    -n T006 -g imageGenerators.MovingPointRandom
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 50    -n T007 -g imageGenerators.Flooding
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 5     -n T008 -g imageGenerators.GaussianCloudMonochrome
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 5000  -n T009 -g imageGenerators.SplashingPoints
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 7     -n T010 -g imageGenerators.GaussianCloudTripleColor
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl n  -dg 25    -n T011 -g imageGenerators.MovingPointLinear
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl y  -dg 25    -n T012 -g imageGenerators.MovingPointLinear
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl n  -dg 900   -n T013 -g imageGenerators.MovingPointRotating
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl y  -dg 900   -n T014 -g imageGenerators.MovingPointRotating
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl n  -dg 25    -n T015 -g imageGenerators.MovingPointRandom
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl y  -dg 25    -n T016 -g imageGenerators.MovingPointRandom
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl n  -dg 50    -n T017 -g imageGenerators.Flooding
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl n  -dg 5     -n T018 -g imageGenerators.GaussianCloudMonochrome
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl n  -dg 5000  -n T019 -g imageGenerators.SplashingPoints
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.05 -rl n  -dg 7     -n T020 -g imageGenerators.GaussianCloudTripleColor
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 10    -n T021 -g imageGenerators.GaussianCloudAndRiver
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 10    -n T022 -g imageGenerators.GaussianCloudGrayScale
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl y  -dg 10    -n T025 -g imageGenerators.SplashingPointsGray
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 10    -n T023 -g imageGenerators.GaussianCloudMonochrome
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl n  -dg 10    -n T024 -g imageGenerators.GaussianCloudTripleColor
java -jar Fisgen_2_0_00.jar -hr 256 -vr 256 -s 2000 -o 0 -rn 0.0  -rl y  -dg 10    -n T025 -g imageGenerators.SplashingPointsGray
```

Fonte: Autoria própria.

Os parâmetros utilizados neste script estão descritos no Quadro 15.

Quadro 15 - Descrição dos parâmetros para geração das imagens sintéticas pelo FISGEN

Parâmetros	Descrição
Fisgen_2_0_00.jar	É a aplicação que gera as imagens
-hr 256	Dimensão da imagem (horizontal) em pixels
-vr 256	Dimensão da imagem (vertical) em pixels
-s 2000	Total de imagens a gerar (2.000 imagens)
-rn 0.05	0.05 indica o nível de ruído gerado dentro da imagem
-rl y	y' indica que haverá uma linha entre os pontos (Reference line)
-dg 25	(Data growth) - quantas classes modelo deveria ter.
-n T012	T012' é o nome do dataset/pasta criada
-g imageGenerators.MovingPointLinear	Classe geradora que será carregada.
-o Officeset	Deslocamento Temporal
-h help	Help

Opções da Classe Geradora (Tipo de imagem)	
Flooding	MovingPointLinear
GaussianCloudAndRiver	MovingPointRandom
GaussianCloudGrayScale	MovingPointRotating
GaussianCloudMonochrome	SplashingPoints
GaussianCloudTripleColor	SplashingPointsGray

Fonte: Autoria própria.

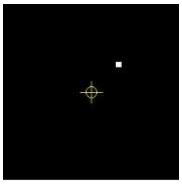
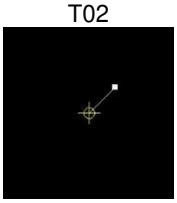
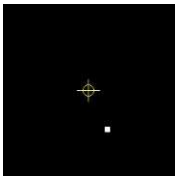
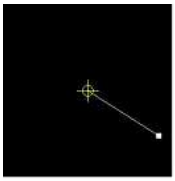
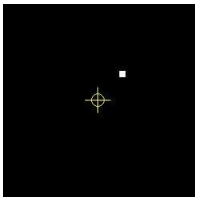
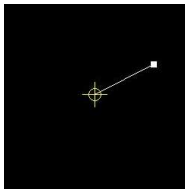
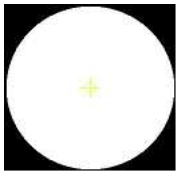
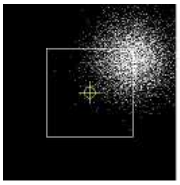
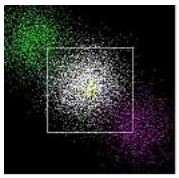
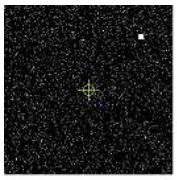
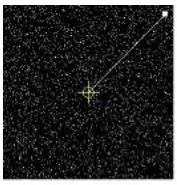
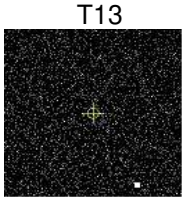
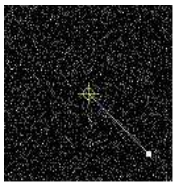
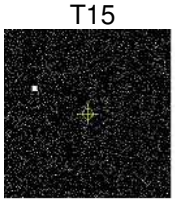
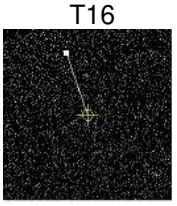
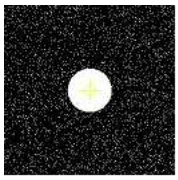
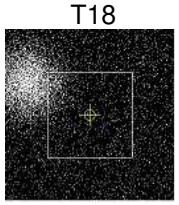
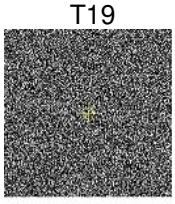
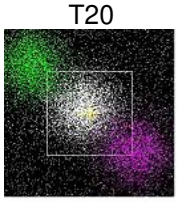
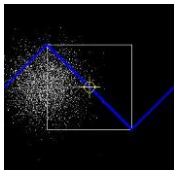
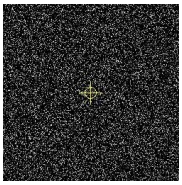
O Quadro 16 ilustra cada *dataset* com o tipo de imagem sintética gerada e suas características e, no Quadro 17, tem-se uma amostra de cada imagem de cada *dataset*.

Quadro 16 - *Dataset* criado com o tipo de imagem e características

Datasets	Metric	Noise Rate	Reference Line	Classes	Images	Group	Images per class	Generator
				Fisen 2				
T001	Distance	0	n	25	2000	25	80,0	imageGenerators.MovingPointLinear
T002	Distance	0	y	25	2000	25	80,0	imageGenerators.MovingPointLinear
T003	Angle	0	n	361	2000	900	5,5	imageGenerators.MovingPointRotating
T004	Angle	0	y	361	2000	900	5,5	imageGenerators.MovingPointRotating
T005	Distance	0	n	25	2000	25	80,0	imageGenerators.MovingPointRandom
T006	Distance	0	y	25	2000	25	80,0	imageGenerators.MovingPointRandom
T007	Coverage	0	n	50	2000	50	40,0	imageGenerators.Flooding
T008	Coverage	0	n	5	2000	5	400,0	imageGenerators.GaussianCloudMonochrome
T009	Coverage	0	n	61	2000	5000	32,8	imageGenerators.SplashingPoints
T010	Coverage	0	n	7	2000	7	285,7	imageGenerators.GaussianCloudTripleColor
T011	Distance	0,05	n	25	2000	25	80,0	imageGenerators.MovingPointLinear
T012	Distance	0,05	y	25	2000	25	80,0	imageGenerators.MovingPointLinear
T013	Angle	0,05	n	361	2000	900	5,5	imageGenerators.MovingPointRotating
T014	Angle	0,05	y	361	2000	900	5,5	imageGenerators.MovingPointRotating
T015	Distance	0,05	n	25	2000	25	80,0	imageGenerators.MovingPointRandom
T016	Distance	0,05	y	25	2000	25	80,0	imageGenerators.MovingPointRandom
T017	Coverage	0,05	n	50	2000	50	40,0	imageGenerators.Flooding
T018	Coverage	0,05	n	5	2000	5	400,0	imageGenerators.GaussianCloudMonochrome
T019	Coverage	0,05	n	61	2000	5000	32,8	imageGenerators.SplashingPoints
T020	Coverage	0,05	n	7	2000	7	285,7	imageGenerators.GaussianCloudTripleColor
T021	Coverage	0	n	4	2000	10	500,0	imageGenerators.GaussianCloudAndRiver
T022	Coverage	0	n	10	2000	10	200,0	imageGenerators.GaussianCloudGrayScale
T023	Coverage	0	n	10	2000	10	200,0	imageGenerators.GaussianCloudMonochrome
T024	Coverage	0	n	10	2000	10	200,0	imageGenerators.GaussianCloudTripleColor
T025	Coverage	0	y	10	2000	10	200,0	imageGenerators.SplashingPointsGray

Fonte: Autoria própria.

Quadro 17 - Amostra de imagens sintéticas geradas para cada *dataset*

T01 	T02 	T03 	T04 
T05 	T06 	T07 	T08 
T09 	T10 	T11 	T12 
T13 	T14 	T15 	T16 
T17 	T18 	T19 	T20 
T21 	T22 	T23 	T24 
T25 			

Fonte: Autoria própria.