UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUCAS CAMPOS TAVANO

ANÁLISE DO POTENCIAL DA LINGUAGEM ELIXIR NO SETOR DE REDES NEURAIS

CURITIBA

2022

LUCAS CAMPOS TAVANO

ANÁLISE DO POTENCIAL DA LINGUAGEM ELIXIR NO SETOR DE REDES NEURAIS

ANALYSIS OF THE POTENTIAL OF THE ELIXIR LANGUAGE IN THE NEURAL NETWORKS SECTOR

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia De Computação do Curso de Engenharia De Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Adolfo Gustavo Serra Seca Neto

CURITIBA

2022

LUCAS CAMPOS TAVANO

ANÁLISE DO POTENCIAL DA LINGUAGEM ELIXIR NO SETOR DE REDES NEURAIS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia De Computação do Curso de Engenharia De Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 30/novembro/2022

João Alberto Fabro
Doutor
Universidade Tecnológica Federal do Paraná

Leandro Batista de Almeida
Mestre
Universidade Tecnológica Federal do Paraná

CURITIBA
2022

RESUMO

Com uma vasta gama de bibliotecas centradas no mercado da aprendizagem de máquina, tais como TensorFlow, NumPy, Pandas, Keras, e outras, Python fez o seu nome como uma das principais linguagens para este setor da programacão. Em fevereiro de 2021, José Valim e Sean Moriarity publicaram a primeira versão da biblioteca *Numerical Elixir* (Nx), uma biblioteca para operações tensoriais escrita na linguagem de programação funcional Elixir criada pelo brasileiro José Valim. A biblioteca Nx visa permitir que a linguagem seja uma boa escolha para operações intensivas em uso de unidades de processamento gráfico. Este trabalho compara os resultados de Python e de Elixir no treinamento de redes neurais convolucionais utilizando conjuntos de dados MNIST e CIFAR-10, concluindo que Python alcançou melhores resultados em geral, possuindo tempo de treino 22,39% mais curto que Elixir para os mesmos cenários. Por um outro lado, este trabalho também conclui que Elixir já se mostra uma alternativa viável para este setor de atuação uma vez atingiu em média boas performances, melhor otimização no uso de memória de acesso randômico (RAM) e se mostrou uma boa alternativa para quem atua no setor de linguagens funcionais.

Palavras-chave: elixir; python; numerical elixir (nx); keras; redes neurais convolucionais.

ABSTRACT

With a wide range of libraries focused on the machine learning market, such as TensorFlow, NumPy, Pandas, Keras, and others, Python has made its name as one of the leading languages for this area of programming. In February 2021, Jose Valim and Sean Moriarity published the first version of the *Numerical Elixir* (Nx) library, a library for tensor operations written in the functional programming language Elixir created by Brazilian Jose Valim. The Nx library aims to enable the language to be a great choice for operations computationally intensive in the use of graphics processing units. This work compares the results of Python and Elixir in training convolutional neural networks using MNIST and CIFAR-10 datasets, concluding that Python achieved better results overall, having 22.39% shorter training time than Elixir for the same scenarios. On the other hand, this work also concludes that Elixir is already a viable alternative for this sector, since it achieved good average performance, better optimization in the use of random access memory (RAM) and proved to be a good alternative for those working in the functional languages industry.

Keywords: elixir; python; numerical elixir (nx); keras; convolutional neural networks.

LISTA DE FIGURAS

Figura 1 – Exemplo de imagens do conjunto MNIST geradas com a ferramenta *Matplotlib* **14**

Figura 2 – Exemplo de imagens do conjunto CIFAR-10 geradas com a ferramenta *Matplotlib* **14**

Figura 3 – Representação visual de uma rede neural artificial com 3 camadas internas. **15**

Figura 4 – Diagrama de alto nível exemplificando a interação entre os três sistemas **19**

Figura 5 – Diagrama de Implantação do sistema Elixir de treinamento de redes neurais. **20**

Figura 6 – Diagrama de Implantação do sistema Python de treinamento de redes neurais. **23**

Figura 7 – Exemplo de como o sistema UnixStats com o *visualization type* ":pretty" imprime no terminal. **24**

Figura 8 – Diagrama de Implantação do sistema UnixStats. **24**

Figura 9 – Gráfico de uso médio de recurso por tempo, medido durante a execução em Elixir. **28**

Figura 10 – Gráfico de uso médio de recurso por tempo, medido durante a execução em Python. **29**

Figura 11 – Gráfico de médio uso de recurso por tempo, medido durante a execução em Elixir. **29**

Figura 12 – Gráfico de médio uso de recurso por tempo, medido durante a execução em Python. **30**

Figura 13 – Gráfico comparativo do uso de *Central Process Unit* (CPU) durante treino de modelo MNIST. **32**

Figura 14 – Gráfico comparativo do uso de CPU. **32**

Figura 15 – Gráfico comparativo do uso de *Random Access Memory* (RAM) durante treino de modelo MNIST. **33**

Figura 16 – Gráfico comparativo do uso de RAM. **34**

Figura 17 – Gráfico comparativo do uso de *Graphics Processing Unit* (GPU) durante treino de modelo MNIST. **35**

Figura 18 – Gráfico comparativo do uso de GPU. **35**
Figura 19 – Gráfico comparativo do uso de *Video Random Access Memory* (VRAM) durante treino de modelo MNIST. **37**
Figura 20 – Gráfico comparativo do uso de VRAM durante treino de modelo MNIST. **37**

LISTA DE TABELAS

Tabela 1 – Tabela com tempos de treino dos *datasets* MNIST e CIFAR-10 em Elixir e Python. **31**

LISTA DE ABREVIATURAS E SIGLAS

Siglas

CPU	*Central Process Unit*
GPU	*Graphics Processing Unit*
RAM	*Random Access Memory*
VRAM	*Video Random Access Memory*
Nx	*Numerical Elixir*
RGB	*Red Green Blue*
OTP	*Open Telecom Platform*
PF	Programação Funcional
POO	Programação Orientada a Objetos
CSV	*Comma-separated values*
CNN	*Convolutional Neural Network*
API	*Application Programming Interface*
XLA	*Accelerated Linear Algebra*
EXLA	*Elixir Accelerated Linear Algebra*
BEAM	*Bogdan Erlang Abstract Machine*

SUMÁRIO

1 INTRODUÇÃO 10

1.1 Objetivos 10

1.2 Organização do Documento 11

2 FUNDAMENTAÇÃO TEÓRICA 13

2.1 *Datasets* 13

2.1.1 Dataset MNIST 13

2.1.2 Dataset CIFAR-10 14

2.2 Redes Neurais Artificiais 15

2.2.1 Redes Neurais Convolucionais 16

2.3 Modelos Avaliados 16

2.3.1 Estrutura do modelo MNIST 16

2.3.2 Estrutura do modelo CIFAR-10 18

3 PROJETO 19

3.1 Elixir Image Recognizer 19

3.1.1 Scidata 20

3.1.2 Axon 21

3.1.3 EXLA 21

3.2 Python Image Recognizer 22

3.2.1 Keras 22

3.3 Unix Stats 23

4 EXPERIMENTO 25

4.1 Preparação 25

4.1.1 Preparação do ambiente para uso da placa de vídeo 25

4.2 Execução 27

4.3 Avaliação 28

5 ANÁLISE 31

5.1 Tempo de treino 31

5.2 Uso de CPU 31

5.3 Uso de RAM 32

5.4 Uso de GPU 34

5.5 Uso de VRAM 36
5.6 Comparação da experiência de desenvolvimento e aprendizagem 37
5.6.1 Documentação 38
5.6.2 Curva de aprendizado 38
6 CONCLUSÃO 40
6.1 Trabalhos Futuros 40
REFERÊNCIAS 42

1 INTRODUÇÃO

Com o anúncio da primeira versão da biblioteca *Numerical Elixir* (Nx) (VALIM, 2021) em fevereiro de 2021 para a linguagem de programação Elixir, tivemos grande interesse em explorar o potencial desta nova biblioteca, uma vez que esta permitiu que Elixir se tornasse viável para se trabalhar com redes neurais. Desta forma buscamos pesquisar e nos aprofundar no impacto desta biblioteca para o setor de redes neurais, além de buscar compreender seu impacto para a própria linguagem de programação Elixir e por fim comparar esta nova biblioteca com outra ferramenta popular deste mercado para se entender melhor os prós e contras de se trabalhar com redes neurais utilizado Elixir.

Neste sentido, além de Elixir, outras linguagens de programação vem atuando nos últimos anos para serem linguagens viáveis neste setor de processamento de dados de tensores n-dimensionais em busca de serem viáveis para se trabalhar com processamento de redes neurais. Um exemplo de linguagem que tem tido forte influência neste mercado tem sido a linguagem Python com sua biblioteca Numpy, possuindo desde 2006 a capacidade de se trabalhar com processamento de tensores (OLIPHANT, 2006). Atualmente, o mercado para o processamento de dados em massa, como a aprendizagem de máquinas, é viável se ser trabalhado em diversas linguagens como Julia, *Java*, C++, R, dentre outras, sendo liderado pela linguagem Python, que atualmente é a linguagem com mais adoção entre desenvolvedores iniciantes e experientes para este setor (DREAMCODE, 2022).

Neste cenário, em fevereiro de 2021, o ecossistema da linguagem Elixir recebeu novas ferramentas como a biblioteca Nx, um novo compilador para realizar operações em GPU o *Elixir Accelerated Linear Algebra* (EXLA), a capacidade de trabalhar com variáveis float16, entre outras melhorias. Estas novas características permitiram que a linguagem Elixir participe no mercado de redes neurais. Assim, este trabalho visa comparar as linguagens de programação Elixir e Python no contexto do processamento de tensores de grande volume, comparando a experiência ao se desenvolver em ambas linguagens, tempo de execução de treino, e a utilização de recursos da máquina como CPU, RAM, GPU, e VRAM ao longo do treino de modelos de redes neurais de reconhecimento de imagem semelhantes.

1.1 Objetivos

O objetivo geral do trabalho é implementar e comparar os resultados alcançados ao treinar modelos de redes neurais nas linguagens Python e Elixir, avaliando quesitos concretos como as diferenças de performance entre implementações nestas linguagens e buscando compreender os motivos dos resultados encontrados.

Além disto, busquei também comparar quesitos qualitativos como complexidade da preparação de ambiente, curva de aprendizado e, por fim, qualidade e quantidade da documentação disponível.

Para isso, foram realizadas as seguintes tarefas:

- Estudo dos conceitos de sistemas inteligentes com foco em redes neurais para reconhecimento de imagens;
- Estudo do impacto de diferentes linguagens de programação na implementação de redes neurais;
- Implementação de algoritmos de reconhecimento de imagem, criação de formas de comparação de *benchmarks* e implementação de uma ferramenta de coletas de dados para ajudar nos *benchmarks*;
- Implementação de um algoritmo de reconhecimento de imagens do dataset MNIST (PAPERSWITHCODE, 2022b) em Python e Elixir;
- Implementação de um algoritmo de reconhecimento de imagens do dataset CIFAR-10 (PAPERSWITHCODE, 2022a) em Python e Elixir;
- Análise do desempenho das implementações considerando as métricas de desempenho: uso de CPU, RAM, GPU, VRAM e tempo de execução;
- Análise do ambiente de desenvolvimento fornecido por ambas as linguagens, comparando complexidade da preparação de ambiente, curva de aprendizado e qualidade e quantidade da documentação disponível;
- Com base nos resultados encontrados, realizar análise da viabilidade da linguagem Elixir na área de redes neurais.

1.2 Organização do Documento

O presente trabalho está organizado em seis capítulos:

- O capítulo 1 contextualiza o projeto e estabelece os objetivos da pesquisa.
- O capítulo 2 contém os conceitos relacionados às tecnologias avaliadas e da análise de desempenho que será utilizada.
- O capítulo 3 define o projeto, explorando os projetos realizados nas linguagens Elixir e Python.
- O capítulo 4 descreve a realização do experimento que inclui os processos de preparação de ambiente, execução dos algoritmos e avaliação dos resultados.
- O capítulo 5 apresenta os resultados alcançados nos experimentos.

- O capítulo 6 apresenta as conclusões obtidas após a avaliação do resultado dos experimentos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica sobre tensores, redes neurais, conjuntos de dados, e sobre as bibliotecas utilizadas em Elixir e Python durante a criação dos sistemas desenvolvidos para este trabalho. Também discute a diferença entre os paradigmas de Programação Funcional (PF) e Programação Orientada a Objetos (POO) no contexto deste projeto e como eles podem ter impacto nos resultados encontrados.

2.1 *Datasets*

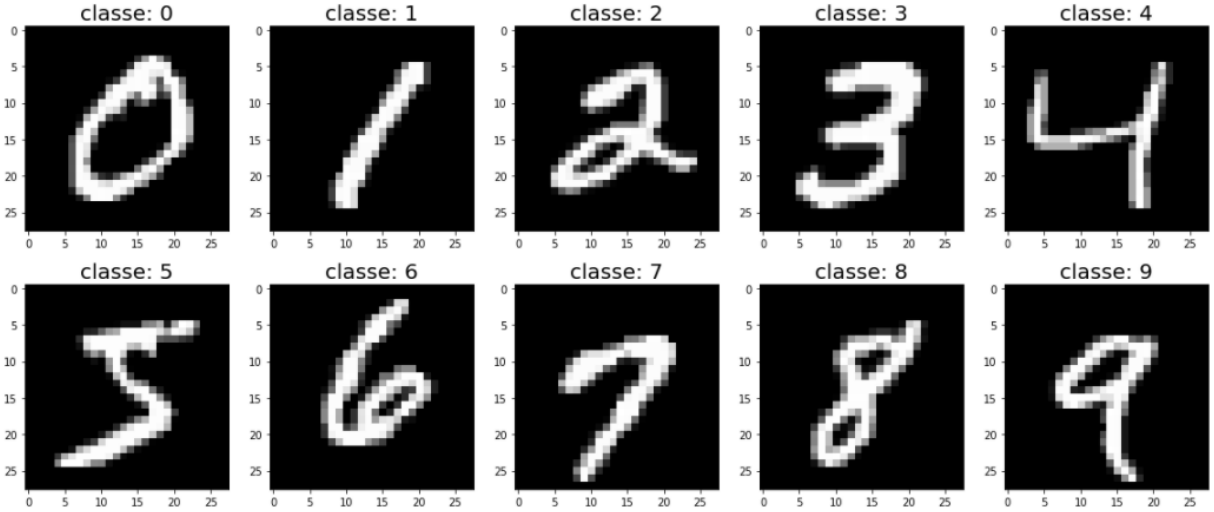
Um *dataset* é um grande conjunto de dados relacionados a um mesmo contexto (CLOUD, 2022). Neste projeto, trabalharei com dois conjuntos de dados: MNIST e CIFAR-10, que serão utilizados para treinar redes neurais convolucionais capazes de classificar imagens. A nomenclatura *Convolutional Neural Network* (CNN) é usada neste trabalho, quando necessário, para indicar redes redes neurais convolucionais.

Para este trabalho foram escolhidos 2 *datasets* simples e já muito bem conhecidos no contexto de redes neurais, para reduzir a complexidade do entendimento e modelagem das CNNs deste projeto, e assim poder focar os estudos realizados neste trabalho nos benchmarks calculados.

2.1.1 Dataset MNIST

MNIST é um conjunto rotulado de imagens de dígitos manuscritos contendo 70 mil entradas; 60 mil representam o conjunto de treino, e 10 mil representam o conjunto de teste. Cada imagem do *dataset* possui uma resolução de 28x28 pixels, totalizando 728 pixels por imagem; assim, é representada em código por uma matriz de 28 por 28. Cada valor contido nesta matriz varia de 0 a 255, 0 representa o preto, e 255 é branco (PAPERSWITHCODE, 2022b).

Figura 1 – Exemplo de imagens do conjunto MNIST geradas com a ferramenta *Matplotlib*

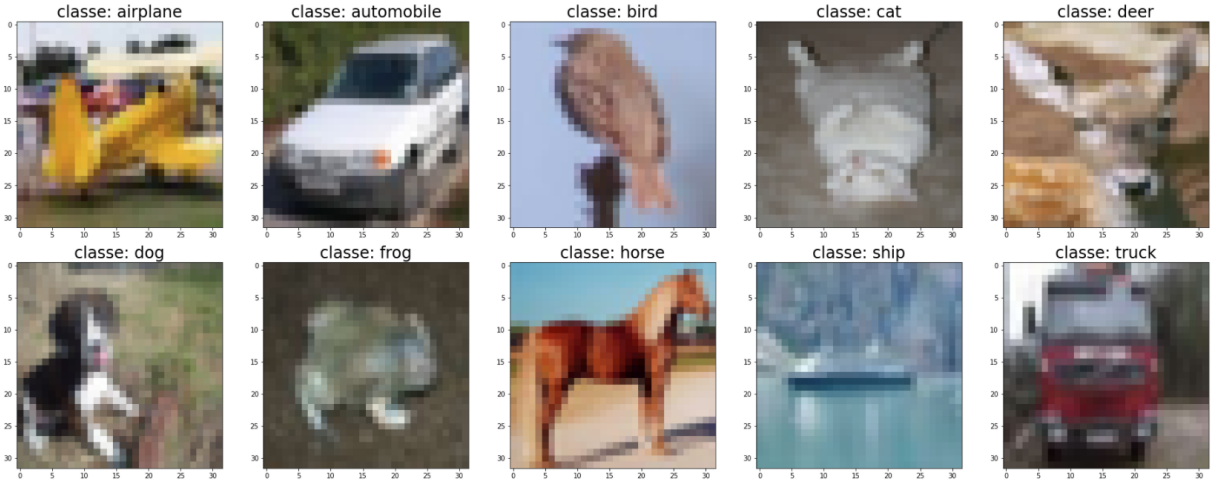


Fonte: Autoria própria.

2.1.2 Dataset CIFAR-10

O CIFAR-10, assim como MNIST, representa um grande conjunto de imagens; no entanto, no caso do CIFAR-10, o conjunto de imagens está relacionado a objetos e animais. As classes contidas no conjunto de dados CIFAR-10 são *airplance*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* e *truck*. Consiste em 66.000 imagens, divididas entre 90% para treino e 10% para testes. Cada imagem é representada por uma matriz de tamanho 32x32 contendo números inteiros, onde cada pixel (elemento da matriz) é representado por 3 números inteiros entre 0 e 255 seguindo o modelo de representação de imagens *Red Green Blue* (RGB) (WIKIPEDIA, 2022).

Figura 2 – Exemplo de imagens do conjunto CIFAR-10 geradas com a ferramenta *Matplotlib*

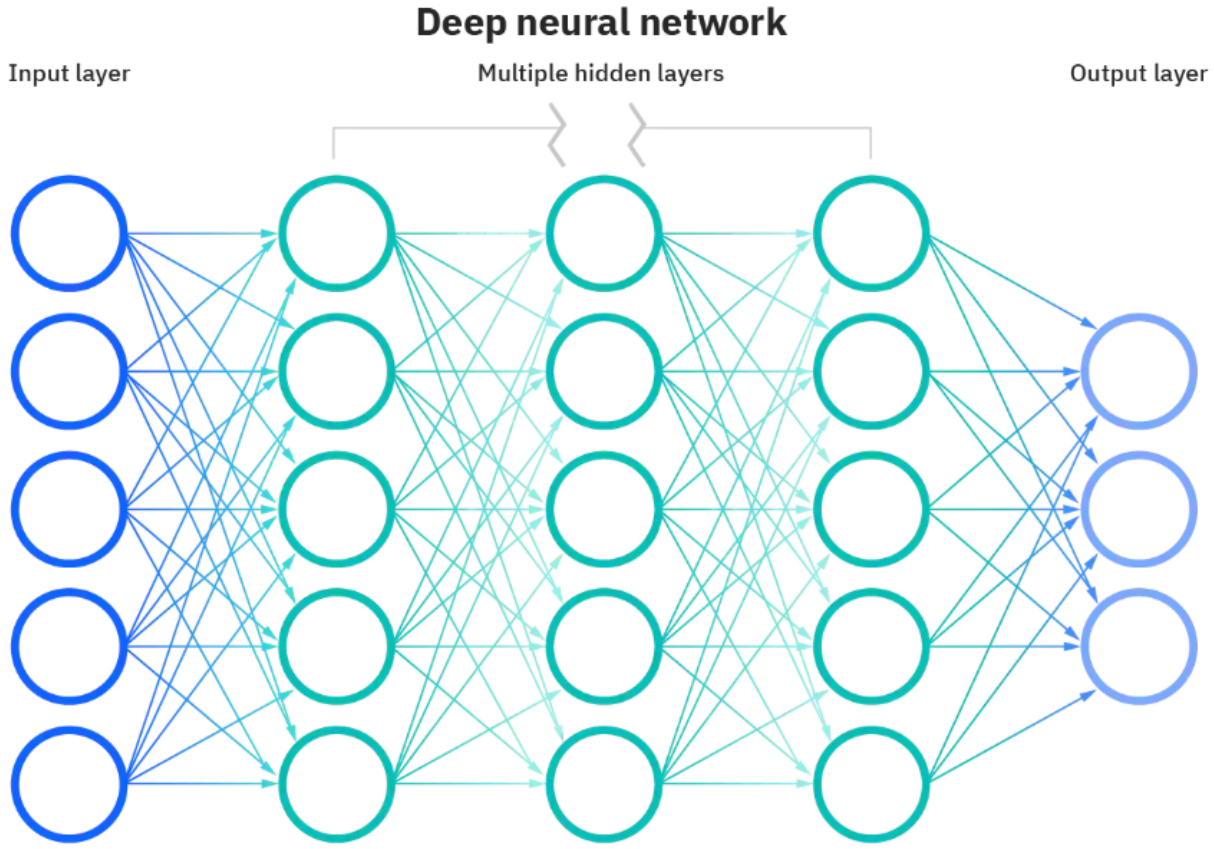


Fonte: Autoria própria.

2.2 Redes Neurais Artificiais

Redes Neurais Artificiais são sistemas inspirados em redes neurais biológicas e que se baseiam no funcionamento de cérebros, trazendo estes conceitos para o contexto de tomadas de decisões matematicamente intensivas. Tais redes neurais podem ser representadas por grafos direcionais multi-camadas, onde cada nó destes grafos é considerado um neurônio artificial de sua rede e é responsável por um operação matemática (O'SHEA; NASH, 2015). Esta representação é exemplificada na imagem 3.

Figura 3 – Representação visual de uma rede neural artificial com 3 camadas internas.



Fonte: https://www.ibm.com/br-pt/cloud/learn/neural-networks.

Cada modelo de uma rede neural artificial é planejado focado em um problema de contexto bem definido. Neste projeto, por exemplo, o reconhecimento de imagens dos datasets estudados. Um modelo de rede neural precisa ser treinado e refinado com exemplos de dados do dataset analisado com o intuito de se encontrar o conjunto de valores que cada neurônio deve conter para que quando se teste novos elementos utilizando uma rede neural treinada, que ela possa gerar uma distribuição probabilística, na qual o elemento desta distribuição com maior valor será considerado o elemento mais provável para ser catalogado, dentre os possíveis elementos do dataset (O'SHEA; NASH, 2015).

No contexto de reconhecimento de imagens, a posição do elemento de maior valor desta distribuição significa o elemento no qual a rede neural catalogou a imagem. Por exemplo, no dataset MNIST a posição do elemento dentro do array de distribuição probilística de maior valor representa o número que a rede neural catalogou como sendo o mais provável ao tentar identificar números do dataset MNIST.

2.2.1 Redes Neurais Convolucionais

Uma rede neural convolucional é um tipo específico de rede neural artificial onde cada nó é responsável por uma operação de convolução[1] sobre a entrada recebida. Um vetor de imagem é utilizado como entrada e processado através da rede para gerar uma pontuação final representada na camada de saída.

2.3 Modelos Avaliados

As CNNs utilizadas para comparar as métricas de desempenho entre Python e Elixir serão exemplificadas através da especificação de modelo neural da biblioteca Axon[2] de Elixir.

2.3.1 Estrutura do modelo MNIST

Esta CNN é uma rede simples e pequena. Por conta desta simplicidade, tentarei explicar detalhadamente esta rede neural, o que não será feito da mesma forma no detalhamento da rede neural do dataset CIFAR-10. Em Elixir, utilizando a biblioteca Axon, a rede neural utilizada para identificar imagens do dataset MNIST é representada em código da seguinte forma:

```
Axon.input({nil, 784}, "input")
  |> Axon.dense(128, activation: :relu)
  |> Axon.dropout(rate: 0.5)
  |> Axon.dense(10, activation: :softmax)
```

Listing 2.1 – Código Elixir que constrói modelo utilizado para identificar imagens do dataset MNIST

Esta rede neural possui uma entrada de tamanho 784 por conta de ser o número de pixels contidos em cada uma das imagens do dataset MNIST como detalhado em 2.1.1. Após

[1] Convolução é uma operação matemática dada pelo somatório elemento a elemento do produto entre duas funções, ao longo da região em que elas se sobrepõem (HAYKIN; VEEN, 2001). No contexto de processamento de imagens a convolução possui uma propriedade de se aplicar filtros em imagens como *blur*, *sharpening*, detecção de bordas, etc (PEDROSA, 2021).

[2] Disponível em https://github.com/elixir-nx/axon

a entrada na rede neural corretamente realizada, cada imagem do dataset é processada por 2 camadas densas.

A primeira camada densa possui tamanho de 128 unidades. Este valor poderia ser outros como 256, 512, 1024, dentre outros. Este valor afeta o tempo de convergência da rede neural para garantir melhor precisão conforma mais alto. Foi selecionado o valor 128 para seguir o modelo exemplo disponibilizado pela equipe do projeto Axon[3]. Esta camada possui uma função de ativação denominada Relu (*Rectified Linear Unit*), a qual mantém o valor de bits com valor numérico positivo e zera o valor de bits com valor negativo (BANERJEE, 2020).

A segunda camada densa possui tamanho 10, uma vez que é a ultima camada da rede neural. Este tamanho representa o número de diferentes tipos de imagens que a rede neural deve classificar, onde cada um destes 10 elementos recebem um peso atrelado a si. Este peso será utilizado para identificar o tipo de elemento no qual a rede neural catalogou a imagem de entrada. Por exemplo, dada saída fictícia:

```
[1, 100, 5, 5, 5, 0, 0, 40, 0, 0]
```

Esta saída representaria que a imagem de entrada tem uma maior probabilidade de ser uma imagem do número 1, uma vez que a segunda posição da lista de saída tem o maior peso dentre todos elementos. Sobretudo, para se facilitar esta análise, é utilizada a função de ativação *softmax* (BANERJEE, 2020), para gerar uma saída na forma de uma distribuição probabilística, onde a soma dos elementos da camada de saída é 1, através de uma média ponderada pelo valor de cada elemento. Sendo assim, para o exemplo acima, a saída com a utilização do *softmax* seria:

```
[0.006, 0.641, 0.032, 0.032, 0.032, 0, 0, 0.256, 0, 0]
```

Concluímos então que a rede neural teria considerado a imagem de entrada como a imagem do dígito 1, uma vez que é o elemento de maior valor desta lista de elementos de saída, com 64,1% de probabilidade de ser o elemento 1, o que ainda não é uma alta probabilidade, porém é a maior dentre todos elementos da saída.

Por fim, entre as camadas densas, existe uma camada de *dropout*. Esta camada é importante para introduzir um fator de aleatoriedade na rede neural, removendo entre as 2 camadas descritas 0,5% dos elementos processados. Esta camada é importante uma vez que redes neurais, assim como outros tipos de sistemas inteligentes, possuem característica de convergirem cedo demais para um resultado final, e assim acabarem ficando presas numa solução não necessariamente ótima. Este problema é conhecido como *overfitting* (EDUCATION, 2021).

[3] Disponível em https://github.com/elixir-nx/axon/blob/main/examples/vision/mnist.exs

2.3.2 Estrutura do modelo CIFAR-10

Esta CNN é uma rede mais complexa que a rede modelada para o *dataset* MNIST. O modelo descrito abaixo é uma forma de se definir este modelo utilizando a biblioteca Axon[4] de Elixir:

```
Axon.input({nil, 3, 32, 32}, "input")
  |> Axon.conv(32, kernel_size: {3, 3}, activation: :relu)
  |> Axon.batch_norm()
  |> Axon.max_pool(kernel_size: {2, 2})
  |> Axon.conv(64, kernel_size: {3, 3}, activation: :relu)
  |> Axon.batch_norm()
  |> Axon.max_pool(kernel_size: {2, 2})
  |> Axon.flatten()
  |> Axon.dense(64, activation: :relu)
  |> Axon.dropout(rate: 0.5)
  |> Axon.dense(10, activation: :softmax)
```

Listing 2.2 – Código Elixir que constrói modelo utilizado para identificar imagens do dataset CIFAR-10

Este modelo começa de forma similar, com uma entrada de formato 3x32x32, por conta do formato das imagens deste *dataset*, como detalhado em 2.1.2, e finaliza de forma similar, com as mesmas 2 camadas densas e a camada de *dropout* no final do modelo.

Sobretudo, esta rede possui tipos novos de funções sendo utilizadas na definição de seu modelo, como: "conv" (convolução), "batch_norm", "max_pool", "flatten"e "kernel_size". O funcionamento aprofundado destas funções e um maior detalhamento da relação delas com o efeito na rede neural modelada pode ser encontrado em Doon, Rawat e Gautam (2018).

No próximo capítulo são explicados os projetos criados para este trabalho e as tecnologias empregadas para o desenvolvimento dos *benchmarks*.

[4] Modelo exemplo disponível em https://github.com/elixir-nx/axon/blob/main/examples/vision/cifar10.exs

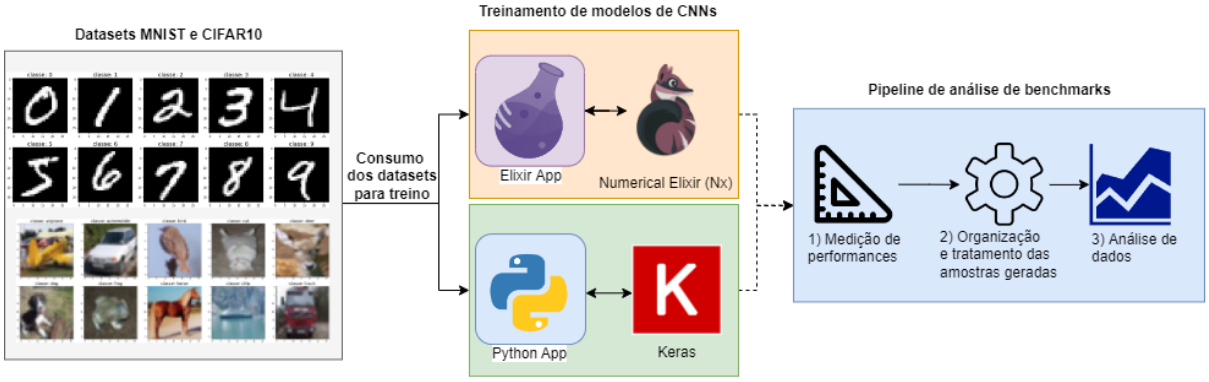
3 PROJETO

Este capítulo apresentará os três sistemas criados para esta monografia:

- Um sistema Elixir denominado **ElixirImageRecognizer**, capaz de modelar e treinar CNNs capazes de classificar imagens dos *datasets* MNIST e CIFAR-10;
- Um sistema Python denominado **PythonImageRecognizer**, capaz de modelar e treinar CNNs capazes de classificar imagens dos *datasets* MNIST e CIFAR-10;
- Um sistema Elixir denominado **UnixStats**, capaz de realizar *benchmark* dos treinos realizados em Python e Elixir.

A figura 4 é uma representação visual de como os sistemas interagem e como juntos compõem o fluxo de treino e análise necessário para os estudos realizados neste trabalho. Além disto, este fluxo será melhor explicado no capítulo 4.

Figura 4 – Diagrama de alto nível exemplificando a interação entre os três sistemas



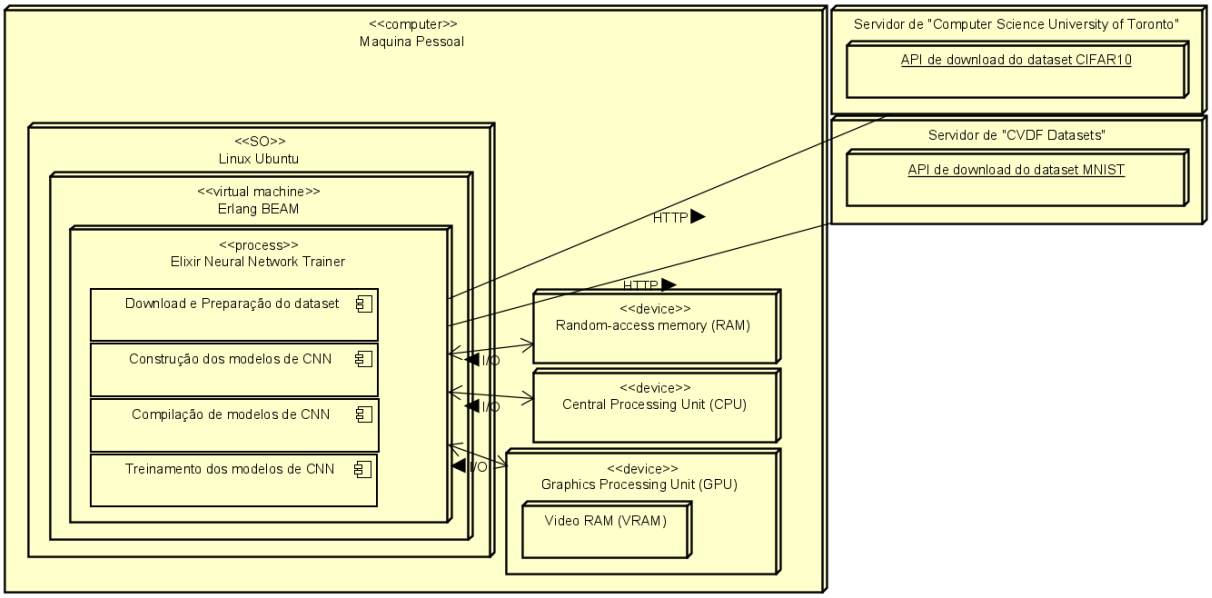
Fonte: Autoria própria.

3.1 Elixir Image Recognizer

O sistema Elixir desenvolvido para o treinamento de CNNs foi projetado com Elixir **1.12.3** e Erlang/OTP **24.0.6** e está disponível publicamente na plataforma Github[1]. Este se utiliza principalmente três bibliotecas do ecossistema Nx[2]: Scidata, Axon, Exla.

[1] Disponível em: https://github.com/sallaumen/elixir_neural_network_labs
[2] Disponível em: https://github.com/elixir-nx

Figura 5 – Diagrama de Implantação do sistema Elixir de treinamento de redes neurais.



3.1.1 Scidata

De acordo com a sua documentação[3], esta biblioteca é responsável pelo fornecimento de *datasets* de uma forma simples para projetos Elixir. Todos os *datasets* fornecidos pela biblioteca não são de autoria própria, cada *dataset* tem no início de seu módulo uma constante denominada “@base_url” contendo a URL para download do dataset. Um ponto forte da biblioteca é sua simplicidade para se baixar estes *datasets* e processá-los como for necessário para então utilizar estes dados para o treino de modelos de redes neurais. Além disto, todos os 12 *datasets* disponibilizados no momento da escrita desta monografia possuem grupo de dados de treino e de teste, para auxiliar no uso conjunto com a biblioteca Axon de Elixir. Atualmente, o Scidata é capaz de fornecer dados de 12 tipos diferentes de *datasets*[4]:

- Caltech101
- CIFAR10
- CIFAR100
- FashinMNIST
- IMDB Reviews
- Iris
- Kuzushiji MNIST

[3] Disponível em: https://github.com/elixir-nx/scidata
[4] Disponível em: https://github.com/elixir-nx/scidata/tree/master/lib/scidata

- MNIST
- SQUAD
- Wine
- Yelp Full Reviews
- Yelp Polarity Reviews

Dentre os *datasets* disponíveis, nesta monografia são utilizados os *datasets* MNIST e CIFAR10.

3.1.2 Axon

De acordo com a sua documentação[5], esta biblioteca é uma *Application Programming Interface* (API) para a biblioteca Nx, simplificando a utilização do Nx para se descrever modelos de redes neurais, permitir o treino das redes neurais descritas e o teste da redes neurais treinadas. Para dar visibilidade do quanto o Axon foi importante para esta monografia, a princípio foi implementada uma rede neural para identificar imagens do dataset MNIST utilizando a biblioteca Nx diretamente, e para este simples *dataset* foram utilizadas 204 linhas de código, enquanto com a biblioteca Axon foram necessárias 102 linhas, ou seja, 50% a menos de código necessário ao utilizar o Axon.

Além disto, a capacidade da biblioteca em se definir redes neurais na forma de estruturas de dados é algo necessário no setor de redes neurais uma vez que facilita a visualização do modelo desenvolvido e agiliza muito o processo de desenvolvimento destes modelos, onde normalmente acabam sendo necessários ajustes constantes até se encontrar um modelo considerado adequado e assim definir como o modelo final para se trabalhar com um dataset. Nesta monografia este processo de refinamento e desenvolvimento do modelo não foi necessário, uma vez que os *datasets* estudados já possuíam modelos muito bem conhecidos e assim foram utilizados os próprios modelos exemplos da biblioteca Axon, com pequenos ajustes para ficar igual ao modelo aplicado em Python.

3.1.3 EXLA

O projeto EXLA[6] é um compilador projetado para se trabalhar com a biblioteca Nx que permite a utilização do compilador *Accelerated Linear Algebra* (XLA) da Google, atuando tanto como back-end para tensores Nx como um compilador para funções **Nx.Defn**[7] (MORIARITY,

[5] Disponível em: https://github.com/elixir-nx/axon.

[6] Disponível em: https://github.com/elixir-nx/nx/tree/main/exla

[7] **Nx.Defn** é uma macro de Elixir introduzida pelo Nx que permite funções serem declaradas pelo termo **defp**

2022). Além disto, o EXLA é também a biblioteca responsável por permitir que códigos escritos com o **Nx.Defn** sejam compilados e executados na GPU.

Este compilador é de extrema importância quando se busca performance no treinamento de redes neurais. Para medida de comparação, utilizando o hardware descrito em 4.1 para se treinar um modelo para identificar imagens do *dataset* MNIST, com a utilização do compilador foram necessários 20,4 segundos, enquanto sem ele foram necessárias aproximadamente 7 horas e 30 minutos para executar o mesmo treino, uma melhora de 1350 vezes em performance.

3.2 Python Image Recognizer

O sistema Python Image Recognizer desenvolvido para o treinamento de CNNs foi projetado com Python **3.8.0** e está disponível publicamente na plataforma Github[8]. Este se utiliza principalmente de uma biblioteca do ecossistema de processamento de dados da linguagem: Keras.

3.2.1 Keras

A biblioteca Keras[9] possui a funcionalidade, assim como o Axon, de ser uma API para o conjunto de funcionalidades do TensorFlow[10], permitindo a criação, treinamento e teste de modelos de redes neurais artificiais, dentre outras funcionalidades.

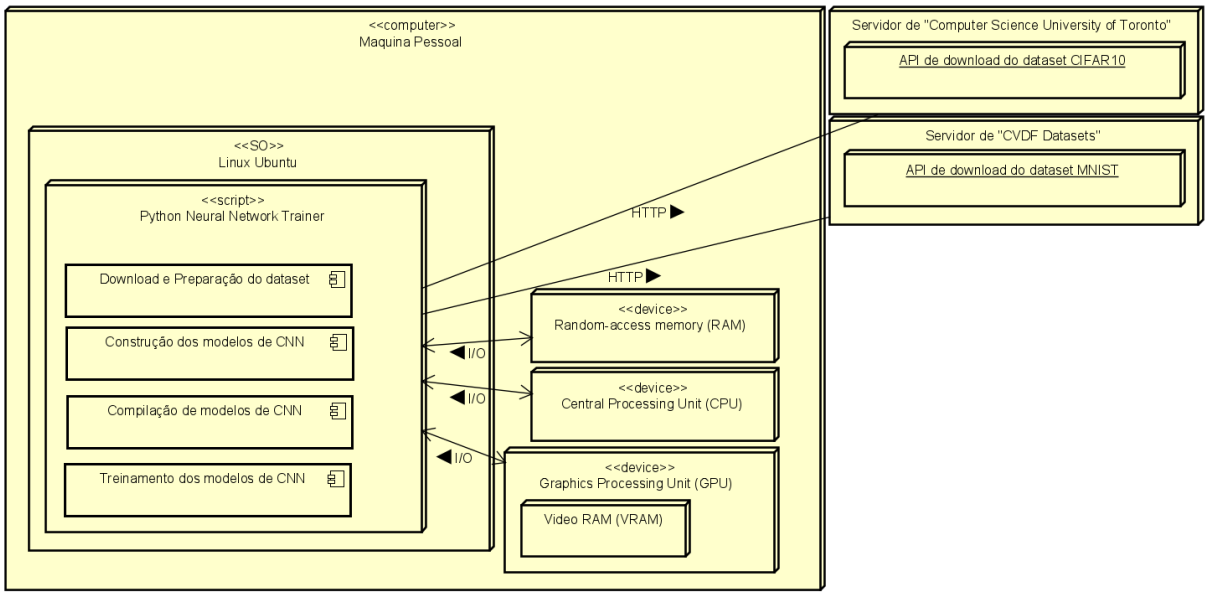
Esta biblioteca é o ponto central do projeto desenvolvido em Python. Com ela fomos capazes de carregar os dados dos *datasets* MNIST e CIFAR-10, modelar suas redes neurais e treiná-las, concluindo assim todos passos necessários para este trabalho em Python.

[8] Disponível em: https://github.com/sallaumen/python_neural_network_labs.

[9] Disponível em: https://keras.io/.

[10] TensorFlow é uma das principais frentes em criação de modelos de aprendizado de máquina, utilizado por diversas bibliotecas, inclusive pelo Nx. Disponível em: https://www.tensorflow.org/.

Figura 6 – Diagrama de Implantação do sistema Python de treinamento de redes neurais.



3.3 Unix Stats

UnixStats é outro sistema em Elixir que desenvolvi para este estudo. É um software com a funcionalidade de medir o uso de CPU, RAM, GPU, e RAM gráfica da máquina executora. Para o executar, o utilizador necessita fornecer três entradas:

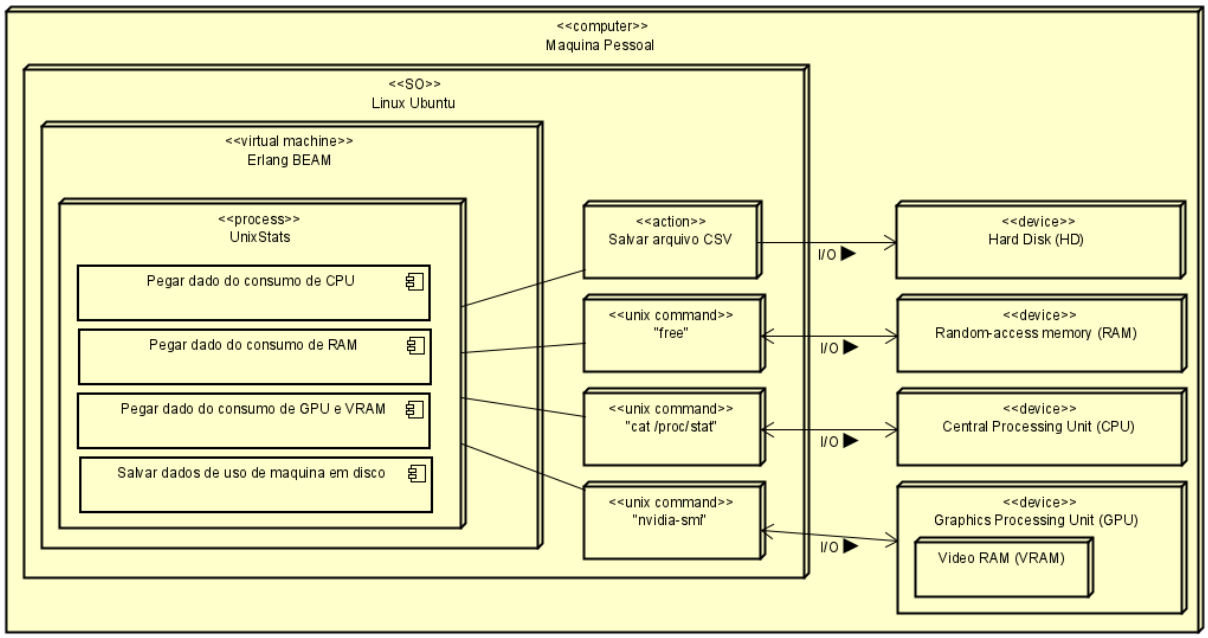
- A entrada “measured process” representa o nome do processo que o UnixStats irá medir, onde para o sistema Elixir (ElixirImageRecognizer) o processo foi “beam.smp” e para o sistema Python (PythonImageRecognizer) foi “python3.8”;
- A entrada “visualization type” pode ser “:pretty” ou “:csv”. O formato escolhido define como os dados medidos serão apresentados ao utilizador do software. A figura 7 exemplifica a forma como o sistema com o “*visualization type*” “:pretty” imprime no terminal;
- A entrada “monitoring time” representa quanto tempo, em segundos, o sistema deve seguir fazendo o *benchmark*.

Figura 7 – Exemplo de como o sistema UnixStats com o *visualization type* ":pretty" imprime no terminal.

```
10:08:21-tavano:~/git/unix_stats (main) $ iex -S mix
Erlang/OTP 24 [erts-12.0.4] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [jit]

Interactive Elixir (1.12.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> UnixStats.measure(15, :pretty, "beam.smp")
Elapsed Time: 0,0    CPU: 4,31735%    RAM: 9,94607%    GPU: 3%    Graphic RAM: 1,6665%
Elapsed Time: 0,1    CPU: 4,32227%    RAM: 9,87509%    GPU: 1%    Graphic RAM: 3,4457%
Elapsed Time: 0,2    CPU: 4,32606%    RAM: 10,3384%    GPU: 1%    Graphic RAM: 3,8466%
Elapsed Time: 0,3    CPU: 4,33013%    RAM: 10,9091%    GPU: 6%    Graphic RAM: 4,2977%
Elapsed Time: 0,4    CPU: 4,33366%    RAM: 11,447%    GPU: 6%    Graphic RAM: 4,7989%
Elapsed Time: 0,5    CPU: 4,33773%    RAM: 11,9568%    GPU: 6%    Graphic RAM: 5,1497%
Elapsed Time: 0,6    CPU: 4,34125%    RAM: 12,4623%    GPU: 6%    Graphic RAM: 5,6259%
```

Figura 8 – Diagrama de Implantação do sistema UnixStats.



4 EXPERIMENTO

Este capítulo descreve a realização do experimento que inclui as etapas de preparação, execução e avaliação.

4.1 Preparação

O ambiente utilizado para se executar o algoritmo desenvolvido em Elixir e em Python foi o mesmo, um computador pessoal utilizando Linux Ubuntu sem interface gráfica como sistema operacional, utilizando os *drivers* NVIDIA descritos em 4.1.1.

Como hardware foram utilizados:

- Placa mãe: ASUSTeK P8H61-M LX3 Plus R2.0
- Processador: Intel Core i5 3570 3.40GHz
- Placa de vídeo: NVIDIA GeForce RTX 3060 Ti
- Memória RAM: 16GB DDR3 1333MHz
- Potência da fonte de alimentação: 850W

4.1.1 Preparação do ambiente para uso da placa de vídeo

Para se processar redes neurais de forma eficiente é necessária uma alta capacidade de processamento de cálculos matemáticos e, desta forma, a utilização da GPU é indispensável, uma vez que este tipo de hardware, proporcionalmente, possui um maior número de transistores dedicados à realização de operações matemáticas com números flutuantes (*floats*) do que uma CPU (DSOUZA, 2020).

Sobretudo, fazer ambos os softwares desenvolvidos funcionarem na GPU em um ambiente único, para garantir uma comparação de *benchmark* justa, foi um processo muito lento, envolvendo muitas iterações falhas no processo de aprendizado até finalmente alcançar o sucesso deste *setup*.

Primeiramente o foco foi fazer cada um dos projetos desenvolvidos funcionarem em sistemas operacionais distintos instalados na mesma máquina. Assim, o primeiro passo realizado foi atuar em como executar o treinamento em Python com o uso da GPU. Python, por ser uma linguagem muito utilizada por desenvolvedores buscando estudar redes neurais, acabou possuindo muitos guias online[1] focados exatamente em ajudar com este *setup*, de forma bem

[1] O tutorial utilizado está disponível em: https://illya13.github.io/RL/tutorial/2020/04/26/installing-tensorflow-on-ubuntu-20.html.

didática, o que permitiu que em pouco tempo, já fosse possível executar treinamentos de CNNs utuilizando a GPU da máquina.

Por outro lado, executar o treinamento com Elixir não foi fácil. O estudo para esta monografia começou em abril de 2021, dois meses apenas após o anúncio da versão inicial da biblioteca e, por conta disto, ainda não haviam documentações oficiais e nem tutoriais ensinando como preparar a máquina para utilizar a biblioteca na GPU. Existiam apenas os arquivos "README" dos projetos disponíveis no Github, que citavam vagamente como executar treinamentos na GPU, porém ainda de forma superficial e que com o que tinha disponível, e mesmo depois alguns meses, este setup ainda não havia sido concluído.

Por volta do terceiro mês atuando neste ponto, e com muito já aprendido sobre este processo, criamos um tópico no *Elixir Forum*[2], onde consegui ajuda de dois desenvolvedores que estavam trabalhando no Nx: José Valim, criador da linguagem Elixir, e Sean Moriarity, autor do livro "Genetic Algorithms in Elixir: Solve Problems Using Evolution". Nesta *issue* consegui o suporte necessário para executar o treinamento das CNNs desenvolvidas para esta monografia utilizando corretamente a GPU. O problema que estava ocorrendo eram algumas variáveis de ambiente que precisavam ser configuradas no sistema operacional, para que a biblioteca EXLA soubesse a versão dos *drivers* NVIDIA utilizados e pudesse assim carregar os devidos pacotes pré-compilados para permitir a comunicação do projeto em Elixir com a GPU. Assim, as variáveis de ambiente configuradas foram:

- **XLA_BUILD**: false
- **XLA_TARGET**: cuda
- **EXLA_TARGET**: cuda
- **EXLA_FLAGS**: –config=cuda
- **TF_CUDA_VERSION**: '11.2'

Sobretudo, ainda havia uma desafio final que era fazer um só sistema operacional com um *setup* e um conjunto de *drivers* único ser capaz de treinar as CNNs necessárias em Python e Elixir. Para isso foram analisados os *drivers* da NVIDIA que cada uma das linguagens fornecia suporte oficial, e então foi refeito o *setup* do sistema operacional utilizando o seguinte conjunto de *drivers*:

- Driver de vídeo NVIDIA: 470.103.01
- NVIDIA CUDA 11.4
- NVIDIA cuDNN 11.2

[2] Disponível em https://elixirforum.com/t/compiling-xla-with-cuda-11-2-support-needed/42528.

Com isso, foi concluída a preparação do ambiente local, permitindo a execução de treino de redes neurais utilizando GPU em Elixir e Python.

4.2 Execução

Estes foram os passos realizados para a execução dos treinamentos, metrificações e análises realizados ao longo desta monografia:

1. Foi preparado o computador com os hardwares descritos em 4.1;
2. Um sistema operacional Ubuntu foi inicializado no hardware, sem utilizar interface gráfica, apenas com terminal;
3. Usando o software tmux[3], o terminal foi dividido em 2 lados independentes;
4. No terminal esquerdo foi inicializado o *software* UnixStats;
5. O terminal direito foi separado para executar os treinamentos da CNN em Elixir e Python com ambos os *datasets* analisados;
6. Foram treinadas 20 redes neurais, 5 para cada combinação entre linguagem e *dataset* estudada neste trabalho e, durante todas as execuções, foi utilizado o UnixStats no terminal esquerdo para medir os *benchmarks* de desempenho das execuções;
7. Para cada execução o UnixStats gerou um arquivo em formato *Comma-separated values* (CSV) com os dados medidos ao longo de 70 segundos;
8. Todos os CSVs foram importados no software Microsoft Excel, para permitir a realização das análises necessárias destes dados;
9. Uma vez que cada execução medida teve um tempo inicial variável devido ao tempo de download do conjunto de dados, foi necessário a realização de uma normalização dos dados para minimizar o efeito do tempo do download nas medições. Por conta deste ajuste, foram utilizadas medidas de 70 segundos no UnixStats, para ter uma margem de até 10 segundos para carregar os dados e assim ter dados normalizados durante 60 segundos de medição.
10. Para cada grupo de cinco amostras, foi criada uma sexta amostra com a média de uso de CPU, RAM, GPU e VRAM;
11. Finalmente, com as amostras dos experimentos realizados, foram criados gráficos para permitir a comparação do desempenho dos recursos analisados, que serão utilizados na seção 4.3 e capítulo 5.

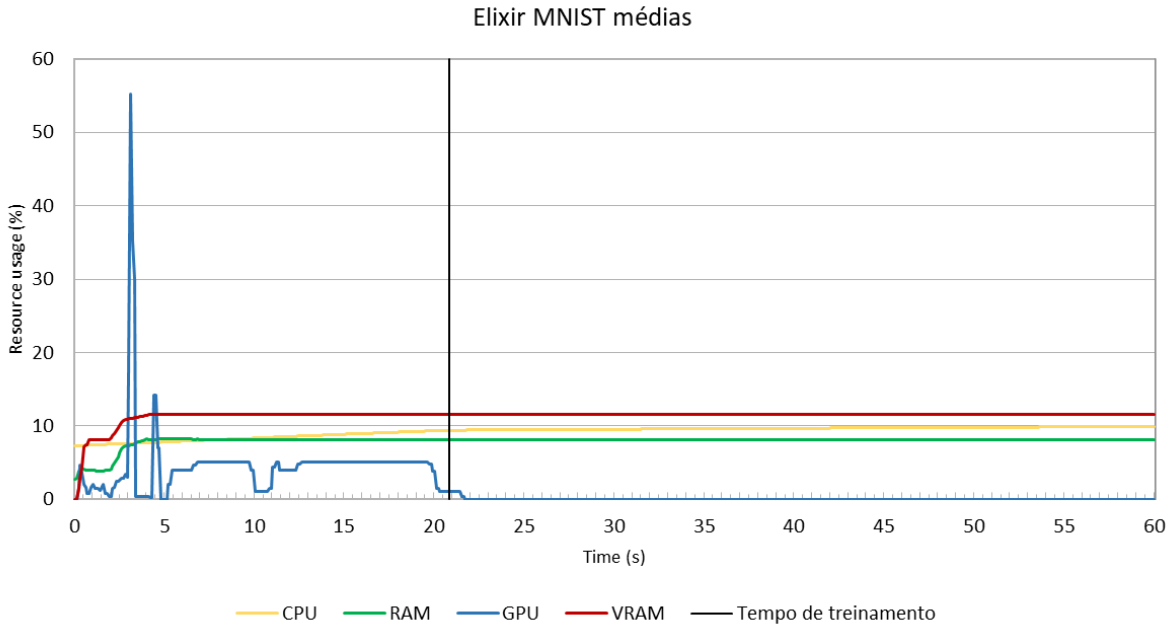
[3] Disponível em: https://github.com/tmux/tmux/wiki.

4.3 Avaliação

Abaixo estão presentes os gráficos de médias de desempenho, resultado dos treinos dos modelos das CNNs desenvolvidas para catalogar imagens dos *dataset* MNIST e CIFAR-10, nas linguagens Elixir e Python, com dados medidos utilizando a aplicação UnixStats. A comparação e análise destes dados será feita no capítulo 5.

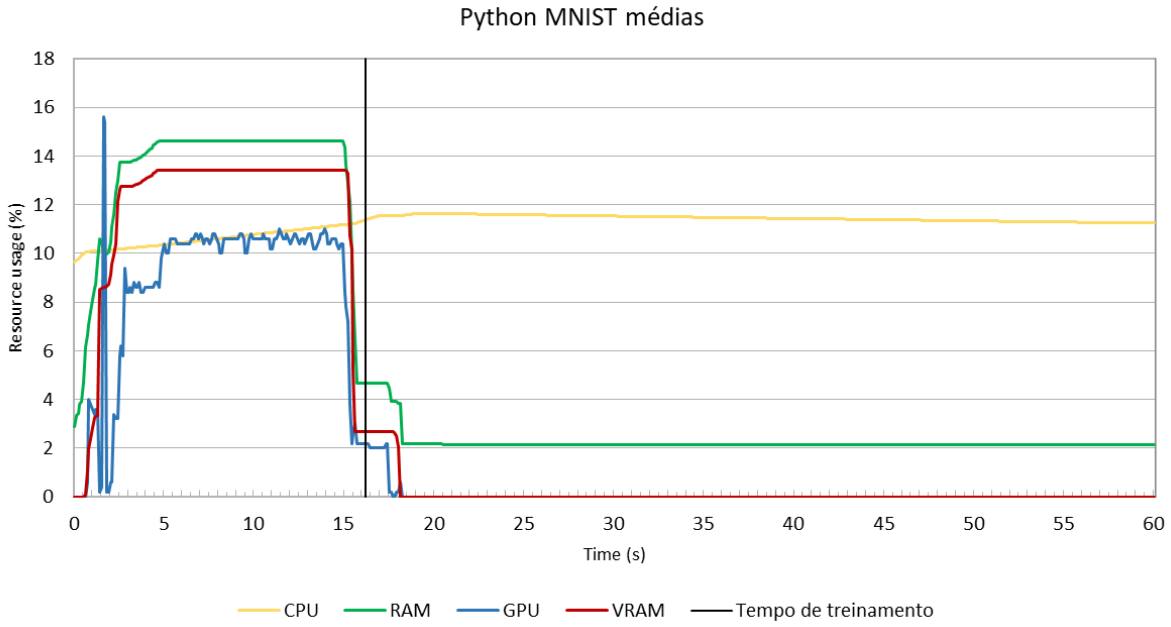
Gráficos de médias de desempenho do treinamento do modelo criado para identificar imagens do *dataset* MNIST.

Figura 9 – Gráfico de uso médio de recurso por tempo, medido durante a execução em Elixir.



Fonte: Autoria própria.

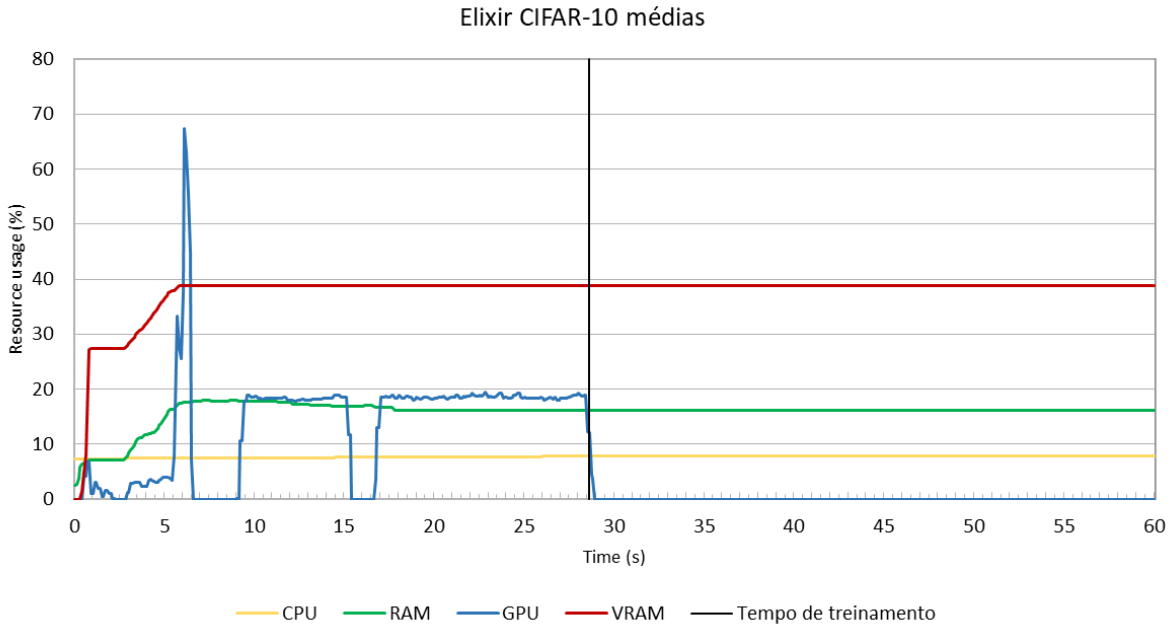
Figura 10 – Gráfico de uso médio de recurso por tempo, medido durante a execução em Python.



Fonte: Autoria própria.

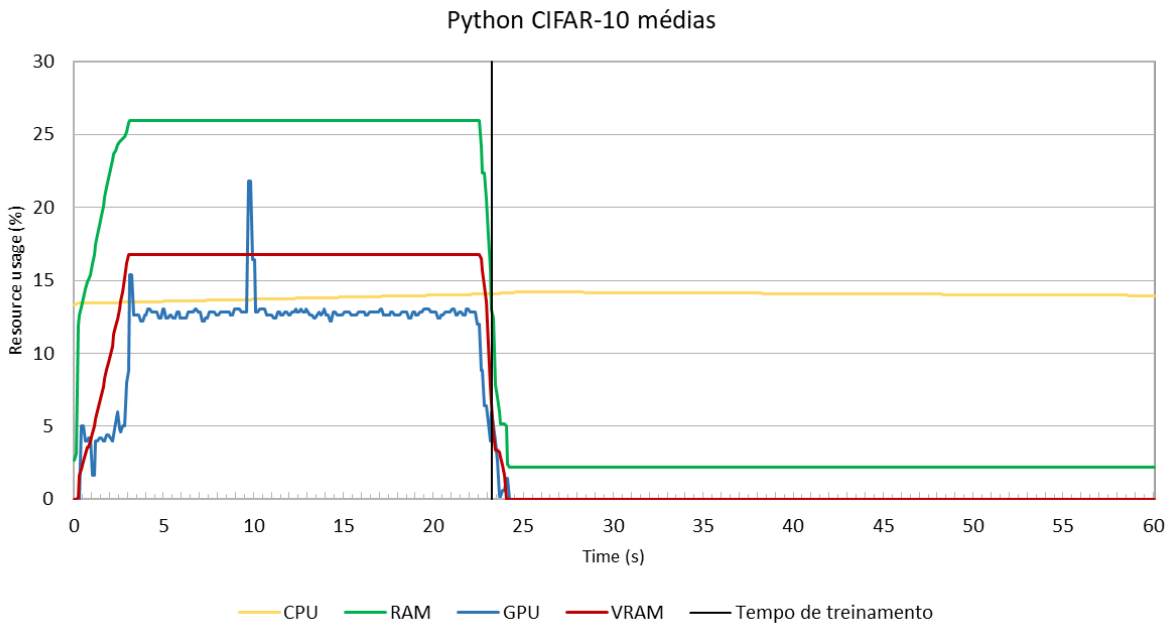
Gráficos de médias de desempenho do treinamento do modelo criado para identificar imagens do *dataset* CIFAR-10.

Figura 11 – Gráfico de médio uso de recurso por tempo, medido durante a execução em Elixir.



Fonte: Autoria própria.

Figura 12 – Gráfico de médio uso de recurso por tempo, medido durante a execução em Python.



Fonte: Autoria própria.

5 ANÁLISE

Neste capítulo apresentaremos os resultados encontrados ao se comparar o desempenho do treino de redes neurais para os *datasets* MNIST e CIFAR-10 nas linguagens Elixir e Python. Os recursos que serão analisados neste capítulo são: tempo de treino, uso de CPU, RAM, GPU e VRAM.

5.1 Tempo de treino

Nesta seção analisaremos o tempo utilizado para o treinamento das CNNs avaliadas neste projeto. A tabela 1 é composta por três colunas. A coluna "*Dataset*" contém o nome do conjunto de dados relativo à amostra da linha, podendo conter os valores "MNIST" e "CIFAR-10". A coluna "Linguagem" se refere a linguagem de programação utilizada na amostra da linha, podendo conter os valores "Elixir" e "Python". Por fim, a coluna "Tempo de treino" possui o tempo total utilizado pela "Linguagem" para treinar um modelo de rede neural convolucional capaz de catalogar imagens da coluna "*Dataset*".

A partir da tabela 1 é possível observar que para o *dataset* MNIST o tempo de treino em Elixir foi 34,21% maior do que o tempo de treino do mesmo *dataset* em Python. Também é possível observar que para o *dataset* CIFAR-10 o tempo de treino em Elixir foi 23,91% maior do que o tempo de treino do mesmo *dataset* em Python.

Tabela 1 – Tabela com tempos de treino dos *datasets* MNIST e CIFAR-10 em Elixir e Python.

Dataset	Linguagem	Tempo de treino total (s)
MNIST	Elixir	20,4
MNIST	Python	15,2
CIFAR-10	Elixir	28,5
CIFAR-10	Python	23,0

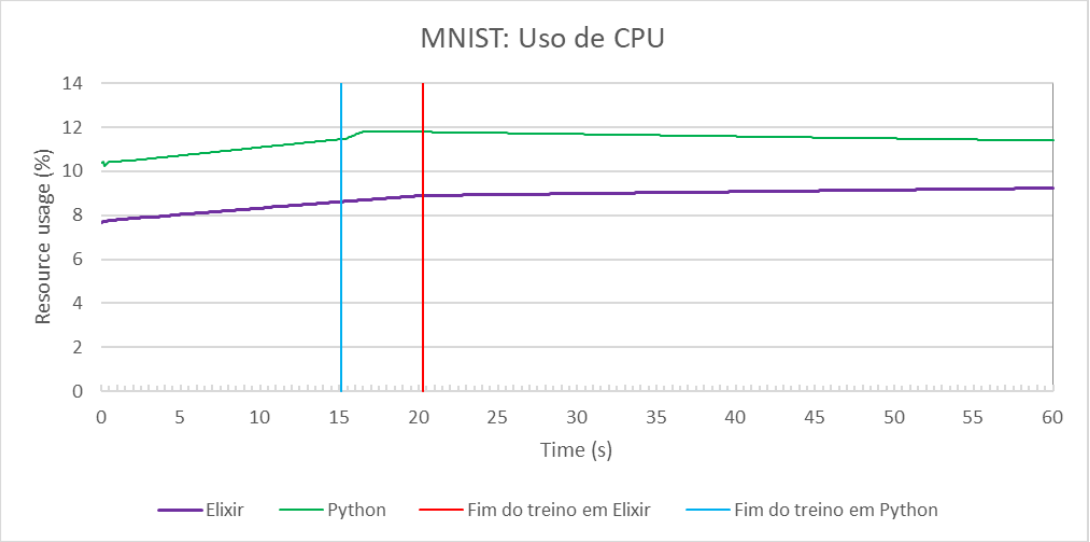
5.2 Uso de CPU

Nesta seção analisaremos o uso de CPU para o treinamento das CNNs avaliadas neste projeto e, para tanto, serão analisados os gráficos 13 e 14.

Ao longo do treino utilizando a linguagem Elixir, a variação do uso de CPU foi de 1,65% durante o treino MNIST e 0,76% durante o treino CIFAR-10. Em Python, a variação do uso de CPU foi de 1,59% durante o treino MNIST e 0,91% durante o treino CIFAR-10.

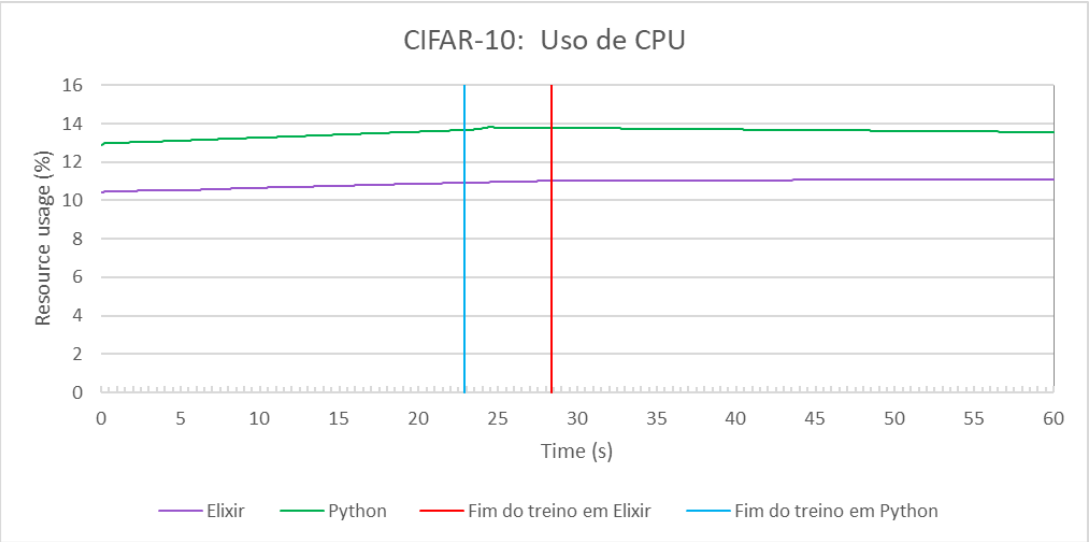
Com isto, observamos que a CPU é pouco utilizada durante o treino em ambas linguagens, uma vez que possui baixa variação, e que em ambas linguagens a variação do uso é muito semelhante durante ambos os treinos realizados.

Figura 13 – Gráfico comparativo do uso de CPU durante treino de modelo MNIST.



Fonte: Autoria própria.

Figura 14 – Gráfico comparativo do uso de CPU.



Fonte: Autoria própria.

5.3 Uso de RAM

Nesta seção analisaremos o uso de RAM para o treinamento das CNNs avaliadas neste projeto e, para tanto, serão analisados os gráficos 15 e 16.

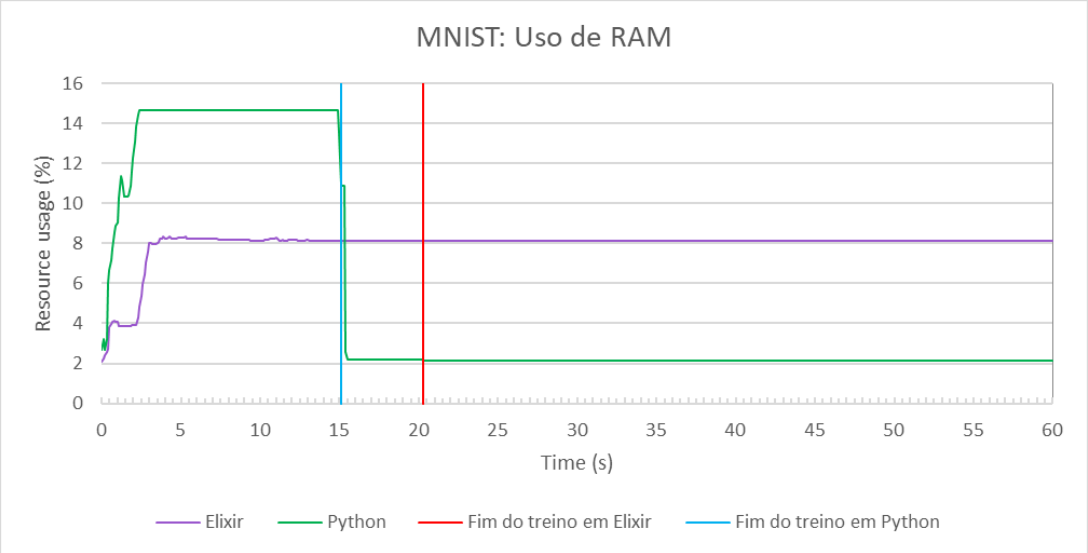
Ao longo do treino utilizando a linguagem Elixir, a variação do uso de RAM foi de 11,92% durante o treino MNIST e 18,71% durante o treino CIFAR-10. Em Python, a variação do uso de RAM foi de 12,52% durante o treino MNIST e 23,95% durante o treino CIFAR-10.

Com isto, é possível notar que a RAM é um recurso importante durante o processo dos treinos de redes neurais em ambas linguagens, o que era esperado, uma vez os conjuntos

de imagens de ambos *datasets* analisados, ficam guardados em memória durante a execução destes treinos. Além disto, também é possível reparar que em Elixir existe um menor uso da RAM do que em Python, nos cenários avaliados, utilizando em média de 5% a menos deste recurso para o treino da rede neural do *dataset* MNIST e 28% a menos deste recurso para o *dataset* CIFAR-10, e uma vez que o tamanho do dado dos *datasets* em ambas linguagens é o mesmo, assim, observando que Elixir aparenta possuir um melhor gerenciamento de RAM nos cenários analisados.

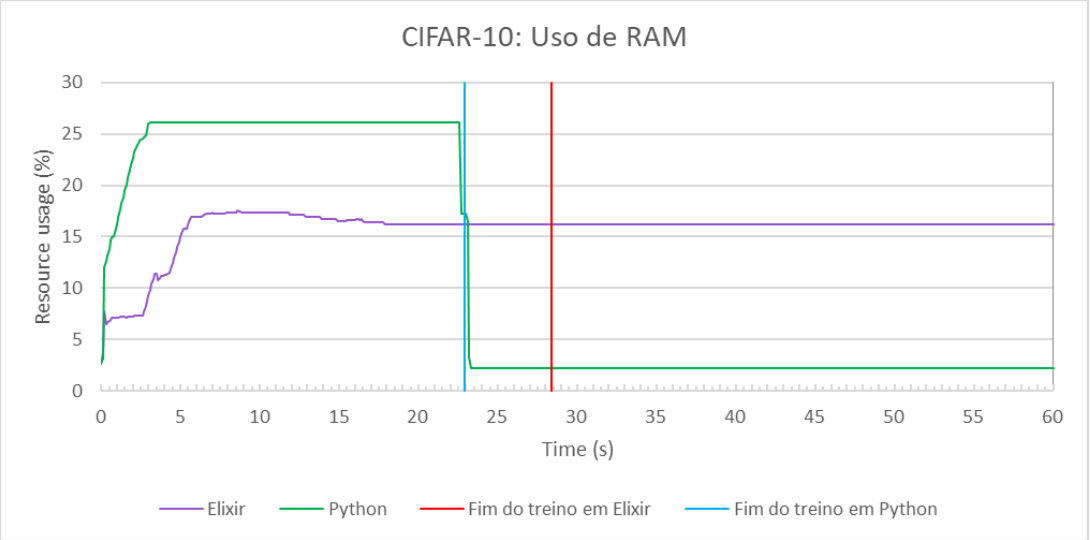
Por fim, observamos que em Python ocorre desalocação da memória RAM ao fim do treinamento. Este comportamento se deve ao fato do treinamento feito em Python ter sido escrito em um código na forma de script, onde o processo finaliza ao final do treino, desalocando a memória utilizada pelo Keras, o que não ocorre em Elixir por conta de Elixir ser executado na máquina virtual do Erlang, a *Bogdan Erlang Abstract Machine* (BEAM) (DEBENEDETTO, 2019). E, ao finalizar o treinamento, por conta da máquina virtual seguir ativa, a RAM segue alocada ao processo Elixir, porém de forma que pode ser reutilizada por outros processos internos da máquina virtual. Este comportamento será visto novamente na seção 5.5.

Figura 15 – Gráfico comparativo do uso de RAM durante treino de modelo MNIST.



Fonte: Autoria própria.

Figura 16 – Gráfico comparativo do uso de RAM.



Fonte: Autoria própria.

5.4 Uso de GPU

Nesta seção analisaremos o uso de GPU para o treinamento das CNNs avaliadas neste projeto e, para tanto, serão analisados os gráficos 17 e 18.

Ao longo do treino utilizando a linguagem Elixir, a variação máxima do uso de GPU foi de 80% durante o treino MNIST e 80% durante o treino CIFAR-10, porém ambas porcentagens ocorrem durante um curto intervalo de tempo de pico de uso, apenas. Para o dataset MNIST este pico ocorre aos 3,4 segundos enquanto para o dataset CIFAR-10 ele ocorre aos 6 segundos. Depois deste pico o uso da GPU fica mais estável; para o dataset MNIST o nível de uso estável foi de 5% e para o dataset CIFAR-10 e esta estabilidade oscila no uso de 16% à 19%.

Por outro lado, ao longo do treino utilizando a linguagem Python, a variação máxima do uso de GPU foi de 19% durante o treino MNIST e 25% durante o treino CIFAR-10, nos curtos intervalos de pico, no qual para o dataset MNIST este pico ocorre aos 1,5 segundos enquanto para o dataset CIFAR-10 ele ocorre aos 9,8 segundos. Depois deste pico o uso da GPU fica mais estável, para o dataset MNIST o nível de uso estável foi de 11%; para o dataset CIFAR-10 é esta estabilidade oscila no uso de 12% à 13%.

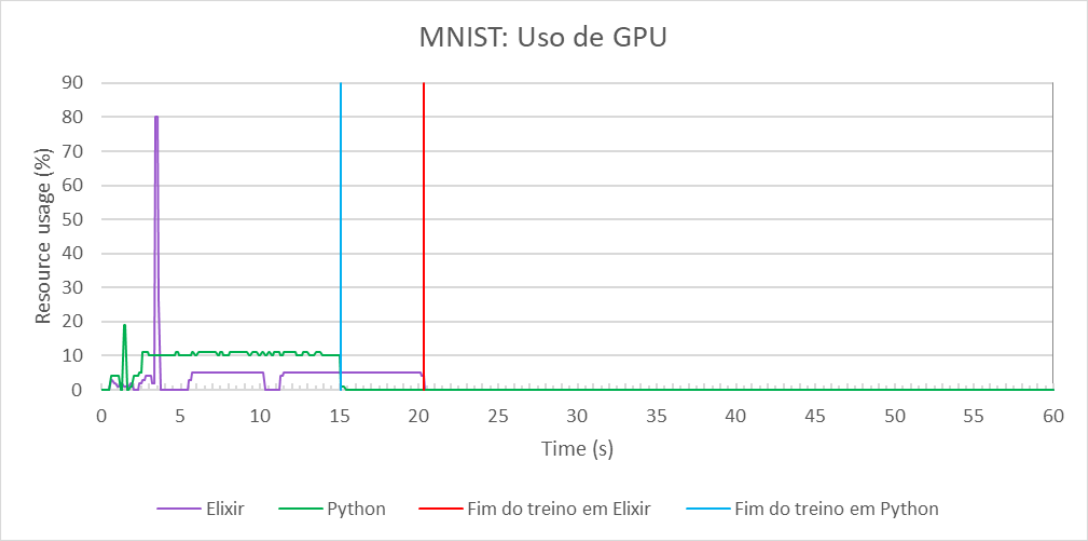
Desta forma, algumas conclusões podem ser tiradas a partir desta análise. Um primeiro ponto é ver que para o Elixir, em ambos treinos há um pico mais alto que em Python, alcançando 80% do uso de GPU, logo no começo do treino, antes mesmo de iniciar o processo da primeira *epoch* da CNN.

Uma segunda observação foi que Elixir possuiu janelas em seus gráficos onde o uso de GPU foi de zero por longos períodos. No treino do modelo para o dataset MNIST, este comportamento ocorreu entre 10,3 segundos e 11,2, uma janela de 0,9 segundos, e para o CIFAR-10 este comportamento ocorreu na janela de 6,4 a 8,8 e novamente entre 15,3 e 16,6,

totalizando uma janela de 3,7 segundos com o uso de GPU em zero. Durante este trabalho, ainda não pudemos concluir a causa raiz destas janelas de uso de GPU zerado, mas este ponto será novamente abordado na seção 6.1 como um ponto de alta relevância para o entendimento em trabalhos futuros em prol de buscar otimizar a biblioteca Nx.

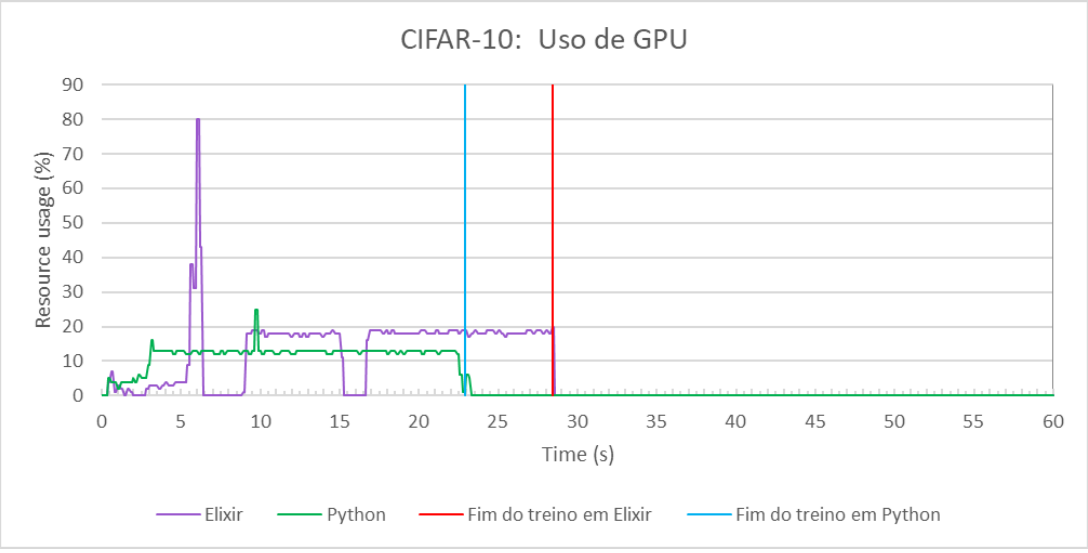
Além disto, de acordo com as amostras coletadas, observamos a inexistência de um padrão muito claro de qual linguagem utiliza mais a GPU uma vez que para o dataset MNIST, Python alcançou porcentagens de uso mais altas, enquanto para o dataset CIFAR-10 Elixir alcançou porcentagens mais altas. Para se concluir qual linguagem utiliza mais GPU, ou então qual possui uma melhor otimização no uso de GPU seria necessário analisar mais casos práticos do treino de CNNs, o que será abordado na seção 6.1.

Figura 17 – Gráfico comparativo do uso de GPU durante treino de modelo MNIST.



Fonte: Autoria própria.

Figura 18 – Gráfico comparativo do uso de GPU.



Fonte: Autoria própria.

5.5 Uso de VRAM

Nesta seção analisaremos o uso de VRAM para o treinamento das CNNs avaliadas neste projeto e, para tanto, serão analisados os gráficos 19 e 20.

Ao longo do treino utilizando a linguagem Elixir, a variação máxima do uso de VRAM foi de 13,42% durante o treino MNIST e 38,85% durante o treino CIFAR-10, e em ambos casos, ao atingir o pico, ele se mantém constante até o fim da amostra coletada. Para o dataset MNIST este pico ocorre aos 2,3 segundos enquanto para o dataset CIFAR-10 ele ocorre aos 3 segundos.

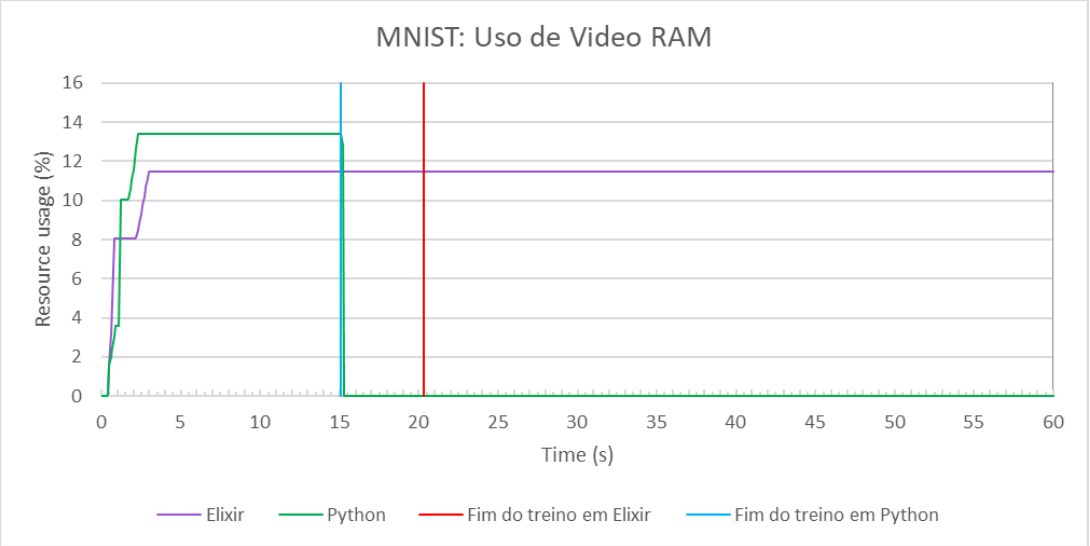
Por outro lado, ao longo do treino utilizando a linguagem Python, a variação máxima do uso de VRAM foi de 11,49% durante o treino MNIST e 38,85% durante o treino CIFAR-10, no qual para o dataset MNIST este pico ocorre aos 3 segundos enquanto para o dataset CIFAR-10 ele ocorre aos 5,7 segundos. Em ambos casos o uso de VRAM fica praticamente constante até o final do processo de treino.

Com a análise feita, identificamos um comportamento bem distinto entre o uso de VRAM entre as linguagens estudadas. Primeiramente é possível notar uma relação com o uso de GPU onde para o dataset MNIST, Python alcançou porcentagens de uso mais altas, enquanto para o dataset CIFAR-10 Elixir alcançou porcentagens mais altas.

Uma segunda análise possível de ser feita com estes dados é de que existe um comportamento único de Elixir que se repete em ambos datasets, o comportamento de manter alocada a VRAM necessária mesmo após o treino ter sido finalizado. A princípio foi cogitado ser um *Memory Leak* do EXLA, e foi aberto um tópico no *Elixir Forum*[1] para solicitar ajuda no entendimento deste comportamento, o qual, assim como na seção 4.1, foi respondido por Sean Moriarity. Com a explicação de Moriarity, foi possível concluir que o comportamento é esperado e padrão do XLA que é utilizado pelo EXLA, onde não é possível garantir a desalocação de VRAM enquanto o processo que alocou a memória ainda está sendo executado, e uma vez que Elixir é executado na máquina virtual do Erlang, a BEAM, esta memória fica relacionada com esta máquina virtual e assim não sendo liberada a não ser que toda a máquina virtual seja finalizada. Este comportamento ocorre similarmente durante a execução dos treinos em Python, mas por conta do treino ser feito em um script, esta memória acaba sendo desalocada ao final da execução do código de treino, como é possível se notar com a análise dos gráficos 19 e 20.

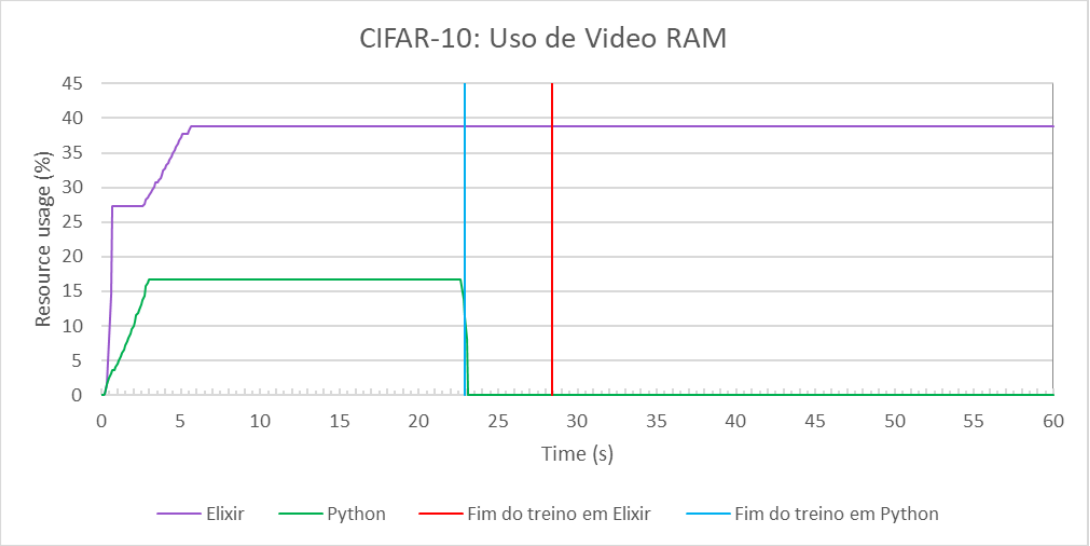
[1] Disponível em: https://elixirforum.com/t/possible-graphic-ram-deallocation-issue-noticed-when-using-nx-with-exla/47629

Figura 19 – Gráfico comparativo do uso de VRAM durante treino de modelo MNIST.



Fonte: Autoria própria.

Figura 20 – Gráfico comparativo do uso de VRAM durante treino de modelo MNIST.



Fonte: Autoria própria.

5.6 Comparação da experiência de desenvolvimento e aprendizagem

Nesta seção serão comparados entre Elixir e Python a quantidade e qualidade das documentações encontrados e curva de aprendizado. Importante ressaltar que esta seção possui diversos pontos de opinião e experiência própria, então isto deve ser levado em consideração.

5.6.1 Documentação

A documentação disponível para dar suporte à implementação realizada em Python foi de ótima qualidade e vasta uma vez que Python já é uma linguagem muito consolidada no setor de processamento de redes neurais, tendo bibliotecas como NumPy, por exemplo, tendo sido disponibilizada ao mercado em 2006 (OLIPHANT, 2006), e a biblioteca *Keras* disponibilizada em 2015. Sendo assim, tendo vários anos até hoje para disponibilizar documentações, artigos e livros, relacionados ao processamento de dados e inteligencia artificial em Python.

Por outro lado, Elixir tem um histórico muito mais curto que Python, tanto em tempo de vida da linguagem como o tempo atuando no mercado de dados e inteligência artificial. O livro "Genetic Algorithms in Elixir" foi publicado em 2021 por Moriarity (2021), sendo o início oficial deste tipo de abordagem de uso da linguagem, e que também inspirou a criação a biblioteca Nx. Sendo assim, a quantidade de documentação em Elixir, desde o começo deste trabalho, se mostrou pequena quando comparada com Python, sendo as principais fontes de documentação o fórum oficial de Elixir[2], o projeto Scholar[3] e as HexDocs[4] das bilbiotecas Nx[5], Axon[6].

Portanto é possível concluir que Python tem um grande histórico no setor de processamento de dados e redes neurais, e por conta disto possui vasta documentação na forma de livros, artigos e guias onlines em geral. Já Elixir possui um menor histórico neste setor e assim uma menor quantidade de documentação. Porém, as documentações oficiais existentes de Elixir foram suficientes para o desenvolvimento dos sistemas criados para o estudo desta monografia, uma vez que possuem ótima qualidade e tem sido aperfeiçoadas continuamente pela equipe responsável pelo projeto Elixir Nx desde seu lançamento.

5.6.2 Curva de aprendizado

Quanto à curva de aprendizado, ela será avaliada levando em consideração o fato de já termos ter trabalhado tanto com Python quanto em Elixir por alguns anos em cada uma das linguagens, e assim possuirmos boa experiência prévia. Tanto Python quanto Elixir tiveram curvas de aprendizado similares para se trabalhar com CNNs.

Em Python foi simples encontrar diversos tutoriais práticos para exercitar redes neurais simples e assim aprender a base da biblioteca Keras através de buscas simples no Google[7]. Em Elixir, a quantidade de tutoriais e guias práticos encontrados foi pequena, porém os materiais encontrados eram todos oficiais da equipe do projeto Nx e assim foram de ótima qualidade, o que proporcionou uma curva de aprendizado boa ao se trabalhar com Nx e Axon para treina-

[2] Disponível em: https://elixirforum.com/
[3] Disponível em: https://github.com/elixir-nx/scholar
[4] Plataforma para disponibilizar documentação de pacotes. Disponível em: https://hexdocs.pm/
[5] Disponível em: https://hexdocs.pm/nx/Nx.html
[6] Disponível em: https://hexdocs.pm/axon/Axon.html
[7] Plataforma de busca disponível em: https://www.google.com/.

mento de redes neurais, levando em consideração o conhecimento prévio na linguagem e a experiência com HexDocs.

6 CONCLUSÃO

Nesta monografia tivemos a oportunidade de avaliar Elixir e Python quanto ao desempenho de treinamento de redes neurais convolucionais, concluindo que, para os datasets analisados, Python apresenta melhor desempenho, e que o uso dos outros recursos de máquina analisados para CPU ambas as linguagens tiveram uso similar. Para RAM Elixir apresentou uma melhor otimização no consumo do recurso, para GPU obtivemos resultados inconclusivos uma vez que não houve padrão do uso de GPU entre os diferentes datasets e, por fim, para VRAM também obtivemos resultados inconclusivos uma vez que, assim como no cenário do uso de GPU, no dataset MNIST Elixir utilizou menos recursos que Python, enquanto que para o dataset CIFAR-10 Elixir utilizou mais recursos que Python.

Quanto à experiência de pesquisa, aprendizagem e desenvolvimento, concluímos que o Python possui mais documentação, material, e variedade de bibliotecas para se trabalhar com redes neurais artificiais. Contudo, a biblioteca Numerical Elixir (Nx) foi um acréscimo significativo para o ecossistema da linguagem Elixir, permitindo ela adentrar novos setores do mercado como o de redes neurais.

Por fim, concluímos que o anúncio do Nx é um marco muito importante para a linguagem Elixir e também para o setor de redes neurais como um todo, uma vez que possuir variedade de linguagens de programação para se trabalhar com redes neurais, ou qualquer outro setor da computação, é muito bom para garantir flexibilidade e liberdade, para que empresas e pesquisadores possam ter a liberdade escolher a linguagem que for mais adequada para seu problema e não ficar fechados a poucas linguagens por falta de opção.

6.1 Trabalhos Futuros

Como trabalhos futuros este trabalho levantou diversos pontos de possíveis pesquisas no setor de redes neurais artificiais em Elixir.

Como primeiro ponto, sugerimos que pesquisadores comparem o desempenho em Elixir e Python em outras redes neurais convolucionais usando outros datasets como: CIFAR-100, Caltech101, Iris, entre outros, para entender se no caso de datasets mais complexos o padrão de comportamento observado nos datasets que nós avaliamos se mantém. Além disto, também ampliar a pesquisa para redes neurais não convolucionais, para buscar entender se os resultados encontrados serão diferentes para outros tipos de redes neurais artificiais ou com outros tipos de abordagens de inteligência artificial como Machine Learing (ML), Generative Adversarial Networks (GAN), Recurrent Neural Networks (RNN), dentre outros tipos.

Um segundo ponto seria se aprofundar no motivo pelo qual tanto para GPU quanto para VRAM foi visto o comportamento de Elixir apresentar menor uso destes recursos com relação a Python para o dataset MNIST e para o dataset CIFAR-10 ter-se notado um comportamento oposto.

Um terceiro ponto seria se aprofundar especificamente no estudo dos pontos que durante o treino das CNNs em Elixir o uso de GPU foi para zero, buscando entender as causas raízes deste comportamento e quem sabe encontrar um ponto de otimização e melhor aproveitamento de recursos disponíveis para o Nx.

Além disso, sugerimos também pesquisas mais abrangentes para se aprofundar nos motivos por trás dos comportamentos observados e entender outros pontos onde o Nx poderia ser melhorado para alcançar melhores resultados.

REFERÊNCIAS

BANERJEE, P. **MNIST - Deep Neural Network with Keras**. 2020. Disponível em: https://www.kaggle.com/code/prashant111/mnist-deep-neural-network-with-keras/notebook. Acesso em: 17 dec. 2022.

CLOUD, G. **Introduction to datasets**. 2022. Disponível em: https://cloud.google.com/bigquery/docs/datasets-intro.

DEBENEDETTO, S. **Elixir and The Beam: How Concurrency Really Works**. 2019. Disponível em: https://medium.com/flatiron-labs/elixir-and-the-beam-how-concurrency-really-works-3cc151cddd61.

DOON, R.; RAWAT, T. K.; GAUTAM, S. Cifar-10 classification using deep convolutional neural network. p. 1–5, 2018.

DREAMCODE. **Top 5 most popular languages for Artificial Intelligence (AI) programming**. 2022. Disponível em: https://www.dreamcodesoft.com/top5-most-popular-languages-for-artificial-intelligence-programming.

DSOUZA, J. **What is a GPU and do you need one in Deep Learning?** 2020. Disponível em: https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d.

EDUCATION, I. C. **What is Overfitting?** 2021. Disponível em: https://www.ibm.com/cloud/learn/overfitting. Acesso em: 17 dec. 2022.

HAYKIN, S.; VEEN, B. V. **Sinais E Sistemas**. Bookman, 2001. ISBN 9788573077414. Disponível em: https://books.google.com.br/books?id=tdNYclZwaYIC.

MORIARITY, S. **Genetic Algorithms in Elixir**. Pragmatic Bookshelf, 2021. Disponível em: https://pragprog.com/titles/smgaelixir/genetic-algorithms-in-elixir/.

MORIARITY, S. **EXLA HexDocs**. 2022. Disponível em: https://hexdocs.pm/exla/EXLA.html.

OLIPHANT, T. E. **Guide to NumPy**. [*S.l.*]: Trelgol, 2006.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. **arXiv preprint arXiv:1511.08458**, 2015.

PAPERSWITHCODE. **CIFAR-10**. 2022. Disponível em: https://paperswithcode.com/dataset/cifar-10.

PAPERSWITHCODE. **MNIST**. 2022. Disponível em: https://paperswithcode.com/dataset/mnist.

PEDROSA, L. **Processando imagens**. 2021. Disponível em: https://www.ic.unicamp.br/~lehilton/cursos/1s2021/mc102w/tarefas/07-imagens. Acesso em: 17 dec. 2022.

VALIM, J. **Nx (Numerical Elixir) is now publicly available**. 2021. Disponível em: https://dashbit.co/blog/nx-numerical-elixir-is-now-publicly-available.

WIKIPEDIA, a. e. l. **RGB**. 2022. Disponível em: https://pt.wikipedia.org/wiki/RGB.