UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Thiago Henrique Segreto Silva

Análise de desempenho de modelos de Deep Learning para previsão de falhas em operações de rosqueamento

São Carlos
2021

Thiago Henrique Segreto Silva

Análise de desempenho de modelos de Deep Learning para previsão de falhas em operações de rosqueamento

Monografia apresentada ao Curso de Engenharia Mecatrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

VERSÃO CORRIGIDA

São Carlos

2021

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

SThiHenrSegrSilva

Segreto Silva, Thiago Henrique
Análise de desempenho de modelos de Deep Learning para previsão de falhas em operações de rosqueamento / Thiago Henrique Segreto Silva; orientador Glauco Augusto de Paula Caurin; coorientador Gustavo Jose Giardini Lahr. São Carlos, 2021.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2021.

1. Operação de rosqueamento. 2. Previsão de falhas. 3. Deep learning. 4. Data augmentation. 5. Análise de desempenho. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: _ Thiago Henrique Segreto Silva _____

Título: _ Análise de desempenho de modelos de Deep Learning para previsão de falhas em operações de rosqueamento_____

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Professor _Glauco Augusto de Paula Caurin_ (Orientador)

Nota atribuída: _9,0_ (__nove_____)

_____
(assinatura)

Professor _Gustavo José Giardini Lahr__

Nota atribuída: _9,0_ (__nove_____)

p/ _____
(assinatura)

Professor _Marcio José da Cunha _____

Nota atribuída: _9,0_ (__nove_____)

p/ _____
(assinatura)

Média: _9,0_ (__Nove__)

Resultado: ____ Aprovado _____

Data: __23_/_07__/_2021__.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☑ NÃO ☐ Visto do orientador _____

AGRADECIMENTOS

Ao Dr. Glauco Caurin, pelo suporte e apoio como orientador.

Ao Gustavo Lahr, por sempre estar disposto a ajudar e ser um grande mentor e amigo

Aos meus pais Marco Antônio e Elizabete Adriana, por todo carinho e apoio que me deram ao longo desses anos.

Agradeço também à AWS por terem disponibilizado recursos usados nesta pesquisa.

RESUMO

SEGRETO SILVA, T. H. **Análise de desempenho de modelos de Deep Learning para previsão de falhas em operações de rosqueamento**. 2021. 198 f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2021.

Operação de rosqueamento é uma das mais comuns em ambientes industriais e se encontra no processo de manufatura de diversas classes de produtos. Para automação desse tipo de operação é fundamental uma etapa de previsão de falhas para evitar perdas de componentes. Mas existem diversas dificuldades a essa etapa devido às incertezas do ambiente industrial e da complexa dinâmica de rosqueamento. Essa última faz com que a modelagem do problema seja extremamente complicada e, muitas vezes, incompleta ou ineficiente. Como alternativa à modelagem clássica, os métodos de aprendizado de máquina têm o potencial de representar as características complexas da operação através da análise de dados experimentais. *Deep learning* é uma das áreas que vem ganhando espaço nesse cenário e já se provou extremamente eficaz em visão computacional e processamento de linguagem natural. Porém, seus modelos necessitam de uma grande quantidade de dados para obterem desempenho significativo, e aliado a dificuldade de extrair dados em operações de rosqueamento, novas estratégias de obtenção de dados são necessárias para aplicações práticas. Uma possível solução, é a geração de dados sintéticos a partir de dados reais pelo processo de *data augmentation*. Nesse contexto, este trabalho se propõe em analisar o desempenho de modelos de *deep learning* treinados em conjuntos de dados artificiais, e verificar o impacto de modelos de diferentes níveis de complexidade na previsão de falhas de rosqueamento.

Palavras-chave: Operação de rosqueamento. Previsão de falhas. Deep learning. Data augmentation. Análise de desempenho.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Elemento opcional. É composto de uma relação alfabética das abreviaturas e siglas utilizadas no texto seguido do seu significado.

AWS	–	Amazon Web Services
CNN	–	Convolutional Neural-Networks
DL	–	Deep Learning
ECR	–	Elastic Container Registry
GPU	–	Graphic Processing Units
JSON	–	JavaScript Object Notation
MLP	–	Multiple Layer Perceptrons
MLOPs	–	Machine Learning Operations
ML	–	Machine Learning
SNS	–	Simple Notification Service

SUMÁRIO

1 INTRODUÇÃO

Rosqueamento é uma das operações mais comuns em ambientes industriais e está presente na manufatura de diversas classes de produtos, desde pequenos aparelhos eletrônicos até grandes maquinários [1]. A automação desse procedimento é fundamental para se conquistar um aumento de produtividade resistente a falhas e com qualidade consistente. Mas, apesar da prevalência e importância dessa operação, a automação completa ainda não é a realidade na indústria devido à sua inerente complexidade [2].

Parte desta complexidade se deve a grande variedade de cenários e parâmetros da operação e a imprevisibilidades do ambiente industrial, o que torna a modelagem do problema particularmente difícil e incompleta [3]. Outra dificuldade se dá pelas tolerâncias mais rigorosas, pois erros podem causar o descarte de peças inteiras ou comprometer componentes sensíveis como baterias. Por fim, durante o processo de rosqueamento ocorrem diferentes estágios de contato mecânico com dinâmicas complexas, e cada uma introduzindo oportunidades para falhas [4]. Neste contexto, uma das estratégias de automação é implementar um estágio de previsão de falhas acoplado com procedimentos de recuperação. Assim, mesmo expostos aos riscos citados acima, é possível antecipá-los e garantir continuidade da operação.

Devido às dificuldades na modelagem, os métodos de previsão baseados em análise de dados são uma solução alternativa vantajosa. Dentre esses, os pertencentes a classe de *deep learning* (DL) possuem um grande potencial para previsões robustas, mas necessitam de um grande volume de dados classificados para atingirem resultados satisfatórios [5]. A extração desses dados possui seus próprios desafios, entre os principais, a dificuldade de se conseguir uma quantidade exemplos de falha significativa. Com apenas 1%-2% das operações de rosqueamento resultando em falha, o conjunto de dados finais é desbalanceado, o que causa o enviesamento nos algoritmos de DL [6]. Também há uma dificuldade intrínseca em se garantir a representatividade desses dados, pois a própria classificação manual de falhas está sujeita a erros na identificação do real estado da peça.

Uma das formas de contornar os conjuntos desbalanceados e a falta de dados é utilizando técnicas de *data augmentation* que expandem artificialmente o conjunto de

dados utilizando heurísticas para garantir a relevância desses novos. A representatividade dos dados pode ser averiguada indiretamente através dos erros de generalização dos modelos de classificação [7].

Outros fatores podem prejudicar o resultado dos modelos de previsão, como por exemplo: complexidade do modelo utilizado, hiperparâmetros mal otimizados e má escolha dos parâmetros extraídos dos dados [8] [pág. 424-427]. Não existe um método para saber qual o principal fator que está deteriorando os modelos, mas com experimentação com pré-processamento de dados e modelos de diferentes complexidades pode-se chegar às principais causas determinístico [8] [pág. 414].

Este trabalho propõe analisar o desempenho de classes de algoritmos de aprendizado na previsão de falhas em operações de rosqueamento robótico a fim de encontrar os principais fatores que o limita. Para isso, foi comparado o impacto do nível de complexidade dos modelos e do tamanho do conjunto de dados em que foram treinados.

A seção 2 apresenta os modelos de DL utilizados, as métricas de classificação escolhidas e explica o processo de otimização dos hiperparêmetros. Também explica o método de análise de desempenho, bem como os tipos de erro envolvidos no uso dos modelos. Na seção 3 está definido o algoritmo de sintetização de dados para a criação dos conjuntos artificiais. As características e o processo de obtenção de dados são explicados na seção 4, onde também está descrito o processo de treinamento e otimização dos modelos em nuvem.

2 DEEP LEARNING

Algoritmos de *machine learning* (ML) são aqueles que possuem a capacidade de realizar tarefas específicas e de se autoajustarem para melhorarem o seu desempenho nessas. Existe uma grande gama de algoritmos que usam diferentes técnicas de regressão e classificação, cada qual com suas vantagens e desvantagens.

DL é uma subclasse de algoritmos de ML, portanto também são algoritmos capazes de aprender a realizar tarefas avaliando o seu desempenho e otimizando seus parâmetros internos conforme esse. O que diferencia essa classe de algoritmos dos modelos de ML clássicos é sua arquitetura de processamento de dados, que usa "neurônios artificiais" para modelar as relações entre os dados de entrada [8] [pág. 191-196].

Neurônios conectados entre si formam estruturas chamadas de redes neurais artificiais, inspiradas nas próprias camadas de neurônios biológicos. O efeito prático dessa estrutura de processamento é o aumento significativo da dimensionalidade do modelo, o que dá a capacidade de representar funções não-lineares complexas através do espaço dimensional com diversos parâmetros. Esse é o principal atributo que torna os algoritmos de DL extremamente eficientes em resolução de problemas complexos [9].

Visão computacional e processamento de linguagem são exemplos de problemas difíceis em que as redes neurais se tornaram a principal técnica de análise e previsão, superando as clássicas de ML e ferramentas heurísticas e estocásticas [10][11]. Apesar do sucesso em ambas as áreas, modelos de DL ainda estão ganhando espaço na análise de séries temporais [12], que estão presentes em diversos problemas práticos devido à natureza temporal dos fenômenos [13].

2.1 Modelos

Abaixo seguem três modelos de DL, dispostos em ordem crescente de complexidade, que é considerada como uma combinação do número de parâmetros e processos intermediários de processamento de dados.

2.1.1 Multilayer Perceptrons (MLP)

A MLP é o modelo mais simples de DL [14], contando com uma estrutura bem definida de camadas de neurônios nas quais cada um está conectado a todos os outros das camadas vizinhas. A *figura 1* mostra a disposição dos *neurônios* em camadas em MLP genérica.

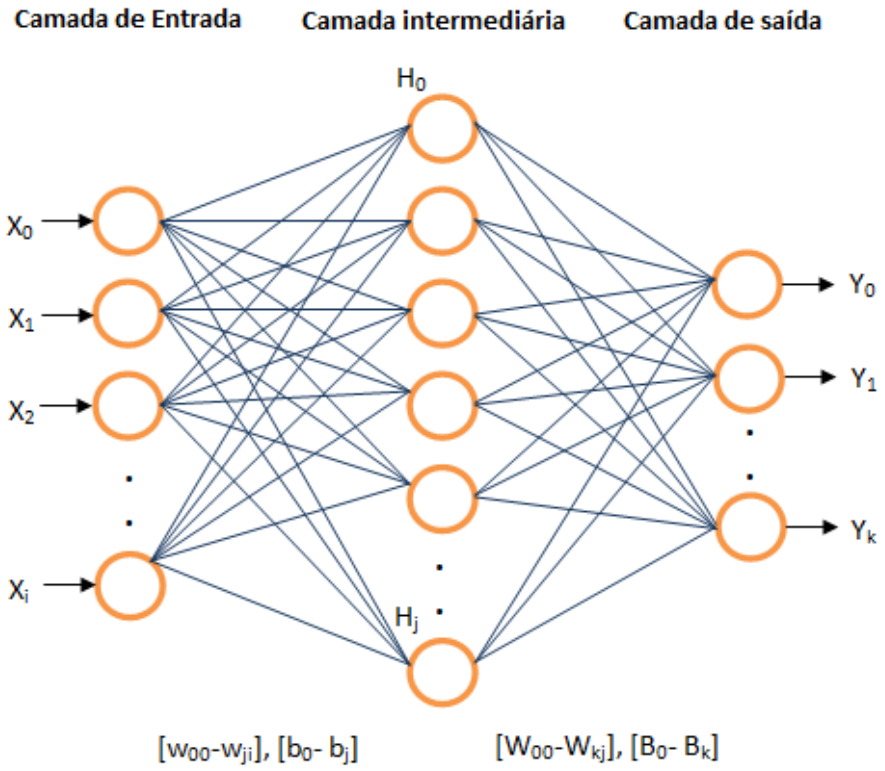

Figura 1 - Esquema genérico de uma MLP (https://www.dotnetlovers.com/article/243/neural-networks-and-mlp)

De camada a camada, os dados de entrada representados pelo vetor X são multiplicados por uma matriz de pesos W, cujos valores representam cada conexão entre os neurônios de duas camadas $\{w_{0,0}, w_{0,1} \dots w_{i-1,i}, w_{i,i}\}$. Depois é aplicada uma função de ativação não linear g com o intuito preservar a dimensionalidade da rede, impedindo que essa se reduza a uma simples transformação linear [15]. A transformação dos dados até antes da última camada pode ser representada pela *equação 1*.

$$H_k = g(W^T * X + b) \qquad 1$$

H é o vetor de valores dos neurônios da camada intermediária, k o índice da camada e b é o vetor de viés, utilizado para garantir um grau de liberdade a mais da função g, permitindo a translação dos resultados da função [16] [pág. 285-287]. Existem diferentes tipos de função de ativação, como por exemplo: sigmoid, tanh e *rectifier linear unit* (ReLU), sendo a última a mais difundida por sua eficácia e rapidez [17].

No caso de problemas de classificação, a última camada possuirá o mesmo número de neurônios que de classes e sua saída será um número no intervalo [0, 1] que representará a certeza da rede sobre a escolha da classe. Assim, tem de se garantir que a soma dos valores de saída seja exatamente 1.

A função *softmax* abaixo (*equação 2*) satisfaz essa condição e sempre mantem a soma total dos valores de saída como valor unitário, sendo K o número de classes e h o vetor de entrada de cada neurônio i da última camada.

$$\sigma(\vec{h})_i = \frac{e^{h_i}}{\sum_{j=1}^{K} e^{h_j}} \qquad 2$$

O último estágio no treinamento do modelo envolve o cálculo do erro nos exemplos de treino e a propagação da última camada para as intermediárias. Com os gradientes descendentes, é possível atualizar os pesos de cada conexão entre neurônios para valores na direção de diminuição dos erros. Existem várias métricas de erro e essas acabam se destacando em cenários diferentes. Por exemplo, para um conjunto de dados com muitos *outliers,* funções polinomiais de erro melhoram o treinamento do modelo quanto maior o seu grau de liberdade. A maior dimensionalidade da função de erro ajuda a diminuir a influência de dados anômalos na atualização da matriz de pesos [18].

Neste trabalho foi utilizada a função perda de entropia cruzada esparsa E que, bastante similar ao erro logístico, é capaz de representar os erros em problemas de classificação [19]. Essa função expressa a diferença entre a distribuição das classes reais p e a das classes previstas q conforme a *equação 3*.

$$E(p, q) = -\sum_{x \in X} p(x) \log q(x) \qquad 3$$

Escolhida a função de erro, aplica-se o método *Stocastic Gradient Descent* (SGD) para propagar o erro da camada de saída para as camadas intermediarias [8] [pág. 197]. Com os gradientes G em cada neurônio calcula-se os novos pesos w_{ij} de sua conexão ponderando o erro com o fator de taxa de aprendizado μ com mostra a *equação 4.*

$$\Delta w_{ji}(x) = -\mu G_j y_i(x) \qquad 4$$

Esse procedimento é repetido múltiplas vezes para cada vetor de entrada X dos dados de treino até que seja percorrido todo o conjunto e satisfaça as condições de parada. Ao final teremos uma matriz de pesos W otimizada para o conjunto de treino e basta verificar se desempenho generaliza para o conjunto de teste, que possui os dados que o modelo não processou.

2.1.2 Convolutional Neural Networks (CNN)

CNNs são os tipos mais utilizados em visão computacional devido ao desempenho sobre humano em muitas aplicações de análise de imagem [20]. Assim como as MLPs, as CNNs também utilizam a mesma estrutura de neurônios conectados em camadas, mas agora com diferenças estruturais que as deixam mais similares ao comportamento dos neurônios de um córtex visual.

A principal diferença é a adição de camadas de convolução que, ao invés de estar completamente conectada, seus neurônios apenas se conectam a uma região limitada da camada anterior. Isso faz com que cada neurônio da camada subsequente apenas abstraia as informações contidas na região, chamadas de *kernel*, que ele observa.

Essa nova forma de se conectar os neurônios de diferentes camadas faz com que haja menos parâmetros para serem processados devido ao menor número de conexões quando comparadas a uma MLP típica [8] [pág. 322]. Para melhorar sua capacidade de abstração devido ao menor número de conexões, a camada convolucional é decomposta em mapas que filtram os dados de entrada e assim treinam sobre aspectos específicos da imagem.

A *figura 2* é uma representação das camadas de convolução com seus respectivos mapas de filtros.

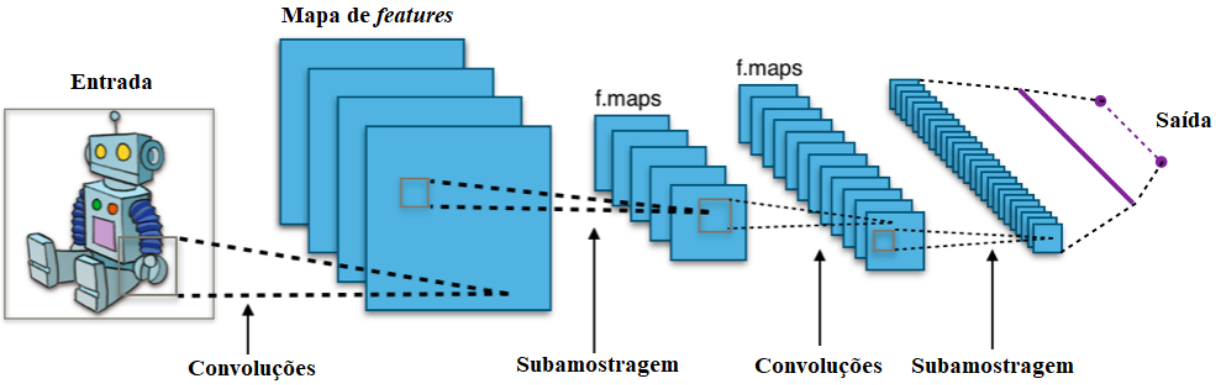

Figura 2 - Esquema genérico de uma CNN
(https://en.wikipedia.org/wiki/Convolutional_neural_network)

A capacidade de abstrair mais com menos conexões gera dois problemas. O primeiro é a grande carga de memória RAM adicional por camada devido aos mapas adicionais por convolução. O segundo é adição de novos hiperparâmetros que tornam extremamente custosos o treinamento do modelo com a otimização da estrutura da rede [16] [pág. 456].

Para amenizar esses problemas, a camada de *pooling* é adicionada entre as camadas intermediárias para reduzir a resolução dos dados de entrada. Para isso, é feito um agrupamento dos dados por kernel, representando-os assim por um único valor em cada neurônio da camada de *pooling*. Porém, a diminuição da resolução dos dados reduz o desempenho final da rede, então deve-se buscar um equilíbrio entre o uso de camadas convolucionais e de *pooling* para atingir um desempenho satisfatório em um tempo de treinamento plausível [21].

No geral, CNNs possuem desempenho equiparável ou ordens de grandeza maior que as MLPs em problemas de alta complexidades por consequência da introdução de novas etapas de abstração [22].

2.1.3 Transformers

Diferente das MLPs e CNNs, as *trasnformers* são redes mais abstratas e usam uma combinação de artefatos ao invés de serem embasada em um conceito de funcionamento único. Dentre seus principais artefatos tem-se os: mecanismos de atenção, camadas de

embedding, MLPs, camadas de normalização. A *figura 3* ilustra um exemplo de uma transforme e seus componentes.

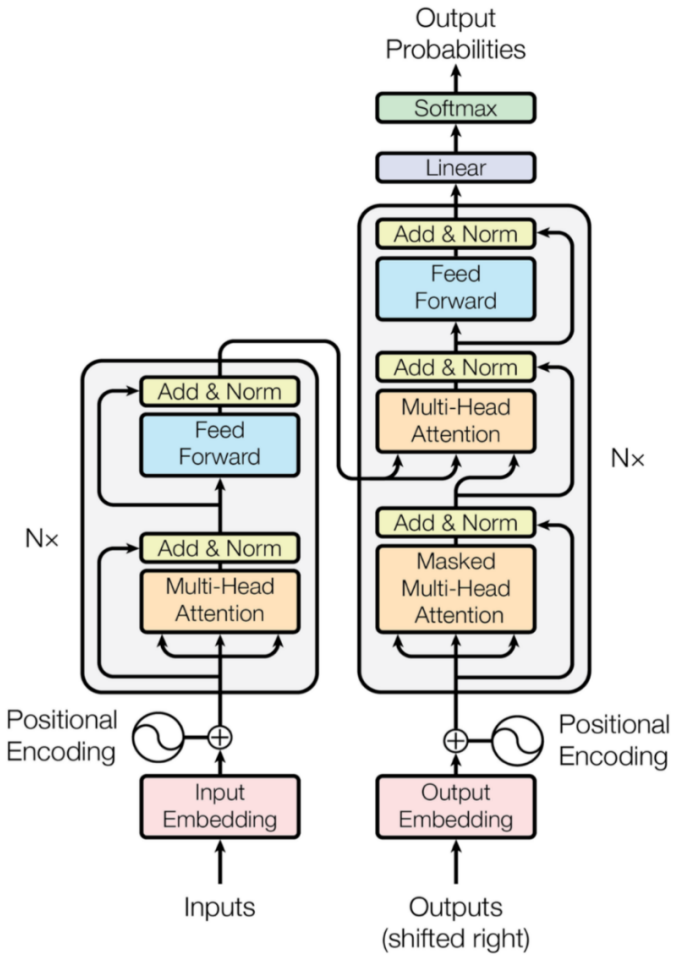

Figura 3 - Esquema genérico de um Transformer [16]

As *transformers* foram desenvolvidas para o estudo de linguagem natural e se tornaram bem-sucedidas no emprego de modelos linguísticos [23]. O motivo de ser um excelente motor desses modelos se deve principalmente aos mecanismos de atenção citados. Esses constroem relações hierárquicas entre os dados que ele recebe [24], determinando a relevância de cada no contexto em que estão inseridos. Os mecanismos são representados na *figura 3* pelas caixas "*masked multi-head attention*".

Na entrada da estrutura existem dois mecanismos, um de codificação (*encoder*) a esquerda e outro de decodificação (*decoder*) a direita. A função desses dois é, respectivamente, codificar os dados de entrada em um formato padrão de 512 bits e eventualmente decodificar os outputs desse formato ao original dos dados. Essa etapa é

necessária pois as *transformers* precisam de uma representação interna dos dados, de forma que todos estejam catalogados para ser mais fácil a identificação e processamentos.

Embedding é outro tipo de processo de codificação crucial no funcionamento de uma transformer, mas diferente do *encoder, embeddings* codificam grupos de dados conforme o contexto em que estão inseridos. Por exemplo, se forem usados para processamento de linguagem, o *encoder* codificaria sentenças inteiras uma vez, enquanto *embeddings* o fariam somente em palavras e o resultado dependeria da sentença em que essas palavras estão contidas.

Todo esse dinamismo serve a um único propósito: processar grandes cadeias de dados correlacionados da forma mais eficiente possível [25]. Essa é exatamente a natureza de um problema de linguagem, processar textos longos com milhares de sentenças e palavras. No contexto de séries temporais, o uso de transformers também é bastante relevante, pois aquelas também são cadeias de dados longas e correlacionadas.

2.2 Métricas de classificação

As métricas mais comuns em problemas de classificação são derivadas da matriz de confusão. Ela representa os possíveis resultados de previsão em relação a classificação correta dos exemplos. As métricas de avaliação dos modelos são escolhidas com base nos diferentes grupos pertencentes à matriz. A *figura 4* é uma matriz de duas dimensões com as principais métricas desempenho.

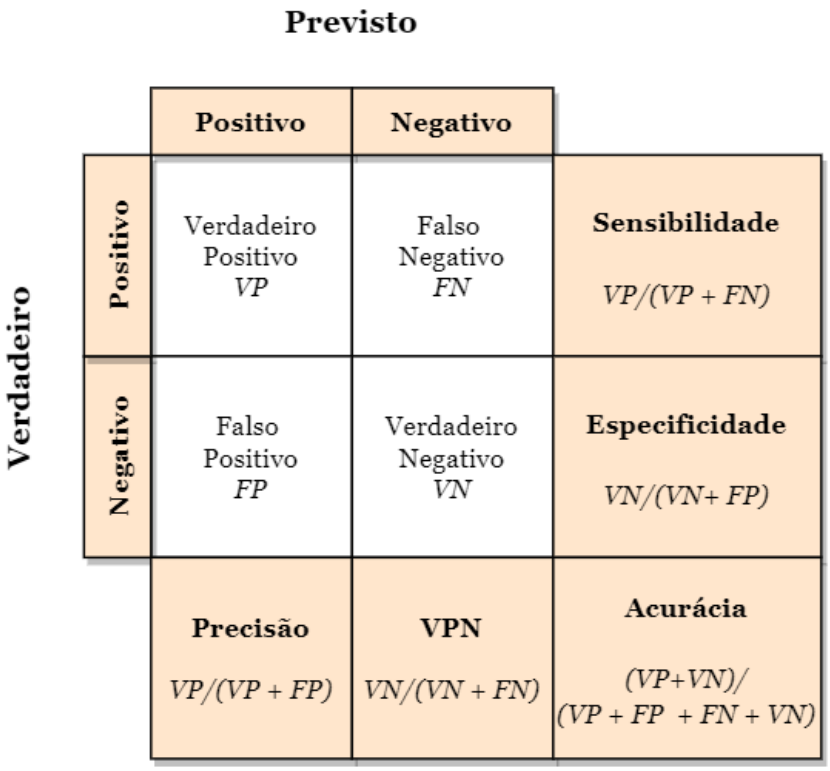

Figura 4 - Matriz de confusão de duas dimensões

A escolha das métricas de desempenho é fundamental para o sucesso de um processo de classificação. Apesar das classes já estarem definidas e devidamente categorizadas, a otimização dos hiperparâmetros do modelo é feita utilizando as métricas como funções objetivas [26]. A princípio não há uma regra definida sobre a escolha das métricas, cada aplicação possui um conjunto mais apropriado para seus requisitos e objetivos [27].

Precisão e sensibilidade são as métricas mais comuns e geralmente possuem uma relação inversa, na tentativa de se otimizar uma compromete-se a outra [28]. Precisão é muito útil quando se quer garantir o menor número de falso positivos com o custo de mais exemplos classificados serem falso negativos. Já a sensibilidade se torna mais importante quando se quer que o máximo de exemplos positivos sejam classificados como tal mesmo com aumento de falsos positivos.

O valor preditivo negativo (VPN) e a especificidade possuem a mesmo relacionamento característico do observado entre previsão e sensibilidade, mas quando se análise a classe negativa, que no caso de classificação de múltiplas classes seriam todas

outras classes além da classe de positiva de referência [26]. Por último, a acurácia representa o desempenho geral dos modelos sobre todas as classes, ou seja, quando as previsões de todas as classes são igualmente importantes, esta é uma das métricas mais desejáveis.

A partir das métricas acima, pode se derivar outras mais apropriadas para atender a outras análises específicas. O valor F1, por exemplo, é uma forma de agregar precisão e sensibilidade para que sejam otimizadas ambas as métricas. Essa é importante em problemas em que ambos falsos positivos e negativos tenham grande relevância e, portanto, a redução do número desses casos deve ser otimizada [27].

Para estudos em que há mais de duas classes, analiza-se cada classe individualmente contra todas as outras a fim de se calcular o valor da métrica de desempenho correspondente. Seguem as fórmulas de precisão, sensibilidade e F1 para multiclasses, sendo k o número de classes.

$$precisão = \frac{VP}{VP + \sum_{i=0}^{k} FP_i} \quad (5)$$

$$sensibilidade = \frac{VP}{VP + \sum_{i=0}^{k} FN_i} \quad (6)$$

$$F1 = 2 * \frac{precisão * sensibilidade}{precisão + sensibilidade} \quad (7)$$

2.3 Hiperparâmetros

Hiperparâmetros são parâmetros internos e externos aos modelos de ML que podem ser otimizados além da etapa de SGD de atualização da matriz de pesos. Os parâmetros internos se referem àqueles que definem o tipo e a estrutura do modelo, e os externos são associados a qualquer configuração da etapa de pré-processamento do conjunto de dados [29].

A otimização desses parâmetros é realizada repetindo-se o processo de treinamento para cada combinação a fim de determinar qual a melhor estrutura de rede neural [29].

2.3.1 Hiperparâmetros MLP

- Neurônios por camada: Este valor normalmente é próximo do tamanho do input, mas pode variar de camada a camada.
- Número de camadas: Quanto maior o número de camadas, maior a capacidade de abstração da rede, mas também introduz problemas em seu treinamento.
- *Dropout:* Fator de regularização que seleciona aleatoriamente quais neurônios vão participar de cada etapa de treino. O objetivo desse parâmetro é amenizar erros causados por alta variância.
- Taxa de aprendizado μ: Valor que determina a intensidade de correção dos pesos após o cálculo de erro e aplicação do SGD.
- *Batch Normalization:* Fator que corrige o desvio padrão de grupos (*batches*) de dados conforme passam pelas camadas da rede. Serve principalmente para estabilizar e acelerar o processo de treinamento, particularmente útil em redes com muitas camadas.
- Função de ativação: apesar de diferentes funções de ativação, ReLU continua sendo a mais utilizada.

2.3.2 Hiperparâmetros CNN

A CNN possui todos os hiperparâmetros de uma MLP e os seguintes adicionais em relação à sua estrutura.

- Número de mapas de filtro por camada: como discutido na seção 2.1.2, cada camada convolucional será um agrupamento de mapas de filtro.
- Tamanho do kernel: tamanho da região que é conectada aos neurônios da camada subsequente.

- *Stride:* Dimensão do deslocamento do kernel entre neurônios. É utilizado para reduzir o tamanho da próxima camada convolucional, diminuindo o tempo de treino.
- *Padding:* Neurônios adicionados à camada, normalmente com peso zero, para determinar um tamanho específico para a próxima camada de convolução

2.3.3 Hiperparâmetros Transformer

- Camadas *transformer:* Camadas abstratas que englobam as camadas *Multi-head,* camadas de normalização e as MLPs internas.
- Número de *heads:* Número de módulos de atenção.
- Tamanho do *embedding:* Tamanho do vetor em que se deseja codificar os dados de entrada na célula de *embedding.*

2.3.4 Cross-validation

Ao se otimizar os hiperparâmetros utilizando o conjunto de teste para verificar o desempenho, pode ocorrer o enviesamento da rede e consequentemente causar erro de generalização. Para evitar esse problema, aplica-se a técnica de *cross-validation* que divide o conjunto de treino em um número k determinado [30]. Treina-se o modelo k vezes, escolhendo uma das divisões como conjunto de teste em cada vez. Dessa maneira, garante-se que o conjunto de teste original fique reservado apenas para a última análise de desempenho, de forma a ter uma avaliação fiel da generalização do modelo para novos dados.

2.4 Análise de desempenho

Os erros e limitações de desempenho dos modelos de ML normalmente recaem sobre os seguintes espectros: enviesamento ou excesso de variância [31]. Conseguir determinar em qual desses está o principal culpado pela limitação de desempenho é

fundamental para se traçar novas estratégias de treino que não desperdicem tempo e recursos computacionais [32]. A *figura 5* ilustra a relação de erros por viés e variância no contexto da complexidade do modelo.

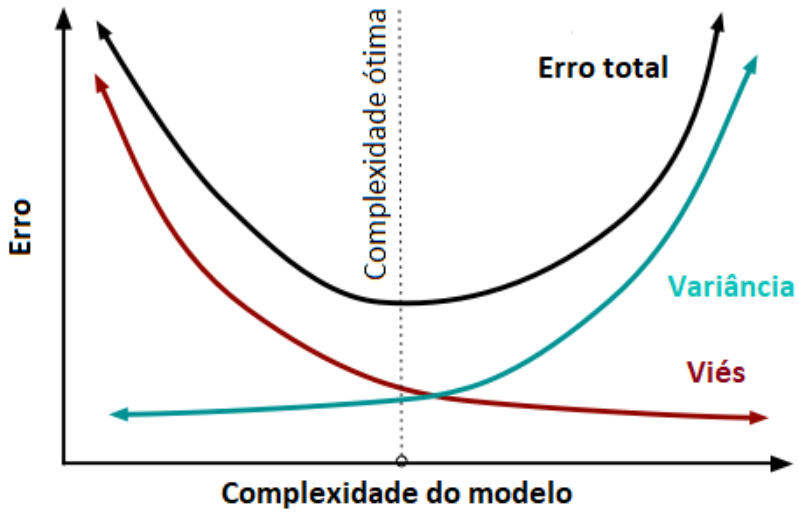

Figura 5 - Comportamento do par viés-variância em relação à complexidade do modelo

Erro por enviesamento se dá pelo fato de o modelo não conseguir representam as relações complexas entre os dados. Dentre os principais motivos, se destacam:

- Pouca quantidade de dados para alimentar o modelo
- Modelo com baixa dimensionalidade (poucos parâmetros para ser representativo)
- Alta regularização dos pesos das conexões entre neurônios, limitando o impacto dos parâmetros internos do modelo

Já a alta variância se dá principalmente por:

- Dados pouco representativos ou anômalos
- Modelo de alta complexidade
- Baixa regularização da matriz de pesos de conexões neurais

Treinamento em conjuntos de dados maiores aumentam o desempenho dos modelos independentemente do tipo de erro [33]. Porém, a magnitude dos ganhos pode indicar que o tamanho do conjunto não é o principal fator limitante do desempenho. No quesito de dimensionalidade do modelo, se os erros de generalização aumentaram com os modelos mais complexos, provavelmente o erro está atrelado à alta variância.

3 DATA AUGMENTATION

Data Augmentation é uma técnica que se tornou bastante popular em implementações de visão computacional. O principal objetivo é gerar dados representativos a partir dos observados e assim melhorar a generalização dos modelos. A representatividade tem que ser aferida pois deve-se garantir que os novos dados artificiais possuam relevância estatística no que diz a respeito da distribuição real dos dados [34].

Apesar da popularidade e eficácia das técnicas, poucas são transferíveis para outras classes de problemas de ML. Isso se deve às particularidades dos dados de imagem, o que torna certas operações de produção de dados artificiais incondizentes com dados de outros tipos. Um exemplo bastante comum é a geração de imagens por rotação que ajuda redes neurais a abstraírem imagens idênticas em perspectivas diferentes. Essa técnica se provou bastante eficaz no campo de visão, mas não obteve o mesmo sucesso em aplicações de séries temporais [35].

Para essas, o método Synthetic *Minority Over-sampling Technique* (SMOTe) é considerado o mais adequado para geração de dados sintéticos [36]. Esse equilibra as populações entre classes diferentes no conjunto de treino afim de eliminar o efeito de enviesamento nos algoritmos. Também pode ser utilizado para expandir um conjunto de dados já equilibrados a fim de se obter maior volume de dados totais, aplicando-o em cada classe separadamente [37].

O seu funcionamento se baseia na criação de dados variados ponderados em relação a distância euclidiana dos vizinhos de mesma classe. Para isso, vetoriza-se os dados da classe de referência e determina-se um o número de vizinhos k dos quais se deseja calcular a média de distância euclidiana, utilizando um ponto pertencente a classe como pivô. A *figura 6* é uma representação geométrica da geração de dados por SMOTe.

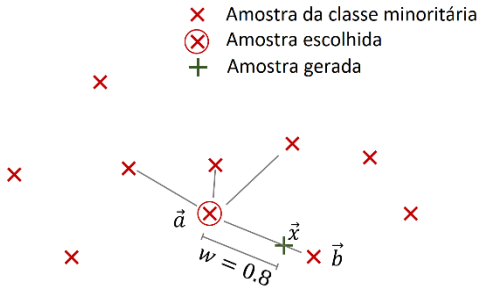

Figura 6 - SMOTe calculando um elemento da classe alvo utilizando 4 vizinhos em torno do ponto de referência. Last, F., Douzas, G., & Bacao, F. (2017). Oversampling for imbalanced learning based on k-means and smote.

4 INFRAESTRUTURA E DADOS

Processos e ferramentas utilizados nas etapas de aquisição e processamento de dados. Segue a descrição dos conjuntos utilizados nesse trabalho, bem como a arquitetura em nuvem utilizada para automação do processo de treinamento dos modelos.

4.1 Conjunto de dados

O conjunto de dados analisados provém do estudo [38] em que foram utilizadas seguintes classes de eventos montado, travado e não montado (*figura 7*). Os dados foram gerados através de operações de rosqueamento com o robô industrial KUKA KR16 utilizando o sensor de força e torque MHS3-50D com tempo de amostragem de 12ms.

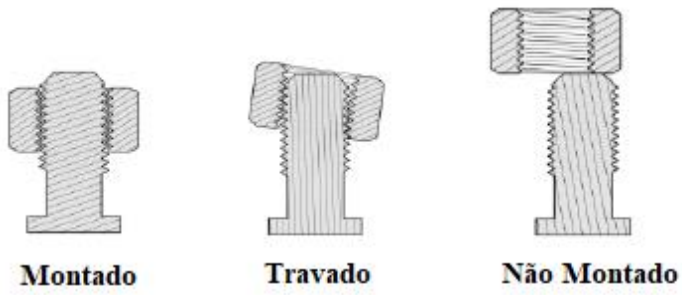

Figura 7 - As três classes da operação de rosqueamento consideradas nos conjuntos de dados

No total há 476 operações de rosqueamento contendo 2560 pontos de dados e 12 séries temporais, que correspondem aos parâmetros de forças axiais, torques, posição angular e linear nos eixos x, y, z respectivamente. As *figuras 8 e 9* exibem as forças e torques em cada eixo e para cada classe de resultado.

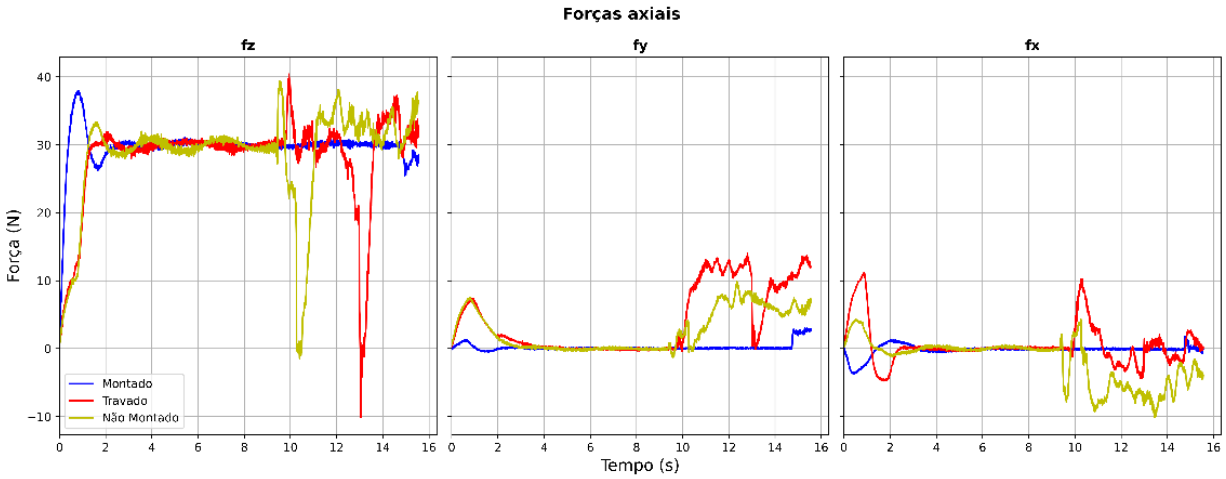

Figura 8 - Forças nos eixos x, y e z de cada classe de experimento

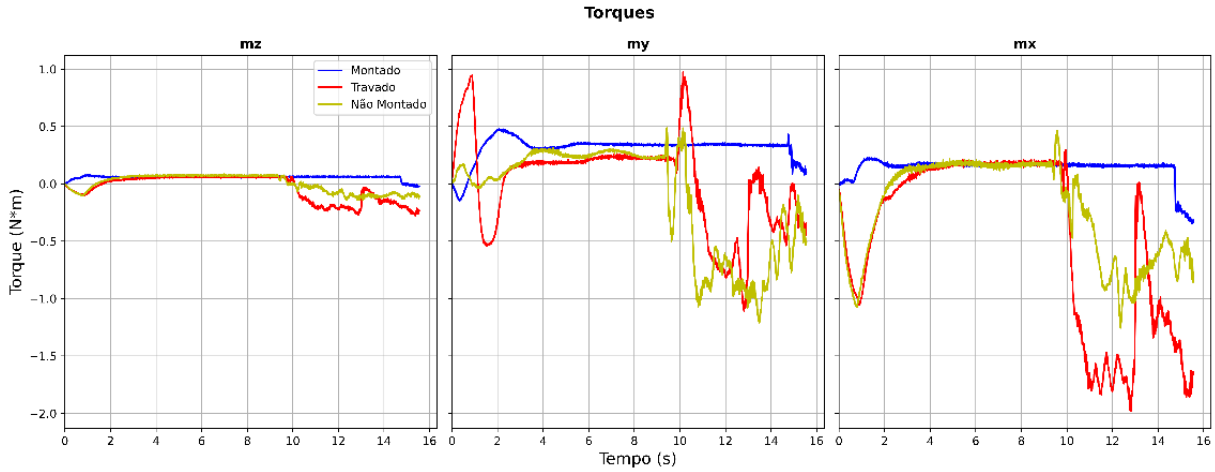

Figura 9 - Torques nos eixos x, y e z de cada classe de experimento

O conjunto de dados foi dividido entre um de treino e de teste pela razão 4:1, respeitando a proporção de cada classe nos exemplares originais. Em seguida foi aplicada a técnica SMOTe para criar o conjunto nivelado de todas as classes, mantendo a quantidade original de exemplos da classe preponderante.

Com os dados balanceados foi gerado um conjunto com número de dados quadruplicados. Estes vão ser utilizados para averiguar o impacto no aumento de dados na performance dos modelos em questão. A *tabela 1* mostra a quantidade de dados relativa a cada conjunto.

Tabela 1 - Número de exemplares de cada classe nos conjuntos de dados

	Original	**Nivelado**	**Quadruplicado**	**Teste**
Montado	243 (64%)	243	972	61 (64%)
Travado	49 (13%)	243	972	12 (13%)
Não Montado	88 (23%)	243	972	23 (23%)
Total	380	729	2916	96

4.2 AWS e MLOPs

Os modelos de DL são considerados computacionalmente custosos devido a plenitude de parâmetros que os compõem somados com aqueles da etapa de pré-processamento. Como discutido, é necessária a otimização desses parâmetros através de repetidas sessões de treino em grandes volumes de dados o que torna a complexidade temporal exponencial [29].

A fim de reduzir o tempo de treinamento e deixar a otimização dos parâmetros dos modelos mais eficiente, a implementação de técnicas de concorrência para processamento em *graphics processing units* (GPUs) se tornou prática padrão em aplicações de DL [39]. Aliado à necessidade do paralelismo, uma infraestrutura autônoma também o é para se manter o funcionamento dos modelos com desempenho consistente, bem como ter agilidade nas análises de diferente modelos e configurações de parâmetros [40].

A combinação das aplicações de ML com ferramentas de automação de engenharia de software constitui o que se chama de *Machine Learning Operations* (MLOps). O conjunto de práticas e diretrizes operativas de MLOps englobam toda a cadeia de treinamento dos modelos até o seu uso em produção e monitoramento de seus desempenhos [40]. A *figura 10* mostra uma típica cadeia de operações dos modelos.

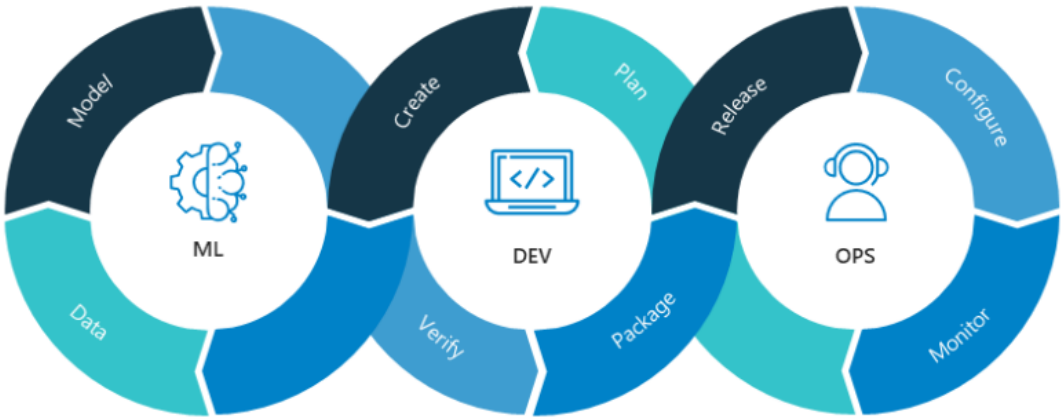

Figura 10 - Ciclo de operação de aplicações de ML

Esse conjunto de técnicas ajudam a simplificar a cadeia de processos do treinamento de modelos e a sua automação traz escalabilidade na criação de estudos de modelos e na execução desses para ensaios em produção. Também se aplica o método de *Continuous Integration / Contininuous Delivery* (CI/CD) para que toda as modificações realizadas sejam facilmente implementadas na arquitetura mantendo a execução dos estudos contínua. Nesse contexto, MLOPs tem como papel fundamental garantir escalabilidade e estabilidade em processos de ML.

A fim de se averiguar a viabilidade de um futuro sistema de DL para previsão de falhas, uma arquitetura em nuvem na Amazon Web Services (AWS) foi desenvolvida para satisfazer os requerimentos discutidos acima. A *figura 11* é um esquemático da arquitetura em nuvem.

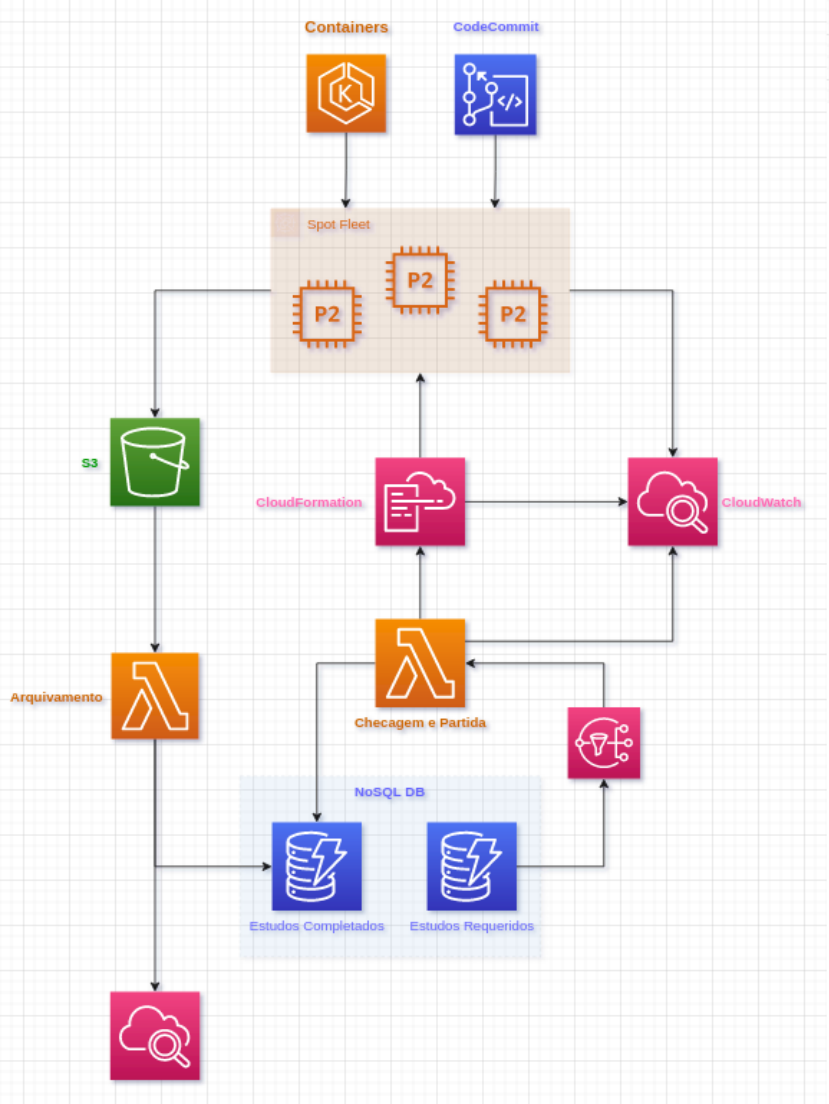

Figura 11 - Arquitetura AWS para produção de modelos

Segue uma breve explicação dos recursos e serviços:

- **Lambda**: Módulos *serverless* que executam *scripts* sem a necessidade de configurar um servidor dedicado.
- **DynamoDB**: Banco de dados não relacional (NoSQL) de valores-chave
- **S3**: Serviço de armazenamento de arquivos
- **Simple notification service (SNS)**: Serviço de envio de mensagens
- **CloudFormation**: Serviço que gerencia a criação de novos recursos da nuvem
- **CloudWatch**: Serviço de monitoramento de recursos da nuvem

- **CodeCommit**: Gerenciador de repositórios git
- **Instâncias EC2 SPOT**: máquinas dedicadas instanciadas por leilão
- **Elastic Conatiner Registry (ECR)**: Serviço de gerenciamento de *containers*

O processo de treinamento e criação de modelos acontece da seguinte forma. Todos os estudos requeridos pelas pesquisas serão salvos em formato *Javascript Object Notation* (JSON) na tabela "Estudos Requeridos" do dynamodb, que por sua vez acionara o SNS para ativação do lambda de checagem de estudos. Esse irá verificar se o estudo já foi realizado comparando as tabelas "Estudo Requeridos" e "Estudos Completados", caso seja realmente um novo estudo, o lambda acionará o *CloudFormation* para instanciar as máquinas *spot* com GPUs. *Containers* com ambiente e bibliotecas configurados para treinar os modelos do *keras* tem suas imagens salvas no ECR e são utilizadas após a confirmação da inicialização das instancias, dando procedimento ao treinamento e otimização dos modelos. O processo de treino e otimização é definido nas configurações especificadas no documento JSON. Por fim, as máquinas irão salvar os melhores modelos no S3 e dar início ao processo de desligamento. Por fim, o S3 ativará o lambda que irá confirmar a execução do estudo e registrar as informações desse na tabela Dynamo "EstudosCompletados".

5 TREINAMENTO DOS MODELOS

Duas estratégias foram adotadas para se fazer a análise de desempenho, treinar os modelos em conjunto de dados de diferentes tamanhos e utilizar modelos de diferentes complexidades em cada um dos conjuntos.

Todo o processo de treinamento foi realizado em ambiente *Python*. A etapa de preparação dos conjuntos de dados foi realizada com auxílio da biblioteca *Scikit-Learn.* Para a construção dos modelos a biblioteca *Keras e* o *tensorflow 2.0* foi escolhido como seu *backend* de processamento, sendo o treinamento realizado em instâncias *Spot* do tipo P da AWS.

Para os conjuntos de dados apresentados na seção 4.1 foi otimizado os hiperparâmetros de cada modelo com a técnica de *cross-validation*, dividindo o conjunto de treino em cinco partes sendo uma utilizada na validação dos parâmetros escolhidos.

Para cada hiperparâmetro foi estabelecido um intervalo de busca, e em cada processo iterativo um valor randômico era selecionado para o estudo, sendo nenhum repetido em estudos consecutivos. Para gerenciá-los, foi utilizada a biblioteca *Optuna,* que tem como padrão o algoritmo de busca *Tree-structured Parzen Estimator.*

Seguem os intervalos de busca analisados:

- Neurônios por camada: [1, 2048]
- Número de camada: [1, 10]
- *Dropout:* [0, 0.5]
- *Batch Normalization:* Gerenciado pelo *Optuna*
- Função de ativação: ReLU nas camadas intermediárias e *softmax* na última.
- Número de camadas por filtro: {32, 64}
- Tamanho do kernel: {1, 3, 5}
- *Stride:* Fixo em 1
- *Padding:* Padrão *'same'* do Keras, que preenche neurônios com zero
- Camadas *transformer:* [1,8]
- Número de *heads:* {2, 4, 6, 8}

- Tamanho do *embedding:* {8, 16}

A precisão na classe de montado foi escolhida como a função objetivo da otimização, pois foi considerado que os falsos positivos de montados em travados são potencialmente mais danosos caso ocorram na operação. Ao fim da otimização, os modelos foram retreinados e seus melhores hiperparâmetros salvos em formato JSON. A *figura 12* representa o fluxograma do processo de treinamento e otimização.

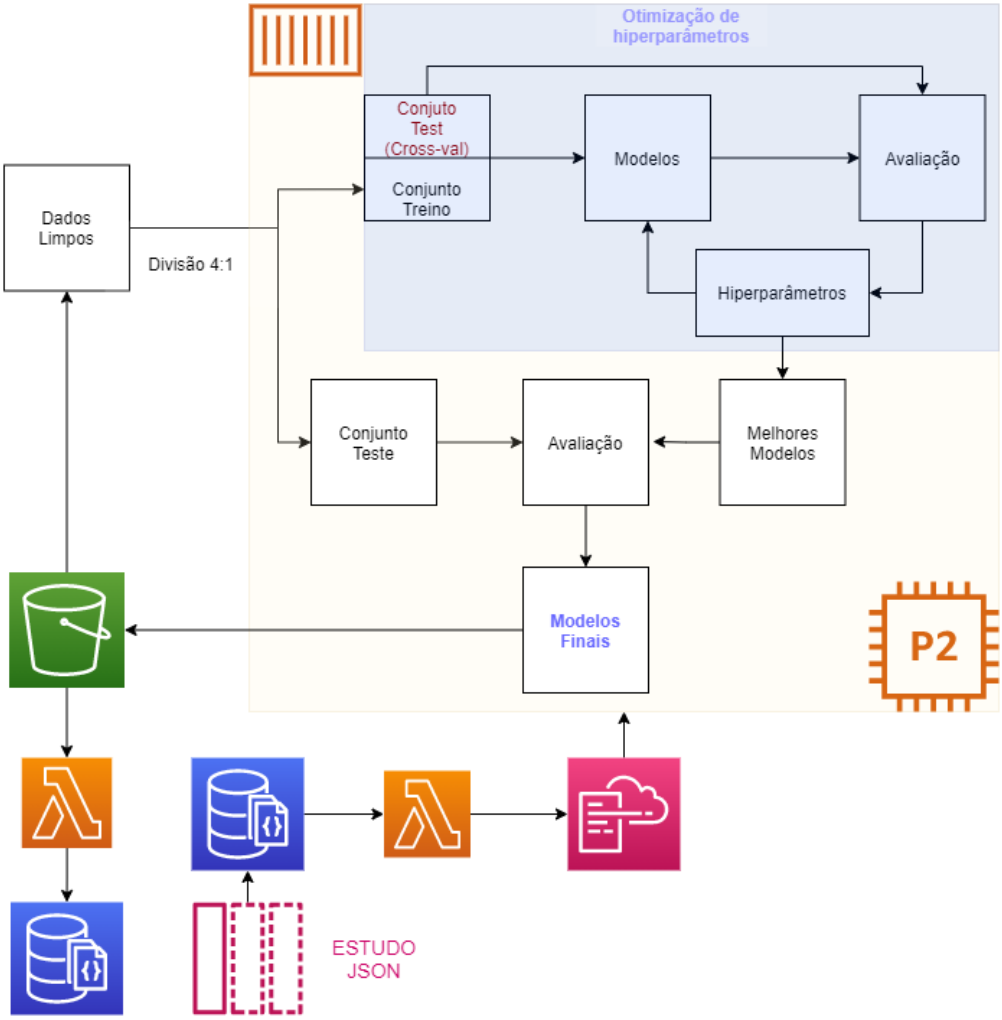

Figura 12 - Fluxograma do processo de treinamento

6 DISCUSSÃO

A primeira análise foi feita em relação ao desempenho dos modelos treinados conjuntos de diferentes tamanhos. Para isso foi calculado a precisão média de cada classe para cada combinação {conjunto, modelo}. Como discutido na seção 5, a precisão foi considerada a métrica mais relevante para o sistema de previsão. A *figura 13* mostra a evolução da precisão dos modelos.

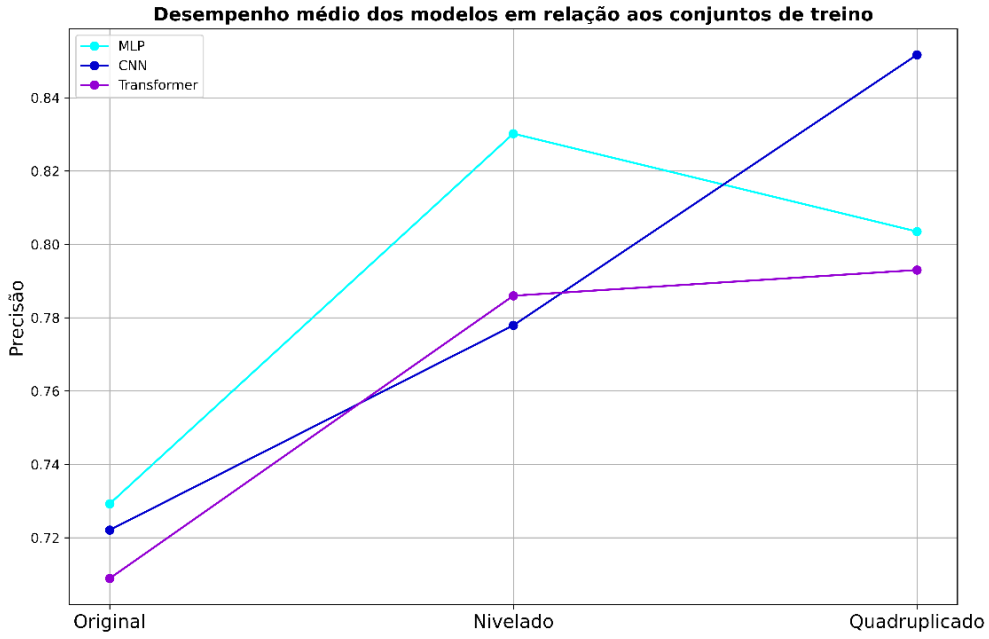

Figura 13 - Precisão média das combinações (conjunto, modelo)

Todos os modelos apresentaram melhora significativa ao serem treinados no conjunto nivelado em relação ao original. Já com o conjunto quadruplicado apenas a CNN teve um ganho na precisão enquanto a MLP deteriorou e a *Transformer* ficou praticamente estacionária. A *tabela 2* contém média das precisões com a proporção de desempenho ganho a cada aumento do conjunto de dados.

Tabela 2 - Precisão média de todas as classes

	MLP	CNN	Transformer
Original	0.73	0.72	0.71
Nivelado	0.83 (14%)	0.78 (8%)	0.79 (11%)
Quadruplicado	0.80 (-3%)	0.85 (10%)	0.79 (1%)

Estratificando as precisões por classe é possível perceber que o impacto dos conjuntos no desempenho foi majoritariamente devido à classificação de travados. A *figura 14* evidencia a melhora em relação à essa classe e mostra que a CNN teve resultados melhores que os outros modelos.

Todos os modelos ou permaneceram, ou deterioraram seus ganhos levemente na classe de "Não montado", e na classe de montados tiveram ganhos pequenos quando comparados aos de travado.

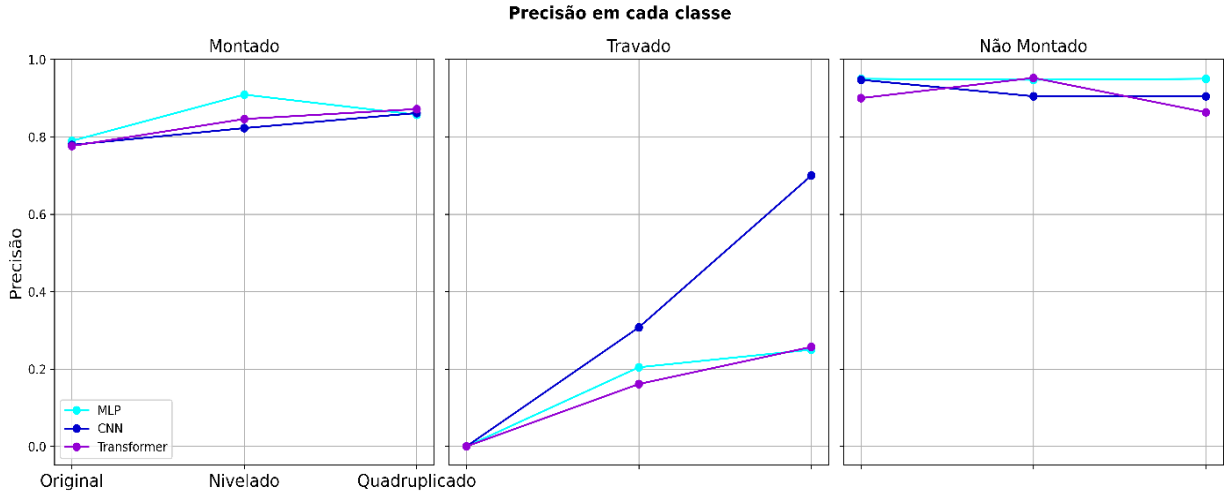

Figura 14 - Precisão estratificada por classe

Para investigar os ganhos em relação a classe "travado", foi utilizado a métrica F1 dos modelos para eliminar a preponderância da precisão em relação à sensibilidade, analisando a média dessas métricas, como mostrado na *equação 7.* Assim pode-se observar o desempenho da CNN isoladamente.

Tabela 3 - Valores de F1 para classe "travado"

	MLP	CNN	Transformer
Original	0	0	0
Nivelado	0.32	0.32	0.27
Quadruplicado	0.31	0.64	0.38

O descolamento da CNN em relação aos outros modelos é evidenciado na última linha da *tabela 3*. Devido ao fato de a classe “travado” possuir apenas doze exemplos no conjunto de testes, essa está suscetível a ter grandes variações no desempenho causado por mudanças marginais. Para verificar esse fenômeno, subtraiu-se uma unidade verdadeiro positivo das duas classes minoritárias “travado” e “não montado” na matriz de confusão da CNN (*tabela 4*) do conjunto quadruplicado.

Tabela 4 - Matriz de confusão da combinação (Quadruplicado, CNN)

		Previstos		
		Montado	**Travado**	**Não montado**
Reais	**Montado**	56	3	2
	Travado	5	7 (- 1)	0 (+ 1)
	Não montado	4	0 (+ 1)	19 (- 1)

Em seguida foi recalculada a métrica F1 na *tabela 2* através das equações 5, 6 e 7 com resultado de 0.778 para a CNN treinada no conjunto quadruplicado. A *figura 15* exibe as precisões médias novamente, mas agora a CNN com deslocamento marginal de uma unidade. Pode-se perceber que o descolamento da CNN em relação aos outros modelos era devido à escassez de classes de “travado”.

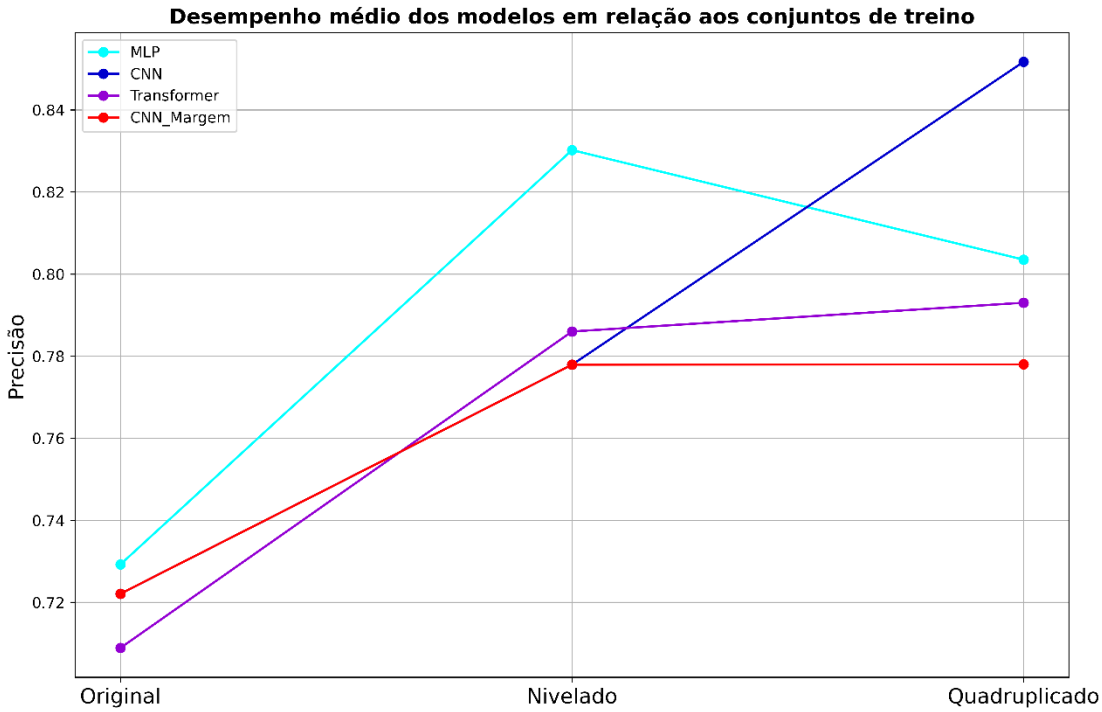

Figura 15 - Média das precisões com erro marginal da CNN

A estagnação do desempenho no conjunto quadruplicado em relação à classe “travado” pode ser explicada por uma deficiência de representatividade do conjunto original, pois a sintetização dos dados por SMOTe usa como referência a distribuição dos dados reais. Portanto se houver enviesamento por dados anômalos, mal classificados ou falta de representatividade, esse erro vai ser propagado para o conjunto artificial.

Com a análise sobre a variação do tamanho dos conjuntos revelando um possível problema de enviesamento, foi feita a comparação do desempenho dos modelos, limitando-se a análise ao conjunto ao quadruplicado.

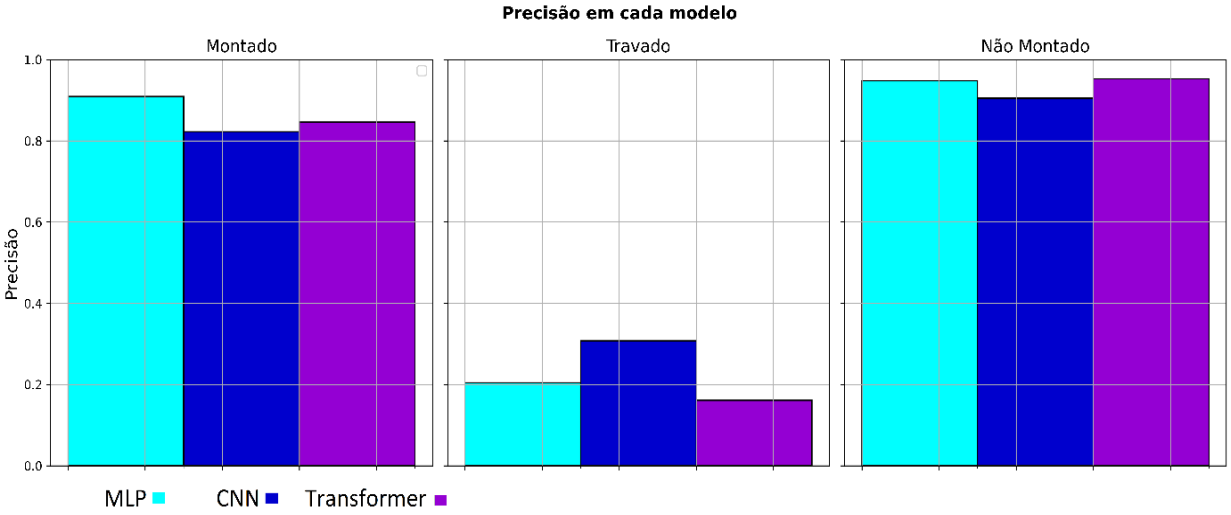

Figura 16 - Precisão de cada modelo para cada classe. Todos os modelos foram treinados apenas pelo conjunto quadruplicado.

A partir da *figura 16* pode-se perceber que não há uma correlação clara entre complexidade do modelo e desempenho. Isso significa que os modelos tiveram erro de generalização comparáveis, sugerindo que a limitação do desempenho não se deve à alta variância.

As análises de complexidade e tamanho de conjunto de dados revelam que existe um erro de enviesamento, pois não houve ganho de desempenho com o aumento do número de dados e aquele se mostrou indiferente à complexidade dos modelos. Portanto, a provável limitação do desempenho é a má representatividade dos dados, que, caso fosse adequada, era esperado uma melhora no desempenho dos modelos treinados no conjunto quadruplicado.

7 CONCLUSÃO

Este trabalho implementou a técnica SMOTe de *data augmentation* para balanceamento e geração de dados sintéticos, nos quais foram treinados os modelos MLP, CNN e transformer de DL. Com auxílio da AWS, os hiperparâmetros dos modelos foram otimizados em nuvem e em seguida foi feita análise do desempenho dos modelos otimizados em cada conjunto de dado obtido.

O balanceamento do conjunto de dados original foi responsável pelo maior desempenho ao analisar os modelos treinados no conjunto nivelado. A maior parte desse ganho ocorreu na melhora da precisão da classe "travado" que era a classe minoritária e mais sujeita ao enviesamento.

Porém, a posterior expansão de dados no conjunto quadruplicado não resultou em avanços significativos com exceção da CNN na classificação de travados. Mas, o descolamento no desempenho foi devido apenas a mudanças marginais da previsão correta da classe, que é resultado dos poucos exemplos de travados no conjunto de teste. Portanto, pode-se considerar que não houve ganhos relevantes no conjunto quadruplicado em relação às classes "travado" e "não montado".

Em seguida, a análise unilateral dos modelos revelou que não houve nenhum impacto significativo no desempenho quando considerada a arquitetura do modelo, o que indica um possível erro por enviesamento e sustenta a hipótese de má representatividade dos dados. Com a estagnação do desempenho no conjunto quadruplicado e a sua invariância em relação aos modelos de diferentes complexidades, a provável causa da limitação do desempenho é um potencial enviesamento no conjunto de dados.

A migração do processo de treinamento para a nuvem agilizou a realização dos estudos permitindo que fossem treinados os modelos mais complexos nos conjuntos maiores em tempo plausível. A arquitetura desenvolvida também será utilizada para estudos futuros para serem treinados em escala, valendo-se da automação de toda a cadeia de processamento.

Nos estudos futuros será feita a comparação da mesma coleção de modelos, mas treinados em outros conjuntos de rosqueamento similares e será realizado novos experimentos com processo de amostragem mais rigoroso e assim ter dados mais representativos.

REFERÊNCIAS

[1] Jia, Z., Bhatia, A., Aronson, R. M., Bourne, D., & Mason, M. T. (2019). **A Survey of Automated Threaded Fastening**. *IEEE Transactions on Automation Science and Engineering*, *16*(1). https://doi.org/10.1109/TASE.2018.2835382

[2] Li, Z. (2015). **Robotics research for 3c assembly automation**. Accessed: Jan. Available: https://app.box.com/s/zcg8qqxt6fw6v4xz22h6

[3] Rigelsford, J. (2004). **Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development**. *Assembly Automation*, *24*(1). https://doi.org/10.1108/aa.2004.03324aae.002

[4] Wiedmann, S., & Sturges, B. (2006). **Spatial kinematic analysis of threaded fastener assembly**. *Journal of Mechanical Design, Transactions of the ASME*, *128*(1). https://doi.org/10.1115/1.2114909

[5] Aronson, R. M., Bhatia, A., Jia, Z., Guillame-Bert, M., Bourne, D., Dubrawski, A., & Mason, M. T. (2017). **Data-Driven Classification of Screwdriving Operations**. In *Springer Proceedings in Advanced Robotics* (Vol. 1). https://doi.org/10.1007/978-3-319-50115-4_22

[6] Aronson, R. M., Bhatia, A., Jia, Z., & Mason, M. T. (2017). **Data collection for screwdriving**. *Robotics Science and Systems, Workshop on (Empirically) Data-Driven Manipulation*, *1*(2), 1–6.

[7] Pipino, L. L., Lee, Y. W., & Wang, R. Y. (2002). **Data quality assessment**. *Communications of the ACM*, *45*(4), 211–218.

[8] Goodfellow, I., Bengio, Y., & Courville, A. (2016). ***Deep learning***. MIT press.

[9] Sejnowski, T. J. (2020). **The unreasonable effectiveness of deep learning in artificial intelligence**. *Proceedings of the National Academy of Sciences of the United States of America*, *117*(48). https://doi.org/10.1073/pnas.1907373117

[10] Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018**). Deep Learning for Computer Vision: A Brief Review**. *Computational Intelligence and Neuroscience*. https://doi.org/10.1155/2018/7068349

[11] Otter, D. W., Medina, J. R., & Kalita, J. K. (2021). **A Survey of the Usages of Deep Learning for Natural Language Processing**. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(2). https://doi.org/10.1109/TNNLS.2020.2979670

[12] Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019). **Deep learning for time series classification: a review**. *Data Mining and Knowledge Discovery*, *33*(4). https://doi.org/10.1007/s10618-019-00619-1

[13] Längkvist, M., Karlsson, L., & Loutfi, A. (2014). **A review of unsupervised feature learning and deep learning for time-series modeling**. *Pattern Recognition Letters*, *42*(1). https://doi.org/10.1016/j.patrec.2014.01.008

[14] Taud, H., & Mas, J. F. (2018). **Multilayer perceptron (MLP)**. In *Geomatic Approaches for Modeling Land Change Scenarios* (pp. 451–455). Springer.

[15] Hayou, S., Doucet, A., & Rousseau, J. (2019). **On the impact of the activation function on deep neural networks training**. In *36th International Conference on Machine Learning, ICML 2019* (Vol. 2019-June).

[16] Aurélien Géron. (2019). ***Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems***. *O'Reilly Media*.

[17] Agarap, A. F. (2018). **Deep learning using rectified linear units (relu)**. *ArXiv Preprint ArXiv:1803.08375*.

[18] Naser, M. Z., & Alavi, A. (2020). **Insights into performance fitness and error metrics for machine learning**. *ArXiv Preprint ArXiv:2006.00887*.

[19] Rubinstein, R. Y., & Kroese, D. P. (2013). ***The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning***. Springer Science \& Business Media.

[20] Chen, L., Wang, S., Fan, W., Sun, J., & Naoi, S. (2016). **Beyond human recognition: A CNN-based framework for handwritten character recognition**. *Proceedings - 3rd IAPR Asian Conference on Pattern Recognition, ACPR 2015*. https://doi.org/10.1109/ACPR.2015.7486592

[21] Shin, H. C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., & Summers, R. M. (2016). **Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning**. *IEEE Transactions on Medical Imaging*, *35*(5). https://doi.org/10.1109/TMI.2016.2528162

[22] Botalb, A., Moinuddin, M., Al-Saggaf, U. M., & Ali, S. S. A. (2018). **Contrasting Convolutional Neural Network (CNN) with Multi-Layer Perceptron (MLP) for Big Data Analysis**. *International Conference on Intelligent and Advanced System, ICIAS 2018*. https://doi.org/10.1109/ICIAS.2018.8540626

[23] Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., & Chao, L. S. (2020). **Learning deep transformer models for machine translation**. *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*. https://doi.org/10.18653/v1/p19-1176

[24] Kobayashi, G. (2021). **Attention is Not Only a Weight: Analyzing Transformers with Vector Norms**. *Journal of Natural Language Processing*, *28*(1). https://doi.org/10.5715/jnlp.28.292

[25] Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. (2020). **Efficient transformers: A survey**. *ArXiv Preprint ArXiv:2009.06732*.

[26] Hossin, M., & Sulaiman, M. N. (2015). **A review on evaluation metrics for data classification evaluations**. *International Journal of Data Mining \& Knowledge Management Process*, *5*(2), 1.

[27] Seliya, N., Khoshgoftaar, T. M., & Van Hulse, J. (2009). **A study on the relationships of classifier performance metrics**. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*. https://doi.org/10.1109/ICTAI.2009.25

[28] Buckland, M., & Gey, F. (1994). **The relationship between Recall and Precision**. *Journal of the American Society for Information Science*, *45*(1). https://doi.org/10.1002/(SICI)1097-4571(199401)45:1<12::AID-ASI2>3.0.CO;2-L

[29] Feurer, M., & Hutter, F. (2019). **Hyperparameter optimization**. In *Automated machine learning* (pp. 3–33). Springer, Cham.

[30] Berrar, D. (2019). ***Cross-Validation***.

[31] Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). **Reconciling modern machine-learning practice and the classical bias–variance trade-off**. *Proceedings of the National Academy of Sciences of the United States of America*, *116*(32). https://doi.org/10.1073/pnas.1903070116

[32] Geman, S., Bienenstock, E., & Doursat, R. (1992). **Neural networks and the bias/variance dilemma**. *Neural Computation*, *4*(1), 1–58.

[33] Al-Jarrah, O. Y., Yoo, P. D., Muhaidat, S., Karagiannidis, G. K., & Taha, K. (2015). **Efficient Machine Learning for Big Data: A Review. *Big Data Research***. https://doi.org/10.1016/j.bdr.2015.04.001

[34] Shorten, C., & Khoshgoftaar, T. M. (2019). **A survey on Image Data Augmentation for Deep Learning**. *Journal of Big Data*, *6*(1). https://doi.org/10.1186/s40537-019-0197-0

[35] Iwana, B. K., & Uchida, S. (2020). **An empirical survey of data augmentation for time series classification with neural networks**. *ArXiv Preprint ArXiv:2007.15951*.

[36] Fernández, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). **SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary**. *Journal of Artificial Intelligence Research*, *61*, 863–905.

[37] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). **SMOTE: Synthetic minority over-sampling technique**. *Journal of Artificial Intelligence Research*, *16*. https://doi.org/10.1613/jair.953

[38] Moreira, G. R., Lahr, G. J. G., Boaventura, T., Savazzi, J. O., & Caurin, G. A. P. (2018). **Online prediction of threading task failure using Convolutional Neural Networks**. In *IEEE International Conference on Intelligent Robots and Systems*. https://doi.org/10.1109/IROS.2018.8594501

[39] Ben-Nun, T., & Hoefler, T. (2019). **Demystifying parallel and distributed deep learning: An in-depth concurrency analysis**. *ACM Computing Surveys*, *52*(4). https://doi.org/10.1145/3320060

[40] Mäkinen, S., Skogström, H., Laaksonen, E., & Mikkonen, T. (2021). **Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?**