UNIVERSIDADE FEDERAL DA GRANDE DOURADOS- UFGD

FACULDADE DE CIÊNCIAS EXATAS E TENOLOGIA

CURSO DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

ALESSANDRO ISHY MEDEIROS

TUIUIÚ DEEP LEARNING: A DEEP LEARNING DEVELOPMENT ENVIRONMENT

DOURADOS/MS

2021

ALESSANDRO ISHY MEDEIROS

TUIUIÚ DEEP LEARNING: A DEEP LEARNING DEVELOPMENT ENVIRONMENT

Orientador: Joinvile Batista Junior

Área de Concentração: Ciência da Computação

Dourados/MS

2021

Tuiuiú Deep Learning: A Deep Learning Development Environment

Joinvile Batista Jr[a], Alessandro I. Medeiros[a]

[a] *Faculdade de Ciências Exatas e Tecnologias, Universidade Federal da Grande Dourados, Dourados, Brasil*

INFORMAÇÕES

Palavras-chave:
Deep Learning Environment

Resumo

Deep learning tem alcançado o estado da arte em aprendizado de máquinas em campos tais como processamento de imagens, visão computacional, reconhecimento de fala, processamento de linguagem natural, robótica, segurança cibernética, e muitos outros. No entanto, a curva de aprendizado para iniciantes é um grande desafio para a sua utilização de forma efetiva. Neste contexto, ferramentas que suportem a execução automática de DNNs (*Deep Neural Networks*), a partir de especificações descritivas ou via interface gráfica, tem um papel decisivo para rápida prototipagem e para aprendizagem de projetos de DNNs, para potencializar o reuso das funcionalidades suportadas por frameworks tais como TensorFlow, PyTorch, Caffe e outros. Neste artigo é reportado o desenvolvimento do Tuiuiú Deep Learning, um ambiente web que suporta especificação, pré-processamento, treinamento e teste de projetos de DNNs, construídas com base nas funcionalidades fornecidas pelos frameworks TensorFlow e PytTorch. Disponibiliza uma interface gráfica amigável com suporte à: visualização e edição da especificação em XML; visualização e configuração de parâmetros das fases de desenvolvimento de DNNs; e visualização das métricas de qualidade resultantes das fases de treinamento e teste.

1. Introdução

Deep learning tem alcançado o estado da arte em aprendizado de máquinas em campos tais como processamento de imagens, visão computacional, reconhecimento de fala, processamento de linguagem natural, robótica, segurança cibernética, e muitos outros [2]. Sua utilização já ultrapassou a fronteira da pesquisa para se tornar realidade em várias aplicações de sucesso do mundo real [3]. No entanto, a curva de aprendizado para iniciantes é um grande desafio para a sua utilização de forma efetiva [5]. Particularmente o ensino de deep learning é de grande importância para geração de profissionais que serão absorvidos por um mercado de trabalho cada vez mais promissor. Neste contexto, ferramentas que suportem a execução automática de DNNs (*Deep Learning Networks*), a partir de especificações descritivas ou via interface gráfica, tem um papel decisivo para rápida prototipagem e para aprendizagem de projetos de DNNs, para potencializar o reuso das funcionalidades suportadas por frameworks tais como TensorFlow [1], PyTorch [17], Caffe [11] e outros.

Neste artigo é reportado o desenvolvimento do Tuiuiú Deep Learning, um ambiente web que suporta especificação, pré-processamento, treinamento e teste[1] de projetos de DNNs, construídas com base nas funcionalidades fornecidas pelos frameworks TensorFlow e PyTorch. O nome Tuiuiú foi inspirado em uma ave que é considerada a ave símbolo do Pantanal, uma região do Brasil com grande biodiversidade.

O Tuiuiú Deep Learning é utilizado para gerar projetos de DNNs, nos contextos de pesquisa ou educacional com a especificação de projetos em arquivo XML (*eXtensible Markup Language*), disponibilizando uma interface gráfica amigável que suporta a visualização e a alteração da especificação em XML, bem como a visualização e a configuração de parâmetros dos artefatos gerados nas fases de pré-processamento. Adicionalmente, é possível alterar os hiperparâmetros[2] de treinamento via interface gráfica, e visualizar a configuração sequencial ou paralela da rede neural, com a possibilidade de selecionar nós da rede para visualização detalhada de sua especificação. Na especificação de cada projeto, descrita em arquivo XML, são definidos os parâmetros que determinam: como obter os dados de entrada e como configurar uma DNN e definir seus hiperparâmetros para dados no formato tabular, imagens, ou texto.

Os pontos fortes da abordagem empregada em Tuiuiú Deep Learning são: (a) a flexibilidade e generalidade de especificar projetos de DNNs em formato XML, dispensando a necessidade de preenchimento de formulários ou da utilização de editores gráficos; (b) a sua padronização à entrada de dados tabulares e imagens; (c) o seu suporte a pesquisadores e a propósitos educativos abrangendo todas as fases de execução de DNNs; e (d) o seu suporte à importação e exportação de código.

Na seção 2, serão descritos os trabalhos correlatos, com base em critérios de comparação. Na seção 3 são descritas as características do Tuiuiú Deep Learning, abordando em subseções: a expressividade e a edição gráfica de sua especificação descritiva em formato XML; o seu suporte à execução das fases de pré-processamento, treinamento, avaliação e teste; suas facilidades de importação e exportação de código; e diferenças do tratamento de imagens e texto em relação a dados tabulares. Na seção 4, são descritas as conclusões e propostas de trabalhos futuros.

[1] Na fase de treinamento a rede os pesos da rede são reajustados para aprender a predizer a partir de um conjunto de exemplos. Os pesos gerados na fase de treinamento são utilizados para predizer a partir de um novo conjunto de exemplos na fase de teste.

[2] Hiperparâmetros são utilizados na configuração das camadas da rede e no gerenciamento das fases de treinamento, validação e teste da rede. O total de vezes que a rede é submetida aos dados de treinamento (*nepochs*) e o tamanho de cada subgrupo de exemplos submetidos à rede (*batch size*) são exemplos de hiperparâmetros.

2. Trabalhos Correlatos

Tuuiuí DLE foi desenvolvido com dois principais objetivos. O primeiro objetivo é o suporte ao desenvolvimento de DNNs baseado em uma especificação a partir da qual a rede é executada automaticamente. Para tornar essa abordagem genérica para qualquer projeto de DNN, adaptadores com código fornecidos pelo usuário podem ser utilizados, para complementar o modelo executável gerado automaticamente a partir da especificação, em casos em que a implementação da obtenção de dados de entrada ou de configuração da DNN não puder ser completamente mapeada a partir da especificação em XML. O segundo objetivo é suporte à execução e visualização dos resultados das fases de pré-processamento de dados, treinamento e teste de projetos de DNNs, complementados pela exportação de código para execução por interpretadores Python.

Para comparar trabalhos com objetivos semelhantes, nesta seção são relacionados trabalhos correlatos que geram automaticamente um modelo ou o código executável a partir de especificações descritivas ou de uma interface gráfica. Para tornar a comparação mais objetiva, é descrito sucintamente como cada ferramenta provê as seguintes características: (a) frameworks suportados; (b) tipo de especificação utilizada; (c) opção de complementar o código gerado automaticamente com código fornecido pelo usuário; (d) exportação de código; (e) tipos de dados de entrada suportados; (f) geração automática de dados; (g) visualização e edição da especificação; (h) visualização dos dados de entrada; (i) pré-processamento de dados; (j) visualização e customização da rede.

WekaDeeplearning4j [14] é uma ferramenta desktop que agrega ao ambiente Weka [6], funções de treinamento de rede neural. Suas características são: (a) framework suportado: DeepLearning4j; (b) tipo de especificação: composta pela descrição do dataset em um arquivo ARFF (attribute-relation file format), que define classes e valores, complementada pela definição de parâmetros da rede fornecidos a partir de uma interface gráfica; (c) código externo complementar: não relatado; (d) exportação de código: não relatado; (e) tipos de dados de entrada : dados tabulares descritos no formato ARFF, texto e arquivos de imagem ou texto com suas localizações descritas no arquivo ARFF; (f) geração automática de dados: geração de dados com colunas numéricas e nominais; (g) visualização e edição da especificação: edição dos dados do arquivo arrf através de uma tabela editável; (h) visualização dos dados de entrada: visualização e análise dos dados através de tabelas e gráficos com estatísticas; (i) pré-processamento de dados: a aplicação Weka disponibiliza filtros para o processamento de dados; (j) visualização e customização da rede: combinações sequenciais de um conjunto de camadas (*layers*) disponibilizadas, com telas específicas para a customização de parâmetros dessas camadas.

Expresso [4] é uma ferramenta desktop com interface gráfica para o desenvolvimento de redes neurais, com foco em redes neurais convolucionais. Suas características são: (a) framework suportado: Caffe; (b) baseada em arquivos prototx do Caffe com configuração dos dados da entrada e modelo da rede via interface gráfica; (c) código externo complementar: não relatado; (d) exportação de código: não relatado; (e) tipos de dados de entrada: texto, LevelDB, matrizes do MATLAB, HDF5, imagens; (f) geração automática de dados: não relatado; (g) visualização e edição da especificação: edição de arquivo prototxt; (h) visualização dos dados de entrada: interface gráfica para a entrada de dados e navegador para visualização de imagens; (i) pré-processamento de dados: o que Caffe suporta através de camadas de Data; (j) visualização e customização da rede: interface gráfica para ajustes e treinamento da rede e para manipulação do modelo treinado.

Barista [12] é uma ferramenta web para o desenvolvimento e monitoramento de redes neurais. Suas características são: (a) framework suportado: Caffe; (b) tipo de especificação: interface gráfica para especificar dados de entrada, modelos e parâmetros da rede; (c) código externo complementar: necessário criar uma camada de código ao framework Caffe; (d) exportação de código: não relatado; (e) tipos de dados de entrada : formatos associados aos data layers do Caffe; (f) geração automática de dados: não relatado; (g) visualização e edição da especificação: customizações via interface gráfica; (h) visualização dos dados de entrada: algumas informações como dimensões e número de itens; (i) pré-processamento de dados: no escopo do Caffe; (j) visualização e customização da rede: editor *drag and drop* para especificação do modelo da rede como combinação camadas sequenciais e paralelas.

NNC (*Neural Network Console*) [10] é uma ferramenta web que provê uma interface gráfica para suportar o desenvolvimento, monitoramento e escolha de redes neurais. Suas características são: (a) framework suportado: nnabla; (b) tipo de especificação: via interface gráfica; (c) código externo complementar: suporta importação de código; (d) exportação de código: suportado; (e) tipos de dados de entrada: CSV (*Comma Separated Values*), matrizes. ou imagens organizadas em diretórios; (f) geração automática de dados: não relatado; (g) visualização e edição da especificação: projeto especificado via interface gráfica; (h) visualização dos dados de entrada: suporta visualização de imagens; (i) pré-processamento de dados: suporta o pré-processamento de imagens; (j) visualização e customização da rede: editor gráfico do tipo arrasta e solta (*drag and drop*) para modelagem de rede sequencial e paralela, com customização de parâmetros via interface gráfica, e monitoramento com gráficos da evolução do treinamento e métricas dos resultados de teste.

PrototypeML [8] é uma ferramenta web que tem como objetivo específico a criação de modelos de redes neurais e a geração do código decorrente do modelo criado. Suas características são: (a) framework suportado: PyTorch; (b) tipo de especificação: representação gráfica de redes neurais como grafos de código de árvore de sintaxe para oferecer suporte a componentes que podem usar loops e lógica condicional e para encapsular segmentos de código PyTorch ou Python normal; (c) código complementar externo: suporta

Tabela 1
Ferramentas relacionadas

	WekaDeepLearning4J	Expresso	Barista	NNC	PrototypeML	Tuiuiú
Frameworks	DeepLearning4J	Caffe	Caffe	NNabla	PyTorch	TensorFlow, PyTorch
Formato próprio e interface gráfica	✓	✓	✓	✗	✗	✓
Formato próprio e editor gráfico	✗	✗	✓	✓	✓	✗
Importação de código	✗	✗	✗	✓	✓	✓
Exportação de código	✗	✗	✗	✓	✓	✓
Tipos de dados	✓	✓	✓	✓	✗	✓
Geração automática de dados	✓	✗	✗	✗	✗	✓
Visualização dos dados de entrada	✓	✓	✓	✓	✗	✓
Fase de pré-processamento	✓	✗	✗	✓	✗	✓
Visualização da rede Neural	✗	✗	✓	✓	✓	✓
Customização da rede Neural	✓	✓	✓	✓	✓	✓

a definição de código relacionado com seus componentes de encapsulamento de código; (d) exportação de código: suportado. As demais características não se aplicam.

O Tuiuiú Deep Learning atende todos os critérios de comparação e suporta todas as fases de execução de um projeto de DNN, para mais de um framework. Sua linguagem de especificação é versátil e viabiliza a utilização de todas as classes dos dois frameworks suportados, sem restrições impostas por formulários de interface gráfica, e dispensando a necessidade de uso de editor gráfico. Suporta edição e visualização da sua linguagem de especificação e dos hiperparâmetros da rede neural, bem como a visualização e inspeção da configuração sequencial ou paralela da rede neural. Suporta importação e exportação de código e facilidades com propósitos educativos.

3. Descrição

Tuiuiú Deep Learning é uma ferramenta web que provê suporte a todas as etapas de desenvolvimento de DNNs. A especificação de projetos de DNNs em XML, é bastante versátil cobrindo a definição do formato de leitura de dados, de parâmetros de pré-processamento de dados tabulares, imagens e texto, de percentuais de dados atribuídos às fases de treinamento, validação e testes, e da configuração e definição de hiperparâmetros de redes sequenciais e paralelas.

Sua interface gráfica amigável, complementa a funcionalidade de edição da especificação em XML, com as seguintes funcionalidades: seleção de projetos e suas versões, associadas a uma determinada configuração de pesos da rede, para execução; visualização das tabelas geradas na etapa de pré-processamento de dados tabulares, com opções configuração de paginação e de navegação nas páginas das tabelas; visualização da configuração da rede com opções de detalhamento da especificação de nós e layers; visualização e edição dos hiperparâmetros da rede; visualização da evolução das métricas, especificadas em XML, durante as fases de treinamento e teste; e visualização de imagens com predições incorretas, informando as predições incorretas para uma dada classe predita.

Inicialmente, o usuário seleciona o projeto que deseja abrir, na interface gráfica do Tuiuiú Deep Learning. Um projeto pode estar associado a várias versões de dados de configuração da rede treinada previamente, e o usuário tem a opção de escolher a versão a partir da qual deseja continuar o treinamento, ou simplesmente criar uma nova versão para realizar o treinamento desde o início.

Para facilitar o entendimento das funcionalidades suportadas, são descritas inicialmente as etapas envolvendo o processamento de dados no formato tabular e, posteriormente, as especificidades dessas etapas para imagens e textos.

3.1. Expressividade e Edição Gráfica da Especificação Descritiva

Dados tabulares podem ser automaticamente gerados em arquivos no formato CSV, a partir de uma especificação no formato XML. Para cada coluna são especificados: nome da coluna e tipo do conteúdo, que pode ser numérico (*integer* ou *float*), *boolean* ou categórico. Para dados numéricos são determinados valores mínimos e valores máximos. Para dados categóricos são especificados os valores possíveis de um tipo enumerado. Na especificação também é definido se valores numéricos ausentes em uma dada coluna deverão ser substituídos por valores médios, ou se a linhas que contém valores ausentes deverão ser removidas. Para valores não numéricos ausentes, a linha sempre é removida. Valores categóricos podem ser codificados como one-hot[3] ou como inteiros. Podem ser especificadas opções para a normalização[4] para dados categóricos codificados como inteiros ou como dados numéricos. Na Figura 1 e Figura 2 é ilustrada a especificação de uma rede neural com várias camadas para predizer riscos

[3]Uma categoria é um enumerado composto de uma lista de valores possíveis, como por exemplo, os valores de estado civil de uma pessoa. Na representação one-hot é associado um vetor com dimensão equivalente ao tamanho da lista. A representação de uma categoria tem valor 1 no índice do vetor correspondente ao índice da lista de valores da categoria e valor 0 para os demais índices. One-hot representa mais adequadamente em uma rede neural, valores de uma categoria que não tem uma relação de ordenação.

[4]Normalização é importante para que uma dada característica não tenha valores mais discrepantes em relação aos valores das demais características. Somente para conceituar um métodos mais simples de normalização, é a divisão de todos os valores de uma dada característica pelo seu maior valor.

```
<data type="feature_label" format="csv" ratio="8,1,1">
    <node name="I0" output="B">
        <feature name="age" content="int" min="18" max="80"
                 null_fill="mean" normalize="MinMax"/>
        <feature name="license_time" content="int"
                 min="0" max="10"
                 null_fill="mean" normalize="MinMax"/>
    </node>
    <node name="I1" output="A">
        <feature name="city_population"
                 content="small, medium, big, metropolitan"
                 normalize="MinMax" encoder="one_hot"/>
        <feature name="residence"
                 content="house, apt, complex, back_alley"
                 normalize="MinMax"/>
    </node>
    <node name="I2" output="B">
        <feature name="residence_garage" content="bool"/>
        <feature name="job_parking_spot" content="bool"/>
        <feature name="anti_theft_device" content="bool"/>
        <feature name="other_driver" content="bool"/>
    </node>
    <label name="risco" classes="high, medium, low"/>
</data>
```

Figura 1: Editor com especificação em XML dos dados de entrada do projeto.

```
<network learning_rate="0.01">
    <node name="A" output="C">
        <layer class_name="Dense" units="16" activation="'relu'"/>
        <layer class_name="Dense" units="32" activation="'relu'"/>
    </node>
    <node name="B" output="C">
        <layer class_name="Dense" units="8" activation="'relu'"/>
        <layer class_name="Dense" units="32" activation="'relu'"/>
    </node>
    <node name="C" output="D, E">
        <layer class_name="Dense" units="16" activation="'relu'"/>
    </node>
    <node name="D" output="F">
        <layer class_name="Dense" units="8" activation="'relu'"/>
        <layer class_name="Dense" units="32" activation="'relu'"/>
    </node>
    <node name="E" output="F">
        <layer class_name="Dense" units="8" activation="'relu'"/>
        <layer class_name="Dense" units="32" activation="'relu'"/>
    </node>
    <node name="F">
        <layer class_name="Dense" units="16" activation="'relu'"/>
        <layer class_name="Dense" units="3" activation="'softmax'"/>
    </node>
</network>
```

Figura 2: Editor com especificação em XML da rede neural do projeto.

de seguros de automóveis, com a especificação de colunas com conteúdo inteiro, booleano e categórico, no escopo da tag data.

A rede neural pode ser especificada como uma sequência de camadas, ou como um grafo com nós compostos de camadas sequenciais e múltiplas entradas. Como consequência da capacidade de representar redes neurais com nós em paralelo, as colunas da matriz de dados da entrada podem ser agrupadas em subgrupos associados a diferentes nós de

```
<S>
    <assign variable="risk" expression="0"/>
</S>
<S>
<if condition="(age GT 70) OR (age LTE 26)">
    <true>
        <S>
            <assign variable="risk" expression="risk + 1"/>
        </S>
    </true>
    <false>
        <S>
            <assign variable="risk"
                    expression="risk + 0.2"/>
        </S>
    </false>
</if>
</S>
<S>
<switch enumeration="city_population">
    <case value="small"/>
    <case value="medium">
        <S>
            <assign variable="risk" expression="risk + 0.3"/>
        </S>
    </case>
    <case value="big">
        <S>
            <assign variable="risk" expression="risk + 0.6"/>
        </S>
    </case>
    <case value="metropolitan">
        <S>
            <assign variable="risk" expression="risk + 1"/>
        </S>
    </case>
</switch>
```

Figura 3: Editor com especificação em XML do algoritmo da geração dos labels.

uma rede. Esse agrupamento dos dados de entrada em nós também pode ser observado na Figura 1, no escopo da tag data, composta por tags node.

Para a especificação da rede neural, cada nó é definido com o seu nome, com a lista de nomes de nós que recebem a sua saída como entrada, e com as suas camadas sequenciais internas. Para cada camada são especificados: o nome da classe e parâmetros relevantes para esta classe, como por exemplo: a quantidade de unidades de cada camada e a função de ativação. A especificação dos parâmetros de cada classe é dinâmica, de forma que o usuário tem a flexibilidade de definir todos os parâmetros de interesse, suportados pela classe utilizada provida pelo framework TensorFlow ou PyTorch. A especificação em XML de uma rede neural com nós em paralelo é ilustrada na Figura 2.

A interface gráfica do Tuiuiú DLE suporta a visualização e a edição da especificação do projeto no formato XML. As facilidades de edição e visualização de erros sintáticos e semânticos da especificação em XML, via interface gráfica, reforçam a versatilidade da utilização da especificação neste formato para o projeto de DNNs.

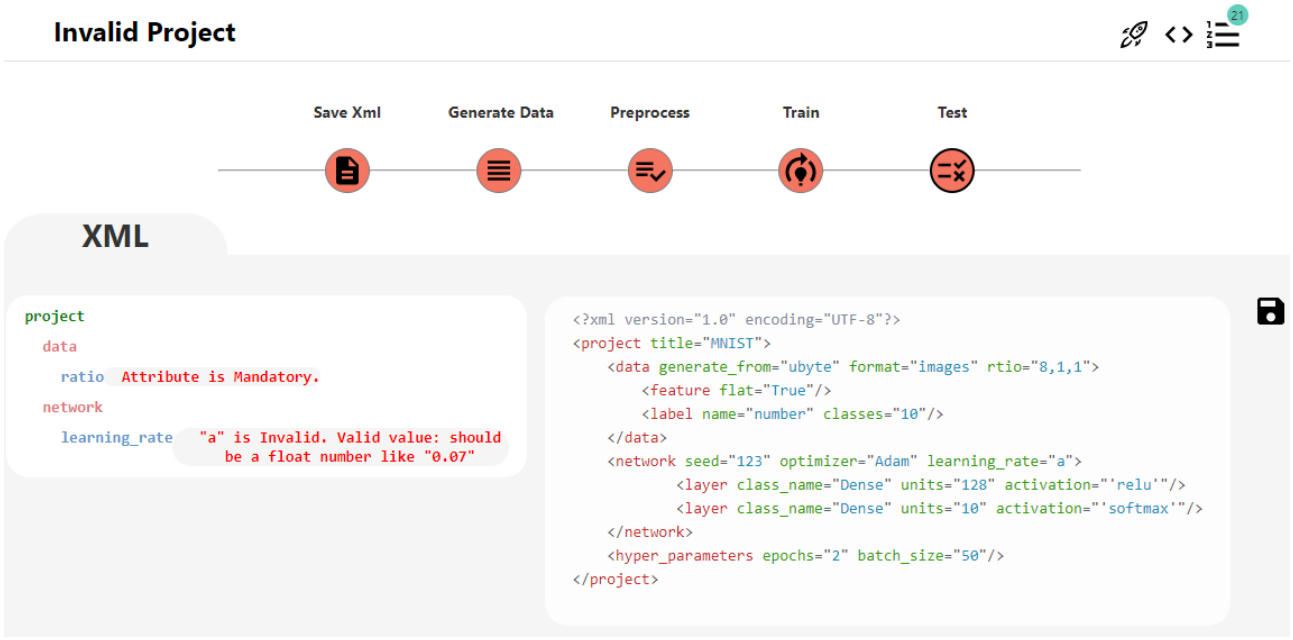

Figura 4: Especificação em XML para ilustrar detecção de erros.

A configuração de camadas e parametrização da rede de forma dinâmica, com plena utilização das funcionalidades e parâmetros disponibilizados pelos frameworks TensorFlow e PyTorch, não impõe restrições associadas à telas específicas de interface gráficas atreladas a subconjuntos de funcionalidades e parâmetros suportados pelos frameworks utilizados. Desta forma, o usuário pode especificar em XML qualquer camada suportada pelos frameworks utilizados, com passagem de qualquer parâmetro válido na linguagem Python, como por exemplo, números, *strings*, listas e dicionários.

Com propósitos educativos, provê geração automática de dados tabulares e algoritmos de geração de labels, especificados em XML. É possível especificar o algoritmo para geração dos labels dos dados tabulares, utilizando comandos de atribuição, condicionais e de iteração. Caso não deseje gerar seus dados tabulares automaticamente, o usuário poderá prover um arquivo CSV, com a representação dos dados no formato tabular. O algoritmo para o cálculo automático do label para cada linha de características gerada aleatoriamente é mostrado na Figura 3. Na Figura 4 é utilizada uma especificação mais compacta com foco na ilustração da identificação de erros de sintaxe ou semântica na descrição da especificação.

3.2. Execução de Pré-processamento, Treinamento, Avaliação e Teste

Após a geração automática dos dados tabulares, é realizada a etapa de processo de pré-processamento. Nesta etapa, as colunas categóricas são convertidas para valores numéricos, os eventuais valores omitidos em linhas da tabela são substituídos pelo valor médio da coluna, ou a linha com valores omitidos é descartada, e os valores numéricos das colunas são normalizados. Essas operações são realizadas de acordo com as especificações fornecidas para o projeto, no formato XML. A interface gráfica do Tuiuiú Deep Learning suporta a visualização das tabelas geradas no formato XML, de maneira iterativa, possibilitando ao usuário reconfigurar quantas linhas deseja visualizar por página na tabela, bem como, navegar nas páginas de cada tabela.

Após a etapa de pré-processamento, é realizada a etapa de treinamento. Conforme ilustrado na Figura 5, a interface gráfica do Tuiuiú Deep Learning suporta a visualização da atualização de métricas de qualidade (*accuracy*[5], *precision*[6], *recall*[7] e *F1 score*[8]) a cada batch realizado. O usuário tem a opção de alterar os hiperparâmetros de treinamento, para avaliar as alterações nos resultados do treinamento. Para torná-las permanentes, essas alterações deverão ser realizadas também na especificação no formato XML.

Na etapa de treinamento, o usuário pode visualizar a configuração da rede, sequencial ou paralela, conforme definida na especificação em XML. Os posicionamentos das arestas entre os nós são determinados automaticamente a partir da implementação de um algoritmo A* [9], que viabiliza a obtenção do caminho mais curto entre nós, evitando o cruzamento entre conexões e nós. A geração automática de conexões entre os nós de uma rede paralela, tornam dispensável a necessidade de uma interface gráfica baseada em um editor *drag and drop*.

O usuário pode selecionar qualquer nó ou camada da rede para visualizar os seus dados de configuração. Se for

[5]Acurácia: quantas predições estão corretas em relação a todas as predições.

[6]Precisão: quantas predições para uma dada classe estão corretas em relação a todas as predições para uma dada classe.

[7]Cobertura: quantas predições para uma dada classe estão corretas em relação a todos os valores reais de uma dada classe.

[8]Pontuação F1: média harmônica entre precisão e cobertura.

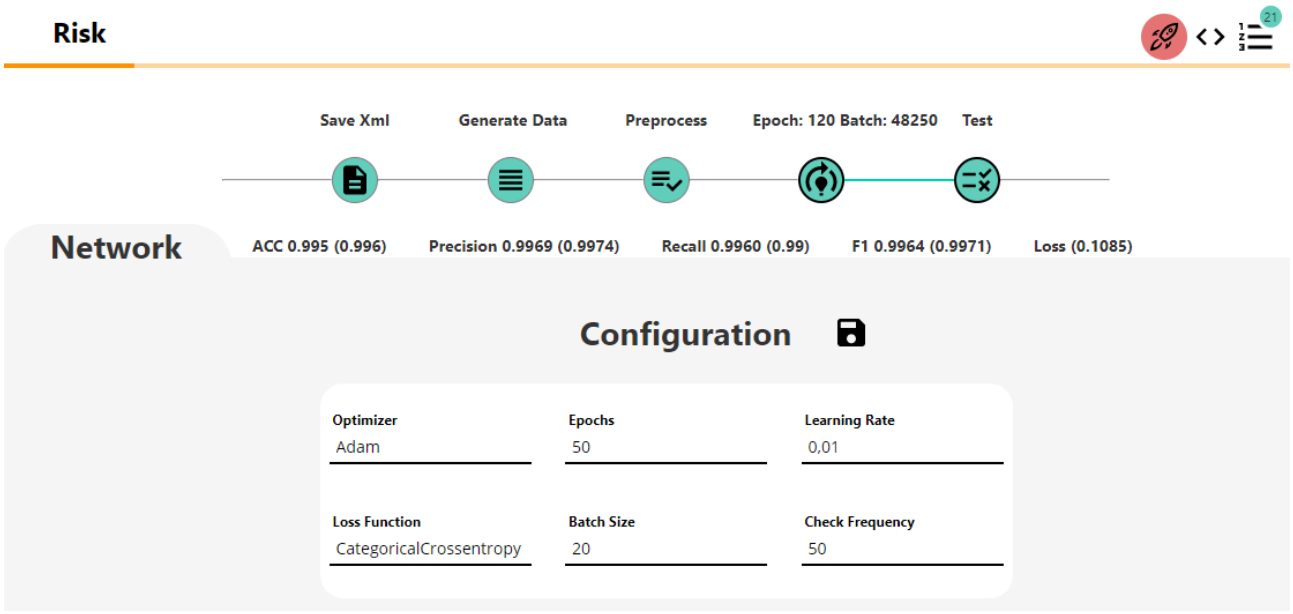

Figura 5: Execução da fase de treinamento e seus parâmetros.

um nó interno da rede, o que exclui o nó de entrada e o nó de saída, irá visualizar as camadas sequenciais que o compõem, podendo selecionar uma das camadas para visualizar sua configuração. Se for um nó de saída, irá visualizar a sua especificação. Se o nó for um nó de entrada deverá selecionar, adicionalmente, a coluna dos dados de entrada, cuja especificação deseja visualizar. Na Figura 6, é ilustrada uma rede com nós em paralelo e a seleção de um nó intermediário da rede, com o detalhamento de suas camadas e o subsequente visualização da especificação de uma de suas camadas. Na Figura 7 é caracterizada somente a região da imagem de saída relevante para ilustrar a seleção de uma dada coluna do nó de entrada e a visualização da especificação da coluna selecionada. De forma semelhante, na Figura 8 é ilustrada a seleção do nó de saída e da visualização da sua especificação. Na etapa de teste é mostrada a evolução dos cálculos das métricas de qualidade, incorporando também os valores gerados na etapa de treinamento.

Também na etapa de treinamento, baseado em uma periodicidade default, ou especificada pelo usuário, ocorre o salvamento dos pesos da rede sendo treinada, caso a perda seja menor do que a perda associada ao salvamento anterior. A cada nova execução do projeto, o usuário tem a opção de escolher uma versão do projeto associada a uma determinada configuração de pesos da rede gerada em treinamentos anteriores, ou executar o treinamento em uma nova versão desde o início.

3.3. Exportação e Importação de Código

Adicionalmente, o usuário tem a opção de solicitar a geração de código para exportação, para execução por interpretadores Python, como por exemplo, o Google Colab. Figura 9 mostra a definição do modelo no código gerado para o projeto especificado na Figura 2, Figura 10 mostra o carregamento de dados e a Figura 11 mostra o treinamento da rede neural.

Finalmente, pode ser necessário complementar o modelo executável gerado a partir da especificação XML. Para tratar essa questão, o Tuiuiú Deep Learning suporta a referência de adaptadores, substituindo camadas ou nós da rede por código Python implementado de acordo com o framework escolhido, o que permite encapsular operações que não podem ser mapeadas diretamente da especificação em XML.

3.4. Especificidades no Tratamento de Imagens e Textos

Após a descrição de todas as etapas de processamento para dados tabulares, são comentadas as especificidades para imagens e textos. Em ambos os casos é mantida a especificação do projeto em um arquivo no formato XML.

A entrada de dados como imagens é padronizada pela utilização de diretórios estruturados em subdiretórios que mapeiam as classes de saída da rede, compostos de arquivos de imagens, cujo redimensionamento de imagens de tamanhos distintos pode ser definido na especificação em XML.

Para o tratamento de imagens, o padrão utilizado para prover os dados de entrada é um diretório composto de subdiretórios associados a cada uma das classes que serão classificadas na saída. Em cada um desses subdiretórios, devem ser providos arquivos com as imagens para as quais a respectiva classe deve ser predita. Opcionalmente, os parâmetros que definem as dimensões e o método utilizado para redimensionamento de imagens com tamanhos distintos podem ser especificados para o projeto em XML.

Com propósitos educativos, também são suportados os formatos utilizados por softwares populares tais como: mnist [15], fashion-mnist [18], ciphar10 e ciphar100 [13]. Na fase de testes é possível visualizar, para cada classe de saída, as

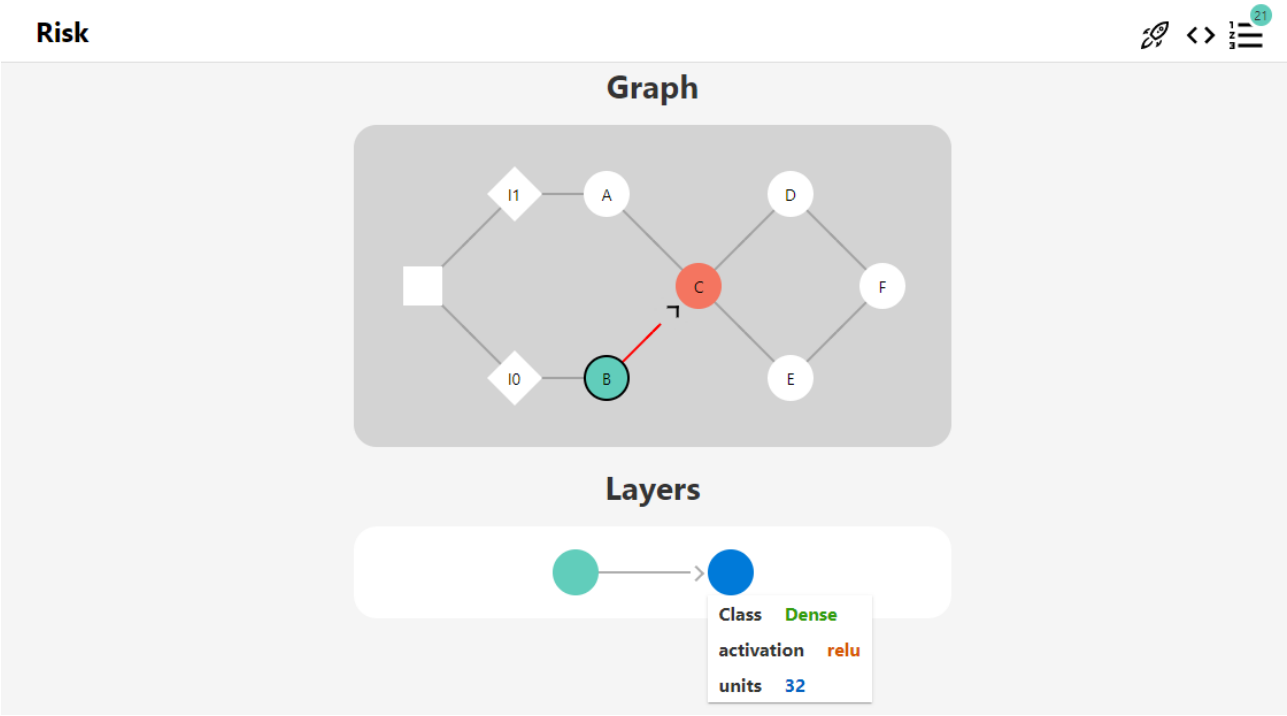

Figura 6: Grafo da rede e camadas de um nó interno selecionado.

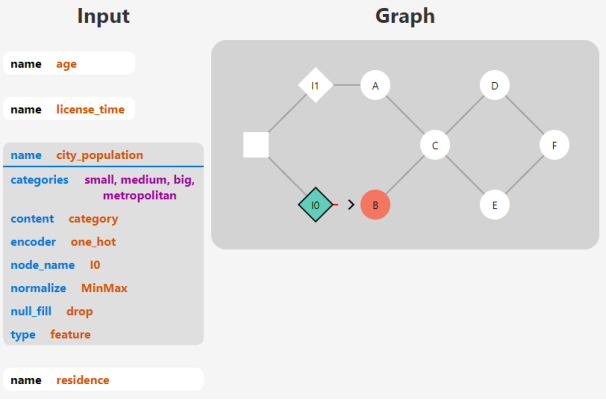

Figura 7: Grafo da rede e especificação do nó de entrada.

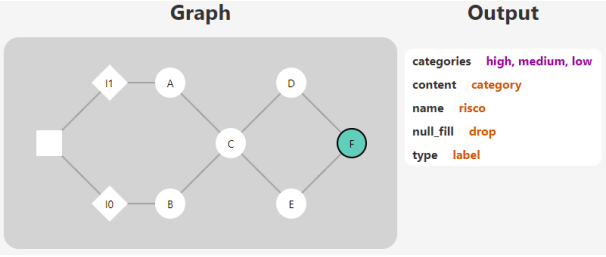

Figura 8: Grafo da rede e especificação do nó de saída.

imagens que foram erroneamente preditas. Na Figura 12, é ilustrado o popular projeto MNIST que tem como objetivo a predição de dígitos de 0 a 9, a partir de imagens escritas à mão. Para cada dígito predito, são reportadas as quantidades de imagens preditas erroneamente e a quantidade total de imagens preditas. Na Figura 12, o usuário selecionou a classe de saída correspondente ao dígito 0, para o qual são mostradas as imagens respectivamente associadas aos dígitos para os quais foram preditas incorretamente. A partir do percentual de erro de predição e das características visuais das imagens preditas incorretamente para cada classe de saída, o usuário obtém uma realimentação para reconfigurar o seu projeto de DNN.

Para o tratamento de textos, o padrão utilizado para prover os dados de entrada são arquivos, no formato TXT, com uma sequência de sentenças, para alimentar as etapas de treinamento e de teste. O título de cada arquivo caracteriza a etapa (emphtrain ou emphtest), o identificador do exemplo, e um valor inteiro associado a uma determinada avaliação. Até o presente momento, a padronização do formato para prover entradas de texto está focada em aplicações de análise de sentimentos [16]. Neste contexto, avaliações dentro de um dado intervalo podem ser mapeadas em uma dada classe de saída, conforme definidas na especificação do projeto em XML. Na especificação do projeto, o usuário deve definir o tamanho máximo de palavras do vocabulário e o tamanho máximo de palavras que serão capturadas em cada sentença. A partir dessas definições, na etapa de pré-processamento, as palavras são mapeadas em representações numéricas. Adicionalmente, o usuário pode especificar o uso do Word2Vec, cuja implementação [7] foi incorporada ao Tuiuiú Deep Learning, para gerar representações numéricas para um dado vocabulário e, posteriormente, reutilizar essas representações em novas execuções de um dado projeto.

```
class CustomNeuralNetModel(Model):
    def __init__(self):
        super(CustomNeuralNetModel, self).__init__()
        self.node_A = Sequential([
            Dense(units=16, activation="sigmoid"),
            Dense(units=32, activation="tanh")
        ])
        self.node_B = Sequential([
            Dense(units=8, activation="relu"),
            Dense(units=32, activation="elu")
        ])
        self.node_C = Dense(units=16, activation="relu")
        self.node_D = Sequential([
            Dense(units=8, activation="sigmoid"),
            Dense(units=32, activation="relu")
        ])
        self.node_E = Sequential([
            Dense(units=8, activation="relu"),
            Dense(units=32, activation="elu")
        ])
        self.node_F = Sequential([
            Dense(units=16, activation="relu"),
            Dense(units=3, activation="softmax")
        ])

    def call(self,input, training=None, mask=None):
        input_A = input[:, 2:4]
        output_A = self.node_A(input_A)

        input_B = [input[:, 0:2], input[:, 4:8]]
        input_B = tf.concat(input_B, 1)
        output_B = self.node_B(input_B)

        input_C = [output_A, output_B]
        input_C = tf.concat(input_C, 1)
        output_C = self.node_C(input_C)

        input_D = output_C
        output_D = self.node_D(input_D)

        input_E = output_C
        output_E = self.node_E(input_E)

        input_F = [output_D, output_E]
        input_F = tf.concat(input_F, 1)
        output_F = self.node_F(input_F)

        return output_F
```

Figura 9: Exportação de código parte 1 - declaração da classe da rede neural.

Além da padronização suportada para dados tabulares e imagens, suporta a entrada de dados tabulares, imagens ou textos no formato numpy, representado por um arquivo com extensão npz.

4. Conclusão e trabalhos futuros

As principais características do Tuiuiú Deep Learning são: (a) suporta todas as etapas de desenvolvimento de DNNs: pré-processamento de dados, treinamento, avaliação e teste; (b) a especificação utilizada em XML é versátil para definir dados de entrada e configuração da rede sequencial e paralela, sem impor restrições associadas às telas

```
def loader(features, labels, batch_size):
    labels = tf.keras.utils.to_categorical(labels)
    data = (features, labels)
    data = Dataset.from_tensor_slices(data)
    return data.batch(batch_size)


def load_npz(path, batch_size):
    train = val = test = None
    data = np.load(path)
    if 'train_features' in data and 'train_labels' in data:
        train = loader(
            data['train_features'],
            data['train_labels'],
            batch_size
        )
    if 'val_features' in data and 'val_labels' in data:
        val = loader(
            data['val_features'],
            data['val_labels'],
            batch_size
        )
    if 'test_features' in data and 'test_labels' in data:
        test = loader(
            data['test_features'],
            data['test_labels'],
            batch_size
        )
    return train, val, test
```

Figura 10: Exportação de código parte 2 - carregamento de dados.

específicas de interface gráficas atreladas a subconjuntos de funcionalidades e parâmetros suportados pelos frameworks utilizados; (c) suporta visualização e edição da especificação e de hiperparâmetros da rede; (d) suporta importação e exportação de código; (e) padroniza entrada de dados tabulares e imagens, suportando o redimensionamento de imagens quando necessário.

Adicionalmente, é amigável para propósitos educacionais, com suporte à: (a) geração automática de dados tabulares a partir da especificação das características das tabela e do algorítmo de geração de labels; (b) entrada de dados de datasets populares como mnist, ciphar10 e ciphar100; (c) suporta a visualização de todas as imagens preditas incorretamente para cada classe de saída, para que o usuário possa avaliar visualmente os tipos de características que estão sendo mal capturado no aprendizado da rede, de forma a poder alterar os hiperparâmetros ou a própria arquitetura da rede utilizada.

O Tuiuiú Deep Learning suporta a entrada de dados tabulares, imagens ou textos no formato numpy, representado por um arquivo com extensão npz. No entanto, para manter o formato original dos dados, a padronização atual para textos suporta basicamente a entrada de informação utilizada para aplicações de análise de sentimentos, para as quais é associada uma avaliação que, conforme definido na especificação do projeto em XML, pode ter seus intervalos de valores (de avaliações) mapeados nas classes de saída da rede. A primeira meta para trabalhos futuros, é a definição

```
metrics = [tf.keras.metrics.CategoricalAccuracy()]
model = CustomNeuralNetModel()
if os.path.exists(weights_path):
    model.load_weights(weights_path)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(
    metrics=metrics,
    loss=tf.keras.losses.CategoricalCrossentropy(),
    optimizer=optimizer
)

callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        weights_path,
        monitor='val_loss',
        verbose=0,
        save_best_only=False,
        save_weights_only=False,
        mode='auto',
        save_freq=save_freq,
        options=None
    )
]
model.fit(
    train,
    epochs=50,
    validation_data=val,
    callbacks=callbacks
)
model.evaluate(test)
```

Figura 11: Exportação de código parte 3 - treinamento da rede neural.

de uma padronização mais abrangente para dados de entrada baseados em textos, para incorporar informações necessárias para tratar outros tipos de aplicações de Processamento de Linguagem Natural.

A segunda meta para trabalhos futuros, é validar possíveis evoluções para a expressividade da linguagem de especificação em XML. A metodologia proposta para esta tarefa, é a prototipagem do mapeamento de um espectro representativo de implementações de DNNs, referenciadas por artigos disponíveis na internet, para especificações XML complementadas por adaptadores. O uso de adaptadores para prover código complementar viabiliza a representação de complexos projetos de DNNs. Entretanto, quanto maior for a expressividade da linguagem de especificação, menor será a dependência do uso de adaptadores, com consequente diminuição de código adicional necessário a ser provido pelo usuário.

Referências

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X., 2016. Tensorflow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, Savannah, GA. pp. 265–283. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[2] Alom, M.Z., Taha, T.M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M.S., Hasan, M., Van Essen, B.C., Awwal, A.A.S., Asari, V.K., 2019. A state-of-the-art survey on deep learning theory and architectures. Electronics 8. URL: https://www.mdpi.com/2079-9292/8/3/292, doi:10.3390/electronics8030292.

[3] Dargan, S., Kumar, M., Ayyagari, M.R., Kumar, G., 2020. A survey of deep learning and its applications: a new paradigm to machine learning. Archives of Computational Methods in Engineering 27, 1071–1092.

[4] Dholakiya, J.H., Kiran, R., Babu, R.V., 2015. Expresso: A user-friendly gui for designing, training and using convolutional neural networks. CoRR, abs/1505.06605 .

[5] Fiore, U., 2019. Neural networks in the educational sector: Challenges and opportunities. Balkan Region Conference on Engineering and Business Education 1, 332–337. doi:10.2478/cplbu-2020-0039.

[6] Garner, S.R., et al., 1995. Weka: The waikato environment for knowledge analysis, in: Proceedings of the New Zealand computer science research students conference, pp. 57–64.

[7] Goldberg, Y., Levy, O., 2014. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. CoRR abs/1402.3722. URL: http://arxiv.org/abs/1402.3722, arXiv:1402.3722.

[8] Harris, D.R., 2020. Prototypeml: A neural network integrated design and development environment. CoRR abs/2007.01097. URL: https://arxiv.org/abs/2007.01097, arXiv:2007.01097.

[9] Hart, P., Nilsson, N., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics 4, 100–107. URL: https://doi.org/10.1109/tssc.1968.300136, doi:10.1109/tssc.1968.300136.

[10] Hayakawa, A., Ishii, M., Kobayashi, Y., Nakamura, A., Narihira, T., Obuchi, Y., Shin, A., Yashima, T., Yoshiyama, K., 2021. Neural network libraries: A deep learning framework designed from engineers' perspectives. CoRR abs/2102.06725. URL: https://arxiv.org/abs/2102.06725, arXiv:2102.06725.

[11] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T., 2014. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 .

[12] Klemm, S., Scherzinger, A., Drees, D., Jiang, X., 2018. Barista - a graphical tool for designing and training deep neural networks. CoRR abs/1802.04626. URL: http://arxiv.org/abs/1802.04626, arXiv:1802.04626.

[13] Krizhevsky, A., Hinton, G., et al., 2009. Learning multiple layers of features from tiny images. Relatório Técnico.

[14] Lang, S., Bravo-Marquez, F., Beckham, C., Hall, M., Frank, E., 2019. Wekadeeplearning4j: A deep learning package for weka based on deeplearning4j. Knowledge-Based Systems 178, 48–50.

[15] LeCun, Y., Cortes, C., Burges, C., 2010. The mnist database of handwritten digits URL: http://yann.lecun.com/exdb/mnist. acessado em 10 de Outubro de 2021.

[16] Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C., 2011. Learning word vectors for sentiment analysis, in: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Portland, Oregon, USA. pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library, in: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[18] Xiao, H., Rasul, K., Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

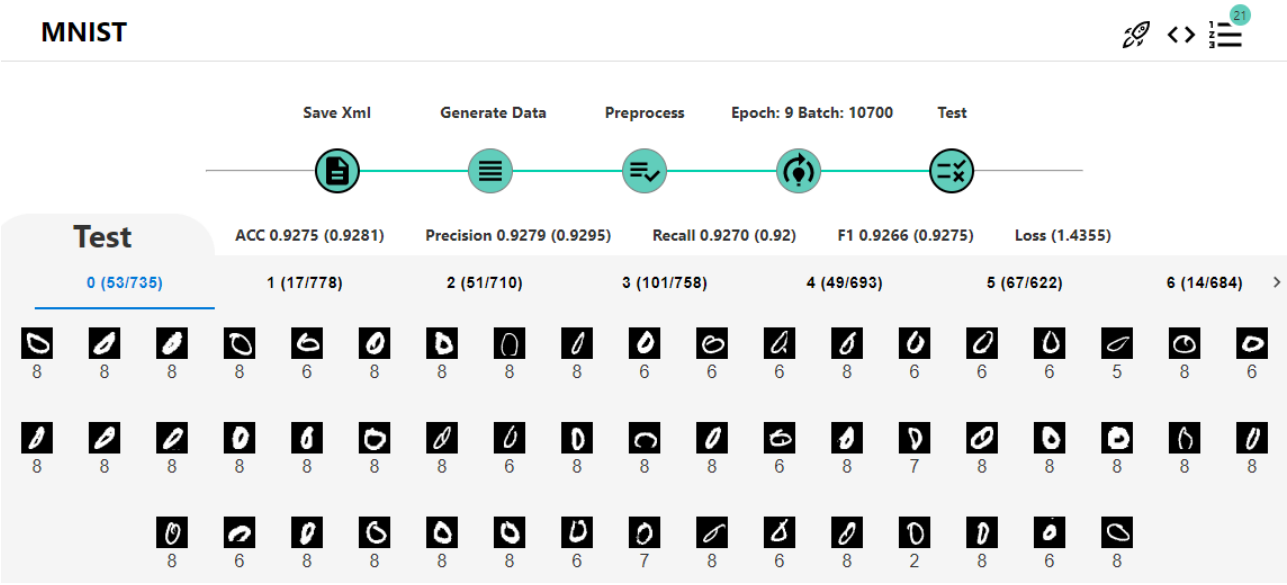

Figura 12: Fase de testes mostrando erros de predição.

CoRR abs/1708.07747. URL: http://arxiv.org/abs/1708.07747, arXiv:1708.07747.