UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL S. VIEIRA
MATEUS I. DO NASCIMENTO

MODELOS DE APRENDIZADO DE MÁQUINA EM SISTEMAS DE WORKFLOWS CIENTÍFICOS

RIO DE JANEIRO
2019

GABRIEL S. VIEIRA
MATEUS I. DO NASCIMENTO

MODELOS DE APRENDIZADO DE MÁQUINA EM SISTEMAS DE WORKFLOWS CIENTÍFICOS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. João Carlos P. da Silva
Co-orientador: Fabrício F. de Faria

RIO DE JANEIRO
2019

Vieira, Gabriel dos Santos
V658m Modelos de aprendizado de máquina em sistemas de workflows científicos / Gabriel dos Santos Vieira, Mateus Ildefonso do Nascimento. – 2019.

46 f.

Orientador: João Carlos Pereira da Silva.
Coorientador: Fabrício Firmino de Faria.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2019.

1. Vistrails. 2. Keras. 3. Workflow científico. 4. Aprendizado de máquina. I. Nascimento, Mateus Ildefonso do. II. Silva, João Carlos Pereira da (Orient.). III. Faria, Fabrício Firmino de (Coorient). IV. Universidade Federal do Rio de Janeiro, Instituto de Matemática. V. Título.

GABRIEL S. VIEIRA
MATEUS I. DO NASCIMENTO

MODELOS DE APRENDIZADO DE MÁQUINA EM SISTEMAS DE WORKFLOWS CIENTÍFICOS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 13 de novembro de 2019.

BANCA EXAMINADORA:

Prof. João Carlos Pereira da Silva
D.Sc - (COPPE/UFRJ)

Fabrício Firmino de Faria
M.Sc - (UFRJ)

Prof. Severino Collier Coutinho
D.Sc - (Leeds)

Felipe Fink Grael
M.Sc - (COPPE/UFRJ)

AGRADECIMENTOS

Gostaríamos de começar os agradecimentos, agradecendo primeiramente a Deus por nos proporcionar saúde e capacidade intelectual para completar este trabalho e concluir os cursos ao longo da nossa graduação. Também gostaríamos de agradecer nosso orientador por estar ao nosso lado nos guiando no decorrer deste trabalho final apontando nossos erros e nos mostrando como corrigi-los. Agradecemos aos nossos professores que nos proveram com seu conhecimento e nos ajudaram a passar por todo esse processo de crescimento. À Universidade Federal do Rio de janeiro, por ser nossa segunda casa durante todos esses anos de graduação e nos propiciar um ambiente de estudo. As nossas famílias por estarem sempre ao nosso lado nos dando forças para seguir buscando cada vez mais conhecimento. Por fim agradecemos aos nossos amigos por estarem juntos conosco nesse processo de crescimento e por tornarem ele o mais tranquilo que foi possível.

RESUMO

Diante do aumento da quantidade de dados somado ao ganho de processamento, surgiram pesquisas utilizando técnicas de aprendizado de máquina capazes de analisar um grande volume de dados. Com o avanço destas pesquisas, foram desenvolvidos formas de extrair conhecimento de diversos tipos de dados tais como: tomografias, músicas, livros, fotos, vídeos e séries temporais. Isso fez com que profissionais de diversos setores utilizassem este método de pesquisa com base na computação para testar suas hipóteses.
O objetivo deste trabalho é utilizar o conceito de workflow científico num experimento de aprendizado de máquina a fim de estruturar a pesquisa e facilitar a integração destes profissionais de modo que seja possível criar experimentos e compartilhar aprendizado utilizando uma linguagem comum a todos. Para isso, utilizamos uma ferramenta de gerenciamento de workflow científico chamada Vistrails adicionamos recursos de aprendizado de máquina nesta ferramenta.

Palavras-chave: Vistrails. Keras. Workflow Científico. Aprendizado de Máquina.

ABSTRACT

Facing a rise of the amount of data followed by the enhance of computer processing, emerged researches utilizing machine learning techniques which are able to analyze big data volume. With the advance of these researches, were developed manners to extract knowledge from multiple types of data such as: tomography, music, books, photos, videos, time series, etc. It made professionals from various fields utilize this research method based on computing to test their hypothesis.
The objective of this work is to utilize the concept of scientific workflow in a machine learning experiment in order to structure the research and facilitate the integration with that professionals so that it is possible to create experiments and share knowledge utilizing a common language to everyone. For this we utilized a scientific workflow management system named Vistrails and we inserted machine learning resources in this system.

Keywords: Vistrails. Keras. Scientific Workflow. Machine Learning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O modo de se fazer ciência passou por grandes mudanças diante do avanço tecnológico. Este nos proporcionou recursos computacionais para tarefas mais complexas e exigentes no âmbito da Inteligência Artificial e outras áreas da computação. O aumento da quantidade de dados somado ao ganho de processamento tornou possível analisar esta grande quantidade e capturar padrões cada vez mais complexos através de modelos de aprendizado de máquina.

É possível analisar todo tipo de informação: tomografias, músicas, livros, fotos, vídeos, séries temporais e etc. Isso fez com que profissionais de diversos setores utilizassem este método de pesquisa com base na computação para testar suas hipóteses. Para integrar todos esses profissionais foi necessário criar uma linguagem compreensível a todos que possibilitasse fazer pesquisa descrevendo um processo experimental de forma pragmática como fluxos de trabalho, mais conhecido como workflow científico, sem exigir nenhum conhecimento profundo de programação.

Workflow científico(MATTOS FABIO C. DA SILVA, 2008) é uma combinação entre workflow e método científico que permite um cientista descrever um experimento científico como um conjunto de tarefas a serem realizadas. Ele é definido através de um sistema de gerenciamento de workflow que utiliza uma notação de compreensão bastante intuitiva. O sistema de gerenciamento é capaz de executar o experimento automaticamente, com pouca ou nenhuma intervenção, utilizando para isso a infraestrutura computacional disponível. Com isso, o workflow científico tornou a automação de experimentos e o uso de plataformas de computação de alto desempenho acessíveis a pesquisadores de diversas áreas do conhecimento.

Os experimentos científicos, independente do domínio de aplicação, devem obedecer a um conjunto rígido de requisitos (WESKE; VOSSEN; MEDEIROS, 1996).

1. Cada experimento deve ser reprodutível e compartilhável para que outros cientistas tenham acesso e façam experimentos semelhantes para confirmar ou refutar os resultados. Os resultados devem ser documentados para que possam ser reaproveitados futuramente.

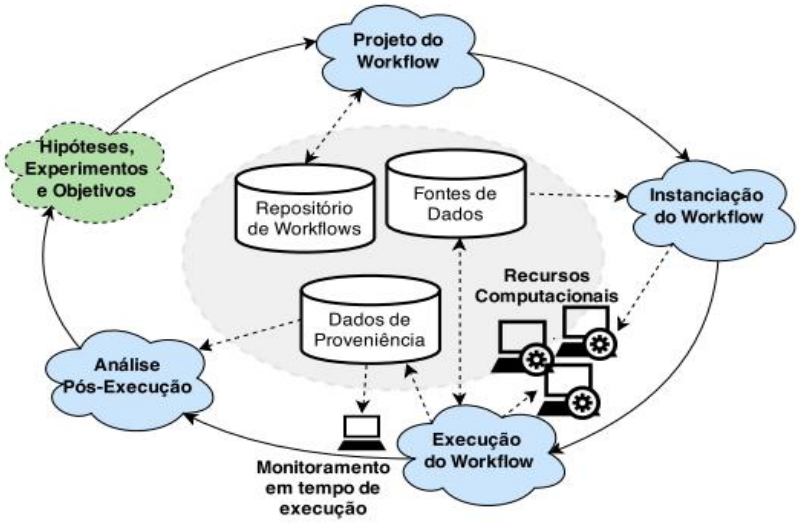
2. Os experimentos devem ter um formato que permita ser conduzido a partir de um ponto inicial e reutilizar ou combinar experimentos realizados anteriormente.

3. Os experimentos devem ser executados em condições controladas. Geralmente, um experimento é uma composição ordenada de etapas (complexas). As condições experimentais deverão ser documentadas e controladas ao longo das etapas. Assim,

caso seja necessário executar o experimento novamente, os mesmos resultados serão obtidos.

Um workflow científico tem seu ciclo de vida composto por quatro fases, ilustradas pela Figura 1 (LUDÄSCHER et al., 2009):

Figura 1 – Fases do ciclo de vida de um workflow científico.



Fonte: Ludäscher et al. (2009)

1. Projeto do Workflow: nesta etapa será definido o modelo e como iremos organizar cada módulo a ser executado, podendo-se reaproveitar ou basear em outros projetos de alguma forma. Cada projeto deve ser salvo, sendo possível reproduzi-lo futuramente. O projeto é feito para testar uma hipótese.

2. Instanciação do Workflow: corresponde à preparação necessária para uma execução particular do workflow. Nela definimos os parâmetros do modelo e fornecemos os dados necessários para executá-lo.

3. Execução do Workflow : corresponde a execução dos componentes do workflow. Nesta etapa serão registrados os dados gerados de cada módulo, além de se obter métricas de performance: tempo de execução, quantidade de memória e ocorrência de erro em cada módulo do workflow facilitando a análise do experimento e agilizando as próximas execuções através de cache em cada módulo.

4. Análise Pós-Execução: corresponde à inspeção e interpretação dos resultados obtidos a partir da execução do workflow. Por exemplo, a análise dos dados de saída gerados pode levar à rejeição da hipótese científica. A análise de informações de proveniência, por exemplo, permite identificar as atividades que prejudicam a execução do workflow. Os resultados obtidos a partir da análise podem conduzir a uma

revisão da hipótese ou do objetivo experimental inicial e, portanto, a um reprojeto do workflow (ou seja, uma nova iteração do seu ciclo de vida).

O objetivo deste trabalho é aplicar o conceito de workflow científico, através de um sistema de gerenciamento, no desenvolvimento de experimentos focados em aprendizado de máquina. Uma aplicação deste foco possui uma série de passos distintos que são correlacionados com cada fase do ciclo de vida de um workflow científico:

1. Projeto: Definir modelo selecionando-se qual base de dados, quais tipos de pré processamento, qual algoritmo de aprendizado e qual método de validação serão utilizados. Nesta fase é decidido quais ferramentas serão utilizadas para validar a hipótese.

2. Instanciação: Configurar os parâmetros exigidos por cada ferramenta. Um projeto de aprendizado de máquina possui milhares de parâmetros que influenciam diretamente no resultado. Uma vez que o projeto foi definido, é possível fazer diversos experimentos apenas alterando os parâmetros, sendo cada combinação de parâmetros uma instância do projeto.

3. Execução: Treinar o modelo de aprendizado documentando seu desempenho para ser analisado futuramente. Alguns treinamentos podem demorar dias, sendo portanto, importante que a ferramenta registre todos os dados obtidos para que seja possível recuperar o que foi feito em caso de falha durante a execução.

4. Análise Pós-Execução: Analisar os dados obtidos após a execução da instância para validar o modelo. Em aprendizado de máquina, validar significa calcular a acurácia do modelo ao executar uma tarefa. Após a análise, o pesquisador tem 3 opções: reconstruir o projeto voltando a primeira etapa, reconstruir a instância voltando a segunda etapa ou finalizar o experimento confirmando a sua hipótese.

Dado a correlação entre workflow científico e aprendizado de máquina, é possível utilizar um sistema de gerenciamento de workflow científico para auxiliar no desenvolvimento de aplicações neste setor. O sistema deve padronizar a linguagem utilizada no experimento de modo que permita a contribuição de todos os cientistas e garantir todos os pré requisitos que um workflow científico possui.

Além disso, ao estruturar o processo de desenvolvimento de aplicações para algoritmos de aprendizado de máquina como um workflow científico, será possível auxiliar os pesquisadores nas suas pesquisas e poderá utilizar essa nova estrutura para finalidades educativas como aulas sobre o tema de aprendizado, exemplos mais explícitos em cada série de passos, etc.

Para exemplificar esse processo de desenvolvimento, será realizado um experimento de classificação de imagens. Neste será realizado a classificação do conjunto de imagens chamado MNIST(LECUN; CORTES, 2010), sendo este amplamente utilizado para explorar novos modelos, *frameworks* e técnicas.

O trabalho está estruturado da seguinte forma: No Capítulo 2 será falado sobre os conceitos básicos, este capítulo irá explicar a base teórica por trás da ferramenta. No Capítulo 3 serão abordadas questões mais técnicas explicando quais funcionalidades foram adicionadas no Vistrails e como elas funcionam. No Capítulo 4 temos um estudo de caso que mostra o passo a passo no processo de criação de um experimento dentro da ferramenta, como mencionado anteriormente, será realizada uma classificação de imagens com o MNIST.

2 CONCEITOS BÁSICOS

Neste Capítulo será abordado os conceitos teóricos sobre Workflow científico, além dos conceitos dos sistemas de gerenciamento deles e por fim uma análise dos recursos do sistema de gerenciamento escolhido.

2.1 MÉTODO CIENTÍFICO

O método científico (ANDERSEN; HEPBURN, 2016) é um conjunto regras que um experimento científico deve seguir. A humanidade tem desenvolvido o seu conhecimento seguindo este processo que poderia ser chamado de boas práticas de pesquisa e é fortemente aceito pela comunidade científica.

Este método é estruturado a partir de uma hipótese buscando reforçá-la ou invalidá-la. A hipótese é uma afirmação definida de modo empírico pelo pesquisador que deseja validá-la aplicando uma série de experimentos seguindo as boas práticas proposta pelo método científico.

Para realizar uma experiência são necessários 3 itens: aparato experimental, procedimento e relatório. O aparato experimental é o material a ser utilizado no experimento, procedimento é uma sequência de atitudes e medidas a serem feitas pelo experimentador e relatório é a análise dos dados obtidos por meio das medidas a fim de obter uma conclusão sobre o experimento.

A primeira regra proposta pelo método científico é ter um ambiente controlado para validar a hipótese. Um ambiente controlado ideal é um espaço onde o pesquisador tem controle sobre todas as variáveis que podem influenciar no experimento. Entretanto, haverá cenário onde não será possível ter controle sobre alguns parâmetros. Neste caso será necessário considerar este ruído no experimento e minimizar o seu impacto na análise. Uma forma de minimizá-lo é repetindo o experimento diversas vezes e utilizar dados estatísticos para reforçar a validação.

A segunda regra é alterar uma variável por vez em cada análise. Tendo um ambiente controlado é possível alterar apenas um parâmetro e manter todos os outros intactos e inferir o impacto deste parâmetro no experimento.

Este método tem se desenvolvido ao longo do tempo sendo bem amplo nos dias de hoje, mas a sua base está nessas duas regras. Vale ressaltar também que o método científico não afirma a hipótese, ele apenas a reforça, uma vez que este método sempre assume a possibilidade de existir um contra-exemplo.

2.2 WORKFLOW CIENTÍFICO

O workflow científico (MATTOS FABIO COUTINHO DA SILVA, 2008) foi definido para auxiliar um experimento aplicando o conceito de fluxos de trabalho num experimento de acordo com o método científico. Ele auxilia criando uma formalização do experimento , organizando todo o processo como um fluxo sequencial composto por módulos. Módulo é uma unidade de processamento que recebe uma entrada e retorna uma saída que pode ser utilizada, ou não, por outro módulo.

Figura 2 – Fluxo de um workflow científico

O processo de definição de workflow científico requer uma ampla discussão entre os membros do grupo de pesquisa. Sua definição envolve tomada de decisão e análises profundas sobre cada uma das etapas do experimento.

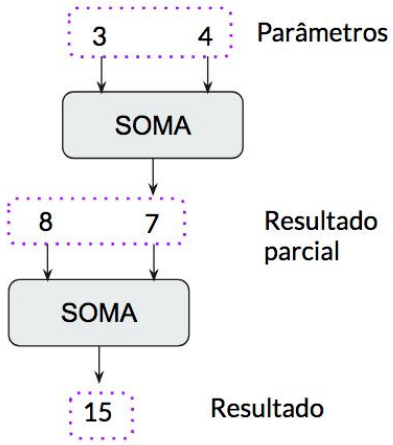
Cada projeto pode possuir características particulares de acordo com as necessidades do experimento. Isso torna necessário usar uma ferramenta que normalize a comunicação entre cada cientista e que permita desenvolver módulos que atendam as necessidades do projeto. Um bom sistema de gerenciamento de workflow científico deve atender esses requisitos.

Um exemplo simples dos módulos é o capaz de executar a operação de soma de dois números, sua saída será a soma dos dois valores fornecidos como entrada. Para somar três números utilizando este módulo, basta utilizar dois deste em sequência onde a saída do primeiro servirá de entrada para o seguinte que vai somar o terceiro valor de entrada como é visto na Figura 3.

Fazendo uma analogia com os 3 itens que um experimento deve ter, o sistema de workflow científico seria o aparato experimental, a instância do projeto seria o procedimento e a análise dos dados após a execução seria o relatório.

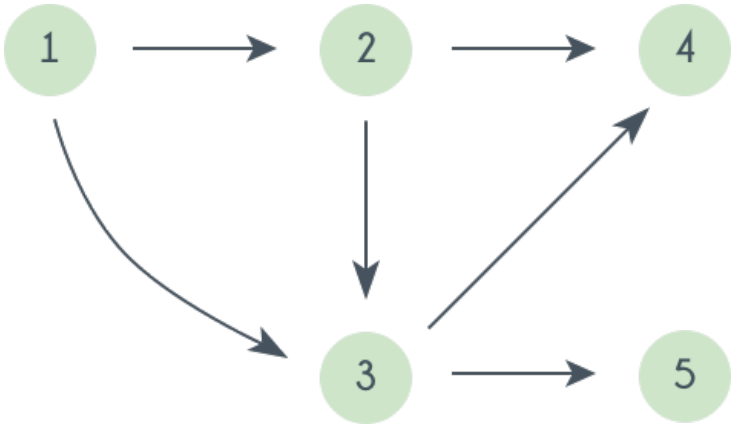
Desenvolver orientado a módulos permite um maior reaproveitamento das funcionalidades já desenvolvidas, pois cada módulo pode ser adicionado em qualquer workflow desde que receba uma entrada no formato esperado. Entretanto, cada workflow está sujeito às particularidades do experimento ao qual está associado. Logo criar uma função que seja capaz de abranger uma quantidade de situações diversas com uma única implementação pode ser uma tarefa árdua. Um bom exemplo desta dificuldade é o pré processamento de dados, visto que há milhares de bases de dados diferentes dificultando o desenvolvimento de um módulo que trate todos os tipos de dados.

Figura 3 – Representação gráfica de um módulo



Um projeto de workflow científico é estruturado como um grafo direcionado acíclico (DAG Directed Acyclic Graph). Tendo em mente que a saída de um módulo pode ser utilizada como entrada em outro módulo, é possível representar um projeto como um grafo direcionado onde uma aresta indica a dependência de um nó do grafo em relação ao outro e cada nó representa um módulo. Se o módulo 1 estiver conectado ao módulo 2, o módulo 2 terá de esperar o 1 terminar de executar para receber os dados necessários e iniciar sua execução.

Figura 4 – Ordem de execução de uma DAG



O grafo na Figura 4, pode representar um workflow científico composto por 5 módulos. Sua execução se inicia com um módulo que não possui dependências (1). Após a execução do módulo sem dependências , serão fornecidos dados para os módulos dependentes a ele (2,3). Uma vez que um módulo recebe dados do módulo anteriormente executado, ele pode executar caso não tenha mais nenhum módulo dependente a ele que ainda não foi

executado (2) que fornecerá dados aos demais módulos. Este processo irá se repetir até que todos os módulos sejam executados. A ordem de execução dos módulos é chamada de ordenação topológica.

É impossível executar um projeto de workflow científico com ciclo na definição do projeto. Se um módulo 1 depende de 2 para executar, 2 depende de 3 e 3 depende de 1, nenhum módulo conseguirá iniciar sua execução travando todo o workflow. Por isso um projeto é estruturado como DAG.

2.3 SISTEMAS DE GERENCIAMENTO DE WORKFLOW CIENTÍFICO

Um sistema de gerenciamento de workflow científico tem como objetivo gerenciar a definição e execução das etapas de processamento através de uma interface intuitiva que permite diversos profissionais realizarem experimentos utilizando uma linguagem padronizada. Ele oferece um conjunto de ferramentas sendo possível definir um projeto organizando os módulos como uma DAG.

Este sistema tem de ser capaz de executar os módulos e ordená-los topologicamente evitando inconsistência entre cada passo do experimento. Existem diversos tipos de sistemas disponíveis com focos diferente. Como o objetivo do trabalho é aplicar o workflow científico no problema de aprendizado de máquina, é preciso que o sistema seja flexível, uma vez que em cada etapa do processo do aprendizado de máquina (pré-processamento, treinamento e validação) existem diversas opções de algoritmos, sendo primordial que o sistema customize e crie módulos de acordo com a demanda do projeto.

Um exemplo de sistema de gerenciamento de workflow é o Watson Studio, um dos serviços do Watson(FERRUCCI, 2012) da IBM, que possui uma interface gráfica e módulos para aprendizado de máquina. É possível criar redes neurais camada a camada customizando a mesma com uma interface gráfica intuitiva.

Embora o Watson Studio seja uma ótima opção, seu código é fechado. Então optamos por um sistema de workflow científico de código aberto, o que abre a possibilidade de criação de novos módulos para prover funcionalidades diferentes. Isso permite uma melhor flexibilidade para a ferramenta visto que pode-se suprir futuras necessidades adicionando diferentes módulos no código.

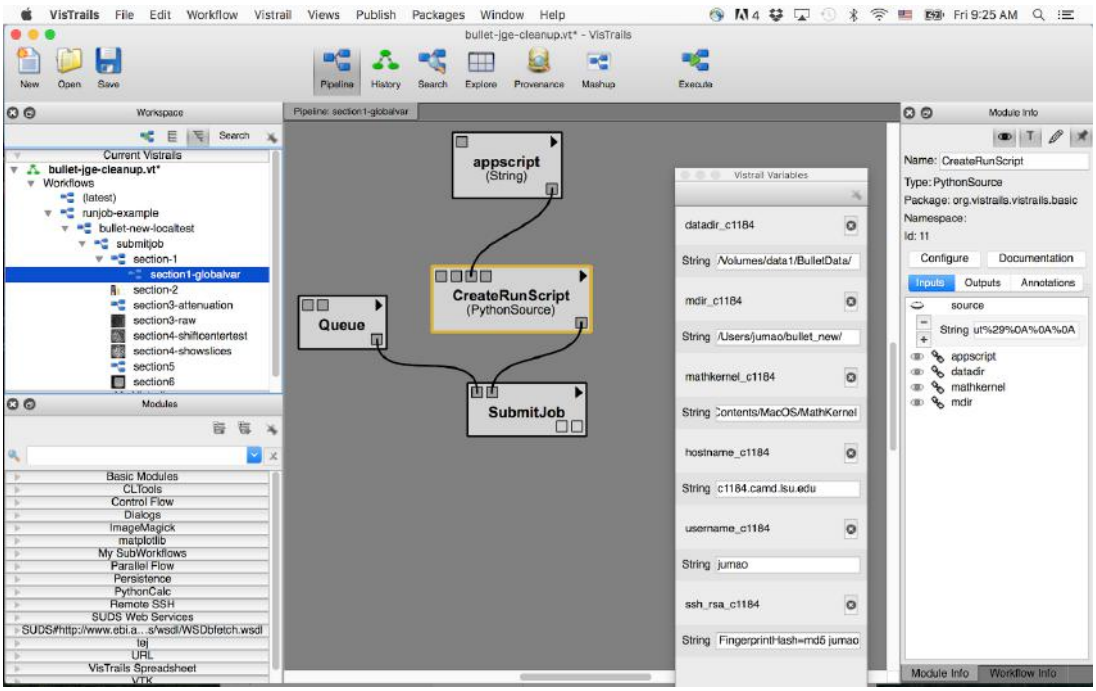
Até o desenvolvimento ser concluído, o workflow passa por muitas mudanças iterativas em sua estrutura e parâmetros. Para auxiliar o processo de desenvolvimento e lidar com essas mudanças, uma interface gráfica é um recurso de alguns sistemas de gerenciamento que facilitam ao desenvolvedor modificar a aplicação inúmeras vezes. Sendo este um dos requisitos, escolhemos um sistema de gerenciamento que possuía essa característica, ele é descrito na seção seguinte.

2.4 VISTRAILS

O software escolhido para este trabalho é chamado Vistrails(CALLAHAN et al., 2006). Ele foi desenvolvido no Instituto de Computação Científica e Imagens da Universidade de Utah, e fornece suporte para exploração e visualização de dados. Ele está escrito em Python e emprega ligações Qt via PyQt que é uma biblioteca dedicada a desenvolver interfaces gráficas escrita em C++.

O Vistrails é um sistema de código aberto, o que permite customizá-lo e compartilhar suas modificações com a comunidade. Ele possui uma representação gráfica do projeto construído sendo possível visualizar os módulos que serão executados sequencialmente. A interface do sistema permite entender todas as etapas do experimento sem precisar entender programação, basta conhecer a função de cada módulo. Entretanto, caso necessário a criação de uma nova funcionalidade há a possibilidade de programar um novo módulo com ela.

Figura 5 – Exemplo de projeto no Vistrails

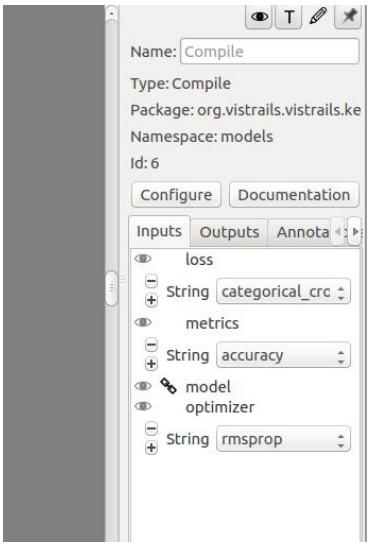


Fonte: Vistrails (2019)

No canto esquerdo da Figura 5 vemos a lista de módulos disponíveis, no centro temos uma visão da arquitetura do projeto onde cada caixa é um módulo e no canto direito é exibido a configuração (parâmetros) do módulo selecionado (CreateRunScript). Percebe-se que o fluxo de execução do projeto é de cima para baixo, onde os módulos do topo já receberam os dados de entrada.

Para parâmetros categóricos é possível adicionar uma lista de opções, como é possível ver na Figura 6, que facilita o processo de experimentação do modelo de aprendizado. Isso

Figura 6 – Seleção de parâmetros no módulo do Vistrals



permite que pesquisadores experimentem novas configurações de parâmetros sem precisar entender a implementação.

Para salvar instâncias com parâmetros diferentes de um projeto em python (sem Vistrails) há duas formas:(i) salvar diversos arquivos alterando apenas os parâmetros utilizados ou (ii) adicionar cada parâmetro via variável de ambiente. As duas formas exigem conhecimento de programação e pode ser extremamente custoso fazer caso o projeto tenha milhares de parâmetros. No Vistrails é possível fazer a mesma coisa sem exigir conhecimento de programação, pois é possível fazer tudo isso via interface gráfica como visto na Figura 5. Entretanto, caso seja necessário, é possível programar novos módulos para atender a necessidade do pesquisador se aproveitando da interface gráfica do Vistrails.

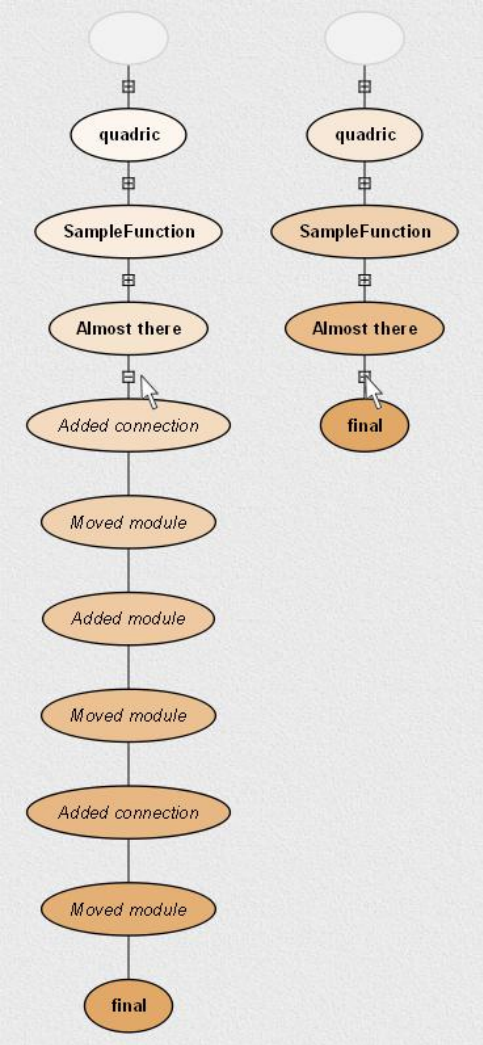
Vistrails abstrai todas as questões de ambiente e facilita a experimentação criando versões do experimento que são possíveis de visualizar por meio de uma árvore como visto na Figura 7. Ele se encarrega do trabalho de gerenciamento do projeto permitindo que o pesquisador foque diretamente no modelo de aprendizado.

Esta abstração e a capacidade de visualização apresentada anteriormente também permitem que o Vistrails se torne uma ferramenta interessante para a educação, visto que aulas relacionadas ao tema de aprendizado podem ficar mais intuitivas e práticas com a utilização destes módulos.

Para instalar o Vistrails há diversas opções disponíveis. Existem versões para Linux, Windows e Mac no site oficial do Vistrails onde estas podem ser baixadas e instaladas ou executadas. Também há a opção de instalar o Vistrails pelo gerenciador de pacotes python chamado Anaconda e além disso pode-se clonar o repositório oficial do Vistrails no github e executar aplicação com o script run.py.

Além do Vistrails, há outros sistemas de código aberto que permitem ao usuário

Figura 7 – Árvore de versionamento via interface gráfica



Fonte: Vistrails (2019)

gerenciar diversos módulos criando workflows para os seus projetos. Exemplos destes sistemas são o Knime (BERTHOLD et al., 2007) e o Apache Airflow (Apache, 2019). O Knime possuí uma interface gráfica assim como o Vistrails e também possuí uma quantidade expressiva de módulos que são utilizados para aprendizado de máquina em si, por causa da capacidade de integração dele é possível desenvolver novos módulos de maneira similar ao Vistrails. Entretanto este sistema não possuí a árvore de versionamento que o Vistrails possuí, o que limita a documentação e o histórico do projeto como um todo. Já o Apache Airflow é utilizado um script onde é definido todo o seu *DAG* para a criação do workflow, ele não possuí uma interface gráfica iterativa, apenas uma para visualização do workflow gerado pelo script. Este sistema possuí integração com diversos serviços da nuvem podendo explorar diversos funcionalidades deste ambiente. Apesar de ter funcionalidades interessantes, o Airflow também não possuí o histórico de versões igual ao Vistrails. Por conta deste recurso poderoso que é a arvore de versionamento do

Vistrails e por ser um diferencial dentre os outros que ele será o sistema utilizado para o desenvolvimento dos módulos de aprendizado.

Devido ao Vistrails ser um sistema de código aberto, toda contribuição feita neste trabalho poderá ser disponibilizada para a comunidade do Vistrails.

3 IMPLEMENTANDO MÓDULOS DE APRENDIZADO DE MÁQUINA NO VISTRAILS

Incorporamos ao Vistrails módulos para cada uma das 4 etapas do processo de aprendizado de máquina: Preparar a base de dados, pré processar a base adquirida, definir e treinar o modelo de aprendizado e validar o resultado obtido.

3.1 PREPARAÇÃO DA BASE DE DADOS

O primeiro passo de um experimento é carregar a base de dados, o que pode ser feito de diversas formas, dependendo do tipo de dado que se está trabalhando. Neste trabalho focamos em carregar dados tabulares no formato CSV e imagens. Dentre os módulos cridos para este trabalho há o *ReadCSV* que carrega dados tabulares e o *ImgLoader* que carrega imagens.

3.1.1 Particionar o dado

Apenas carregar o dado por completo não permite que o pesquisador tenha flexibilidade para utilizar o dado. Então implementamos dois módulos *SplitCol* e *Sample*. *SplitCol* remove colunas do dado tabular, permitindo ao pesquisador validar algumas hipóteses, como a importância, a respeito da característica presente em cada coluna na base de dados. *Sample* retorna uma amostra da base, sendo que o tamanho da amostra pode ser configurado via parâmetro, permitindo ao pesquisador testar o modelo em um conjunto menor de dados e validar algumas hipóteses mais rapidamente já que treinar toda a base de dados é um processo que pode levar horas ou até dias.

3.1.2 Consumo dos dados em aprendizado de máquina

O consumo de dados durante o treinamento do modelo é feito de forma iterativa. O processo ocorre da seguinte forma: Primeiro o modelo de aprendizado inicia todos os seus parâmetros com valores aleatórios e depois consome uma amostra da base de dados para atualizar seus parâmetros. Este processo se repete até consumir toda a base.

Durante o treinamento não é preciso carregar toda a base de dados de uma só vez sendo que cada etapa da iteração consome apenas uma amostra. Isso permite ao modelo carregar apenas uma amostra da base e descartá-la após a sua análise. Felizmente a própria linguagem Python permite a execução deste processo por meio de um recurso chamado *Generator*.

3.2 PRÉ PROCESSAMENTO

Após carregá-los, é preciso preparar os dados para o modelo de aprendizado de máquina. Os dados em aprendizado de máquina são um enorme conjunto de exemplos que serão analisados pelo modelo. Cada exemplo fornecido ao modelo tem de estar num formato de um vetor n-dimensional onde cada dimensão contém um valor numérico que representa uma característica dos dados. Caso os dados carregados não estejam neste formato, é necessário convertê-lo ao formato numérico aplicando técnicas de pré-processamento. Além de preparar os dados para o modelo, o pré-processamento é capaz de otimizar o treinamento através de técnicas que criam uma melhor representação dos dados facilitando o treinamento. Cada tipo de dado possui um conjunto de técnicas de pré processamento, a seguir abordaremos as técnicas que adicionamos no Vistrails.

3.2.1 One Hot Vector

Esta técnica de pré processamento consiste em transformar dados categóricos em dados numéricos da como visto na Tabela 1.

Tabela 1 – Representação vetorial para dados categóricos utilizando o one hot vector

classes	dimensão 1	dimensão 2	dimensão 3
carro	1	0	0
avião	0	1	0
barco	0	0	1

A ideia desta técnica é criar uma dimensão para cada valor categórico e marcar com o valor 1 a dimensão associada a este valor. No exemplo da Tabela 1 a primeira dimensão do vetor está associada a palavra carro, a segunda a avião e a terceira a barco. Logo, se quisermos informar ao modelo que temos um carro devemos marcar a primeira dimensão com o valor 1 e 0 nas dimensões restantes e para os outros veículos o processo é análogo. Ao carregar um dado tabular o módulo *ReadCSV* identifica colunas categóricas e aplica esta técnica automaticamente.

3.2.2 Embedding

Este método é uma otimização do *One Hot Vector*, onde ao invés de se criar uma dimensão para cada classe categórica possível, ele codifica cada classe num vetor de tamanho fixo cujo valor pode ser configurado via parâmetro, no exemplo abaixo (tabela 2) cada classe foi codificada num vetor de duas dimensões. A vantagem desta técnica é que além de utilizar menos memória, ela considera a distribuição das classes dentro da base de dados e define os valores de cada vetor de modo que a distância entre cada vetor codificado representa a similaridade entre as classes.

Tabela 2 – Representação vetorial para dados categóricos utilizando o embedding

classes	dimensão 1	dimensão 2
carro	0.53	0.52
moto	0.51	0.58
cachorro	-0.86	-0.13

A tabela 2 contém os vetores codificados para as classes carro, moto e cachorro. Se calcularmos a distância euclidiana entre dois vetores quaisquer, teremos a similaridade entre eles. Podemos ver que as classes *carro* e *moto* estão mais próximas enquanto a classe *cachorro* está distante de ambos. Uma vez que a representação vetorial nos forneça, mesmo que indiretamente, a similaridade entre as classes, o modelo de aprendizado será capaz de considerar esta informação ao analisar a base de dados podendo gerar melhores resultados durante o treinamento. Este método não nos diz exatamente o que cada dimensão representa, mas podemos inferir seu significado através da análise dos vetores obtidos. Para aplicar esta técnica de pré-processamento no Vistrails, devemos adicionar o módulo *Embedding* após o módulo *Sequencial*. O módulo *Sequencial* é o responsável por iniciar uma rede neural criando sua primeira camada. O tópico redes neurais será abordado na subseção 3.3.2.

3.2.3 Normalização

Uma vez que o modelo de aprendizado de máquina considera apenas valores numéricos, o seu valor absoluto pode influenciar na forma como o modelo interpreta os dados. Quanto maior o valor, maior será seu peso durante o treinamento. É possível que as dimensões da base de dados estejam em escalas diferentes, o que pode tornar o modelo enviesado pelas dimensões de maior magnitude durante o treinamento.

Para resolver este problema, é necessário normalizar os dados colocando todas as dimensões na mesma escala. Existem diversos tipos de normalização e para este trabalho foram implementados dois métodos de normalização : MinMaxScaler e StandardScaler. MinMaxScaler utiliza o valor mínimo e o máximo para normalizar os dados entre num intervalo entre 0 e 1. StandardScaler utiliza a média e o desvio padrão para normalizar os dados.

3.2.4 Ampliação de dados

Ampliação de dados (*augmentation*) (PEREZ; WANG, 2017), é um conjunto de técnicas para gerar mais exemplos em um conjunto de dados. No escopo deste trabalho foram utilizadas algumas técnicas que aumentam o conjunto de imagens em um dataset. Utilizou-se técnicas como *rotacionar, inverter, cortar as imagens*, entre outras, que auxiliam no processo de obtenção de mais exemplos relacionados ao conjunto de imagens a priori.

Muitas vezes, a base de dados não possui muitos exemplos de algumas categorias e por isso técnicas de ampliação de dados são utilizadas frequentemente para obter mais desses exemplos, respeitando o padrão em que se poderia observar os objetos para não criar imagens que poderiam não representar imagens reais.

3.3 DEFINIÇÃO DO MODELO DE APRENDIZADO

Para construir e, posteriormente, treinar o modelo de aprendizado, incorporamos a biblioteca Keras (CHOLLET et al., 2015) ao Vistrails criando módulos para o desenvolvimento de modelos de aprendizado utilizando redes neurais. Agora por meio deste trabalho é possível criar e aplicar redes neurais e utilizar o recurso de gerenciamento de workflow do Vistrails durante a construção do modelo. Grande parte do desenvolvimento de uma aplicação de aprendizado de máquina é otimizar os hiperparâmetros, onde estes são os parâmetros definidos antes do processo de treinamento, via experimentação. O Vistrails nos dá um processo estruturado de experimentação facilitando a otimização dos hiperparâmetros.

3.3.1 Keras

O Keras(CHOLLET et al., 2015) é uma API de alto nível, escrita em Python, para o desenvolvimento de redes neurais. Ela utiliza bibliotecas de mais baixo nível, como o Tensorflow, CNTK e Theano, permitindo criar redes neurais com alto nível de performance de modo fácil. O Keras possui um inventário de redes neurais já treinadas prontas para serem utilizadas, e este inventário contém os modelos mais famosos que são ou já foram estado da arte para determinado tipo de tarefa.

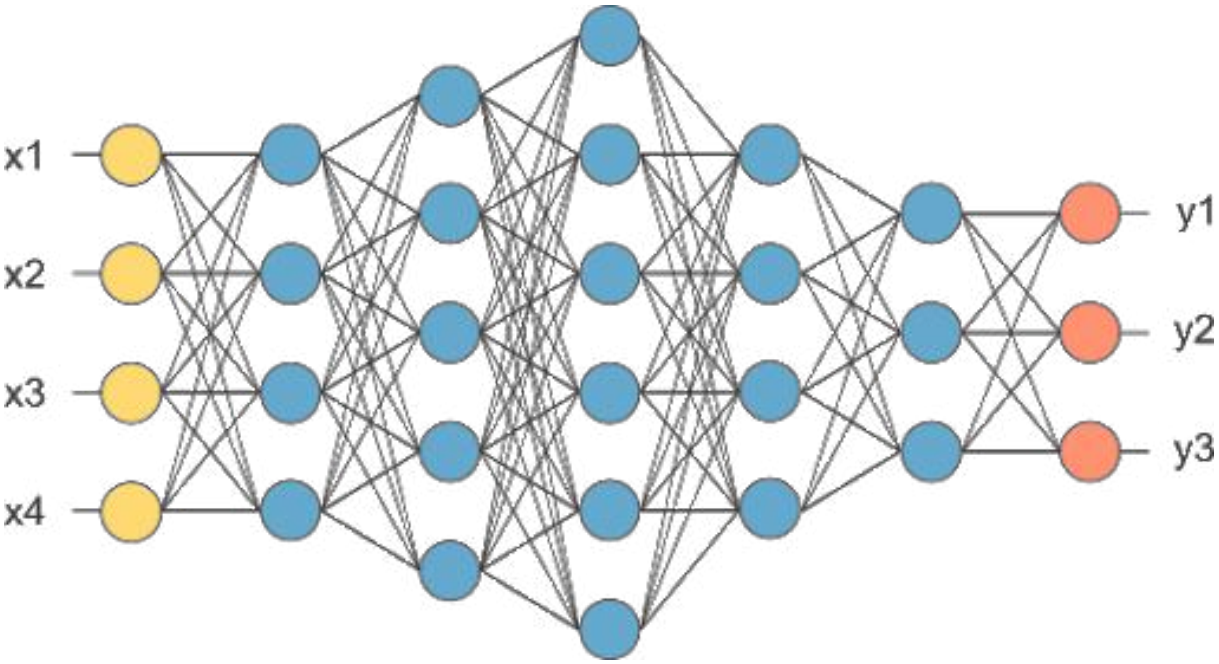
No trabalho utilizamos esta API para desenvolver módulos capazes de criar redes neurais e definir a arquitetura da rede via interface gráfica do Vistrails e importamos o inventário de redes neurais para utilizar como critério de comparação ou reutilizar a sua arquitetura e construir uma nova rede neural.

3.3.2 Redes Neurais

A rede neural é composta por camadas (layers) e cada camada pode possuir uma função de ativação (activations). Existem diversos tipos de redes neurais com propósitos diferentes. No trabalho foram adicionados 3 tipos: MLP *(multilayer perceptron)*, recorrente e convolução.

No esquerda da Figura 8 temos a primeira camada que é responsável por receber a entrada diretamente da base de dados. As camadas intermediárias, que estão entre a primeira e a última, são as camadas escondidas e elas são responsáveis por analisar e reconhecer padrões existentes na base de dados. A última camada é a saída da rede que será o valor a ser analisado após o treinamento.

Figura 8 – Arquitetura de uma rede neural



Para criar as redes neurais no Vistrails, foram criados 3 conjuntos de módulos: *Models*, *Layers* e *Activations*. *Model* define a configuração da rede enquanto *Layer* e *Activation* definem a arquitetura da rede.

3.3.2.1 Models

Models contém dois módulos: *Sequential* e *Compile*. *Sequential* é responsável por iniciar a criação da rede neural definindo o formato da entrada que é a primeira camada na Figura 8. *Compile* é responsável por definir como o treinamento da rede será feito, nele informamos quais serão os dados de treino, teste e validação, como os pesos da rede neural serão inicializados e qual função de custo otimizará os pesos da rede.

O módulo *Sequential* recebe um parâmetro para definir o tamanho da camada de entrada e o módulo *Compile* recebe três parâmetros categóricos: *Loss*, *Metrics* e *Optimizer*. *Loss* é a função de custo que é utilizada para calcular o erro do modelo, *Optimizer* é o método de otimização do modelo a partir do erro obtido e *Metrics* é o cálculo da taxa de acerto do modelo. Para cada um desses parâmetros, o Keras disponibiliza uma lista de opções sendo possível utilizá-los no Vistrails.

3.3.2.2 Layers

Layers contém os principais tipos de camadas de rede neurais disponíveis no Keras:

- *Dense*, camada mais simples de uma rede neural que consiste em conectar cada

neurônio da camada anterior com todos da camada seguinte. Exemplos dessa camada são vistos na Figura 8.

- *LSTM (HOCHREITER; SCHMIDHUBER, 1997)* (Long Short-Term Memory) tipo de camada otimizada para analisar dados temporais.
- *Conv2D (LECUN PATRICK HAFFNER, 1999)* tipo de camada utilizada para analisar dados de duas dimensões como, por exemplo, imagens.
- *Dropout (SRIVASTAVA et al., 2014)*, técnica de regularização para evitar overfitting.
- *Flatten (JIN AYSEGUL DUNDAR, 2014)* é utilizada para conectar uma *Conv2D* a uma *Dense*, ela consiste em reduzir a dimensão de uma *Conv2D* para 1D.
- *Embedding (GAL; GHAHRAMANI, 2016)*, camada utilizada para criação de representação vetorial densa para dados categóricos.
- *Batch Normalization (IOFFE; SZEGEDY, 2015)*, técnica de normalização dos pesos da camada para evitar overfitting.

3.3.2.3 Activations

Activation contém as funções de ativação que podem ser adicionadas em cada camada para otimizar a performance do treinamento da rede. A função de ativação é responsável por definir se o neurônio da camada envia o sinal para o neurônio seguinte ou não. Cada módulo em *Activation* corresponde a uma função ativação distinta. Foram adicionadas ao Vistrails:

- *ReLU (NAIR; HINTON, 2010)*, função de ativação mais simples que retorna zero quando a entrada é negativa.

$$f(x) = \begin{cases} x, & \text{Se } x \geq 0 \\ 0, & \text{Se } x < 0 \end{cases}$$

- *LeakyReLU (MAAS; HANNUN; NG, 2013)* variação do *ReLU* que ao invés de zerar quando a entrada for negativa ele multiplica por uma constante alpha definida via parâmetro.

$$f(x) = \begin{cases} x, & \text{Se } x \geq 0 \\ \alpha x, & \text{Se } x < 0 \end{cases}$$

- *PreLU (HE et al., 2015b)* uma otimização do *LeakyReLU* que otimiza a escolha do alpha automaticamente.

- *ELU (CLEVERT; UNTERTHINER; HOCHREITER, 2015)* variação do *ReLU* utilizando uma função exponencial.

$$f(x) = \begin{cases} x, & \text{Se } x \geq 0 \\ \alpha(e^x - 1), & \text{Se } x < 0 \end{cases}$$

- *ThresholdedReLU (CHIENG et al., 2018)* é o *ReLU* truncado por uma constante c.

$$f(x) = \begin{cases} c, & \text{Se } x \geq c \\ x, & \text{Se } x \geq 0, x < c \\ 0, & \text{Se } x < 0 \end{cases}$$

- *Softmax (GOODFELLOW; BENGIO; COURVILLE, 2016)*, muito utilizada na última camada, aumenta a variância entre os neurônios da camada facilitando a classificação do modelo. Diferente das funções anteriores que utilizam apenas o valor do neurônio, o *Softmax* utiliza toda a camada para obter a ativação do neurônio.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

3.3.2.4 Construindo uma rede neural no Vistrails

Podemos criar uma rede neural no Vistrails em 3 passos , como visto na Figura 9:

1. Adicionar o módulo Sequential configurando a entrada do modelo (primeira camada).

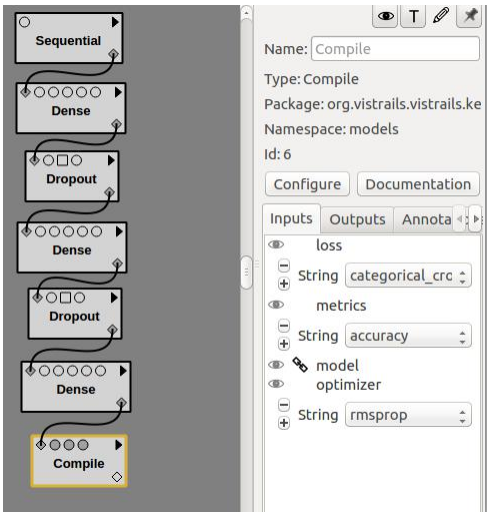
2. Adicionar as camadas com ou sem função de ativação definindo a estrutura da rede neural.

3. Adicionar o módulo Compile definindo o processo de treinamento e a partição dos dados.

Além dos módulos para a construção de redes neurais, foram importados para o Vistrails modelos de redes neurais de convolução para a classificação de imagens treinados com a base de dados ImageNet(DENG et al., 2009). As redes neurais importadas para o Vistrails são:

- *VGG16 (SIMONYAN; ZISSERMAN, 2014)*, a arquitetura *VGG* faz uma melhoria em relação a classificação de imagens contra uma das melhores redes do ano de 2015 a AlexNet substituindo kernels de convolução muito largos por kernels de tamanho 3x3 apresentando uma melhora significativa. A arquitetura *VGG16* possui 16 camadas como o nome sugere.

Figura 9 – Exemplo de uma rede neural no Vistrails



- *VGG19(SIMONYAN; ZISSERMAN, 2014)* possui as mesmas características que o *VGG16* porém ele possui mais três camadas na rede totalizando 19.

- *ResNet (HE et al., 2015a)* ou redes residuais são redes que ao invés de treinarem a partir das características das imagens como as outras redes normalmente fariam, ela treina a partir do resíduo ou seja da diferença das características já aprendidas para as características que faltam ser aprendidas. Foi adicionada a versão da rede que possui 50 camadas.

- *InceptionV3 (SZEGEDY et al., 2015)* foi uma rede criada para ter uma boa performance e menos parâmetros que seus antigos concorrentes *VGG16* e AlexNet e ainda sim foi vencedor do concurso de classificação do ImageNet em 2015.

- *Xception (CHOLLET, 2016)* , uma rede de convolução profunda criada pelo Google que significa *Extreme version of Inception*. Esta rede possui uma melhor performance na base de dados do imageNet que o seu antecessor *InceptionV3*.

- *InceptionResNetV2 (SZEGEDY; IOFFE; VANHOUCKE, 2016)* , uma evolução do modelo conhecido como *InceptionV4* ou *InceptionResNet* e como o nome sugere é uma evolução do modelo *inceptionV3* acoplando características do modelo *ResNet* de redes residuais. Adicionamos apenas a versão 2 do *InceptionResNet* pois apenas esta estava disponível para ser utilizada pelo Keras. O *InceptionResNetV2* é uma evolução do *InceptionResNet* visto que ele tende a ter um treinamento bem mais rápido que seu antecessor.

- *DenseNet (HUANG; LIU; WEINBERGER, 2016)* , um modelo criado em 2017 possui menos parâmetros que o modelo *ResNet* e ainda possui uma acurácia maior

utilizando o conjunto de dados do *ImageNet* comparado a este modelo. Foram adicionados os modelos com 121, 169 e 201 camadas que são os mais utilizados na literatura.

- *MobileNet (HOWARD et al., 2017)* , arquitetura de rede convolucional mais utilizada em um ambiente mobile ou onde há uma limitação de poder computacional. Nessa rede há uma troca de precisão por velocidade.

Todas elas podem ser utilizadas com os pesos do treinamento na base de dados do ImageNet ou com pesos inicializados aleatoriamente para classificação de imagens no conjunto de dados desejado. Na implementação deste trabalho há opção de carregar estes pesos da rede já treinada no ImageNet por meio de um parâmetro booleano do módulo chamado *"weightsImagenet"*.

Além das redes em si também é necessário detalhar sobre os otimizadores que as redes utilizam. Os otimizadores são funções que tentam minimizar a função de custo de um problema. Nas redes esta função de custo se dá com o erro entre o valor da predição de classificação e o valor real da mesma. Dentre os otimizadores que existem estes foram os que foram criados módulos no trabalho:

- SGD (BOTTOU; BOUSQUET, 2008), o Gradiente descendente estocástico (conhecido como SGD), é um método iterativo que otimiza uma função objetivo que é diferenciável. Ele pode ser considerado como uma aproximação estocástica do otimizador Gradiente descendente, já que ele substitui o gradiente (que é calculado para o dataset completo) por uma estimação do mesmo (calculada de um subconjunto selecionado aleatoriamente dos dados).

- Adagrad (DUCHI; HAZAN; SINGER, 2011), é um otimizador com taxas de aprendizado que são específicas para cada parâmetro, elas são adaptadas relativamente a quão frequentemente um parâmetro é atualizado durante o treinamento. Quanto mais atualizações um parâmetro recebe, menor é a taxa de aprendizado.

- RmsProp (TIELEMAN; HINTON, 2012), RMSprop foi introduzido nas apresentações de uma aula online de redes neurais lecionada por Geoffrey Hinton da Universidade de Toronto. Hinton não publicou o RMSprop em um artigo formal, porém ele ainda sim se tornou um dos métodos mais populares de otimização para aprendizado profundo.

- Adadelta (ZEILER, 2012), é uma versão mais robusta do Adagrad que adapta a taxa de aprendizado baseada em uma janela das atualizações dos gradientes, ao invés de acumular todos os gradientes passados. Dessa maneira, este otimizador continua aprendendo mesmo quando muitas atualizações foram feitas. Comparado

com o Adagrad, na versão original do Adadelta não é necessário definir uma taxa de aprendizado inicial.

- Adam (KINGMA; BA, 2014), pode ser visto como uma combinação do RMSprop e o SGD com momentum. Ele utiliza o gradiente ao quadrado para escalar a taxa de aprendizado assim como o RMSprop e ele tira vantagem do momentum por utilizar a média móvel do gradiente ao invés de apenas o gradiente como o SGD com momentum.
- Adamax (KINGMA; BA, 2014), É uma variante do Adam baseada na norma infinito.
- Nadam (DOZAT, 2016), Assim como Adam é essencialmente o RMSprop com momentum, Nadam é o RMSprop com o Nesterov momentum.

3.4 TREINAMENTO DO MODELO DE APRENDIZADO

Após definir o projeto, é preciso selecionar os parâmetros de cada módulo. Por conta da interface gráfica do Vistrails, a inserção de parâmetros nos módulos criados previamente se faz intuitivamente. Além disso, o Vistrails possui uma funcionalidade que é a de guardar o histórico das versões do workflow representando estas como árvores, como ilustrado na Figura 7. Por conta desta funcionalidade é possível guardar versões com diferentes parâmetros nos modelos apresentados na seção anterior assim como todas as mudanças nos modelos utilizados, tornando o processo de instanciação capaz de ser reutilizável e capaz de documentar todos os passos realizados no projeto. Para parâmetros categóricos, foi adicionado um checklist com os possíveis valores a serem adicionados. A listagem dos valores categóricos facilita a experimentação dos parâmetros.

Após definir o projeto e seus parâmetros, é possível executar o projeto e iniciar o aprendizado do modelo. Durante a execução, o Vistrails irá armazenar no cache a saída de cada módulo criando "checkpoints"durante a execução. Criou-se um módulo chamado *ModelFit* responsável por executar e treinar o modelo criado resultando em uma visualização do decorrer do algoritmo ao longo das épocas de treinamento. Além disso foi adicionado ao Vistrails módulos para carregar e salvar um modelo treinado. Após o treinamento, é importante que o pesquisador salve o modelo para poder reutilizá-lo em um novo projeto ou treiná-lo novamente.

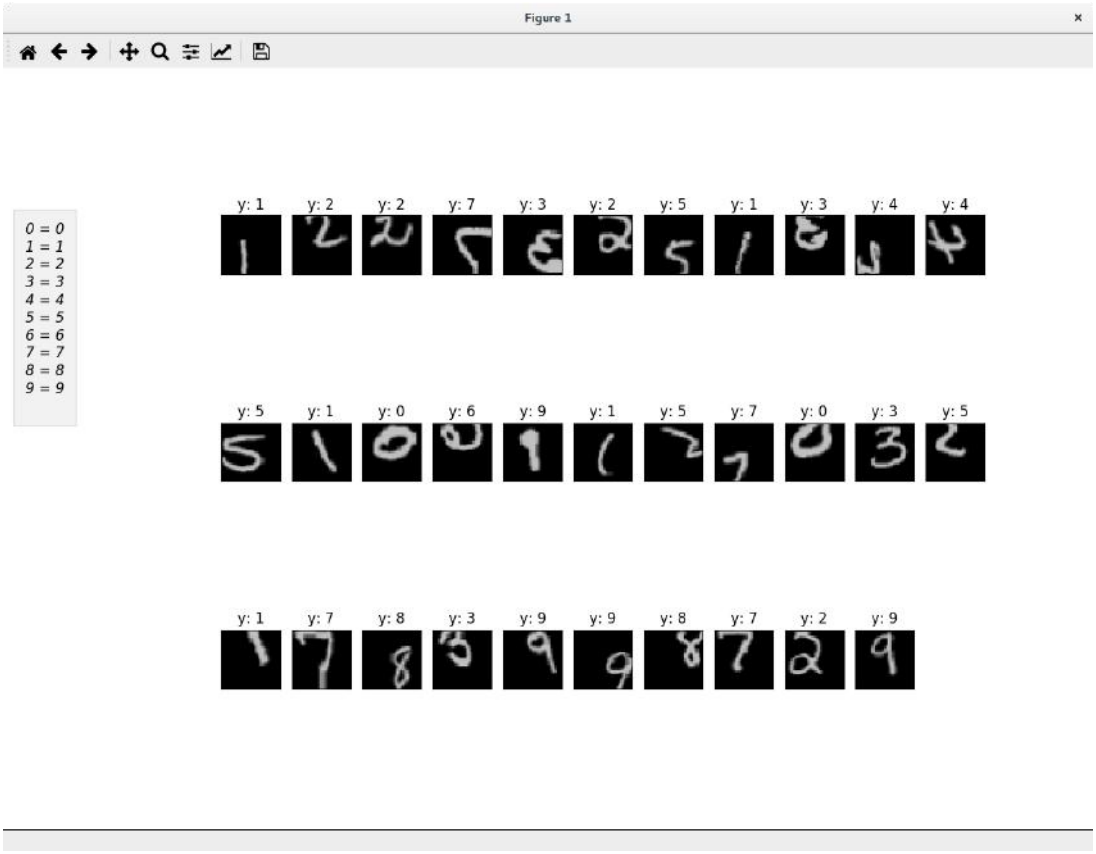
3.5 ANALISE DO RESULTADO DO TREINAMENTO

3.5.1 Visualização de dados

Por mais que o Vistrails possua a própria interface de visualização de dados, esta não atendia aos requisitos do trabalho pois possuia parâmetros que divergiam das saídas esperadas dos módulos criados anteriormente, então foi utilizada a biblioteca matplotlib

para integrar as visualizações dos módulos criados previamente. Foi criado um módulo de visualização que recupera um *batch* de imagem e o exibe. A quantidade de imagens que serão exibidas é definida pelo módulo de carregamento de imagem, em seu parâmetro *batchSize*. Por fim qualquer modificação feita na imagem em seu pré processamento também será exibida. Um exemplo de uma visualização com um batch de 32 imagens e com rotação de imagem e outras técnicas no pré processamento é vista na Figura 10.

Figura 10 – Exemplo da visualização de dados de imagens no Vistrails

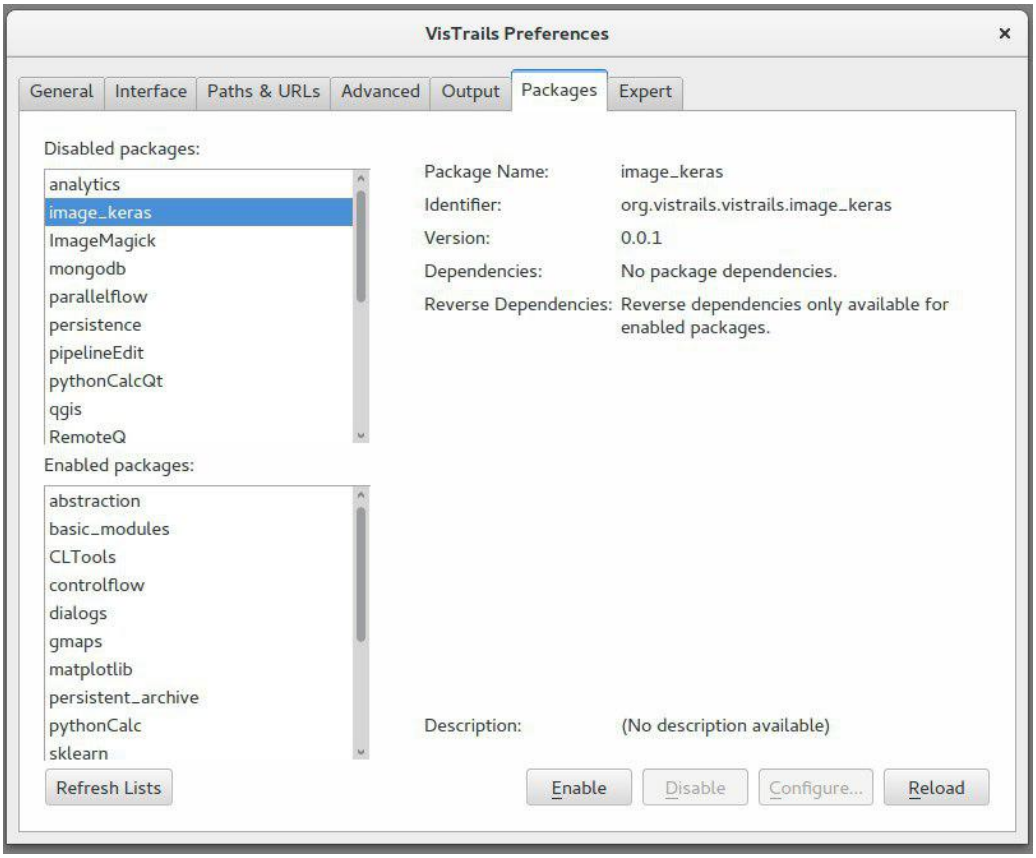


4 ESTUDO DE CASOS

Foi realizado um estudo de casos envolvendo os módulos criados no Vistrails e uma tarefa de classificação de imagens. Essa tarefa de classificação utiliza o conjunto de dados conhecido como MNIST (LECUN; CORTES, 2010), composta por imagens de números escritos à mão de 0 a 9. A classificação deste dataset é utilizada como base para testes de performance e tutoriais iniciais para a tarefa de classificação de imagens utilizando métodos de inteligência artificial. A ideia desta tarefa é conseguir treinar um modelo para aprender as características das imagens daqueles números e com isso ser possível, dado uma nova imagem de um dígito escrito à mão, predizer que número é aquele.

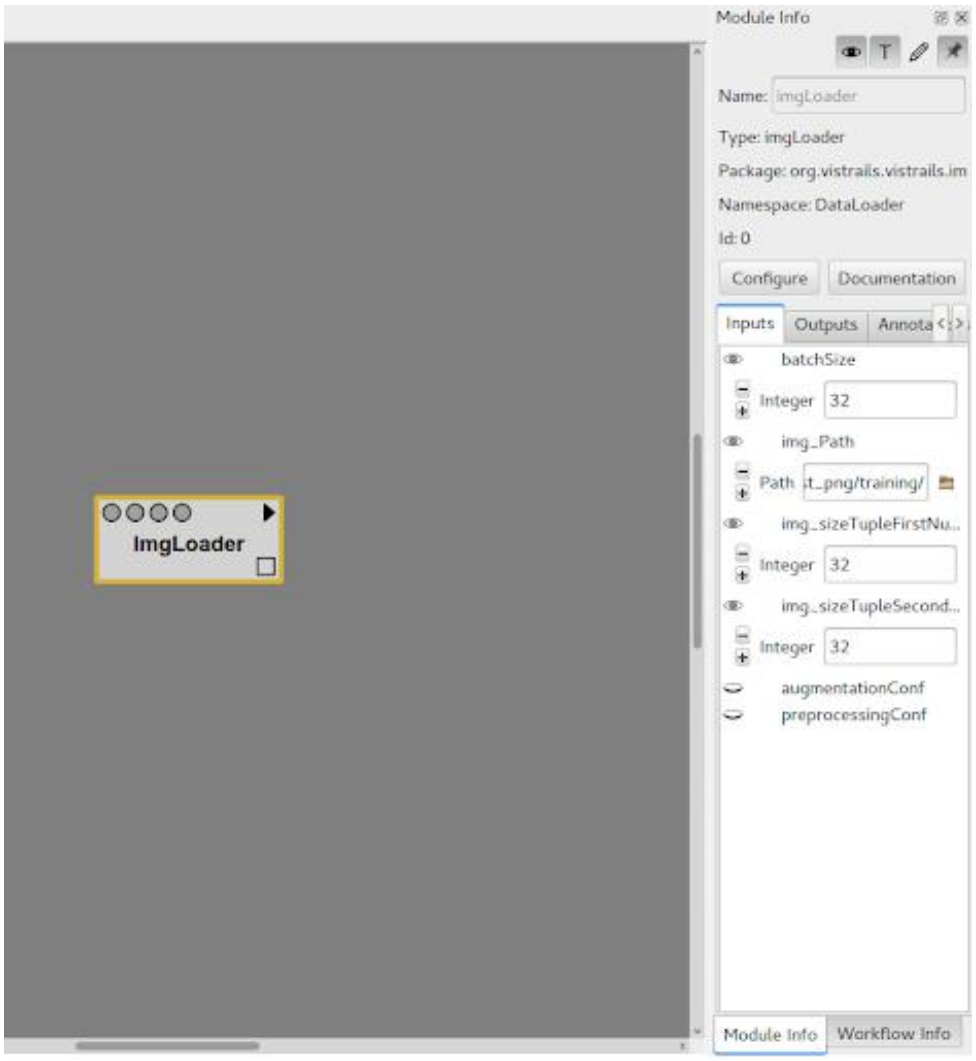
O primeiro passo para a realização da tarefa é habilitar o pacote de módulos criado no Vistrails para lidar com o carregamento e classificação de imagens. O pacote criado é chamado de *image_keras* como é possível ver na Figura 11. Nessa figura há pacotes que estão desabilitados pois não são pacotes padrão do Vistrails e temos os pacotes habilitados que são. Para habilitar um pacote basta selecionar o pacote em questão e clicar em *enable* e de forma análoga para desabilitar clicar em *disable*.

Figura 11 – Carregando um pacote no Vistrails



Após esse primeiro passo de habilitar o pacote é possível carregar as imagens que serão utilizadas para a classificação utilizando o módulo específico para tal chamado *ImgLoader*, no caso deste estudo é utilizado o dataset com suas classes divididas em diferentes pastas, cada pasta é divida entre os números de 0 a 9 e cada uma contém exemplos de imagens dos respectivos números. Por questão de simplicidade, no carregamento o dataset já está particionado em duas pastas para treinamento e validação. Na Figura 12 é possível ver este módulo sendo utilizado, nos parâmetros ao lado direito, mais especificamente no parâmetro *img_path* é possível observar a pasta onde esses arquivos de treinamento estão sendo carregados.

Figura 12 – Carregando um módulo no Vistrails

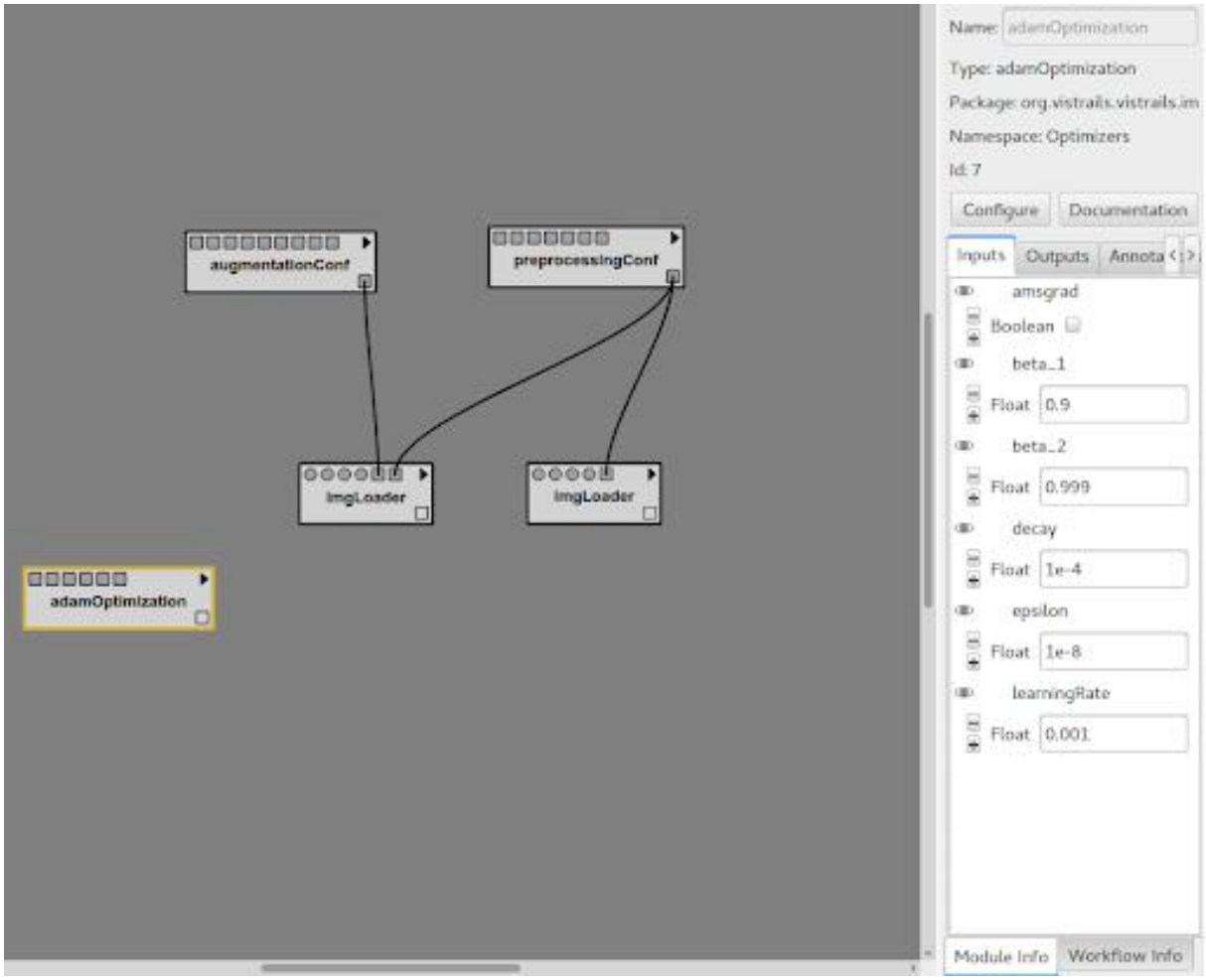


Após inserir os outros parâmetros obrigatórios como o tamanho da imagem e a quantidade de exemplos sendo utilizada por vez, é possível utilizar técnicas de ampliação de dados ou pré processamento para criar mais exemplos ou aprimorar os dados. Estas técnicas são realizadas utilizando dois outros módulos que estão presentes no mesmo pacote que

foi habilitado anteriormente são estes o módulo *augmentationConf* e *preprocessingConf*. Eles possuem apenas parâmetros de configuração opcionais que permitem ao usuário customizar como ele preferir quais técnicas sejam utilizadas em seu conjunto de dados.

Após inserir os módulos de carregamento das imagens, pode-se escolher qual modelo de rede de convolução será utilizado para a tarefa de classificação. Para esta tarefa foi escolhida a rede MobileNet por ser uma rede de convolução que consome pouco recurso computacional se comparado às outras redes, ela foi escolhida com essa característica visto que os computadores utilizados para os testes iniciais eram bem limitados. Dentre os módulos de otimizadores que foram criados foi escolhido para este trabalho o otimizador chamado Adam pois possui excelentes resultados tanto em performance quanto em velocidade de treinamento. A criação do módulo, bem como a adição de seus parâmetros podem ser vistos na Figura 13, além do Adam também é possível ver que neste trabalho foi inserido um módulo de pré processamento tanto nos dados de treino como nos de validação e um módulo de ampliação de dados no conjunto de dados de treino.

Figura 13 – Exemplo com o otimizador Adam



Para concatenar os módulos vistos anteriormente na Figura 13 assim como o módulo da rede MobileNet que não está presente na figura, é preciso utilizar o módulo chamado *ModelFit*. Este é o responsável para de fato treinar o modelo segundo as especificações inseridas previamente em cada módulo, inserindo nos parâmetros do *ModelFit* é possível especificar quantas épocas e passos o modelo do MobileNet será treinado.

E por fim, podemos também inserir um módulo de visualização das imagens para observar como elas estão sendo utilizadas pelo modelo de classificação. Executando o workflow com sucesso, o console da aplicação que está executando o vistrails executará os comandos dos módulos e treinará o modelo, este console irá mostrar o treinamento ao longo das iterações e a cada iteração ele irá apresentar os valores da função de erro (*loss* e *val_ loss*) e acurácia (*acc* e *val_ acc*), este treinamento é visto na Figura 14. Após esse treinamento é apresentada uma janela com o batch de imagens do módulo de visualização, assim como o gráfico da métrica utilizada para avaliação do modelo e o valor da função de custo ao longo das épocas.

Figura 14 – Console executando o treinamento

```
10/10 [==============================] - 8s 792ms/step - loss: 1.4736 - acc: 0.4938 - val_loss: 1.5675 - val_acc: 0.4406
Epoch 75/100
10/10 [==============================] - 8s 791ms/step - loss: 1.5160 - acc: 0.4781 - val_loss: 1.7943 - val_acc: 0.3812
Epoch 76/100
10/10 [==============================] - 8s 800ms/step - loss: 1.5894 - acc: 0.4281 - val_loss: 1.4551 - val_acc: 0.4500
Epoch 77/100
10/10 [==============================] - 8s 786ms/step - loss: 1.5654 - acc: 0.4500 - val_loss: 1.4075 - val_acc: 0.4813
Epoch 78/100
10/10 [==============================] - 8s 787ms/step - loss: 1.4553 - acc: 0.4656 - val_loss: 1.5295 - val_acc: 0.4625
Epoch 79/100
10/10 [==============================] - 8s 788ms/step - loss: 1.4875 - acc: 0.4500 - val_loss: 1.4220 - val_acc: 0.4719
Epoch 80/100
10/10 [==============================] - 8s 790ms/step - loss: 1.4276 - acc: 0.4469 - val_loss: 1.3188 - val_acc: 0.4969
Epoch 81/100
10/10 [==============================] - 8s 793ms/step - loss: 1.3775 - acc: 0.5000 - val_loss: 1.4571 - val_acc: 0.5000
Epoch 82/100
10/10 [==============================] - 8s 791ms/step - loss: 1.4233 - acc: 0.4969 - val_loss: 1.5342 - val_acc: 0.4969
Epoch 83/100
10/10 [==============================] - 8s 787ms/step - loss: 1.3724 - acc: 0.4938 - val_loss: 1.4256 - val_acc: 0.4781
Epoch 84/100
10/10 [==============================] - 8s 788ms/step - loss: 1.3882 - acc: 0.5062 - val_loss: 1.4387 - val_acc: 0.5156
Epoch 85/100
10/10 [==============================] - 8s 791ms/step - loss: 1.2919 - acc: 0.5406 - val_loss: 1.5388 - val_acc: 0.4969
Epoch 86/100
10/10 [==============================] - 8s 788ms/step - loss: 1.3682 - acc: 0.4938 - val_loss: 2.2914 - val_acc: 0.3656
Epoch 87/100
10/10 [==============================] - 8s 793ms/step - loss: 1.3685 - acc: 0.4813 - val_loss: 3.8445 - val_acc: 0.3344
Epoch 88/100
10/10 [==============================] - 8s 795ms/step - loss: 1.4261 - acc: 0.4844 - val_loss: 2.9869 - val_acc: 0.3063
Epoch 89/100
10/10 [==============================] - 8s 791ms/step - loss: 1.3191 - acc: 0.5406 - val_loss: 2.6858 - val_acc: 0.3719
Epoch 90/100
10/10 [==============================] - 8s 788ms/step - loss: 1.1617 - acc: 0.5938 - val_loss: 2.2471 - val_acc: 0.4281
Epoch 91/100
10/10 [==============================] - 8s 803ms/step - loss: 1.1986 - acc: 0.5812 - val_loss: 1.7346 - val_acc: 0.4969
Epoch 92/100
10/10 [==============================] - 8s 794ms/step - loss: 1.2017 - acc: 0.5938 - val_loss: 1.7858 - val_acc: 0.4531
Epoch 93/100
10/10 [==============================] - 8s 788ms/step - loss: 1.2928 - acc: 0.5344 - val_loss: 1.6580 - val_acc: 0.4906
Epoch 94/100
10/10 [==============================] - 8s 799ms/step - loss: 1.3297 - acc: 0.5438 - val_loss: 1.9266 - val_acc: 0.4156
Epoch 95/100
10/10 [==============================] - 8s 791ms/step - loss: 1.2298 - acc: 0.5500 - val_loss: 1.9853 - val_acc: 0.4562
Epoch 96/100
10/10 [==============================] - 8s 787ms/step - loss: 1.1698 - acc: 0.5687 - val_loss: 2.9533 - val_acc: 0.3875
Epoch 97/100
10/10 [==============================] - 8s 792ms/step - loss: 1.2188 - acc: 0.5531 - val_loss: 2.0369 - val_acc: 0.4938
Epoch 98/100
10/10 [==============================] - 8s 782ms/step - loss: 1.2291 - acc: 0.5469 - val_loss: 2.9058 - val_acc: 0.4250
Epoch 99/100
10/10 [==============================] - 8s 794ms/step - loss: 1.2893 - acc: 0.5375 - val_loss: 1.9661 - val_acc: 0.4844
Epoch 100/100
10/10 [==============================] - 8s 786ms/step - loss: 1.1208 - acc: 0.5750 - val_loss: 1.6653 - val_acc: 0.5625
```

Após essa execução sendo realizada com sucesso é possível ver nos gráficos da Figura 15 que a acurácia do modelo treinado aumentou consideravelmente assim como sua função de minimização do erro diminuiu ao longo das épocas na curva de validação. Isso caracteriza que de fato o modelo treinado está conseguindo predizer as classes com acurácia de por volta de 55%, como visto no gráfico, onde em contrapartida um modelo aleatório de previsão apenas teria uma acurácia em torno de 10% visto que há 10 classes para serem classificadas. Além disso após essa execução sendo realizada com sucesso os módulos inseridos no canvas de workflow serão imbuídos da cor verde como é possível ver na Figura 16, simbolizando que a execução ocorreu sem nenhum erro.

Figura 15 – Gráfico da execução ao longo das épocas

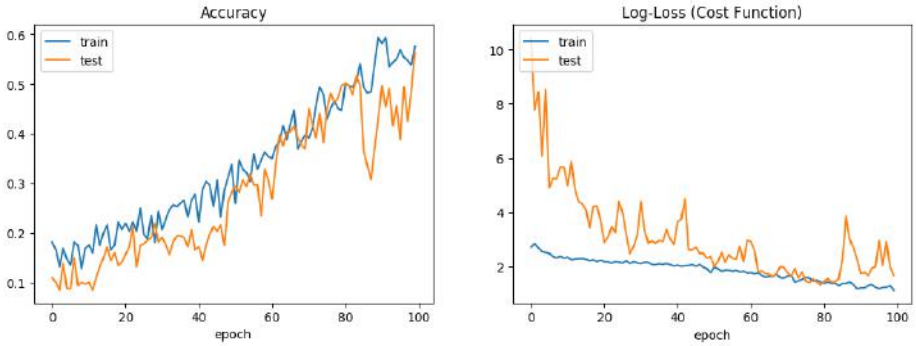
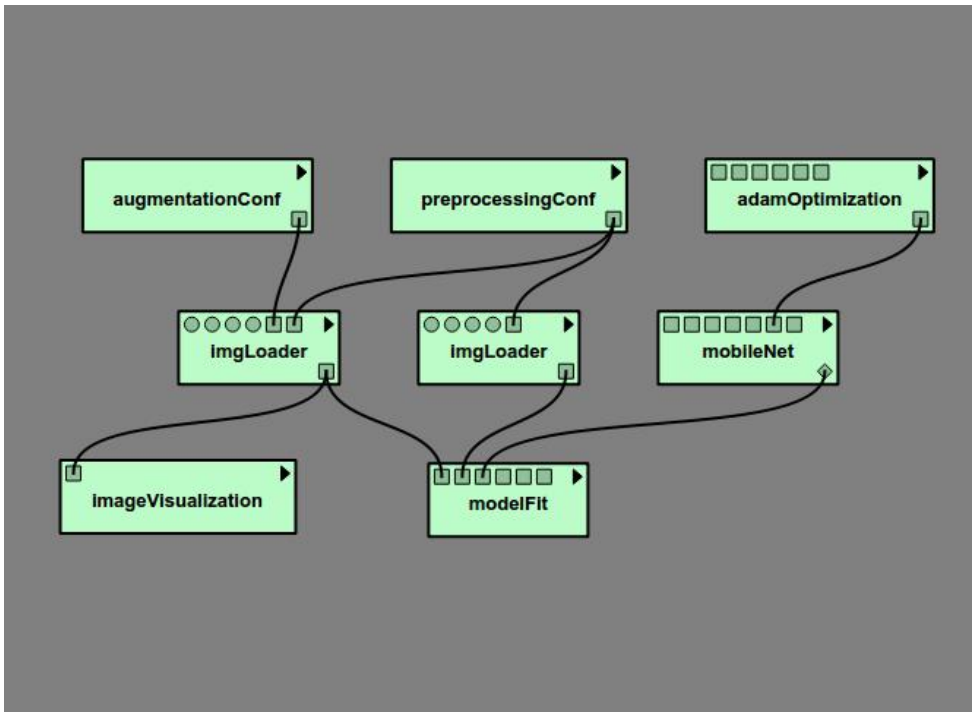


Figura 16 – Projeto finalizado



5 CONCLUSÃO

O trabalho teve um foco na experiência do usuário e não no desenvolvimento. Logo, o que deveria ser apenas uma migração das funcionalidades do Keras no Vistrails, tornou-se um desafio e dificultou o processo, visto que o código tinha de ser robusto e genérico suficiente para permitir uma flexibilidade no uso sem que seja necessário alterar o código fonte. Cada módulo desenvolvido teve de ser repensado durante o desenvolvimento pois diversos cenários imprevisíveis surgiram durante a sua validação. Vale pontuar que os módulos de pré processamento foram os que mais demandaram reimplementação visto que ele lida com diversos tipos de dados.

Quando pensamos na usabilidade e na coesão entre diversas implementações, diversos desafios de arquitetura de software surgem. Qual é a melhor forma de organizar os módulos? Como permitir que o usuário utilize cada módulo de diversas formas sem que seja necessário alterar a implementação do módulo? Visto que o Vistrails é um projeto open source, como criar um código limpo e claro que seja fácil de entender possibilitando contribuições? Neste trabalho foi preciso pensar também no código escrito por ser uma ferramenta de código aberto.

Neste trabalho aprendemos a importância de dar visibilidade a pesquisa possibilitando contribuições e este conceito não só se aplica apenas em aprendizado de máquina, podemos aplicar isso em outras áreas tanto em outras correlacionadas as ciências exatas que possuam um fluxo de execução dos seus algoritmos, como de outras áreas científicas que possuam uma série de passos que possam ser reproduzidos em suas pesquisas. É muito importante darmos acessibilidade ao conhecimento permitindo que pessoas de outros setores entendam o que está sendo feito e ainda poder contribuir com a sua visão de mundo.

5.1 TRABALHOS FUTUROS

Um dos trabalhos futuros seria aplicar técnicas de Transfer learning entre modelos no Vistrails podendo compartilhar os pesos do modelo via interface gráfica. Outra possibilidade é aplicar automaticamente técnicas de word embedding em dados categóricos.

Um tipo de função que se faz necessária no futuro é a de múltiplas formas de visualização de dados, tanto de forma estatística sobre a amostra de dados, como de formas a representar dados com muitas dimensões em dimensões menores para extrair uma visualização de como os dados se comportam naquele ambiente.

Também é necessário implementar módulos com outras arquiteturas consolidadas e que estão em utilização em abrangência como as redes generativas adversárias (Gans) (GOODFELLOW et al., 2014), alguns algoritmos de boosting como o extreme gradient boost (XGboost) (CHEN; GUESTRIN, 2016) e etc.

E para que o trabalho possa ser utilizado sem a necessidade de instalação e também possa ser utilizado de lugares distintos, para melhorar sua praticidade, é interessante que fosse implementada uma forma de acessá-lo via uma interface web provendo esse serviço sobre demanda.

REFERÊNCIAS

ANDERSEN, H.; HEPBURN, B. Scientific method. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Summer 2016. [S.l.]: Metaphysics Research Lab, Stanford University, 2016.

Apache. **Apache Airflow Home Page**. 2019. [Online; accessed 10-October-2019]. Disponível em: <https://airflow.apache.org/>. Acesso em: 10 oct.2019.

BERTHOLD, M. R. et al. KNIME: The Konstanz Information Miner. In: **Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)**. [S.l.]: Springer, 2007. ISBN 978-3-540-78239-1. ISSN 1431-8814.

BOTTOU, L.; BOUSQUET, O. The tradeoffs of large scale learning. In: PLATT, J. C. et al. (Ed.). **Advances in Neural Information Processing Systems 20**. Curran Associates, Inc., 2008. p. 161–168. Disponível em: <http://papers.nips.cc/paper/3323-the-tradeoffs-of-large-scale-learning.pdf>.

CALLAHAN, S. P. et al. Vistrails: Visualization meets data management. In: **Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2006. (SIGMOD '06), p. 745–747. ISBN 1-59593-434-0. Disponível em: <http://doi.acm.org/10.1145/1142473.1142574>.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. **CoRR**, abs/1603.02754, 2016. Disponível em: <http://arxiv.org/abs/1603.02754>.

CHIENG, H. H. et al. Flatten-t swish: a thresholded relu-swish-like activation function for deep learning. **CoRR**, abs/1812.06247, 2018. Disponível em: <http://arxiv.org/abs/1812.06247>.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. **CoRR**, abs/1610.02357, 2016. Disponível em: <http://arxiv.org/abs/1610.02357>.

CHOLLET, F. et al. **Keras**. 2015. <https://keras.io>.

CLEVERT, D.-A.; UNTERTHINER, T.; HOCHREITER, S. Fast and accurate deep network learning by exponential linear units (elus). **CoRR**, abs/1511.07289, 2015.

DENG, J. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: **CVPR09**. [S.l.: s.n.], 2009.

DOZAT, T. Incorporating nesterov momentum into adam. In: . [S.l.: s.n.], 2016.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. Jul, p. 2121–2159, 2011.

FERRUCCI, D. A. Introduction to "this is watson". **IBM J. Res. Dev.**, IBM Corp., Riverton, NJ, USA, v. 56, n. 3, p. 235–249, maio 2012. ISSN 0018-8646. Disponível em: <http://dx.doi.org/10.1147/JRD.2012.2184356>.

GAL, Y.; GHAHRAMANI, Z. A theoretically grounded application of dropout in recurrent neural networks. In: **Proceedings of the 30th International Conference on Neural Information Processing Systems**. USA: Curran Associates Inc., 2016. (NIPS'16), p. 1027–1035. ISBN 978-1-5108-3881-9. Disponível em: <http://dl.acm.org/citation.cfm?id=3157096.3157211>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. 184–185 p. <http://www.deeplearningbook.org>.

GOODFELLOW, I. J. et al. Generative adversarial nets. In: **Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2**. Cambridge, MA, USA: MIT Press, 2014. (NIPS'14), p. 2672–2680. Disponível em: <http://dl.acm.org/citation.cfm?id=2969033.2969125>.

HE, K. et al. Deep residual learning for image recognition. **CoRR**, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>.

HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. **CoRR**, abs/1502.01852, 2015. Disponível em: <http://arxiv.org/abs/1502.01852>.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **CoRR**, abs/1704.04861, 2017. Disponível em: <http://arxiv.org/abs/1704.04861>.

HUANG, G.; LIU, Z.; WEINBERGER, K. Q. Densely connected convolutional networks. **CoRR**, abs/1608.06993, 2016. Disponível em: <http://arxiv.org/abs/1608.06993>.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. **CoRR**, abs/1502.03167, 2015. Disponível em: <http://arxiv.org/abs/1502.03167>.

JIN AYSEGUL DUNDAR, E. C. J. Flattened convolutional neural networks for feedforward acceleration. dez. 2014. Disponível em: <https://arxiv.org/pdf/1412.5474.pdf>.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **CoRR**, abs/1412.6980, 2014. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14>.

LECUN PATRICK HAFFNER, L. B. Y. B. Y. Object recognition with gradient-based learning. jan. 1999. Disponível em: <http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf>.

LECUN, Y.; CORTES, C. MNIST handwritten digit database. 2010. Disponível em: <http://yann.lecun.com/exdb/mnist/>.

LUDÄSCHER, B. et al. Scientific workflows: Business as usual? In: DAYAL, U. et al. (Ed.). **Business Process Management**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 31–47. ISBN 978-3-642-03848-8.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In: **in ICML Workshop on Deep Learning for Audio, Speech and Language Processing**. [S.l.: s.n.], 2013.

MATTOS FABIO C. DA SILVA, N. R. S. M. S. d. C. A. Gerência de workflows científicos: Uma análise crítica no contexto da bioinformática. 2008. Disponível em: <https://www.cos.ufrj.br/uploadfile/1204740464.pdf>.

MATTOS FABIO COUTINHO DA SILVA, N. R. S. M. S. d. C. A. Gerência de workflows científicos: Uma análise crítica no contexto da bioinformática. fev. 2008. Disponível em: <https://www.cos.ufrj.br/uploadfile/1204740464.pdf>.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: **Proceedings of the 27th International Conference on International Conference on Machine Learning**. USA: Omnipress, 2010. (ICML'10), p. 807–814. ISBN 978-1-60558-907-7. Disponível em: <http://dl.acm.org/citation.cfm?id=3104322.3104425>.

PEREZ, L.; WANG, J. The effectiveness of data augmentation in image classification using deep learning. **CoRR**, abs/1712.04621, 2017. Disponível em: <http://arxiv.org/abs/1712.04621>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **CoRR**, abs/1409.1556, 2014.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, p. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V. Inception-v4, inception-resnet and the impact of residual connections on learning. **CoRR**, abs/1602.07261, 2016. Disponível em: <http://arxiv.org/abs/1602.07261>.

SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. **CoRR**, abs/1512.00567, 2015. Disponível em: <http://arxiv.org/abs/1512.00567>.

TIELEMAN, T.; HINTON, G. **Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude**. 2012. COURSERA: Neural Networks for Machine Learning.

Vistrails. **Version Tree View**. 2019. [Online; accessed 10-October-2019]. Disponível em: <https://www.vistrails.org/usersguide/dev/html/version_tree.html#version-tree-view>. Acesso em: 10 oct.2019.

WESKE, M.; VOSSEN, G.; MEDEIROS, C. B. Scientific workflow management: Wasa architecture and applications. Universität Münster. Angewandte Mathematik und Informatik, 01 1996.

ZEILER, M. D. ADADELTA: an adaptive learning rate method. **CoRR**, abs/1212.5701, 2012. Disponível em: <http://arxiv.org/abs/1212.5701>.

APÊNDICES

APÊNDICE A – MÓDULOS ADICIONADOS NO VISTRAILS

Módulo	Descrição
imgLoader	Carrega as imagens, atribui a quantidade de imagens em um batch e define o tamanho das imagens.
augmentationConf	Define as configurações de ampliação de dados
preprocessingConf	Define as configurações de pré processamento
imageVisualization	Visualiza o batch de imagens com o tamanho definido pelo imgLoader com o pré processamento se ele tiver sido realizado
sgdOptimization	Módulo que define os parâmetros do otimizador SGD para ser usado em um modelo
rmsPropOptimization	Módulo que define os parâmetros do otimizador RMSprop para ser usado em um modelo
adagradOptimization	Módulo que define os parâmetros do otimizador Adagrad para ser usado em um modelo
adadeltaOptimization	Módulo que define os parâmetros do otimizador Adadelta para ser usado em um modelo
adamOptimization	Módulo que define os parâmetros do otimizador Adam para ser usado em um modelo
adamaxOptimization	Módulo que define os parâmetros do otimizador Adamax para ser usado em um modelo
nadamOptimization	Módulo que define os parâmetros do otimizador Nadam para ser usado em um modelo
resnet50	Utiliza o modelo de classificação de imagem Resnet50
inceptionV3	Utiliza o modelo de classificação de imagem inceptionV3
VGG16	Utiliza o modelo de classificação de imagem VGG16
mobileNet	Utiliza o modelo de classificação de de imagem mobileNet
xception	Utiliza o modelo de classificação de de imagem xception
VGG19	Utiliza o modelo de classificação de de imagem VGG19
inceptionResNetV2	Utiliza o modelo de classificação de de imagem inceptionResNetV2
denseNet121	Utiliza o modelo de classificação de de imagem denseNet121
denseNet169	Utiliza o modelo de classificação de de imagem denseNet169

Sequential	Responsável por iniciar a rede neural criando a primeira camada.
Compile	Responsável por configurar o treinamento da rede neural.
Fit	Responsável por executar o treinamento da rede neural.
FitGenerator	Responsável por executar o treinamento da rede neural com um grande volume de dados.
Evaluate	Responsável por exibir a performance da rede neural.
SaveModel	Responsável por salvar o modelo treinado.
LoadModel	Responsável por carregar o modelo treinado.
Dense	Camada mais simples de uma rede neural que consiste em conectar cada neurônio da camada anterior com todos da camada seguinte.
LSTM	Tipo de camada otimizada para analisar dados temporais.
Conv2D	Tipo de camada otimizada para analisar dados de duas dimensões como, por exemplo, fotos
Flatten	Utilizada para conectar uma Conv2D a uma Dense, ela consiste em reduzir a dimensão de uma Conv2D para 1D
Embedding	Camada otimizada para analisar dados categóricos
Batch Normalization	Técnica de normalização dos pesos da camada para evitar overfitting.
ReLU	Função de ativação mais simples que retorna zero quando a entrada é negativa.
LeakyReLU	Variação do ReLU que ao invés de zerar quando a entrada for negativa ele multiplica por uma constante alpha definida via parâmetro.
PreLU	Uma otimização do LeakyReLU que otimiza a escolha do alpha automaticamente.
ELU	Variação do ReLU utilizando uma função exponencial.
ThresholdedReLU	É o ReLU truncado por uma constante c.
Softmax	Muito utilizada na última camada, aumenta a variância entre os neurônios da camada facilitando a clas-sificação do modelo.
ReadCSV	Importa um dado do tipo csv no Vistrais
DataClassification Preprocessing	Recebe o dado no formato CSV e prepara o dado para um modelo classificador.
Sample	Retorna uma amostra da base de dados.
SplitCol	Particiona o dado por um conjunto de colunas.