UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ASTROLAB: UMA FERRAMENTA PARA IDENTIFICAÇÃO E CLASSIFICAÇÃO DE CORPOS CELESTES

WILLIAM MAURÍCIO GLÜCK

BLUMENAU
2016

WILLIAM MAURÍCIO GLÜCK

ASTROLAB: UMA FERRAMENTA PARA IDENTIFICAÇÃO E CLASSIFICAÇÃO DE CORPOS CELESTES

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Aurélio Faustino Hoppe - Orientador

BLUMENAU
2016

ASTROLAB: UMA FERRAMENTA PARA IDENTIFICAÇÃO E CLASSIFICAÇÃO DE CORPOS CELESTES

Por

WILLIAM MAURÍCIO GLÜCK

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
Membro:	Prof. Daniel Theisges dos Santos, Mestre – FURB
Membro:	Profª. Luciana Pereira de Araújo, Mestra – FURB

Blumenau, 05 de julho de 2016

Dedico este trabalho a meus amigos e familiares.

AGRADECIMENTOS

A minha família, pelo apoio e incentivo.

A toda a turma do Bruf, minha segunda família, por sempre estarem do meu lado quando precisei.

A todas as pessoas que estiveram próximas de mim durante os últimos 4 anos e meio da minha vida, de conhecidos à amigos.

A meu gerente Silvio Etges, por toda a paciência e incentivo nesse momento final da graduação.

Ao meu orientador Aurélio Faustino Hoppe, pela paciência, pressão e apoio durante o desenvolvimento desse trabalho.

Imagination will often carry us to worlds that never were. But without it we go nowhere.

Carl Sagan

RESUMO

A classificação morfológica de galáxias é fundamental para que a astronomia possa ter um maior entendimento sobre a origem e evolução do universo. Este trabalho descreve o desenvolvimento de uma ferramenta que realiza a classificação morfológica de galáxias através de redes neurais convolucionais. O *design* utilizado possui duas camadas de convolução, duas de *max pooling*, uma densa totalmente conectada com *dropout* e uma de saída utilizando *softmax*. A base de dados utilizada foi criada a partir do *Sloan Digital Sky Survey* (SDSS) e dos dados disponibilizados pelo projeto Galaxy Zoo 2. Ela contém 107.620 imagens para treino e 978 imagens para teste. Fez-se necessário a criação de uma etapa de processamento visando reduzir o ruído nas imagens utilizadas. A ferramenta foi desenvolvida na linguagem de programação Python juntamente com bibliotecas para realizar o processamento das imagens e a criação da rede neural. Os testes realizados apresentam uma taxa de acerto de 86.40% na base de testes e 97.5% na base de treino.

Palavras-chave: Classificação morfológica de galáxias. Redes neurais convolucionais. Processamento de imagens.

ABSTRACT

Galaxy morphological classification is essential for astronomy. It allows a greater understanding of the origin and evolution of the universe. This work describes the development of a tool that performs morphological classification of galaxies using convolutional neural networks. The used design has two convolution layers, two max pooling layers, a fully dense connected layer using dropout and a output layer with softmax. The database used was created from the Sloan Digital Sky Survey (SDSS) and data provided by the project Galaxy Zoo 2. It contains 107,620 images for training and 978 test images. It was needed the creation of a processing step to reduce noise in the images used. The tool was developed in the Python programming language together with libraries to perform image processing and the creation of the neural network. Tests show a 86.40% success rate on the test dataset and 97.5% at the training dataset.

Key-words: Galaxy morphological classification. Convolutional neural networks. Image processing.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GPU – *Graphics Processing Unit*

RF – Requisito Funcional

RNA – Rede Neural Artifical

RNF – Requisito Não Funcional

SDSS – Sloan Digital Sky Survey

SVM – *Support Vector Machines*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Classificar galáxias, por quê? Segundo Ball (2008), a classificação de galáxias não é um fim, mas sim um meio para um maior entendimento da física das galáxias. Na literatura, tem-se vários exemplos de áreas que através de bons métodos de classificação levaram a melhora no entendimento de determinado assunto. Elfattah (2013) cita que a tabela periódica é um bom exemplo. Ela foi organizada de acordo com as características próprias de cada elemento, resultando no entendimento da estrutura do átomo e consequente descobrimento de novos elementos. Da mesma forma, o entendimento das galáxias pode dar importantes dicas da origem e evolução do universo.

Antes de 1920, a astronomia ainda não tinha certeza se galáxias eram sistemas separados da Via Láctea ou apenas um tipo de nébula que a integrava (HUBBLE, 1936). Isso é um exemplo do estado inicial que se encontra o estudo de galáxias e, explica porque, ainda hoje, existem estudos tentando realizar classificações básicas de galáxias (BALL, 2008). O projeto Galaxy Zoo 2 que classificou em 2013 a morfologia de 304.122 galáxias, é um exemplo disso (WILLETT et al., 2013).

Atualmente, um observatório pode capturar mais de dez mil imagens astronômicas por noite. O processo tradicional de análise é realizado por um humano especialista. Durante o processo, imagens que contém objetos de interesse, estrelas e galáxias, por exemplo, são adicionadas a coleção do observatório. Caso o objeto de interesse seja uma galáxia, ela é então classificada de acordo com algum sistema de classificação morfológica. Esse tipo de classificação vem se tornando impraticável frente ao enorme volume de dados (JENKINSON, 2014).

Para Jenkinson (2014), a solução para este problema está na utilização de técnicas de processamento de imagens e aprendizado de máquina. Essas técnicas podem ser utilizadas na criação de classificadores automatizados com capacidade de lidar com situações não lineares, como as apresentadas pela distribuição dos parâmetros em galáxias.

Diante do exposto, o trabalho apresenta uma ferramenta que, a partir de imagens astronômicas, realize a classificação morfológica de galáxia.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver uma ferramenta que, aplicando técnicas de processamento de imagem e inteligência artificial em imagens astronômicas, realize a classificação morfológica de galáxias.

Os objetivos específicos do trabalho são:

a) processar imagens de galáxias realizando a diminuição de ruído;
b) criar e treinar uma rede neural artificial convolucional que aprenda, a partir de um conjunto de dados de treinamento, a classificar galáxias a partir das suas características morfológicas.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. No primeiro é realizada a introdução do trabalho e seus objetivos. O segundo capítulo apresenta a fundamentação teórica relevante para o desenvolvimento, explicando conceitos gerais e apresentando trabalhos correlatos ao proposto. O terceiro capítulo descreve, passo a passo, o desenvolvimento da ferramenta, listando seus requisitos, especificação e as técnicas e ferramentas utilizadas. Um exemplo de operacionalidade também é demonstrado e os resultados obtidos são discutidos. Por fim, o quarto capítulo apresenta as conclusões e possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está dividido em 3 seções. A seção 2.1 descreve o sistema Hubble de classificação morfológica de galáxias. A seção 2.2 apresenta a estrutura e funcionamento de uma rede neural convolucional. Por fim, na seção 2.3, são apresentados os trabalhos correlatos com abordagens para classificação de galáxias, através de processamento de imagens e classificação com algoritmos baseados em redes neurais.

2.1 SISTEMA HUBBLE DE CLASSIFICAÇÃO MORFOLÓGICA DE GALÁXIAS

Especulações sobre a natureza das galáxias datam de meados do século 18. Ainda no ano de 1921 se discutia se esses nebulosos objetos vistos no céu eram parte da nossa galáxia ou entidades separadas. No ano de 1924 essa discussão teve fim em estudos realizados por Hubble (HUBBLE, 1936).

Em 1926, Edwin Hubble criou o sistema Hubble de classificação morfológica de galáxias. É o sistema de classificação mais conhecido nos dias de hoje e mantém sua essência básica desde sua criação (BALL, 2008). Conforme descreve Serrano (2010), em sua primeira versão, com base na aparência óptica de imagens de galáxias em placas fotográficas, Hubble propôs três classes gerais: elípticas, espirais e irregulares.

Todas as observações realizadas por Hubble partiram de centenas de galáxias. Das mais brilhantes as mais vagas, a coleção visava ser uma amostra relevante que podia ser utilizada para a descoberta de características comuns a outras galáxias (HUBBLE, 1936).

O sistema Hubble é também chamado de sequência Hubble. O nome derivou da crença de que a morfologia de galáxias representava uma evolução temporal. Nessa linha evolutiva as galáxias elípticas teriam se formado e passado por mais dois estágios em sequência: lenticulares e elípticas. Por isso, ainda hoje, as galáxias elípticas são ditas de tipo precoce, e as espirais de tipo tardio (SERRANO, 2010).

Após alterações, em 1936, a versão final do sistema de Hubble apresentava 4 tipos de classificação: elíptica, lenticular, espiral e irregular. O tipo hipotético S0 foi adicionado diante da necessidade de um estado intermediário entre as classes elípticas e espirais, enquanto que o tipo irregular agrupava todas galáxias que não se encaixavam na sequência. (SERRANO, 2010).

Segundo Ata et al. (2009, p.157), todos os tipos e subtipos são denotados por uma sigla de identificação. Hubble propôs as seguintes classificações: (i) Elíptica: E0, E3, E5, e E7; (ii) Espiral: S0, Sa, Sb, Sc, e Sd; (iii) Espiral barrada: SBa, SBb, e SBc; e (iv) Irregular: Im, e

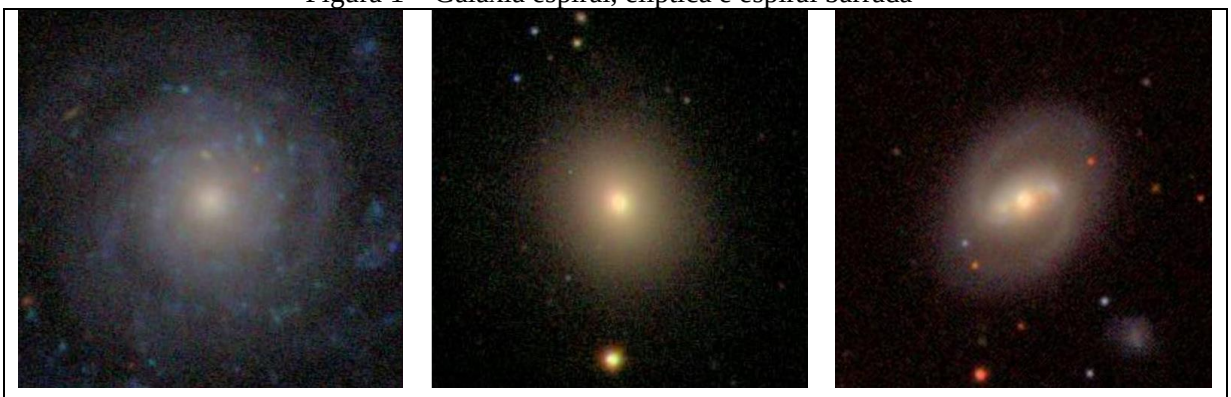
Ibm. Entre as diversas características levantadas para realizar a classificação, conforme exposto por Ata et al. (2009), 5 merecem destaque especial:

a) o tamanho do núcleo central em relação ao disco externo. Se dividem em galáxias com núcleo, mas sem disco (elípticas ‘E’), com disco, mas sem núcleo (espirais de tipo tardio ‘Sd’ e irregulares ‘Irr’) e com núcleo e disco (de acordo com o tamanho do núcleo, do maior para o menor respectivamente, ‘S0’, ‘Sa’, ‘Sb’ e ‘Sc’);
b) a presença ou ausência de braços em espiral. Se dividem em galáxias com braços em espiral (‘Sa’, ‘Sb’, ‘Sc’ e ‘Sd’), com disco suave, mas sem braços em espiral (‘S0’) e com disco irregular, mas sem braços em espiral (‘Irr’);
c) a proximidade dos braços em espiral em relação ao centro da galáxia. Se dividem em galáxias com os braços em espiral próximos do núcleo (‘Sa' e ‘Sb’) e com os braços em espiral afastados do núcleo (‘Sc’ e ‘Sd’);
d) a granulosidade dos braços em espiral. Se dividem em galáxias com baixa granulosidade (‘Sa’ e ‘Sb’) e com alta granulosidade (‘Sc’ e ‘Sd’);
e) a presença ou ausência de barras. Galáxias muitas vezes apresentam arranjos lineares que são distintos do seu bojo central, eles são conhecidos como barras. Quando uma galáxia espiral possui barras ela tem um ‘B’ adicionado em sua notação.

De forma geral as galáxias de tipo precoce (elípticas) são identificadas como globos ou elipses. Elas possuem centros brilhantes que decaem em brilho rapidamente em direção a borda. Não apresentam, em sua maioria, estruturas visíveis que se destacam (HUBBLE, 1926). Galáxias lenticulares, por sua vez, representam um estado intermediário onde galáxias elípticas começam a apresentar um disco sem braços visíveis. Elas evoluem para galáxias espirais quando braços começam a se tornar visíveis no seu disco (HUBBLE, 1936).

As galáxias espirais se dividem em barradas e normais. Elas se diferenciam pelo seu núcleo, que pode ou não possuir um arranjo em forma de barra. A evolução das galáxias espirais faz com que braços inicialmente discretos e próximos do núcleo fiquem mais visíveis e afastados, e o núcleo, inicialmente brilhante, fique cada vez mais apagado (HUBBLE, 1936). A Figura 1 exibe uma galáxia espiral, uma galáxia elíptica e uma galáxia espiral barrada.

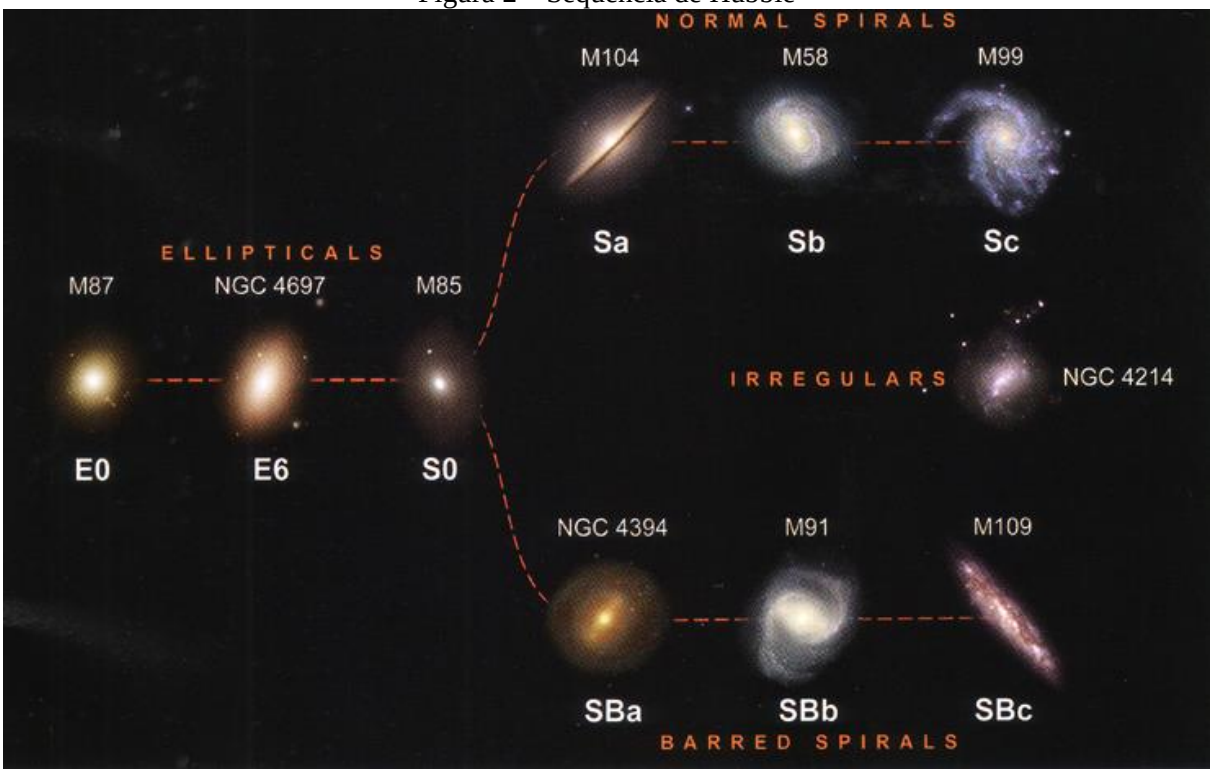
Figura 1 – Galáxia espiral, elíptica e espiral barrada



Galáxia espiral | Galáxia elíptica | Galáxia espiral barrada

O diagrama da sequência Hubble pode ser visualizado na Figura 2. A imagem exibe a evolução temporal da morfologia das galáxias. À esquerda as galáxias elípticas de tipo precoce, à direita as galáxias espirais e espirais barradas.

Figura 2 – Sequência de Hubble



Fonte: Ata et al. (2009).

Segundo Ball (2008) apesar de se basear apenas na aparência óptica da luz das galáxias próximas, a sequência Hubble se correlaciona muito bem com muitos outros parâmetros físicos gerais destas galáxias. Apesar de sua longa existência, ela ainda é o método de classificação morfológica de galáxias mais utilizado nos dias de hoje (SERRANO, 2010).

2.2 REDES NEURAIS CONVOLUCIONAIS

O propósito de um algoritmo classificador, conforme descrito por Ball (2008, p.21), é mapear um conjunto de entradas para a saída correta (ou saídas). As entradas são os parâmetros para o objeto em questão e a saída a classificação.

Ball (2008, p.20) afirma que Redes Neurais Artificiais (RNAs) são estruturas que simulam a estrutura neural do cérebro, no sentido de que elas consistem em neurônios agindo como nodos que são capazes de realizar processamento. Esse tipo de algoritmo é recomendado em situações de classificação no qual os dados de entrada podem apresentar ruído e complexidade maiores das que métodos de classificação tradicional conseguem lidar (MITCHELL, 1997).

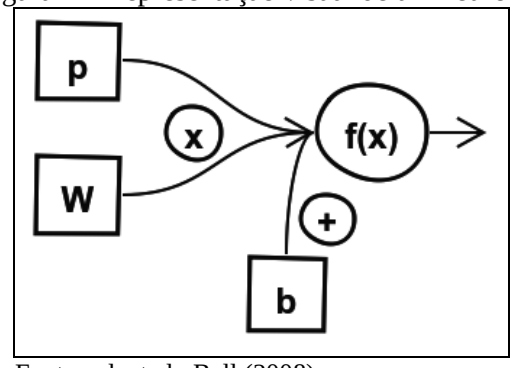
A estrutura básica de um neurônio pode ser definida em entrada (p), saída (a), pesos (W), unidade *bias* (valor originado de um neurônio especial que não possui entrada de dados e é representado pela letra b. Serve para aumentar a capacidade de adaptação da rede neural) e uma função de transferência (f) que opera sobre as entradas para gerar a saída (Figura 3). Pesos e entrada são multiplicados e após a soma da unidade *bias* sobre o resultado a função de ativação é aplicada, gerando a saída. Os neurônios podem possuir um ou mais parâmetros de entrada. (BALL, 2008). A Figura 4 exibe a representação visual de um neurônio.

Figura 3 – Definição matemática de um neurônio

$$a = f(\mathbf{W}\mathbf{p} + b)$$

Fonte: Ball (2008).

Figura 4 – Representação visual de um neurônio



Fonte: adaptado Ball (2008).

Ball (2008) descreve algumas das funções de transferência utilizadas por redes neurais para classificação:

a) *hard limit*, se a entrada é inferior a determinado valor a saída é 1, caso a entrada seja superior a este valor a saída é 0;
b) linear, a saída é um múltiplo linearmente dimensionado da saída;

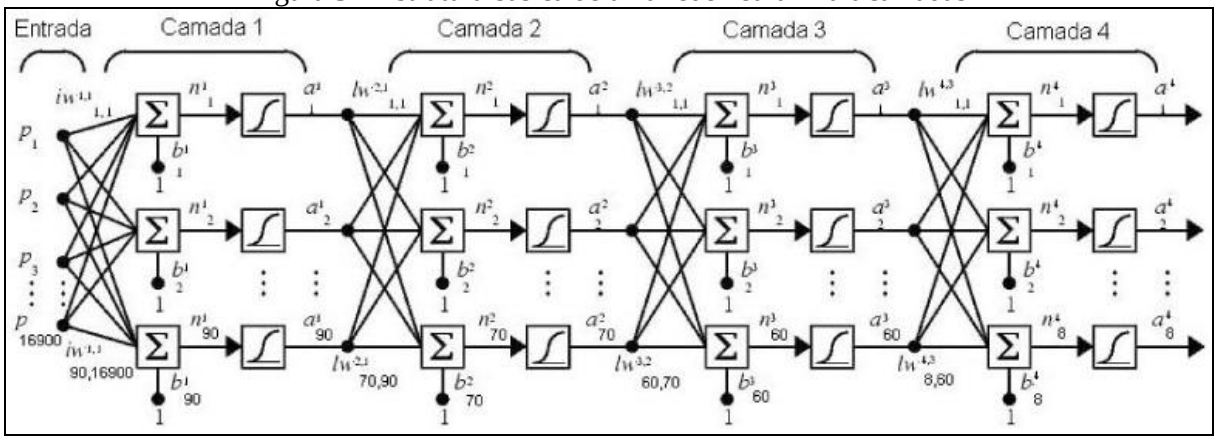
c) sigmoide, a saída é um valor entre 0 e 1;

d) *tahn*, similar a função de transferência sigmoide, a saída é um valor entre -1 e 1.

A função de ativação ReLU, apresentada no ano 2000, teve rápida adoção nos últimos anos. Diante de melhores resultados apresentados durante treinamento, a maioria dos pesquisadores atualmente a recomenda. Seu valor de saída é sempre entre -1 e 1 (HEATON, 2015).

Cada neurônio isolado pode ser generalizado para um conjunto de neurônios organizados em uma camada de um ou mais neurônios (S), onde um vetor (R) representa os parâmetros de entrada e uma matriz (S por R) representa os pesos. Uma rede neural pode ser formada por uma ou mais camadas de neurônios, com quantidade de neurônios variável. Em uma de suas estruturas mais básicas, cada camada interna recebe de entrada a saída da camada anterior. A Figura 5 mostra visualmente a estrutura de uma rede neural multicamadas com múltiplas saídas (BALL, 2008).

Figura 5 – Estrutura básica de uma rede neural multicamadas



Fonte: Ball (2008).

Para que a classificação realizada pela rede neural tenha uma porcentagem de acerto relevante é necessário que os valores da matriz de pesos sejam ajustados, visando diminuir a soma dos erros das classificações. O erro das classificações é calculado através de uma função de custo, que tem seu valor reduzido com algum algoritmo de redução de gradiente. Esse treinamento pode ser realizado de maneira supervisionada, através de um conjunto de dados previamente mapeados corretamente, ou de maneira não supervisionada (BALL, 2008). Em uma rede neural que possui aprendizado não supervisionado, segundo Elfattah et al. (2013, p. 170), não há nenhum mestre que diz o que é verdadeiro ou falso. Assim, nenhum conhecimento anterior sobre a estrutura dos dados é necessário para que o algoritmo possa ser utilizado. O algoritmo mais comumente utilizado para realizar o treinamento de uma rede neural de forma supervisionada é o *backpropagation* (BALL, 2008).

Conforme exposto por Ball (2008) a fase de treinamento, independente do algoritmo utilizado, não procura atingir o cenário ideal. Algumas técnicas podem ser adotadas para evitar que o treinamento continue indefinitivamente. Para redes *percepton* multicamadas as seguintes técnicas são utilizadas:

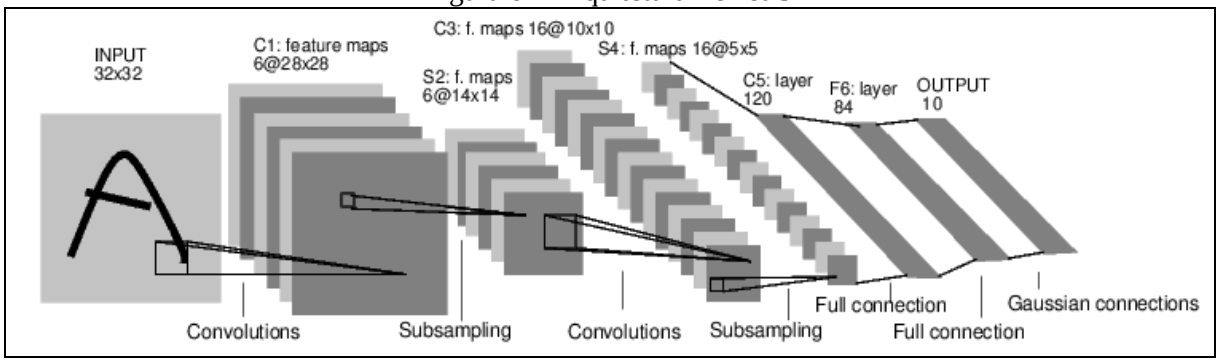
a) treinar até que o erro esteja abaixo de determinado valor;
b) treinar um número determinado de vezes;
c) treinar um período de tempo determinado.

Muitos outros tipos de arquiteturas existem: *feedforward* generalizada (forma generalizada da *perceptron* multicamadas onde uma camada de neurônios pode pular uma ou mais camadas de neurônios, conseguindo resolver problemas de maneira mais eficiente), mapas auto organizáveis (realiza treinamento de maneira não supervisionada), entre outras, cada qual com suas vantagens e características (ATA, 2009).

Uma arquitetura que teve grande impacto na área de visão computacional é o modelo convolucional. Com base em anos de estudo sobre a biologia do olho, essas redes neurais levaram a área de reconhecimento de imagens a outro patamar. Elas utilizam campos sobrepostos de entrada de dados para simular características presentes nos olhos biológicos. Através dessas técnicas, possuem a habilidade de processar escala, rotação e sujeira em imagens (HEATON, 2015).

Dos diversos designs de redes neurais convolucionais, a rede LeNet-5, apresentada por LeCun (1998), pode ser dada como um exemplo clássico utilizado para classificação de imagens. Seu formato original possui três tipos de camadas: convolucionais, de *max pooling* e camadas densas. LeCun (1998) utilizou essa configuração para a realizar a classificação de dígitos e caracteres. A Figura 6 mostra a estrutura da rede neural LeNet-5.

Figura 6 – Arquitetura LeNet-5



Fonte: LeCun (1998).

Segundo Heaton (2015), o objetivo primordial de uma camada de convolução é a detecção de bordas, linhas e outros elementos visuais. Os parâmetros básicos que uma camada

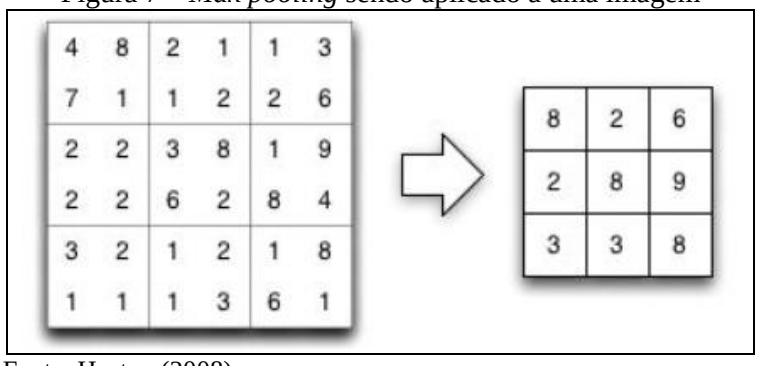
convolucional necessita para sua definição são: Número de filtros que serão aplicados, tamanho dos filtros, *stride* (tamanho dos passos que os filtros devem realizar ao percorrerem a imagem), *padding* em pixels que deve ser adicionado na imagem original e a função de ativação a ser utilizada.

O componente básico de uma camada convolucional, o filtro, é um objeto em forma de quadrado que percorre a imagem da esquerda para direta e de cima para baixo, se locomovendo de acordo com o seu *stride*. Durante cada passo os pesos da camada são multiplicados aos pixels e somados à unidade *bias.* Sobre o resultado dessa operação é aplicada a função de ativação. Filtros são os responsáveis pela detecção de características importantes nas imagens. (HEATON, 2015).

A camada de *max-pooling* não realiza treinamento nem aplica função de ativação. Ela tem como único objetivo reduzir a dimensão dos dados recebidos, mantendo sempre os pixels de maior valor. São utilizadas normalmente após uma camada de convolução. Esta técnica pode reduzir o sobre ajuste (treinamento excessivo que diminui a capacidade de generalização da classificação realizada pela rede neural) dos pesos durante o treinamento (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Os parâmetros que definem uma camada de *max-pooling* são dimensão e *stride*. Caso uma camada de *max-pooling* tenha dimensão de 2 e *stride* de 2, por exemplo, a imagem de entrada será reduzida pela metade. Caso exista mais de um canal de cor na imagem, o pixel de maior valor entre todos os canais é o mantido (HEATON, 2015). A Figura 7 dá um exemplo de *max-pooling* sendo aplicado a uma imagem.

Figura 7 – *Max pooling* sendo aplicado a uma imagem



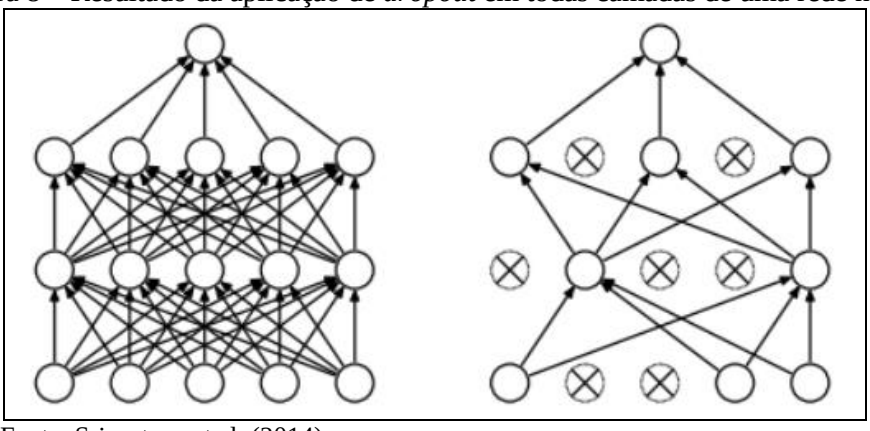
Fonte: Heaton (2008).

As camadas finais da rede LeNet-5 formam uma rede neural *perceptron* multicamadas tradicional. Algumas camadas podem ser adicionadas à LeNet-5 visando melhora na representação e resultado da classificação realizada, como uma camada de regularização *dropout* e uma camada de saída com a função de ativação *softmax* (HEATON, 2015).

Dropout é um algoritmo de regularização que comumente é adicionado a redes neurais na forma de camada. A diferença desta camada para uma densa tradicional é que ela possui uma probabilidade, que pode ser alterada, de eliminar neurônios e suas ligações temporariamente durante o treinamento. Seu funcionamento se baseia na redução da chance de dois neurônios criarem uma dependência muito forte. Isso resulta em menos sobre ajuste (HEATON, 2015).

Srivastava et al. (2014) conseguiu resultados superiores em todos os testes realizados após a aplicação da técnica. O algoritmo pode ser aplicado em camadas seguidas ou isoladas. A Figura 8 exibe uma rede neural *feed-forward* multicamadas sem *dropout* (esquerda) e após a aplicação da técnica nas três camadas (direita).

Figura 8 – Resultado da aplicação de *dropout* em todas camadas de uma rede neural



Fonte: Srivastava et al. (2014).

Softmax é uma função de ativação normalmente encontrada na camada final de uma rede neural. Diferente da função de ativação linear na qual apenas o maior valor de saída representa a classificação realizada, a função *softmax* retorna um vetor indicando a probabilidade de cada possibilidade no processo de classificação. O Quadro 1 exibe código em Python que implementa a função *softmax.*

Quadro 1 – Implementação da função de ativação *softmax*

```
01 def softmax(neuron_output):
02       total_sum = 0
03       for value in neuron_output:
04             total_sum = total_sum + value
05       total_sum = math.exp(total_sum)
06       prob = []
07       for i in range(len(neuron_output))
08             prob[i] = math.exp(neuron_output[i]) / total_sum
09       return prob
```

Fonte: adaptado de Heaton (2015).

Uma rede neural com uma camada de saída utilizando a função de ativação linear, terá como resultado um vetor que pode ter como soma um valor maior que 1. A função *softmax* realiza a soma de todos os valores, calcula a exponencial dessa soma e divide esse valor pela

exponencial de cada item. Isso transforma a saída em um vetor de probabilidades, que tem como soma 1 (HEATON, 2015).

2.3 TRABALHOS CORRELATOS

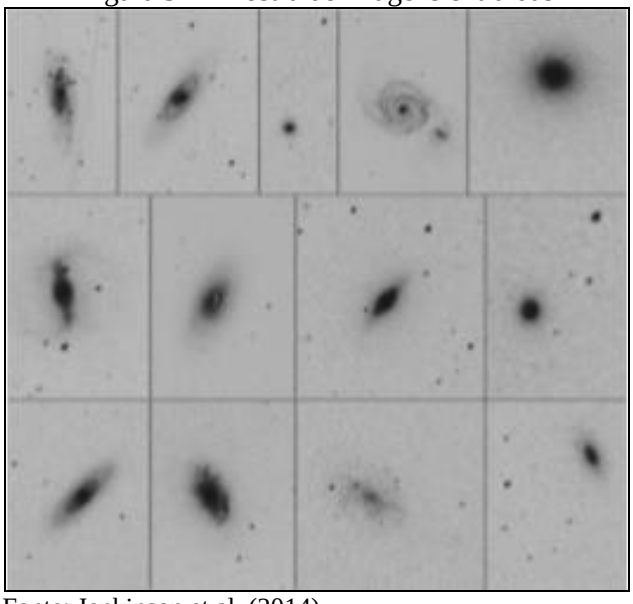
A seguir são relacionados três trabalhos correlatos. A seção 2.3.1 apresenta o artigo de Jenkinson et al. (2014) denominado "*Machine Learning and Image Processing in Astronomy with Sparse Data Sets*". Na seção 2.3.2 é apresentado o artigo "*Automated Classification techniques of Galaxies Using Artificial Neural Networks Based Classifiers*" de Ata et al. (2009). A seção 2.3.3 apresenta o artigo "*An Intelligent Approach for Galaxies Images Classification*" de Elfattah et al. (2013). E por fim, a seção 2.3.4 apresenta um comparativo entre os trabalhos correlatos.

2.3.1 *Machine Learning and Image Processing in Astronomy with Sparse Data Sets*

Jenkinson et al. (2014) focam no desenvolvimento de um sistema automático de classificação de galáxias, utilizando imagens baseadas no método de representação esparsa de dados. Em segundo plano, foi estudada a classificação baseada em imagens melhoradas com *alpha-rooting*, *heap transform* e *paired transforms*.

A partir da coleta os dados, o sistema implementado por Jenkinson et al. (2014) realiza o pré-processamento e a extração de imagens contendo possíveis galáxias. Nesta etapa foram aplicados algoritmos de subtração e filtragem não-linear, detecção de bordas e cálculo das características morfológicas. Os resultados dessas etapas podem ser visualizados na Figura 9, onde são exibidas as imagens extraídas.

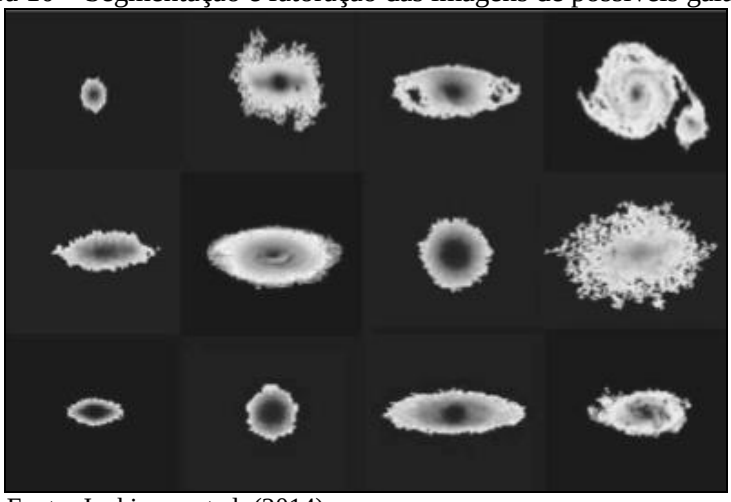
Figura 9– Amostra de imagens extraídas



Fonte: Jenkinson et al. (2014).

Após a identificação de possíveis galáxias é realizada a segmentação das imagens e posteriormente a fatoração de vetores, técnica que reduz a dimensionalidade de dados. Isso facilita futuras análises visuais através de técnica estatísticas para vetores (CAI, BAU e HE, 2011). A Figura 10 apresenta o resultado da etapa de segmentação e fatoração das imagens.

Figura 10 – Segmentação e fatoração das imagens de possíveis galáxias.



Fonte: Jenkinson et al. (2014).

O algoritmo de aprendizado de máquina utilizado para a classificação foi o *Support Vector Machine* (SVM), sobre um conjunto de dados de apenas 17 galáxias. Dente elas, 14 foram utilizadas para treinamento e 3 para classificação. A taxa de acerto foi de 100%, mas, segundo Jenkinson et al. (2014, p. 203), esse valor pode diminuir à medida que o número de galáxias produz variabilidade dos valores de características experimentados.

2.3.2 Automated Classification techniques of Galaxies Using Artificial Neural Networks Based Classifiers

Ata et al. (2009) propõem-se a analisar e classificar morfologicamente imagens de galáxias, realizando a comparação de resultados de 10 tipos diferentes de redes neurais: *perceptron* multicamadas, *feedforward* generalizado, modular, Jordan/Elman, Análise de componentes principais, base radial, auto organizável, recorrente atrasada, recorrente e SVM.

O sistema desenvolvido foi dividido nas fases de pré-processamento da imagem, extração de características, treinamento dos algoritmos e classificação. O conjunto de dados utilizado é formado de 10 imagens obtidas do catálogo Zsolt Frei, disponibilizado pelo departamento de ciências astrofísicas de Princeton (FREI, 1999).

No pré-processamento, as características extraídas se agrupam em morfológicas (M) e componentes principais (CP). Segundo Ata et al. (2009, p. 158), características morfológicas são aquelas baseadas em características visualmente perceptíveis. Elas foram divididas em 9

subgrupos: área, centroide, *bounding box*, eixo principal, eixo menor, excentricidade, orientação, área convexa e extremos. A partir destas características, tenta-se encontrar as componentes principais, que são utilizadas para reduzir o tamanho de um conjunto de dados mantendo o máximo possível de informação sobre ele (ATA et al., 2009). As imagens foram comprimidas usando vetores base de componentes principais de 4, 8, 16 e 24 elementos, preservando 70%, 75%, 80% e 85% da imagem original, respectivamente no conjunto de dados (ATA et al., 2009).

Todas as combinações de dados entre as características morfológicas e componentes principais foram realizadas. Os melhores resultados podem ser visualizados no Quadro 2.

Quadro 2 – melhores resultados obtidos

Tipo de rede neural	**Dados utilizados**	**Taxa de erro %**
Perceptron multicamadas	M + 24CPs	0,855
Feedforward generalizado	M + 24CPs	1,657
Modular	M + 24CPs	3,670
Jordan/Elman	M + 24CPs	0,922
Análise de componentes principais	M	5,942
Base radial	M	5,942
Auto Organizável	M + 16CPs	4,109
Recorrente Atrasada	M + 16CPs	2,457
Recorrente	M + 24CPs	3,354
Support Vector Machine	M + 24CPs	0,471

Fonte: adaptado de Ata et al. (2009).

A solução implementada usando SVM foi a que atingiu os melhores resultados, tendo 99,529% de acerto. Os piores resultados foram atingidos pela rede basial. Ata et al. (2009) indica que as características morfológicas são aparentemente mais eficientes que as características das componentes principais, dado que os melhores resultados foram o que mantiveram a maior parte dos dados brutos. Isso se deve a menor susceptibilidade a variações em galáxias da mesma categoria encontrada nas características morfológicas.

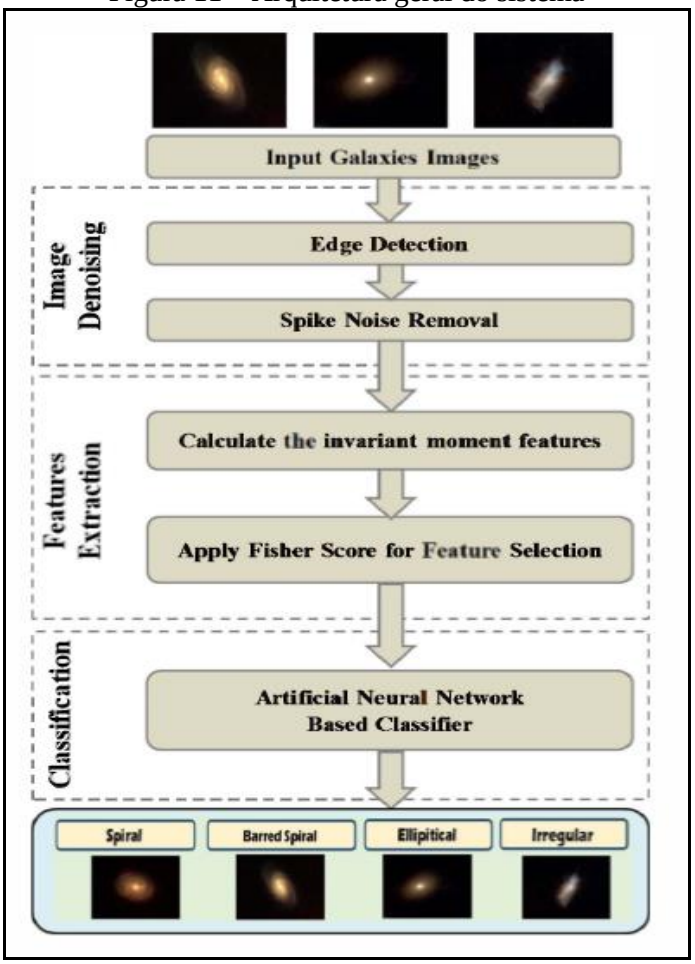
2.3.3 An Intelligent Approach for Galaxies Images Classification

Elfattah et al. (2013) propõem o desenvolvimento de uma solução automática, inteligente e eficiente para a realização da classificação morfológica de galáxias pela sequência Hubble.

Para realizar a classificação, os autores utilizam uma rede neural artificial e realizam a extração de características baseadas em momento. A solução proposta é dividida em três

fases: remoção de ruído, extração de características e classificação. As etapas que formam cada uma das fases são exibidas na Figura 11 (ELFATTAH et al., 2013).

Figura 11 – Arquitetura geral do sistema



Fonte: Elfattah et al. (2013).

A primeira etapa tenta garantir que a imagem não possua ruído, contendo apenas a galáxia desejada. Segundo Elfattah et al. (2013), o tipo de ruído que domina as imagens de galáxias é o ruído de pico. Para eliminá-lo, utiliza-se um algoritmo de detecção de borda.

Após a etapa de remoção de ruídos, as imagens são enviadas para a fase de extração de características. Nesta fase, o algoritmo de momentos invariantes, que já tem sua eficiência comprovada, é utilizado para realizar a extração de características invariantes a escala, rotação e translação (ELFATTAH et al., 2013). A aplicação do algoritmo de Fisher define quais características são mais relevantes para a classificação.

Na etapa de classificação os autores utilizaram uma rede neural recorrente atrasada e uma rede neural auto organizável, para posterior comparação de resultados. Uma rede neural recorrente atrasada, segundo Elfattah et al. (2013, p. 169), é um tipo de rede neural artificial (RNA), que é capaz de modelar qualquer processo não-linear arbitrariamente complexo. Já uma rede neural auto organizável é um tipo de rede neural artificial que não necessidade de

um conjunto de dados classificado para realizar seu treinamento, pois utiliza aprendizado não supervisionado para produzir um mapa, normalmente bidimensional, dos exemplos de entrada (ELFATTAH et al., 2013).

Para realização de testes da solução proposta foram utilizadas 108 imagens de galáxias do catálogo Zsolt Frei, contendo os tipos mais comuns de morfologias. Elas foram divididas em um conjunto para treinamento, contendo 24 imagens e um conjunto de testes, contendo 68 imagens. Os resultados podem ser verificados no Quadro 3.

Quadro 3 – Resultados obtidos

Tipo de Rede Neural	**Taxa de Erro %**
Recorrente Atrasada	2,704
Auto Organizável	1,502

Fonte: adaptado de Elfattah et al. (2013).

Na análise de resultados, percebe-se que após a fase de extração de características através do algoritmo de momento invariante, a rede neural auto organizável foi a que apresentou os melhores resultados na etapa de classificação (ELFATTAH et al., 2013).

2.3.4 Comparativo entre os trabalhos correlatos

O Quadro 4, montado com base nas informações obtidas a partir dos trabalhos descritos anteriormente, lista as principais características de cada um dos trabalhos correlatos.

Quadro 4 – Comparativo dos trabalhos correlatos

Características/ trabalhos relacionados	Jenkinson et al. (2014)	Ata et al. (2009)	Elfattah et al. (2013)
Origem da base de dados	Criada pelos autores	Zsolt Frei	Zsolt Frei
Quantidade de imagens de treino	14	10	24
Quantidade de imagens de teste	3	10 (retiradas da base de treino)	68
Algoritmos utilizados para processamento das imagens	*Rooting transform*, *Heap transform*, *Paired transform*.	-	Detecção de bordas e remoção de ruído de ponta
Algoritmos utilizados para classificação	SVM	Redes percepton multicamadas, *feedforward* generalizada, modular, Jordan/Elman, PCA, de base radial, auto organizáveis, recorrente atrasada, recorrente e SVM.	Mapas auto organizáveis e SVM
Tipo de dados de entrada	Representação esparsa das imagens	Características morfológicas e PCA	Momentos invariantes com Fisher Score
Melhor taxa de acerto de classificador supervisionado	100	99.53 (SVM)	97.29 (SVM)

A base de dados utilizada por Jenkinson et al. (2014) foi montada pelos autores a partir de imagens astronômicas do telescópio do Instituto Nacional de Astrofísica, Óptica y Electrônica (INAOE), de Tonantzintla, México. Foram criadas 17 imagens, 14 utilizadas para treino e 3 para teste do classificador. Ata et al. (2009) e Elfattah et al. (2013) utilizaram a base de dados criada por Frei (1999), formada por 113 imagens.

Todos trabalhos correlatos utilizaram bases de dados pequenas que possuíam imagens balanceadas e com ruído mínimo. Elfattah et al. (2013) e Jenkinson et al (2014), propõem como trabalho futuro a utilização de base de dados maiores e mais complexas, contendo imagens desbalanceadas e com ruído.

A etapa de processamento mais complexa foi a realizada por Jenkinson et al (2014), já que os demais utilizaram imagens já processadas da base Zsolt Frei. Apesar da utilização de diversos algoritmos para a classificação, o melhor resultado de classificador supervisionados foi apresentado pelo SVM em todos os trabalhos.

A representação dos dados utilizada como entrada na rede neural foi variada. Nenhum trabalho utilizou a imagem inteira, mas sim técnicas visando a redução da dimensionalidade dos dados originais.

3 DESENVOLVIMENTO

Neste capítulo serão apresentadas as etapas de desenvolvimento. A seção 3.1 apresenta os requisitos. A seção 3.2 apresenta a especificação. Na seção 3.3 é apresentada de forma detalhada a implementação da ferramenta. A seção 3.4 apresenta os resultados obtidos e sugestões de futuras melhorias.

3.1 REQUISITOS

Os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) da ferramenta Astrolab são:

a) permitir que o usuário realize o processamento de imagens astronômicas (RF);
b) permitir que o usuário realize o treinamento da rede neural que será utilizada para classificação (RF);
c) permitir que o usuário informe uma imagem de galáxia e realize a classificação morfológica da mesma (RF);
d) permitir que o usuário visualize a taxa de sucesso durante o treinamento da rede neural (RF);
e) utilizar a linguagem Python para implementação da ferramenta (RNF);
f) utilizar a biblioteca de visão computacional OpenCV para suporte as operações de processamento de imagem (RNF);
g) utilizar a biblioteca de aprendizado de máquina TensorFlow para criação da rede neural (RNF);
h) ser implementado na língua inglesa (RNF).

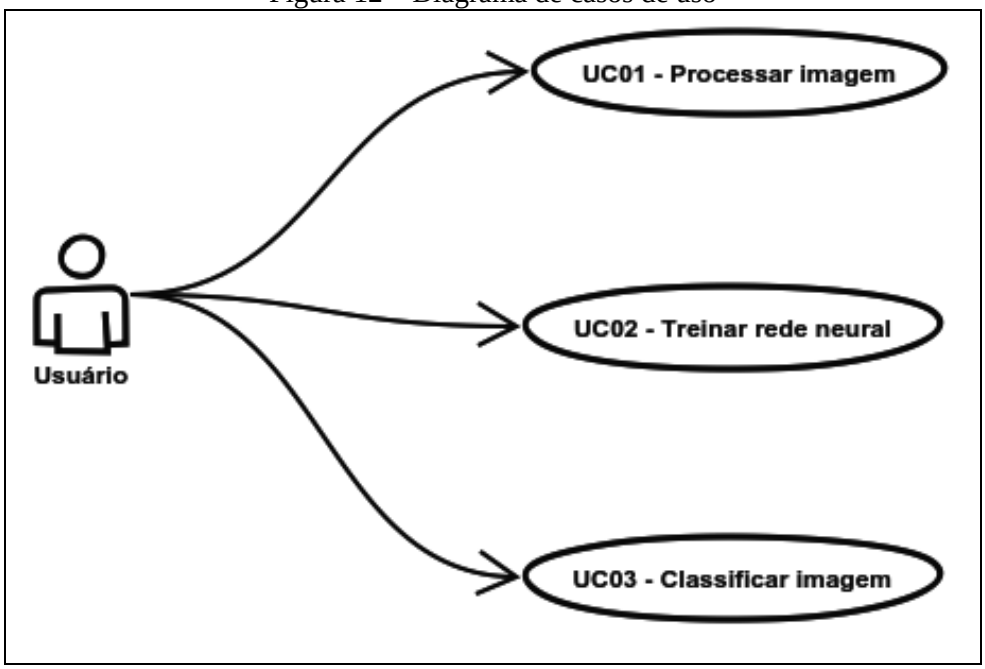
3.2 ESPECIFICAÇÃO

A especificação da ferramenta foi realizada utilizando a *Unified Modeling Language* (UML), o paradigma de programação orientada a objetos e a ferramenta sketchboard.me para elaborar os diagramas de casos de uso, de classes e de atividades.

3.2.1 Diagrama de casos de uso

Nesta seção são apresentados os casos de uso da ferramenta. Foi identificado apenas um ator, o `usuário`, o qual realiza uma das três ações disponibilizadas pela ferramenta. A Figura 12 exibe o diagrama de casos de uso com as ações fundamentais disponibilizadas pela ferramenta.

Figura 12 – Diagrama de casos de uso



O caso de uso `UC01 – Processar imagem` realiza o processamento de uma imagem, removendo possíveis sujeiras, a redimensionando e criando uma imagem a partir do resultado. O primeiro passo do usuário é informar o caminho da imagem que será processada, ou o caminho da pasta com as imagens que devem ser processadas. O segundo passo é informar à ferramenta o caminho que a imagem ou imagens devem ser salvas. Por fim, o usuário informa o tamanho que deve ser utilizado no redimensionamento.

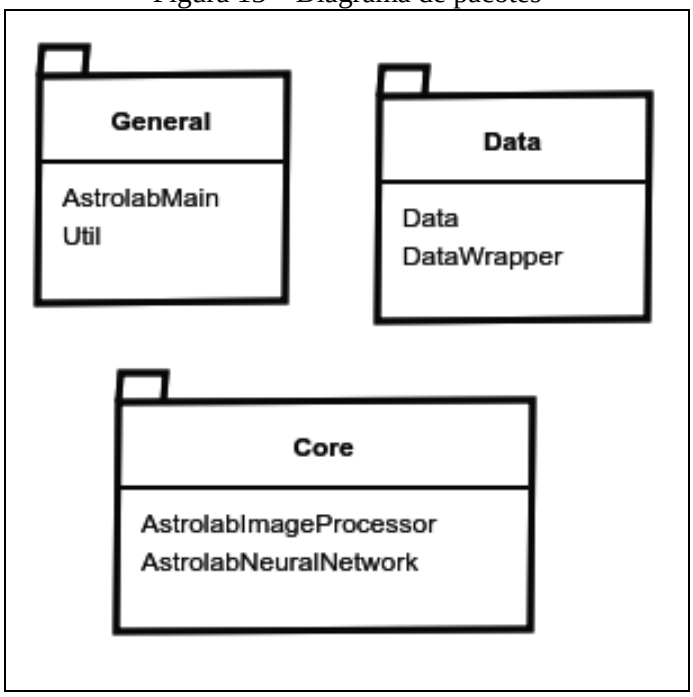
O caso de uso `UC02 – Treinar rede neural` executa o treinamento da rede neural para futuras classificações. Primeiramente deve-se informar o caminho que contém as imagens de treinamento. Estas imagens devem estar nomeadas numericamente. O usuário informa em seguida o caminho do arquivo de rótulos, contendo as classes de cada uma das imagens de treinamento. A ordem dos registros no arquivo de rótulos deve condizer com a da numeração das imagens. O caminho das imagens de treinamento deve ser então informado para que a rede neural carregue os dados. Informações adicionais são solicitadas ao usuário antes de iniciar o treinamento: tamanho das imagens, número de classes de saída, iterações de treinamento e tamanho do *batch* de dados que deve ser utilizado em cada iteração.

O caso de uso `UC03 – Classificar imagem` realiza a classificação de uma imagem. O usuário informa o caminho da mesma e a rede neural realiza a sua classificação. A imagem que foi informada é exibida ao usuário, o que possibilita validação visual da classificação. O detalhamento dos casos de uso da aplicação está no apêndice A.

3.2.2 Diagrama de pacotes

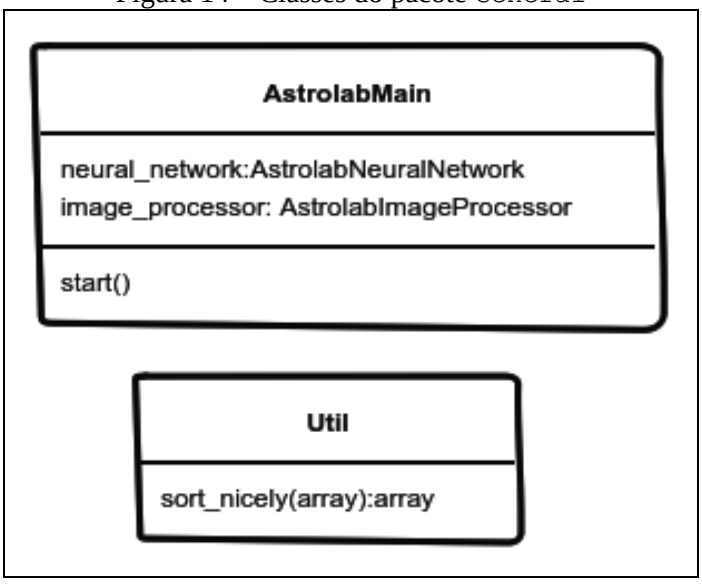
Nesta seção são descritas as classes que compõem a ferramenta Astrolab. A Figura 13 exibe o conjunto de pacotes que compõem a aplicação: `General`, `Data` e `Core`.

Figura 13 – Diagrama de pacotes



O pacote `General` engloba a classe utilitária `Util` e a classe `AstrolabMain`, responsável por controlar a interação do usuário com a ferramenta. Ambas foram agregadas em um único pacote visando comodidade. Os principais métodos e atributos de cada uma das classes podem ser visualizados na Figura 14.

Figura 14 – Classes do pacote `General`

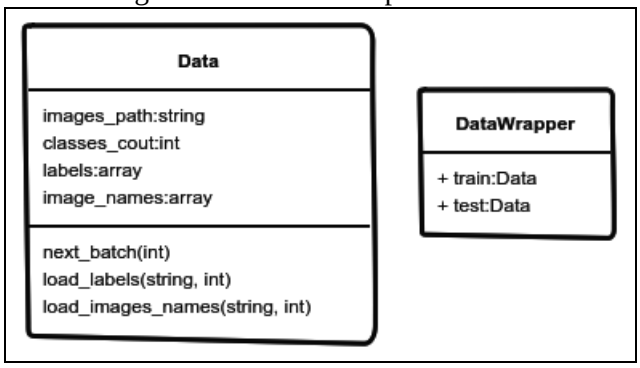


A classe `Util` possui apenas um método que realiza a ordenação numeral ascendente de uma lista que pode ser formada por números ou texto contendo números. O método é

utilizado durante a leitura das imagens no treinamento. A classe `AstrolabMain` cria uma interface via console com o usuário a partir do método `start`.

O pacote `Data` engloba as classes responsáveis pela representação dos dados utilizados pela ferramenta. Duas classes fazem parte do pacote: `Data` e `DataWrapper`. Seus principais métodos e atributos podem ser visualizados na Figura 15.

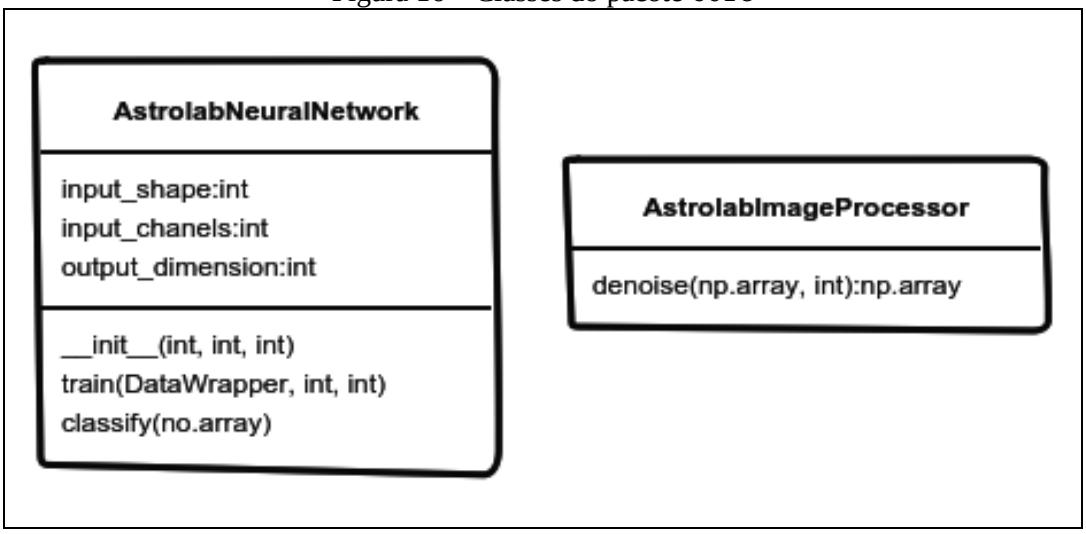
Figura 15 – Classes do pacote `Data`



A classe `Data` é responsável por retornar os *batchs* de imagens e rótulos durante a fase de treinamento e validação da rede neural. As imagens são carregadas em memória apenas durante a criação do *batch*, a partir do nome e caminho que são mantidos pela classe. A classe `DataWrapper` tem como objetivo apenas organizar os dados de maneira clara, evitando que as informações necessárias para o treinamento da rede neural sejam passadas uma por uma.

O pacote `Core` engloba as classes que implementam a funcionalidade base da ferramenta Astrolab: `AstrolabNeuralNetwork` e `AstrolabImageProcessor`. A Figura 16 apresenta as duas classes com seus principais métodos e atributos.

Figura 16 – Classes do pacote `Core`



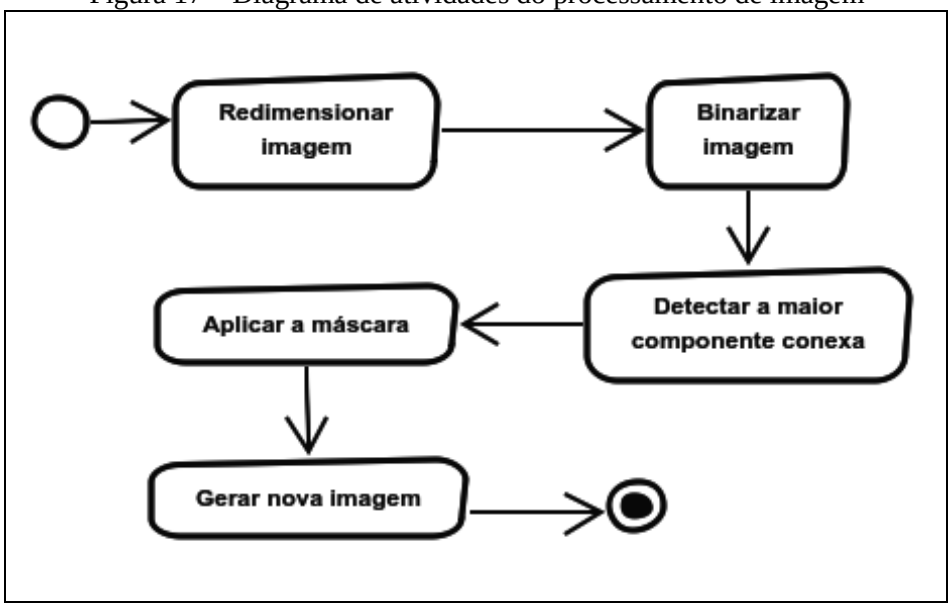
A classe `AstrolabImageProcessor` é a responsável por realizar o processamento da imagem. Ela possui apenas um método, que recebe uma imagem e um valor inteiro indicando

o tamanho que a imagem retornada deve possuir (altura e largura são sempre iguais). A classe `AstrolabNeuralNetwork` é a responsável por realizar o treinamento dos pesos que serão utilizados pela rede neural realizar a classificação. Seus principais métodos são `train`, que realiza o treinamento com base em um `DataWrapper`, e `classify` que a partir de uma imagem e pesos ajustados realiza a tarefa de classificação.

3.2.3 Diagramas de atividade

Para facilitar o entendimento das ações realizadas pela ferramenta, foram criados três diagramas de atividades: processamento de imagem, treinamento da rede neural e classificação. A Figura 17 exibe as atividades realizadas durante o processamento das imagens.

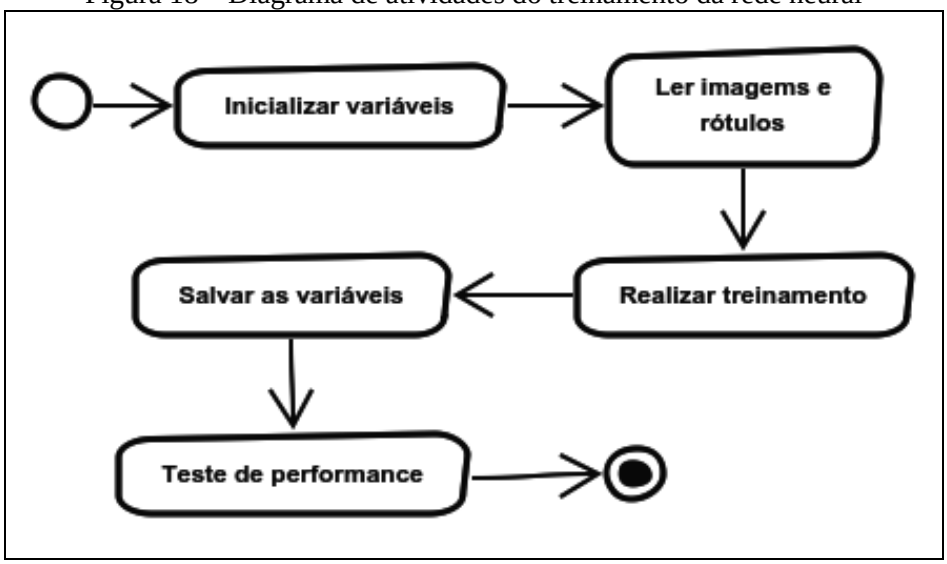
Figura 17 – Diagrama de atividades do processamento de imagem



O processamento de uma imagem pode ser dividido em seis etapas. Inicialmente uma imagem é dada como entrada e redimensionada para um tamanho informado pelo usuário. Em seguida, a imagem redimensionada é binarizada. A partir da imagem binarizada a maior componente conexa é encontrada. Por fim, a maior componente conexa é aplicada como máscara sobre a imagem original e uma nova imagem é gerada.

A Figura 18 exibe o diagrama de atividades do treinamento da rede neural.

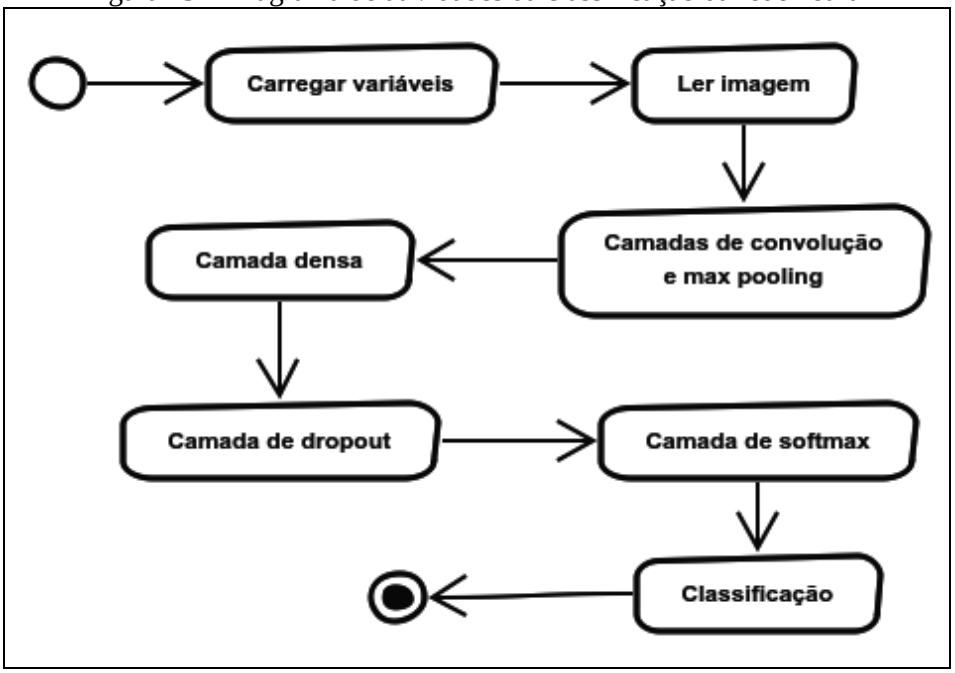
Figura 18 – Diagrama de atividades do treinamento da rede neural



As variáveis (pesos e *bias*) são inicializadas e as imagens/rótulos são lidos a partir dos caminhos passados. O treinamento, que visa diminuir a função de custo utilizando o algoritmo de *backpropagation* é aplicado, ajustando as variáveis. Ao terminar o treinamento, as novas variáveis são salvas e um teste de performance é realizado.

A Figura 19 exibe o diagrama de atividades do processo de classificação da rede neural.

Figura 19 – Diagrama de atividades da classificação da rede neural



O primeiro passo é realizar o carregamento das variáveis previamente ajustadas. A imagem é carregada em memória e passa pelas camadas de convolução e *max pooling* para que a extração de características seja realizada. A camada densa tenta encontrar juntamente

com a camada de *dropout* padrões nos valores extraídos. Os valores encontrados são enviados a camada de *softmax* que transforma o resultado em uma probabilidade. Por fim, a maior probabilidade é considerada como classificação e exibida ao usuário.

3.3 IMPLEMENTAÇÃO

Nesta seção são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação. A seção 3.3.1 apresenta de forma detalhada as técnicas e ferramentas utilizadas. A seção 3.3.2 descreve a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A ferramenta foi criada utilizando a linguagem de programação Python 3.5.1 e as bibliotecas OpenCV 3.1 para as tarefas de processamento de imagens e a TensorFlow 0.9 para a implementação da rede neural convolucional. Ambas são de código aberto e tem como dependência em comum a biblioteca Numpy, utilizada para manipulação de imagens (leitura e gravação) e realização de cálculos matemáticos. Para implementação das chamadas REST durante a criação da base de dados, a biblioteca Requests, uma simplificação da biblioteca nativa Python Urllib, foi utilizada.

As seções a seguir descrevem com amostras de código fonte como foi realizada a implementação das seguintes etapas de desenvolvimento: processamento de imagens, geração da base de dados para treinamento da rede neural e criação da rede neural convolucional que realiza a classificação das imagens.

3.3.1.1 Processamento das imagens

A primeira etapa no processamento de uma imagem é realizar o seu redimensionamento. O código responsável por esta tarefa recebe uma imagem e um valor inteiro indicando o tamanho para o qual ela será redimensionada. O código que realiza esta etapa pode ser visto no Quadro 5.

Quadro 5 – Redimensionamento da imagem recebida.

```
01 image_center = resize_to_size / 2
   image = cv2.resize(image,(resize_to_size, resize_to_size),
02 interpolation = cv2.INTER_CUBIC)
```

A próxima etapa utiliza o algoritmo de limiarização para realizar a binarização da imagem. A técnica utiliza um valor limite e transforma todos os pixels com valor inferior ao limite em zero. Os restantes são definidos como 255. A imagem binarizada é então utilizada para encontrar todos os contornos existentes (Quadro 6).

Quadro 6 – Escala de cinza, binarização e busca de contornos.

```
01 binary = np.empty(image.size)
02 finalImage = np.empty(image.size)
03
04 _, binary = cv2.threshold(image, 20, 255, cv2.THRESH_BINARY)
05 _, contours, hierarchy = cv2.findContours(binary, cv2.RETR_LIST,
06 cv2.CHAIN_APPROX_NONE)
```

Em posse dos contornos é encontrada a maior componente conexa presente. Ela deve englobar o ponto central da imagem. Caso exista uma componente conexa que possua o ponto central da imagem, uma máscara binarizada desse contorno é aplicada sobre a imagem original, criando uma nova imagem que possui apenas a área da maior componente conexa da imagem original. Caso nenhuma componente conexa cumpra com os requisitos definidos, `None` é retornado, isso evita que outro astro que possa estar presente seja o mantido (Quadro 7).

Quadro 7 – Busca da maior componente conexa e aplicação da máscara.

```
01 biggerContourIndex = None
02       maxContourSize = 0
03       count = 0
04
05       for contour in contours:
06             contourSize = cv2.contourArea(contour)
07             if contourSize > maxContourSize and
                          cv2.pointPolygonTest(contour, (image_center,
                           image_center), False) == 1:
08                   maxContourSize = contourSize
09                   biggerContourIndex = count
10             count += 1
11
12       if biggerContourIndex:
13             mask = np.zeros(binary.shape, np.uint8)
14             mask = cv2.drawContours(mask, contours, biggerContourIndex,
                                       (255,255,255), cv2.FILLED)
15
16             maskedImage = cv2.bitwise_and(image, gray, mask=mask)
17
18             return maskedImage
19       else:
20             return None
```

Uma iteração é realizada na lista de componentes conexas encontradas (linha 5). Caso alguma componente cumpra com os requisitos definidos (linha 7), ela é utilizada para criar uma máscara (linha 14). A máscara é uma imagem binarizada que possui os pixels da área da maior componente como 255, e os demais pixels como 0. Por fim, uma nova imagem é criada a realizando um `and` binário sobre os valores da máscara e da imagem original já redimensionada (linha 16). A Figura 20 exibe a imagem binarizada, a imagem original com os contornos que foram encontrados e a máscara criada a partir da maior componente conexa. A Figura 21 exibe lado a lado uma imagem antes e após o processamento redirecionamento e remoção de ruídos.

Figura 20 – Etapas de processamento de uma imagem

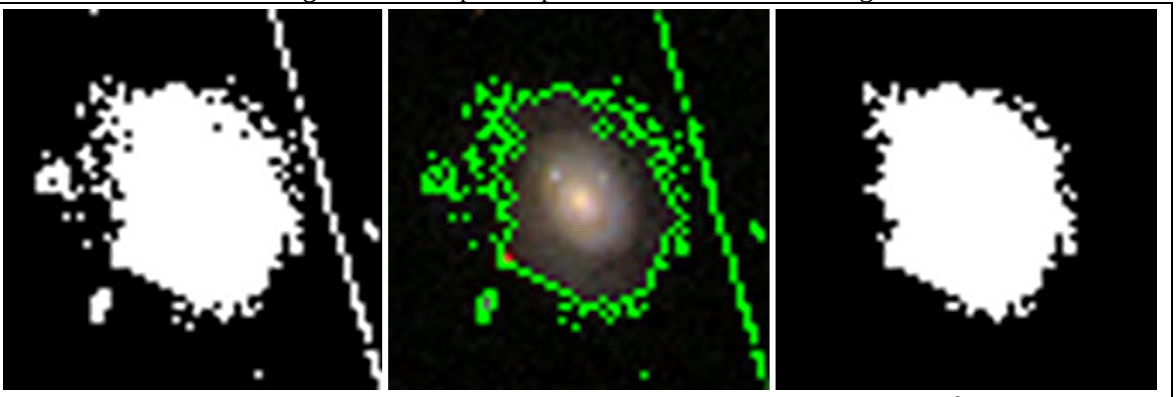


Imagem binarizada | Contornos encontrados | Máscara

Figura 21 – Resultado do processamento

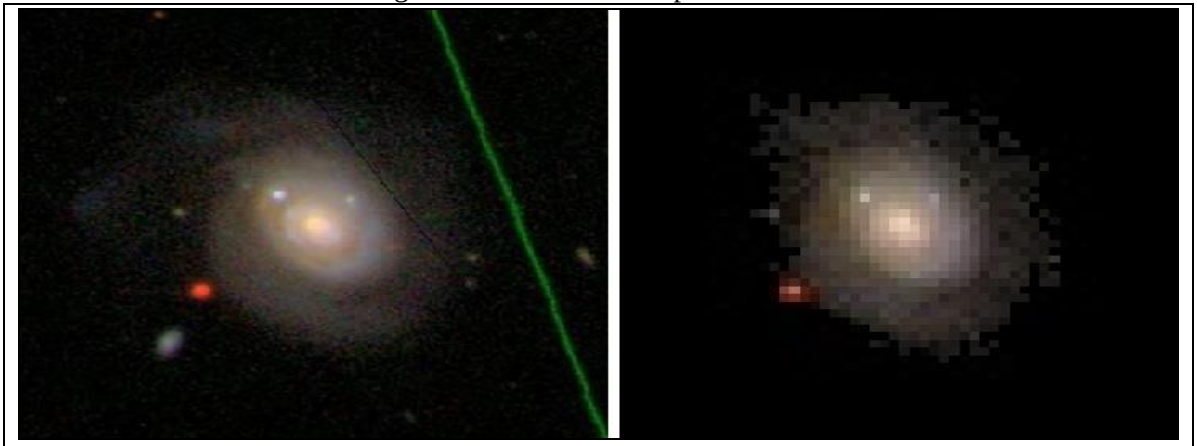


Imagem original | Imagem redimensionada

A galáxia da Figura 21 foi reduzida de 512x512 para 56x56 e passou pela remoção de ruídos com sucesso. Na imagem processada foi mantida apenas a galáxia.

3.3.1.2 Geração da base de dados

A geração da base de dados pode ser dividida em três passos: *download* das imagens, processamento das imagens e geração do arquivo de rótulos com base nas imagens processadas.

As imagens foram adquiridas da base de dados do *Sloan Digital Sky Survey* (SDSS). O SDSS é um levantamento astronômico que está mapeando um quarto do céu. Quando finalizado, terá uma base de mais de 40TB de dados. A interface utilizada para acessar esses dados foi o serviço ImageCutout, que disponibiliza imagens do céu com base em posições escolhidas pelo usuário (declinação e ascensão reta) (NIETO-SANTISTEBAN; SZALAY; GRAY, 2003).

As informações de ascensão reta e declinação passadas para o serviço ImageCutout, assim como as classes utilizadas para a criação do arquivo de rótulos, foram retiradas da tabela 5 disponibilizada pelo projeto Galaxy Zoo 2. Galaxy Zoo 2 é um projeto científico que

realizou a classificação morfológica de 304.122 galáxias retiradas da base de dados do SDSS. As classificações, realizadas por cidadãos cientistas voluntários, concorda em mais de 90% com as realizadas por astrônomos profissionais (WILLETT et al., 2013).

O serviço ImageCutout recebe como parâmetros a declinação, ascensão reta, tamanho da imagem e escala. O *download* é realizado com base nas informações contidas na tabela 5 da base de dados do projeto Galaxy Zoo. As imagens salvas recebem como nome o número da linha da tabela 5 de onde as informações passadas ao serviço são retiradas. A escala é corrigida manualmente quando o *zoom* é inadequado. O Quadro 8 mostra o trecho de código que realiza a chamada passando como parâmetro os valores lidos.

Quadro 8 – Código realizando a chamada REST para download de uma imagem

```
formatted_url = url.format(ra=content[1], dec=content[2], scale=scale)
      while True:
            try:
                  response = requests.get(
                        formatted_url,
                        stream=True,
                        timeout=1
                  )
                  if response.status_code == 200:
                        filename = str(spamreader.line_num) + '.jpg'
                        with open(filename, 'wb') as f:
                              response.raw.decode_content = True
                              shutil.copyfileobj(response.raw, f)
                              break
                  else:
                        print("Tentando novamente, retornou erro")
                        time.sleep(1)
            except:
                  print("Exceção ocorreu, tentando novamente")
                  time.sleep(1)
```

Após as imagens serem adquiridas, é realizado o processamento das mesmas. A partir das imagens processadas é gerado um arquivo de rótulos. O arquivo de rótulos utiliza novamente a tabela 5 do projeto Galaxy Zoo. Todas galáxias espirais e espirais barradas são classificadas como tipo tardio (1) e todas elípticas como tipo jovem (0). O código que escreve no arquivo de rótulos pode ser visto no Quadro 9.

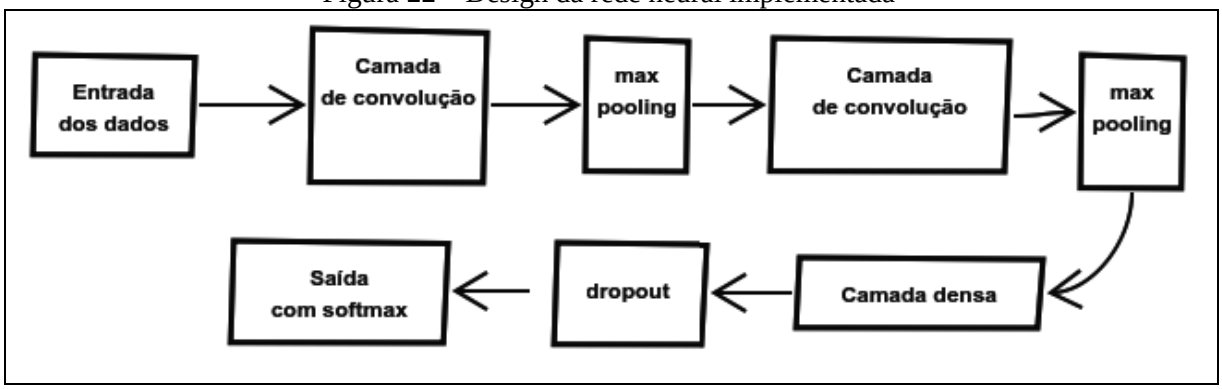
Quadro 9 – Criação do arquivo de rótulos.

```
01 if content:
02       if str(spamreader.line_num) + '.jpg' in images:
03             print(content[0])
04             if content[0].startswith('E'):
05                   spamwriter.writerow([spamreader.line_nu, 0])
06             elif content[0].startswith('S'):
08                   spamwriter.writerow([spamreader.line_num, 1])
09             else:
10                   spamwriter.writerow([spamreader.line_num, 2])
```

3.3.1.3 Criação da Rede Neural Convolucional

A rede neural implementada possui duas camadas de convolução intercaladas com duas camadas de *max pooling*, uma camada densa, e uma camada de saída que utiliza *softmax*. *Dropout* foi utilizado para evitar sobre ajuste. Na Figura 22 é exibida graficamente a estrutura da mesma.

Figura 22 – Design da rede neural implementada



A biblioteca de código aberto da Google, TensorFlow, foi utilizada na criação da rede neural. Seu funcionamento é baseado em grafos. O programador deve definir o grafo de execução do algoritmo que está implementando. Cada nó do grafo é um *placeholder*, que quando executado na sessão, percorre todo caminho do grafo necessário para sua criação (ABADI et al., 2015).

O primeiro passo para a configuração da rede neural é informar o formato das imagens de entrada (altura ou largura, que são sempre iguais após o processamento), número de canais utilizados (três para RGB) e a quantidade de classes da saída. Na linha 3 é calculado o tamanho final da imagem após as camadas de convolução, e na linha 4 a dimensão das entradas (Quadro 10).

Quadro 10 – Inicialização dos valores básicos que serão utilizadas pela rede neural

```
01 self.input_shape = input_shape
02 self.input_channels = input_channels
03 self.input_shape_conved = int(self.input_shape / 4)
04 self.input_dimension = self.input_shape * self.input_shape
05 self.output_dimension = output_dimension
```

Em posse dos valores informados, é realizada a definição do grafo de execução da rede neural. O grafo define *placeholders* e variáveis para uso futuro no treinamento e classificação. Os *placeholders* de entrada e rótulos são definidos em apenas duas linhas de código (Quadro 11), onde informa-se o tipo de valor contigo nos mesmos e a dimensão de cada um. A quantidade de itens é definida como `None`, o que permite ao usuário definir o tamanho do *batch* de treinamento posteriormente.

Quadro 11 – Inicialização dos *placeholders* de entrada e rótulos

```
01 self.x = tf.placeholder(tf.float32, shape=[None, self.input_dimension])
02 self.correct_y = tf.placeholder(tf.float32, shape=[None,
   self.output_dimension])
```

O Quadro 12 exibe a definição da fase de extração de características da rede neural, formada por quatro camadas. O processo pode ser resumido em quatro etapas que se repetem para cada conjunto de duas camadas (uma de convolução seguida por uma de *max pooling*): inicialização das variáveis (peso e *bias)*, convolução, aplicação da função de ativação e realização do *max pooling*. O valor resultante desse processo é enviado então para a próxima camada.

Quadro 12 – Definição da fase de extração de características da rede neural

```
01 conv1_W = self.create_random_weights([5, 5, 1, 32])
02
03 conv1_b = self.create_random_biases([32])
04 x_reshaped = tf.reshape(self.x, [-1, self.input_shape,
05                                              self.input_shape, 1])
06 conv1_o = tf.nn.relu(self.do_conv2d(x_reshaped, conv1_W) + conv1_b)
07 pool1_o = self.do_max_pool_2x2(conv1_o)
08
09 conv2_W = self.create_random_weights([5, 5, 32, 64])
10 conv2_b = self.create_random_biases([64])
11 conv2_o = tf.nn.relu(self.do_conv2d(pool1_o, conv2_W) + conv2_b)
12 pool2_o = self.do_max_pool_2x2(conv2_o)
```

Para realizar a inicialização dos pesos, o método `create_random_weights` recebe uma lista de valores que informa o formato da variável a ser criada. Os dois primeiros valores correspondem ao tamanho de cada filtro de convolução, o terceiro informa o número de canais da entrada (3 para imagens coloridas, por exemplo) e o último é o número de canais da saída (igual ao número de filtros de convolução aplicados). O método garante também que os pesos possuam ruído e não sejam inicializados como zero. O método `create_random_biases` inicializa uma variável com valores constantes 0.1 e dimensão passada por parâmetro.

Após a inicialização dos pesos, é realizado o redimensionamento de `x` para um tensor de 4 dimensões (linha 3), formato necessário para a aplicação da convolução. O valor -1 informa que a dimensionalidade dos dados deve ser mantida e o valor 1 informa o número de canais. Os demais valores indicam a altura e largura da imagem.

No método `do_conv_2d`, que recebe a imagem e pesos como parâmetro, é realizada a convolução dos valores (nenhum *padding* é aplicado na imagem e é utilizado *stride* de um). Ao resultado da convolução é somado o *bias* e aplicada a função de ativação ReLU. O valor retornado é então enviado a camada de *max pooling*. O método `max_pool_2x2` recebe como parâmetro o resultado da convolução e utiliza blocos de 2x2 o que reduz a dimensão da imagem pela metade (a quantidade de canais fica inalterada). A implementação dos métodos `do_conv_2d` e `do_max_pool_2x2` pode ser visualizada no Quadro 13.

Quadro 13 – Implementação da convolução e *max pooling*

```
01 def do_conv2d(self, x, W):
02       return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
03
04 def do_max_pool_2x2(self, x):
05       return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2,
                                                          1], padding='SAME')
```

O resultado das etapas de extração de características é redimensionado e utilizado como entrada na camada densa. A imagem neste ponto está com um quarto do tamanho original e 64 canais. A camada densa possui 1024 neurônios e é responsável por realizar o processo de classificação. Este consiste em inicializar os pesos e *bias*, redimensionar os dados para um *batch* de vetores, multiplicar a entrada e os pesos, somar o *bias* e aplicar a função de ativação no resultado (Quadro 14).

Quadro 14 – Definição da camada densa

```
01 d1_W = self.create_random_weights([self.input_shape_conved *
   self.input_shape_conved * 64, 1024])
02 d1_b = self.create_random_biases([1024])
03
04 pool2_o_reshaped = tf.reshape(pool2_o, [-1, self.input_shape_conved *
                                  self.input_shape_conved * 64])
05 d1_o = tf.nn.relu(tf.matmul(pool2_o_reshaped, d1_W) + d1_b)
```

Para evitar sobre ajuste dos pesos, aplica-se *dropout* antes da camada de saída. Um *placeholder* é criado para controlar a probabilidade que um neurônio tem de ser mantido durante o *droupout*. Isso permite que ele seja aplicado durante a fase de treinamento e removido na fase de testes. A camada de saída aplica o algoritmo *softmax* para transformar os valores em probabilidades, são recebidas 1024 entradas e o número de saídas é igual ao definido na variável `output_dimension`. O *placeholder* `predicted_y` representa a previsão

realizada pela rede neural. A definição da etapa de *dropout* e *softmax* pode ser visualizada no Quadro 15.

Quadro 15 – Etapa de *dropout* e *softmax*

```
01 self.resistence = tf.placeholder(tf.float32)
02 d1_o_dropout = tf.nn.dropout(d1_o, self.resistence)
03
04 d2_W = self.create_random_weights([1024, self.output_dimension])
05 d2_b = self.create_random_biases([self.output_dimension])
06 self.predicted_y = tf.nn.softmax(tf.matmul(d1_o_dropout, d2_W) + d2_b)
```

O grafo de execução da classificação já está finalizado neste ponto, mas ainda é necessário criar os *placeholders* que são necessários para o treinamento da rede neural: a função de custo, a função que minimiza a função de erro, a previsão correta e a acurácia da previsão (Quadro 16).

Quadro 16 – Definição dos placeholders utilizados para treinamento

```
01 cross_entropy = tf.reduce_mean(-tf.reduce_sum(self.correct_y *
   tf.log(self.predicted_y), reduction_indices=[1]))
02 self.train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
03 correct_prediction = tf.equal(tf.argmax(self.predicted_y, 1),
   tf.argmax(self.correct_y, 1))
04 self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

A função de custo utilizada é a de entropia cruzada, função de custo padrão para algoritmos de classificação. Ela calcula a diferença da classificação desejada e da classificação realizada pela rede neural. A função de minimização da função de custo utilizada é a de gradiente descendente estocástico com otimizador Adam. Ela atualiza os pesos a cada *batch* de treinamento, diferente da função de gradiente descendente tradicional. Cada atualização é realizada com base no gradiente da função de custo (encontrado através da sua derivada) (HEATON, 2015).

O *placeholder* de acurácia é criado para que o usuário receba retorno da taxa de sucesso durante e após o treinamento. Para criá-lo é feita uma comparação entre os tensores de previsão da rede neural e de rótulos, onde apenas o maior valor é considerado e arredondado para 1. Com o grafo finalizado e todos os *placeholders* e variáveis definidas, é possível executar o treinamento e classificação através da rede neural.

3.3.1.3.1 Treinamento

O código que realiza o treinamento pode ser visualizado no Quadro 17. O primeiro passo é inicializar as variáveis, criadas durante a definição do grafo de execução da rede neural. Um objeto `saver` é criado para persistir os pesos e *bias* atualizados ao fim do treinamento. O número de iterações, o tamanho do *batch* de treinamento e um `DataWrapper`

são passados como parâmetro. O objeto de dados já deve possuir as imagens e rótulos de treino e teste carregados.

Quadro 17 – Treinamento da rede neural

```
01 self.session.run(tf.initialize_all_variables())
02 saver = tf.train.Saver()
03
04 for i in range(iterations):
05
06       batch = data_wrapper.train.next_batch(batch_size)
07       if i % 100 == 0:
08             train_accuracy = self.accuracy.eval(feed_dict = {
                     self.x: batch[0],
                     self.correct_y:batch[1],
                     self.resistence:0.5
               })
09             print("Step %d, training accuracy %g"%(i, train_accuracy))
10
11       self.train_step.run(feed_dict={
               self.x: batch[0],
               self.correct_y:batch[1],
               self.resistence:0.5
         })
12
13 batch = data_wrapper.train.next_batch(1000)
14 print("Test accuracy %g"%self.accuracy.eval(feed_dict={
         self.x: batch[0],
         self.correct_y: batch[1],
         self.resistence:1.0
   }))
15
16 saver.save(self.session, "./trained_variables.ckpt")
```

Para cada iteração um passo de treinamento é realizado em um *batch* de imagens. Os pesos são ajustados reduzindo a função de custo. A resistência dos nós é definida como 50% durante o treinamento. A cada 100 iterações é realizado um teste de performance com o *batch* corrente e exibido ao usuário. O teste consiste em realizar a classificação das imagens passadas e calcular a taxa de acerto atingida.

A cada 1000 iterações as imagens de teste e seus respectivos rótulos são utilizados para realizar o teste real de performance da rede. Todos os nós são mantidos (resistência dos nós é de 100%) e os novos pesos ajustados são salvos para uso durante a classificação.

3.3.1.3.2 Classificação

A classificação da rede neural utiliza as variáveis inicializadas e ajustadas durante a fase de treinamento. Apenas uma imagem é passada como parâmetro. O objeto `saver` é utilizado novamente, desta vez para recuperar as variáveis salvas previamente e adicioná-las a sessão. O *placeholder* `predicted_y`, ao ser executado, realiza todas etapas necessárias previamente definidas no grafo de execução da rede neural. A maior probabilidade retornada

da classificação é exibida ao usuário juntamente com a imagem passada, permitindo validação visual do resultado da classificação. O Quadro 18 exibe o código da implementação.

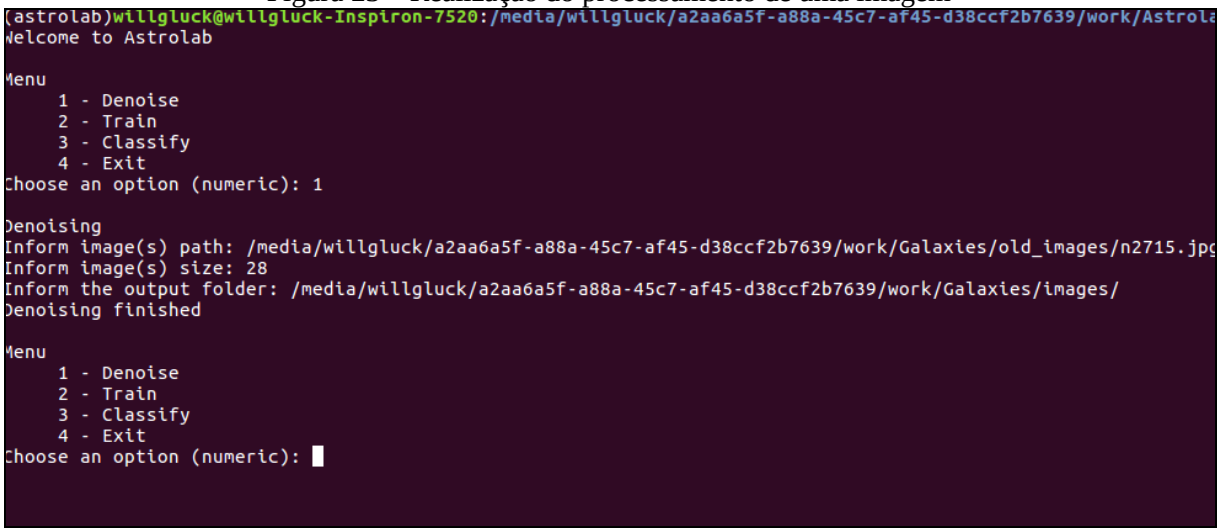
Quadro 18 – Classificação de uma imagem pela rede neural

```
01 saver = tf.train.Saver()
02 saver.restore(self.session, "./trained_variables.ckpt")
03
04 classification =  self.session.run(tf.argmax(self.predicted_y, 1),
   feed_dict={self.x : [image], self.resistence:1.0})
05
06 print("Prediction: " + str(classification[0]))
07 plt.imshow(image.reshape(28, 28), cmap=plt.cm.binary)
08 plt.show()
```

3.3.2 Operacionalidade da implementação

Nesta seção será exibida a operacionalidade da ferramenta. A iteração com o usuário é feita através de uma aplicação *console*. Quatro opções são disponibilizadas: processar uma imagem (1), treinar uma rede neural (2), realizar a classificação de uma imagem (3) e sair da aplicação (4). A Figura 23 exibe a realização do processamento de uma imagem.

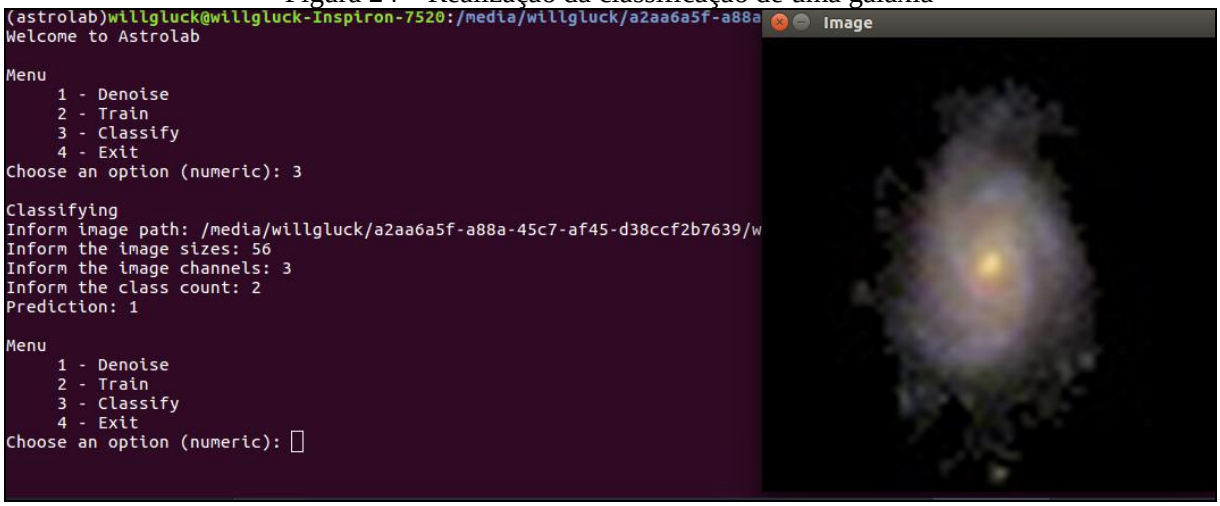
Figura 23 – Realização do processamento de uma imagem



Primeiramente é solicitado ao usuário o caminho da imagem que será processada. Caso o caminho represente uma pasta, a aplicação irá realizar o processamento de todas as imagens contidas nela. Por fim, o usuário deve informar o tamanho e o caminho das novas imagens que serão criadas.

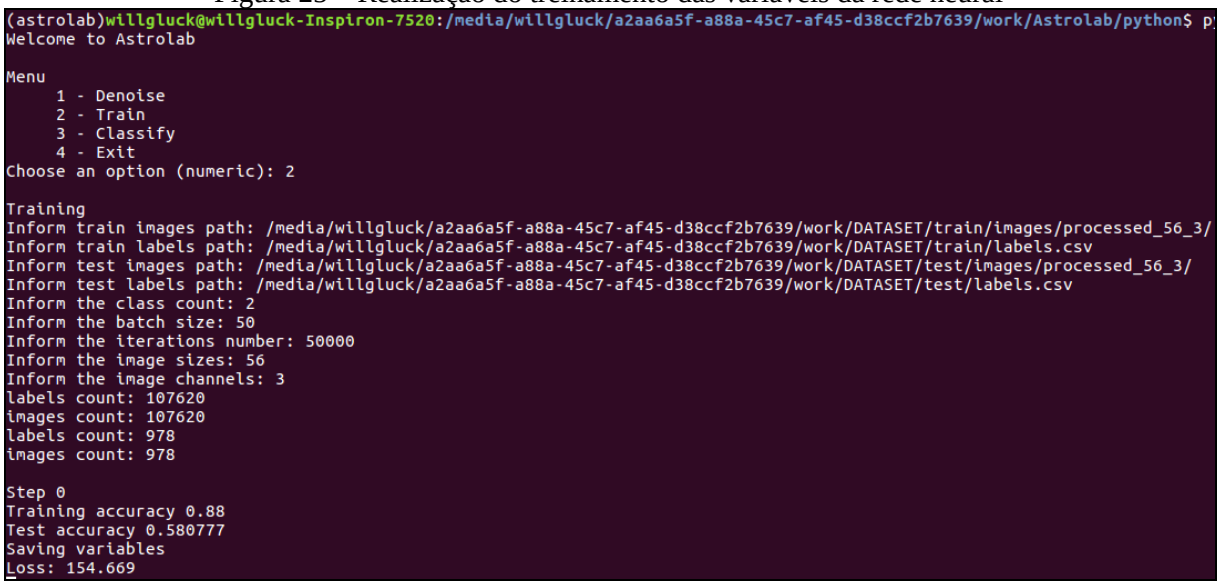
A Figura 24 exibe a realização de uma classificação. Após informada a opção 3, é solicitado ao usuário o caminho da imagem. Caso nenhuma classificação ou treinamento tenha sido realizado até então, a aplicação irá solicitar que o usuário informe o formato dos dados: tamanho das imagens, número de canais dos dados e quantidade de classes de classificação.

Figura 24 – Realização da classificação de uma galáxia



Caso o usuário nunca tenha treinado a rede a classificação não será realizada. Para realizar o treinamento a opção 2 deve ser informada. O primeiro passo é informar onde se encontram as imagens e arquivos de rótulos para teste e treino. O usuário deverá então passar o número de classes de classificação, tamanho do *batch* de treinamento, número de iterações do treinamento, tamanho das imagens utilizadas e por fim, número de canais das imagens. A Figura 25 exibe o início de um treinamento sendo realizado.

Figura 25 – Realização do treinamento das variáveis da rede neural



O usuário recebe durante o treinamento algumas informações via controle. A cada 100 iterações são exibidos o resultado da função de custo e a taxa de acerto nas imagens do *batch* de treinamento. A cada 1000 iterações as variáveis são salvas, um teste é realizado na base de dados de testes e o resultado é exibido ao usuário. Ao atingir as iterações definidas o processo de treinamento finaliza e os pesos estarão ajustados e prontos para a etapa de classificação.

3.4 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados experimentos realizados com o aplicativo e os resultados obtidos. Na seção 3.4.1 são apresentados testes de performance realizados, tanto do treinamento realizados na rede neural como do processamento das imagens. A seção 3.4.2 apresenta um comparativo com os trabalhos correlatos.

3.4.1 Testes de performance

Os testes foram realizados com diferentes configurações. Os testes tinham como objetivo descobrir o melhor número de iterações e tamanho de imagem ideal a ser utilizado. Imagens grandes demais tornam o treinamento computacionalmente custoso, mas imagens muito pequenas podem perder informações importantes para ajuste correto dos pesos.

O tamanho inicial utilizado durante a fase de testes foi de 28x28, tamanho inspirado na base de dados MNIST de dígitos. MNIST é uma base de dados largamente utilizada para testes e experimentos em visão computacional (LECUN et al., 1998). A base foi utilizada para realizar a validação do design da rede neural. Uma taxa de acerto de 99.2% foi atingida em 20.000 iterações de 50 imagens. A base de treino era de 60.000 imagens e a de teste 10.000 imagens. A Figura 26 exibe um conjunto de 100 imagens retiradas da base de dados MNIST.

Figura 26 – 100 imagens da base de dados MNIST

Validada a rede neural, uma base com imagens de galáxias foi criada. 107.620 imagens foram separadas para treino e 978 imagens para teste. Elas foram divididas em duas classes: galáxias de tipo precoce e de tipo tardio. O Quadro 19 exibe a quantidade de imagens de cada uma das classes na base de treino e teste.

Quadro 19 – Detalhamento da base de dados

Classe da galáxia/Base de dados	Treino	Testes
Tipo precoce	47.376	410
Tipo tardio	60.244	568

A Tabela 1 apresenta resultados de testes realizados com diferentes números de iterações e tamanho de imagens variados, utilizando o seguinte design: duas camadas de convolução com *max-pooling* 2x2 (32 filtros de 5x5 na primeira, 64 filtros de 5x5 na segunda), uma camada densa com *dropout* (1024 neurônios. *Dropout* em 50%) e uma camada de saída utilizando a função de ativação *softmax*.

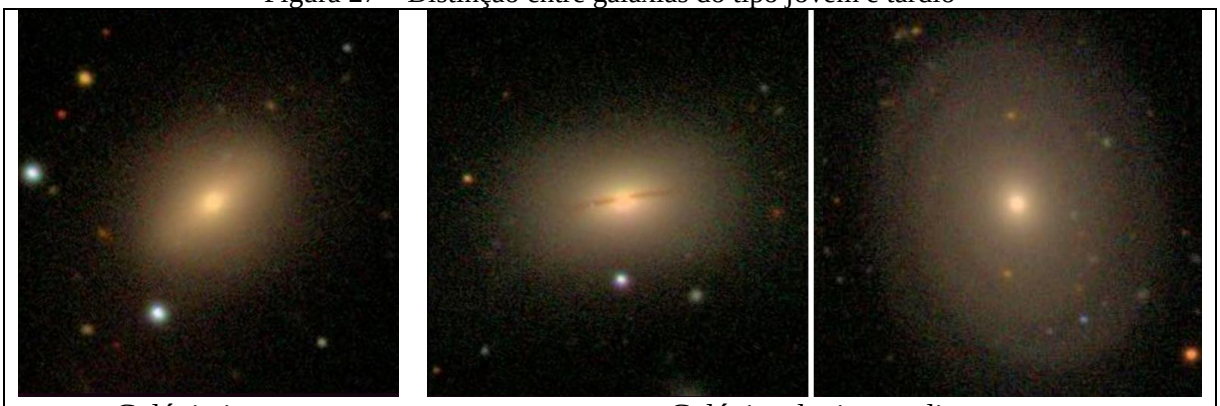
Tabela 1 – Testes de performance com diferentes números de iterações

Teste	Tamanho imagens	Iterações	Duração	Acerto (%)
1	28x28	20.000	1:00:23	73.0
2	28x28	40.000	2:17:15	79.34
3	28x28	60.000	3:20:57	80.57
4	56x56	10.000	2:45:19	81.59
5	56x56	15.000	4:06:16	84.66
6	56x56	28.000	7:20:33	85.68
7	56x56	56.000	13:34:338	86:40
8	56x56	77.000	19:29:10	83.94

O melhor resultado atingido foi com as imagens de tamanho 56x56, cerca de 6.40% superior ao melhor resultado conseguido com as imagens reduzidas para 28x28. Isso indica possível perda de informações nas imagens de 28x28, prejudicial para o treinamento. A partir de 60.000 iterações o ganho de performance foi mínimo ou nulo. A função de custo se manteve estagnada, e apenas pequenas variações na classificação aconteciam devido a aplicação de *dropout*.

Um *script* em Python foi criado durante a fase de testes para identificar os possíveis problemas que a rede neural enfrentou em classificar corretamente as galáxias. Um dos problemas identificado foi a confusão entre galáxias de tipo jovem e tipo tardio nas etapas iniciais (Figura 27).

Figura 27 – Distinção entre galáxias do tipo jovem e tardio



Galáxia jovem Galáxias de tipo tardio

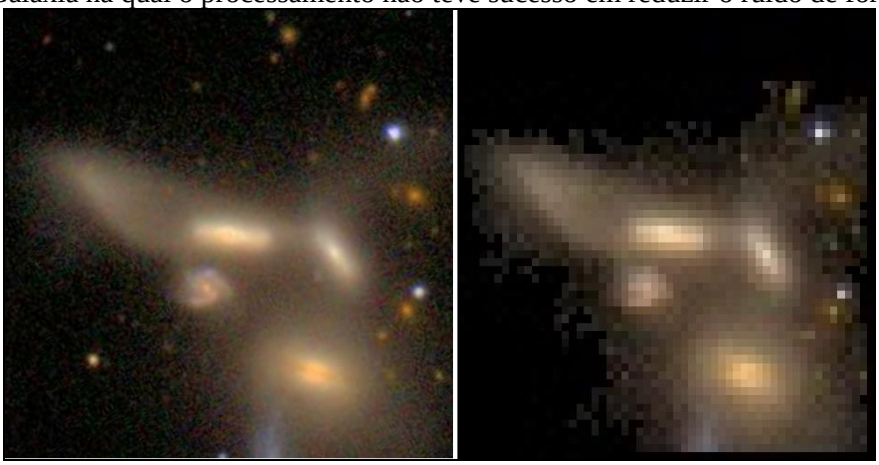
A galáxia que se encontra a esquerda foi classificada corretamente pela rede neural com 100% de chance de ser uma galáxia de tipo jovem e 0% de ser de tipo tardio. As duas galáxias a direita foram classificadas incorretamente. Elas são de tipo tardio, pois já apresentam discretas estruturas em seu disco. A galáxia mais à direita teve como classificação 71.51% de chance de ser de tipo jovem e 28.49% de chance de ser de tipo tardio. Isso indica que os discos foram identificados, mas outras características como cor e formato acabaram se sobressaindo na classificação.

Quando as estruturas em espiral são mais visíveis a rede não apresentou dificuldades em realizar a classificação correta. As galáxias exibidas na Figura 28 são todas de tipo tardio e foram classificadas corretamente com mais de 98% de certeza na classificação.

Figura 28 – Galáxias de tipo tardio classificadas corretamente

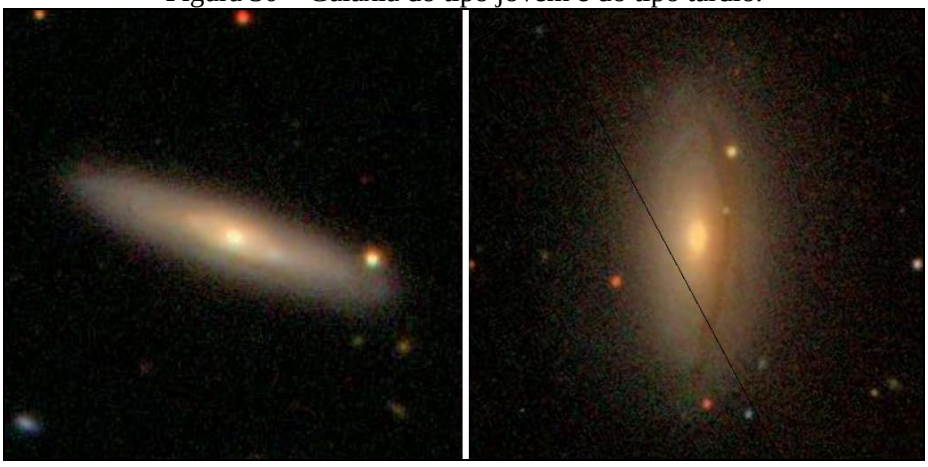
Outro problema identificado foi a incapacidade da fase de processamento das imagens em reduzir o ruído de forma adequada diante da alta complexidade de algumas imagens. A Figura 29 exibe a galáxia original à esquerda e a galáxia processada à direita.

Figura 29 – Galáxia na qual o processamento não teve sucesso em reduzir o ruído de forma adequada



O valor utilizado no algoritmo de limiarização foi alterado em uma tentativa de reduzir esse problema, mas isso causou perda de braços em galáxias espirais e o valor original foi mantido. Galáxias vistas lateralmente também se tornaram um problema para classificações realizadas pela rede neural. Pequenas variações no núcleo e disco definem se a galáxia é de tipo jovem ou tardio (Figura 30).

Figura 30 – Galáxia do tipo jovem e do tipo tardio.



A indicação de disco na galáxia da direita é o que a define como uma galáxia de tipo tardio. Possivelmente, devido a diversos exemplos de galáxias tardias vistas lateralmente e poucas galáxias de tipo jovem, a rede acabou classificando algumas galáxias de tipo jovem (esquerda) como de tipo tardio (direita). Imagens de maior qualidade (112x112), uma fase de processamento aprimorada e uma base de dados maior, são alguns dos pontos levantados que podem melhorar o resultado atingido pela rede.

O Quadro 20 exibe a matriz de confusão das classificações realizadas na base de testes. Das 978 imagens, 845 foram classificadas corretamente e 133 tiveram classificação incorreta.

Quadro 20 – Matriz de confusão das classificações realizadas na base de testes

		Predito	
		Precoce	Tardia
Verdadeiro	Precoce	348	62
	Tardia	71	497

3.4.2 Comparativo com trabalhos correlatos

O Quadro 21 mostra um comparativo entre a ferramenta desenvolvida e os trabalhos correlatos.

Quadro 21 – Comparativo dos trabalhos correlatos e a ferramenta desenvolvida

Características/ trabalhos relacionados	Jenkinson et al. (2014)	Ata et al. (2009)	Elfattah et al. (2013)	Trabalho proposto
Origem da base de dados	Criada pelos autores	Zsolt Frei	Zsolt Frei	SDSS e Galaxy Zoo
Quantidade de imagens de treino	17	113	24	107.620
Quantidade de imagens de teste	17 (retiradas da base de treino)	10 (retiradas da base de treino)	68	978
Algoritmos utilizados para processamento das imagens	*Rooting transform*, *Heap transform*, *Paired transform*.	-	Detecção de bordas e remoção de ruído de ponta	Limiarização e componentes conexas.
Algoritmos utilizados para classificação	SVM	Redes percepton multicamadas, *feedforward* generalizada, modular, Jordan/Elman, PCA, de base radial, auto organizáveis, recorrente atrasada, recorrente e SVM.	Mapas auto organizáveis e SVM	Rede neural convolucional com *dropout* e *softmax*.
Tipo de dados de entrada	Representação esparsa das imagens	Características morfológicas e PCA	Momentos invariantes com Fisher Score	Imagens processadas
Melhor taxa de acerto de treinamento supervisionado	100	99.53	97.29	86.40 (na base de testes) e 97.5 (na base de treino)

Conforme proposto no trabalho desenvolvido por Elfattah et al. (2013) e Jenkinson et al. (2014), o trabalho desenvolvido utiliza uma base de dados maior e mais desbalanceada. Ela contém casos de astros sobrepostos, galáxias se encontrando e ruído. A base de dados é um dos motivos levantados para explicar a diferença da taxa de acerto alcançada pela ferramenta.

O algoritmo classificador utilizado também é diferente do encontrado nas demais implementações, bem como as técnicas utilizadas para o processamento das imagens. O tipo

de entrada utilizado foi a imagem inteira, sem a fase de extração de características encontrada nos demais trabalhos.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de uma ferramenta que realiza classificação morfológica de galáxias a partir de imagens astronômicas. Os objetivos principais eram realizar redução de ruído nas imagens utilizadas, treinamento de uma rede neural e classificação utilizando os pesos treinados.

A etapa de processamento conseguiu reduzir o ruído da base de dados parcialmente, apresentando problema em imagens com duas galáxias muito próximas, astros sobrepostos, ou corpos estranhos aparecendo. Novos algoritmos e técnicas devem ser testados futuramente em busca de melhores resultados.

O treinamento da rede neural, apesar da dificuldade encontrada na etapa de processamento das imagens, foi satisfatório. Taxas de acerto acima de 97% foram atingidas nas imagens de treino. Testes de performance na base de testes (978 imagens) foram realizados com os pesos treinados e atingiram taxa de acerto acima de 86%. Os longos períodos de treinamento foram incômodos e inviabilizaram uma quantidade maior de testes, especialmente em diferentes *designs* de rede.

A ferramenta foi desenvolvida na linguagem de programação Python juntamente com a biblioteca TensorFlow para a criação da rede neural convolucional. Todo o processamento de imagens foi realizado utilizando a biblioteca OpenCV. Toda iteração com o usuário é realizada via *console*.

A base de dados utilizada para validação da implementação foi montada através do serviço ImageCutout do SDSS, com base nas informações disponibilizadas pelo projeto Galaxy Zoo 2. Foram mais de 100.000 imagens baixadas, processadas e utilizadas para treino da rede neural. A criação da base de dados (imagens, processamento, criação dos arquivos de rótulos) foi realizada através de scripts escritos na linguagem de programação Python.

É possível concluir que a implementação realizada apresenta uma alternativa na realização de classificação morfológica de galáxias. Por fim, este trabalho pode servir como base para futuros trabalhos na área.

4.1 EXTENSÕES

Algumas das extensões possíveis para este trabalho são:

a) criação de uma interface gráfica para o usuário;
b) aumentar a base de dados utilizada para treinando e teste;
c) otimizar a etapa de processamento das imagens a fim de reduzir o número de casos com ruído excessivo, como presença de duas galáxias na mesma imagem;

d) fazer testes com outros *designs* de rede para comparação de performance;
e) realizar testes com outros tamanhos de imagens para comparação de performance;
f) implementar outros algoritmos classificadores para comparação de performance na mesma base de dados;
g) utilizar G*raphics Processing Unit* (GPU) para treinamento da rede neural.

REFERÊNCIAS

ABADI, Martín et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Disponível em: <http://arxiv.org/pdf/1603.04467v2.pdf>. Acesso em: 22 jun. 2016. No prelo.

ATA, M. M. et al. Automated classification techniques of galaxies using artificial neural networks based classifiers. **2009 International Conference On Computer Engineering & Systems,** [s.l.], p.157-161, dez. 2009. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/icces.2009.5383290.

BALL, Nicholas M., **Morphological Classification of Galaxies Using Artificial Neural Networks**. 2008. 80 f. Dissertação (Mestrado em Ciência) – Curso de Astronomia, Universidade de Sussex. Disponível em: <http://arxiv.org/pdf/astro-ph/0110492.pdf?origin=publication_detail>. Acesso em 04 set. 2015.

BANERJI, Manda et al. Galaxy Zoo: reproducing galaxy morphologies via machine learning . **Monthly Notices Of The Royal Astronomical Society,** [s.l.], v. 406, n. 1, p.342-353, 30 abr. 2010. Oxford University Press (OUP). http://dx.doi.org/10.1111/j.1365-2966.2010.16713.x.

CAI, Deng; BAO, Hujun; HE, Xiaofei. Sparse concept coding for visual analysis. **Cvpr 2011,** [s.l.], p.2905-2907, jun. 2011. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/cvpr.2011.5995390.

ELFATTAH, Mohamed Abd et al. An intelligent approach for galaxies images classification. **13th International Conference On Hybrid Intelligent Systems (his 2013),** [s.l.], p.167-172, dez. 2013. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/his.2013.6920476.

HEATON, Jeff. **Artifical Intelligente for Humans:** Volume 3: Deep Learning and Neural Networks. [s. L.]: Createspace Independent Publishing Platform, 2015. 374 p.

HUBBLE, E. P.. Extragalactic nebulae. **Apj,** [s.l.], v. 64, p.321-369, dez. 1926. IOP Publishing. http://dx.doi.org/10.1086/143018. Disponível em: <http://adsabs.harvard.edu/abs/1926ApJ....64..321H>. Acesso em: 28 maio 2016.

HUBBLE, Edwin. **The Realm of the Nebulae.** New Haven: Yale University Press, 1936. 207 p.

JENKINSON, John et al. Machine learning and image processing in astronomy with sparse data sets. **2014 Ieee International Conference On Systems, Man, And Cybernetics (smc),** [s.l.], p.200-203, out. 2014. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/smc.2014.6973907.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E.. ImageNet Classification with Deep Convolutional Neural Networks. In: NEURAL INFORMATION PROCESSING SYSTEMS 2012, 26., 2012, Lake Tahoe. **Proceedings of the conference held 12-15.** [s.l]: Mit Press, 2012. p. 1 - 9. Disponível em: <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>. Acesso em: 22 jun. 2016.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings Of The Ieee,** [s.l.], v. 86, n. 11, p.2278-2324, 1998. Institute of Electrical & Electronics Engineers (IEEE). http://dx.doi.org/10.1109/5.726791. Disponível em: <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>. Acesso em: 21 jun. 2016.

MITCHELL, Tom M. (Tom Michael). **Machine learning**. New York : McGraw-Hill, 1997. 414 p, il. (McGraw-Hill series in computer science).

NIETO-SANTISTEBAN, María A.; SZALAY, Alexander S.; GRAY, Jim. ImgCutout, an Engine of Instantaneous AstronomicalDiscovery. In: ASTRONOMICAL DATA ANALYSIS SOFTWARE AND SYSTEMS (ADASS), 13., 2003, Strasbourg. **Proceedings... .** San Francisco: Astronomical Society Of The Pacific, 2004. v. 314, p. 666 - 669. Disponível em: <http://adsabs.harvard.edu/full/2004ASPC..314..666N>. Acesso em: 18 jun. 2016.

SERRANO, Rodney Delgado. **The Evolution of the Hubble Sequence:** morpho-kinematics of distant galaxies. 2010. 189 f. Tese (Doutorado) - Curso de Astronomy And Astrophysics, Laboratoire Galaxies, Etoiles, Physique Et Instrumentation, Observatorie de Paris, Paris, 2010. Disponível em: <http://arxiv.org/pdf/1201.6406v1.pdf>. Acesso em: 20 set. 2015.

ZSOLT, Frei. **Galaxy Catalog.** 1999. Recuperado 2002 por Princeton University, Department of Astrophysical Sciences. Disponível em: <http://www.astro.princeton.edu/ frei/catalog.html>. Acesso em: 19 set. 2015.

WILLETT, K. W. et al. Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey. **Monthly Notices Of The Royal Astronomical Society,** [s.l.], v. 435, n. 4, p.2835-2860, 22 set. 2013. Oxford University Press (OUP). http://dx.doi.org/10.1093/mnras/stt1458. Disponível em: <http://arxiv.org/pdf/1308.3496v2.pdf>. Acesso em: 14 jun. 2016.

APÊNDICE A – Detalhamento dos casos de uso

Nesta seção são apresentados os detalhamentos dos casos de uso, com descrição, ator, pré-condições e cenários.

No caso de uso `UC01 – Processar imagem` o `usuário` pode realizar o processamento de uma imagem, removendo possíveis sujeiras, a redimensionando e criando uma nova imagem a partir do resultado. Detalhes sobre o caso de uso estão descritos no Quadro 22

Quadro 22 – `UC01 – Processar imagem`

Número	01
Caso de Uso	Processar imagem
Ator	`Usuário`
Pré-condições	Estar com a ferramenta aberta e possuir a imagem a ser processada
Cenário principal	1. O `Usuário` seleciona a opção `Denoise` (1) da ferramenta. 2. O `Usuário` informa o caminho da imagem. 3. O `Usuário` informa o tamanho das novas imagens que serão criadas. 4. O U`suário` informa o caminho que a nova imagem será salva. 5. A ferramenta executa o processamento da imagem.
Fluxo alternativo	No passo 2 o `usuário` pode informar uma pasta contendo diversas imagens. Isso irá repetir os passos seguintes para cada uma das imagens encontradas na pasta.

No caso de uso `UC02 – Treinar rede neural` o ator pode executar o treinamento da rede neural para futuras classificações. Detalhes sobre o caso de uso estão descritos no Quadro 23.

Quadro 23 – `UC02 – Treinar rede neural`

Número	02
Caso de Uso	Treinar rede neural
Ator	`Usuário`
Pré-condições	Estar com a ferramenta aberta e possuir uma base de dados para treino e outra para teste. Ambas devem ser formadas por imagens que possuem suas classes mapeadas em um arquivo de rótulos.
Cenário principal	1. O `Usuário` seleciona a opção `Train` (2) da ferramenta. 2. O `Usuário` informa o caminho das imagens de treino. 3. O `Usuário` informa o caminho do arquivo de rótulos das imagens de treino. 4. O `Usuário` informa o caminho das imagens de teste. 5. O `Usuário` informa o caminho do arquivo de rótulos das imagens de teste. 6. O `Usuário` informa a quantidade de classes diferentes existentes nos dados. 7. O `Usuário` informa o tamanho do *batch* a ser utilizado durante o treinamento. 8. O `Usuário` informa o número de iterações que devem ser utilizadas durante o treinamento. 9. O `Usuário` informa o tamanho das imagens. 10. O `Usuário` informa o número de canais das imagens. 11. A ferramenta exibe *feedback* ao usuário durante treinamento.

No caso de uso `UC03 – Classificar imagem` o usuário pode realizar a classificação morfológica de uma galáxia a partir de sua imagem. Detalhes sobre o caso de uso estão descritos no

Quadro 24 – `UC03 – Classificar imagem`

Número	03
Caso de Uso	Classificar imagem
Ator	`Usuário`
Pré-condições	Estar com a ferramenta aberta, possuir variáveis já treinadas pela rede neural e ter uma imagem processada de uma galáxia com a mesma dimensão das imagens utilizadas durante o treinamento.
Cenário principal	1. O `Usuário` seleciona a opção `Classify` (3) da ferramenta. 2. O `Usuário` informa o caminho da imagem. 3. O `Usuário` informa o tamanho das imagens. 4. O `Usuário` informa o número de canais das imagens. 5. O `Usuário` informa a quantidade de classes de classificação 6. A ferramenta exibe a classificação realizada (um valor de 0 a n-1, onde n é o número de classes de classificação) e a imagem.
Fluxo alternativo	Caso uma classificação ou treinamento já tenha sido realizada anteriormente desde a sua última inicialização, a aplicação vai do passo 2 diretamente para o passo 6.