UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Victor Gutemberg Oliveira Marques

Avaliação do desempenho das redes neurais convolucionais na detecção de ovos de esquistossomose

RECIFE
2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Victor Gutemberg Oliveira Marques

Avaliação do desempenho das redes neurais convolucionais na detecção de ovos de esquistossomose

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Engenharia da Computação, orientada pelo professor Germano Crispim Vasconcelos.

RECIFE
2017
UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Victor Gutemberg Oliveira Marques

Avaliação do desempenho das redes neurais convolucionais na detecção de ovos de esquistossomose

Monografia submetida ao corpo docente da Universidade Federal de Pernambuco, defendida e aprovada em 14 de julho de 2017.

Banca Examinadora:

_____

Orientador

Doutor Germano Crispim Vasconcelos

_____

Examinador

Doutor Robson do Nascimento Fidalgo

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Marques e Jilmara, por terem me dado muitas oportunidades na minha vida que me trouxeram até aqui e incluo os meus irmãos, Camilla e Gabriel, por estarem sempre ao meu lado durante todo o percurso da faculdade.

Agradeço aos professores e funcionários do Centro de Informática (CIn) da UFPE que me permitiram ter acesso à uma formação de qualidade excepcional. Foram alguns anos de aprendizado que valeram a pena e apesar de todos os sofrimentos passados com muitas provas e projetos sei que farão muita falta no futuro.

Agradeço aos meus amigos que tornaram todo esse trajeto mais agradável. Por todos os momentos de risadas, alegrias e desesperos que só quem passou conseguirá entender.

Agradeço também à empresa PickCells por ter fornecido todas as imagens de lâminas contendo ovos de esquistossomo utilizadas nesse trabalho e também pelo aprendizado na área de aprendizagem de máquinas.

RESUMO

A esquistossomose mansoni é uma doença parasitária causada pelo agente Schistosoma mansoni. Essa parasitose está atrás apenas da malária em número de infectados no mundo. Cerca de 200 milhões de pessoas no âmbito global estão infectadas e aproximadamente 12 milhões de pessoas somente no Brasil [1].

O diagnóstico da doença é realizado através da identificação de ovos do parasita em lâminas contendo amostras de fezes do paciente possivelmente infectado [2].

Esse trabalho busca automatizar a detecção de ovos de esquistossomose em imagens de lâminas através de técnicas de aprendizagem de máquina e processamento de imagens. As redes neurais convolucionais será utilizada como extrator de características para fazer a classificação de objetos presentes em imagens das lâminas.

Palavras-chave: esquistossomose, diagnóstico, rede neurais convolucionais

ABSTRACT

Schistosomiasis mansoni is a parasitic disease caused by the agent Schistosoma mansoni. This parasite is only behind malaria in the number of infected in the world. Around 200 million people globally are infected and approximately 12 million people in Brazil alone [1].

The diagnosis of the disease is performed by identifying parasite eggs on slides containing stool samples from the possibly infected patient [2].

This work seeks a possible automation in the detection of schistosomiasis eggs in slide images through machine learning techniques and image processing. The convolutional neural networks will be used as an extractor of characteristics to make the classification of objects present in images of the blades.

Keywords: schistosomiasis, diagnosis, convolutional neural networks

Sumário

Lista de Figuras

LISTA DE TABELAS

TABELA DE SIGLAS

Sigla	Significado
MLP	*Multilayer Perceptron*
CNN/ConvNet	*Convolutional Neural Network*
ROC	*Receiver Operating Characteristic*
AUROC	*Area under the Receiver Operating Characteristic*
ReLU	*Rectified Linear Unit*

1. Introdução

A esquistossomose mansoni é uma doença parasitária causada pelo agente Schistosoma mansoni. Esse parasita apresenta duas fases de desenvolvimento. Na primeira fase o agente se aloja nos caramujos gastrópodes aquáticos podendo posteriormente infectar o ser humano. O ser humano é o hospedeiro principal mais frequente. É nele que os parasitas chegam à fase adulta e se reproduzem depositando seus ovos no hospedeiro. Os ovos são posteriormente eliminados através das fezes poluindo o ambiente e possivelmente infectando novas pessoas [3].

O diagnóstico da doença é realizado através da identificação de ovos de esquistossomose nas fezes do hospedeiro. O processo manual se inicia com a preparação da lâmina utilizando amostras de fezes do paciente. Posteriormente toda a lâmina é analisada por um laboratorista que irá verificar a existência de ovos [2]. Por ter dependência humana esse processo pode sofrer variâncias, decorrentes do cansaço visual ou pouca experiência do laboratorista. A utilização de meios computacionais para detecção dos ovos em imagens de lâminas poderia auxiliar o diagnóstico da doença, além de possivelmente torna-lo mais rápido.

Redes neurais convolucionais (ConvNets) têm sido amplamente utilizadas na classificação de imagens. Existe uma vasta quantidade de soluções propostas para o problema de classificação de imagens. Em um dos ramos de estudo se situam as soluções que aplica o uso de Redes Neurais com *Deep Learning*, sendo as ConvNets contidas dentro desse ramo. Essa técnica tem obtido ótimos resultados, chegando próximo do desempenho humano em alguns casos, como evidenciado por Krizhevsky [4].

Esse trabalho se propõe a analisar o desempenho das redes neurais convolucionais na classificação de ovos do esquistossomo. Imagens de lâminas contendo ovos de esquistossomose foram obtidas com auxilio de um microscópio. Segmentos contendo os ovos foram recortados e utilizados para criação da base de imagens utilizadas para os testes realizados. Os segmentos podem então ser classificados em duas classes que estão contendo ovo ou não contendo.

1.1. Objetivos

O objetivo geral desse trabalho é desenvolver um modelo computacional para detecção de ovos de esquistossomose visando auxiliar o diagnóstico da doença. O objetivo específico será analisar a eficiência dos modelos de redes neurais convolucionais na identificação de ovos do parasita causador da esquistossomose em imagens lâminas com amostra de fezes de pacientes infectados.

2. Conceitos Básicos

Este capítulo irá focar no entendimento básico do problema e introdução aos termos e técnicas usadas ao longo do trabalho.

2.1. Banco de imagens

O banco de imagens utilizado neste trabalho será o mesmo utilizado no artigo proposto por André Firmo em 2010 [5]. O processo completo é explicado no artigo citado e será brevemente descrito. As imagens foram obtidas com auxilio de um microscópio e uma câmera a ele acoplada com auxilio de um suporte. Um laboratorista treinado fez a varredura das lâminas e ao achar setores com a presença de ovos uma imagem era capturada. Também foram capturadas imagens sem a presença de ovos, pois o método utilizado no artigo exigia exemplos que não continham ovos. As imagens obtidas possuem 640 de largura por 480 de altura. Devemos notar que cada um dos pixels da imagem é composto por 3 componentes que são as cores vermelha, verde e azul. Graças a isso podemos considerar cada imagem como uma matriz de tamanho 640x480x3.

É importante ressaltar que as imagens apresentam alguns problemas de padronização. As capturas não utilizaram isolamento do ambiente, o que resultou em imagens com diferentes luminosidades e até mesmo gradientes de luminosidade. É possível observar na Imagem 1 por exemplo que a luminosidade apresenta variações comparando a parte superior e inferior da lâmina. Já na Imagem 2 os ovos estão localizados em áreas pouco iluminadas, o que torna difícil a diferenciação entre o ovo e o plano de fundo.

Após a captura das imagens foi necessário a marcação da posição exata dos ovos. Com auxílio de um software de visualização todas as imagens foram revistas pelo laboratorista que pôde selecionar as posições dos ovos. Cada posição é descrita pelas coordenadas x e y além da altura e largura do retângulo selecionado. Um exemplo de imagem e suas marcações é visto na Imagem 1.

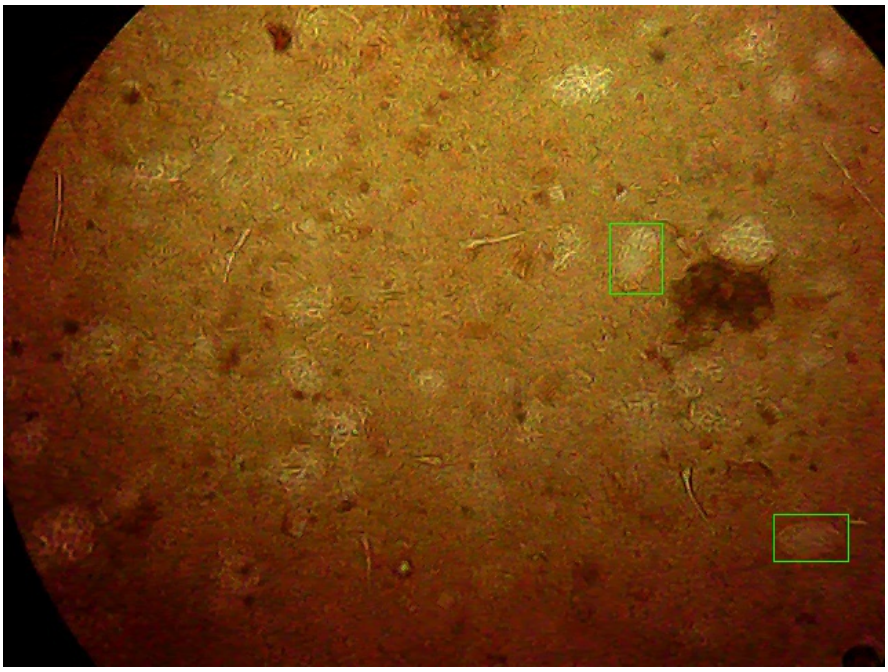


Imagem 1 - Captura da lâmina contendo variância de luminosidade

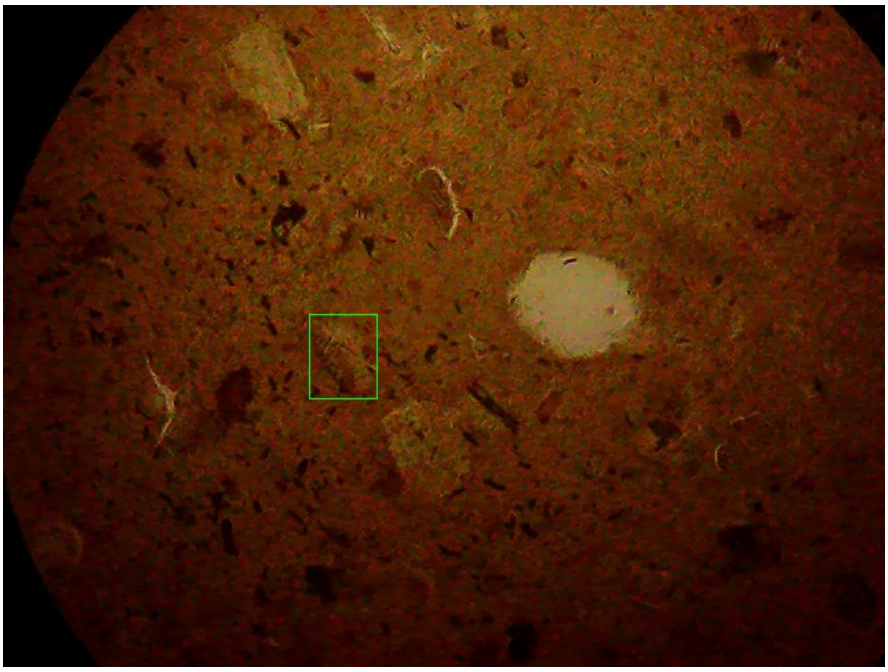


Imagem 2 - Captura da lâmina contendo ovo em área pouco iluminada

2.1.1.Segmentação

Os métodos de classificação utilizados neste trabalho têm entradas de tamanho fixo. Para poder utiliza-los as imagens devem ser previamente segmentadas e redimensionadas de forma a obter fragmentos contendo os

possíveis objetos a serem classificados. Foram feitas algumas tentativas de segmentação utilizando métodos como SLIC, Sobel, *Felzenszwalb* e *Quickshift*, porém devido a falta de padrão do bando de imagem e da baixa resolução nenhum dos métodos obteve bons resultados.

Para obtenção dos segmentos a lista de imagens e coordenadas dos ovos foi fornecida a um script de pré-processamento. No primeiro passo as marcações retangulares foram convertidas em quadrados. Essa conversão foi feita aumentando a dimensão de menor tamanho e centrando a imagem de forma que o ovo ficará no meio. Após esse processo os segmentos foram recortados da imagem dando origem à base da classe positiva.

Para obtenção dos segmentos da classe negativa o algoritmo proposto por André Firmo [5] foi utilizado como auxílio. O artigo utiliza um método chamado *Haar Cascades* proposto por Paul Viola e Michael J Jones em 2004 para a detecção facial [6]. Esse método após treinado recebe como entrada uma imagem e gera uma lista contendo marcações de segmentos que foram classificados como ovos. Os parâmetros utilizados foram relaxados de forma que foi possível obter uma grande quantidade de segmentos marcados incorretamente. Essa abordagem foi escolhida pois é possível obter alguns segmentos que têm aparência semelhante à de um ovo, mesmo não sendo. Com isso podemos ter mais certeza de que o conjunto da classe negativa não conterá apenas exemplos contendo somente plano de fundo. Caso isso acontecesse os segmentos semelhantes aos ovos poderiam ser erroneamente classificados como pertencentes a classe positiva. O primeiro segmento da Imagem 4 (b) por exemplo é similar aos ovos contidos nos segmentos positivos.

A Imagem 3 apresenta a lâmina da Imagem 1 contendo em vermelho as marcações do *Haar Cascades* e em verde as marcações feitas pelo laboratorista. Nesta imagem é possível observar que a lâmina contém outros artefatos que se assemelham aos ovos, como por exemplo o segmento marcado com o número 1. As marcações que possuem área em comum maior do que 20% com as marcações do laboratorista são eliminadas pois possuem partes do ovo.

Todos os segmentos após o recorte foram redimensionados para o tamanho 28x28 afim de padronizar o tamanho e possibilitar o uso dos classificadores.

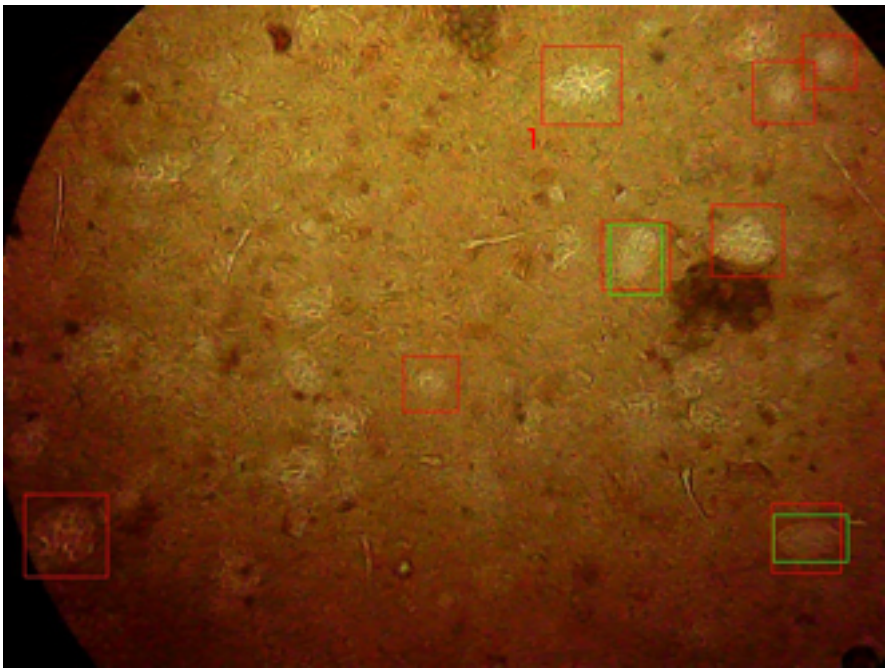


Imagem 3 - Imagem de lâmina marcada (em vermelho) pelo *Haar Cascades*

(a) Exemplos da classe positiva (ovos) (b) Exemplos da classe negativa

Imagem 4 - Exemplos de segmentos de imagens

2.2. Sampling

Após a etapa de segmentação das imagens das lâminas é necessário separar os segmentos obtidos em três subconjuntos. Esses subconjuntos serão utilizados para o treinamento, validação e teste dos métodos a serem comparados. A divisão é feita de forma que a quantidade imagens de cada classe seja proporcional ao

tamanho da subdivisão. A divisão escolhida foi de forma que 50% das imagens de cada classe serão para treinamento, 25% para validação e 25% para teste.

A quantidade de exemplos da classe negativa é muito maior que a classe positiva, contendo respectivamente 15649 e 2392. Antes da utilização das imagens no treinamento e validação é necessário balancear as quantidades de imagens em cada classe. Caso isso não fosse feito a método de aprendizagem poderia apresentar uma performance boa para um tipo de classe e não para a outra. Neste trabalho serão abordadas três formas de balanceamento diferentes.

2.2.1.Under sampling

Esse método de balanceamento irá fazer a redução da quantidade de segmentos da classe majoritária de forma que o número de segmentos se torne igual ao da classe minoritária. Para realizar tal atividade o método de clusterização k-médias será utilizado. Esse método foi descrito por MacQueen, James em 1967 [7] como o “particionamento de uma população N-dimensional em k conjuntos”. Para cada um dos k conjuntos será escolhido um protótipo, que é idealmente um ponto com a menor distância a todos os segmentos contidos naquele conjunto. Por essa propriedade cada um dos protótipos pode ser usado como representante daquele conjunto, já que apresentam semelhança a todos os elementos ali contidos. Neste caso o número k será igual ao tamanho da classe minoritária o que irá fazer com a classe majoritária seja reduzida a k representantes.

2.2.2.Over sampling

O primeiro método de balanceamento por *over sampling* não utiliza nenhum tipo de inteligência. Para tornar a quantidade de segmentos de cada classe igual uma cópia randômica dos segmentos da classe minoritária é realizada até que as quantidades se igualem.

O segundo método a ser utilizado é o *Synthetic Minority Oversampling Technique* (SMOTE) proposto por Chawla et al em 2002 [8]. O método funciona criando amostras sintéticas através dos exemplos da classe minoritária. Para gerar

novas amostras o primeiro passo é calcular os k vizinhos mais próximos de cada uma das amostras, considerando apenas as da classe minoritária. Após essa etapa um certo número de vizinhos é escolhido. Esse número é definido pela quantidade de amostras que se deseja criar. Por exemplo, se o objetivo é aumentar o número de amostras em 200% é necessário escolher aleatoriamente dois vizinhos dentre os k, de forma que cada amostra irá criar duas novas. Após isso a diferença entre a amostra e seu vizinho é calculada e multiplicada por um valor aleatório entre 0 e 1. O resultado da operação anterior é então somado a amostra inicial gerando o resultado final. Essa operação é mais facilmente vista na equação abaixo, em que x é a amostra a base, x' o seu vizinho e δ um valor aleatório entre 0 e 1.

$$x_{new} = x + (x' - x) * \delta$$

2.3. Aprendizagem de Máquina

O objetivo geral dos métodos de aprendizagem de máquina (AM) é conseguir adquirir conhecimento através de uma base de exemplos do problema a ser resolvido. Os exemplos são fornecidos ao método de aprendizagem escolhido e através desse processo um conjunto de características do problema é identificado e armazenado. Neste trabalho será utilizado o método de treinamento supervisionado, explicado em mais detalhes na subsecção abaixo.

2.3.1. Treinamento Supervisionado

A primeira suposição a ser feita quando se deseja usar o treinamento supervisionado é que existe uma base de exemplos do problema que foram previamente classificados com ajuda de um especialista do assunto. Por exemplo, caso se deseje treinar um classificador de cachorros e gatos, o primeiro passo é obter um conjunto de imagens dos dois animais e manualmente classifica-las. O conjunto de imagens previamente classificadas então é usado para o treinamento do classificador. Para este trabalho o conjunto de imagens consiste em segmentos que contêm ovos do esquistossomo e segmentos que não os contêm que foram

previamente classificados com ajuda de um laboratorista. O processo de obtenção e classificação das imagens de treinamento é explicado em mais detalhes no capítulo "Obtenção das Imagens".

Esse é o tipo de aprendizagem mais utilizado, tanto por técnicas tradicionais quando por técnicas de *deep learning* [9]. Em geral os métodos de aprendizagem recebem uma entrada e geram como saída um vetor contendo uma pontuação para cada classe. Quanto maior essa pontuação, maior a certeza do sistema de que a entrada pertence aquela classe. O objetivo do treinamento é então fazer com que o sistema responda com a maior pontuação à classe que a entrada pertence. Como neste caso a resposta já é conhecida o sistema deve então adaptar os seus parâmetros internos de forma que a classe correta apresente a maior pontuação. Os exemplos de treinamento são apresentados e os parâmetros internos são ajustados até que a taxa de erro atinja um limiar definido.

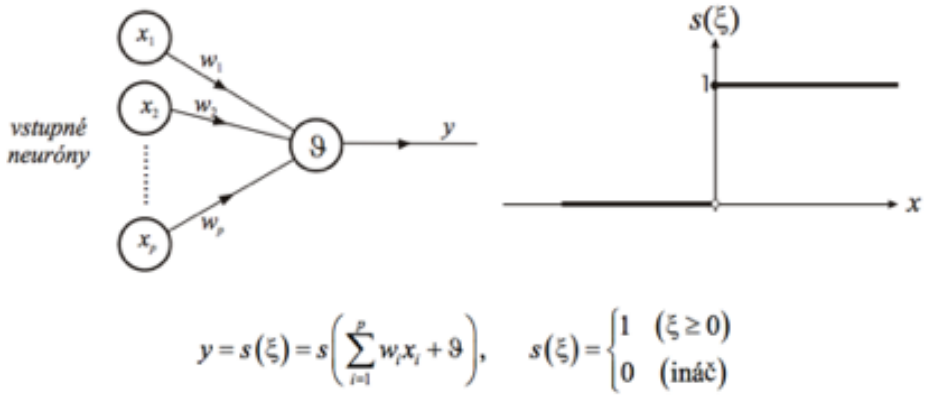
Um dos métodos de atualização dos parâmetros internos bastante utilizado no treinamento de redes multicamada é o *backpropagation* proposto por Robert em 1988 [10]. De forma resumida, o método funciona inicialmente calculando o erro da camada de saída, comparando com o valor esperado. Os pesos entre a camada de saída e a camada anterior são atualizados para diminuir o erro e o gradiente do erro é repassado à camada anterior. Esse processo é realizado até que todos os pesos até a camada de entrada sejam atualizados.

2.4. Sistema Inteligente

Dois métodos de classificação foram escolhidos afim de comparar a performance obtida por eles. Os métodos escolhidos foram os de *Multilayer Perceptron* (MLP) e as redes neurais convolucionais. Apesar de ter sido proposto em 1986 [11], o MLP ainda é amplamente utilizado na indústria atualmente. Já as redes neurais convolucionais são mais recentes e tem se popularizado na classificação de imagens.

2.4.1.Multilayer Perceptron (MLP)

O modelo de *multilayer perceptron* surgiu como uma evolução do *perceptron*. O *perceptron* é um algoritmo de aprendizagem proposto por Rosenblat em 1957 [12]. Nesse modelo uma unidade de aprendizagem recebe múltiplas entradas em que cada entrada está associada a um peso. Cada valor de entrada é multiplicado ao seu peso associado e o resultado é somado a um valor constante chamado bias. O resultado dessa soma é então passado a uma função de ativação que irá responder com zero ou um dependendo do valor total da soma. O conhecimento adquirido é armazenado nos pesos que conectam à entrada.



$$y = s(\xi) = s\left(\sum_{i=1}^{p} w_i x_i + \vartheta\right), \qquad s(\xi) = \begin{cases} 1 & (\xi \geq 0) \\ 0 & (\text{iná\v{c}}) \end{cases}$$

Imagem 5 - Modelo geral do *perceptron*[1]

Em 1986, Rumelhart, Hinton e Williams publicaram o artigo chamado "*learning internal representation by error propagation*" [11] em que descrevem um modelo de *perceptron* em múltiplas camadas. Uma grande vantagem apresentada por esse modelo em relação ao *perceptron* simples é a sua capacidade de solucionar problemas não linearmente separáveis.

Esse modelo possui uma arquitetura em camadas, que podem conter números variados de neurônios em cada camada. Essas são organizadas de forma que os neurônios de cada camada recebem como entrada a saída da camada anterior, o que é chamado de *feed-forward*. Outra restrição é que as unidades de

1 Disponível em <http://www2.fiit.stuba.sk/~cernans/nn/nn_texts/neuronove_siete_priesvitky_02_Q.pdf> acesso julho, 2017

cada camada não se comunicam entre si [13]. Existem três tipos de camadas que compõem a rede. A primeira é a camada de entrada que tem como papel receber os dados. Como o objetivo dessa camada é apenas receber os dados, os pesos associados a cada entrada é um. Posteriormente podem vir uma ou mais camadas escondidas. Esse tipo de camada extrai informações dos dados recebidos pela camada anterior e as repassam à camada a frente. Por último temos a camada de saída que conterá a resposta do sistema ao dado fornecido a camada de entrada.

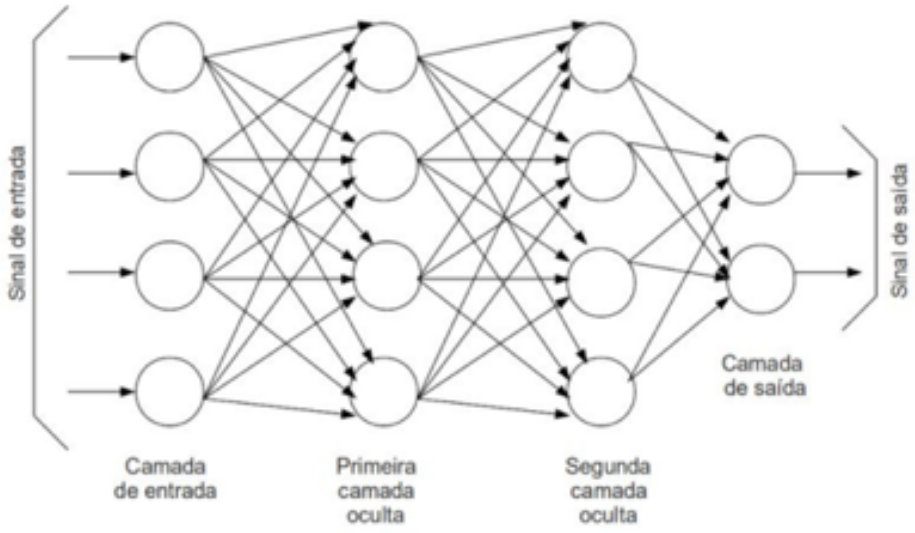


Imagem 6 - Arquitetura em múltiplas camadas (MLP)[2]

2.4.2.Deep Learning

Nos algoritmos tradicionais, tais como o MLP, os dados podem precisar passar por um processo de engenharia de características (*feature engineering)* para que seja possível realizar a classificação. Esse processo é realizado por um especialista que define quais atributos do dado devem ser utilizados. Alguns dos atributos podem não ter grande relevância durante a classificação, portanto não devem ser utilizados. Esse processo requer grande conhecimento sobre o problema e pode ser custoso e demorado dependendo da complexidade do dado.

[2] Disponível em <https://alexandrevolpi.wordpress.com/tag/dino/> acesso Julho, 2017

Os algoritmos de *deep learning* são um subconjunto dos métodos de *feature learning* [9]. Esses algoritmos buscam extrair de forma automática as características necessárias para classificação dos dados. Esse objetivo é alcançado utilizando múltiplas camadas intermediárias simples, porém não lineares. Cada uma dessas camadas é treinada e transforma o dado recebido em uma representação mais abstrata que é repassada a camada posterior. Através do treinamento, as camadas se especializam na extração ou filtro de diferentes atributos específicos do problema. Por esse processo é possível derivar atributos de mais alto nível, partindo da representação pura do dado fornecido. Na classificação de imagens, por exemplo, a primeira camada pode extrair curvas e repassar essa informação para a camada posterior. A segunda camada por então combinar algumas dessas curvas formando partes de objetos. Já a terceira camada compõe objetos mais complexos que então serão usados para classificação.

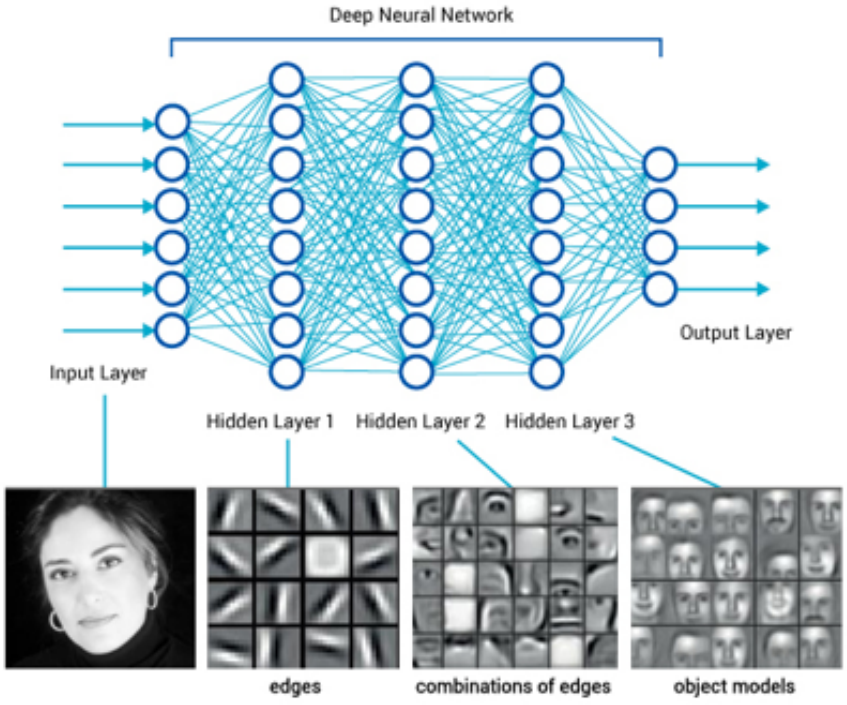


Imagem 7 - Exemplo de arquitetura *deep learning*[3]

A arquitetura apresentada na Imagem 7 se assemelha bastante a arquitetura da rede MLP apresentada na Imagem 6. A principal diferença dessas duas redes

[3] Disponível em <https://www.nextbigfuture.com/2016/08/wave-computing-massively-parallel-deep.html> acesso Junho, 2017

está na quantidade de nós nas camadas escondidas, além da quantidade de camadas escondidas utilizadas. Os neurônios de uma camada são ligados aos neurônios da camada anterior através de pesos, que é onde o conhecimento adquirido é armazenado. As redes de *deep learning* podem conter centenas de milhões de ligações [9] que precisam ser ajustadas para a resolução de cada problema. Um dos métodos que pode ser usado nessa tarefa é o gradiente descendente estocástico, descrito na parte de treinamento da MLP.

Algumas áreas como visão computacional e reconhecimento de fala já foram fortemente impactadas pelo uso de arquiteturas *deep learning*. Outras áreas como linguagem natural, processamento de texto e recuperação de informações tem um grande potencial de impacto por esse tipo de rede [14].

2.5. Métricas de avaliação dos resultados

Após o treinamento os diferentes métodos de aprendizagem deverão ser avaliados para que possam ser comparados. Para realizar a avaliação algumas métricas comumente utilizadas em problemas de classificação serão utilizadas.

2.5.1. Matriz de confusão

A matriz de confusão é uma ferramenta importante pois permite analisar de forma rápida o desempenho de cada sistema. Os valores que compõem a matriz são obtidos fornecendo os segmentos do conjunto de teste ao método de classificação e comparando sua predição com a classe correta de cada segmento.

Imagem 8 - Matriz de confusão de problema contendo duas classes[4]

Após a comparação os valores são classificados em quatro possíveis opções:

1. **Verdadeiro Positivo (VP):** segmentos que pertencem a classe positiva e foram classificados como positivos. No caso desse trabalho, são segmentos que contêm ovos do esquistossomo e foram classificados corretamente.
2. **Falso Positivo (FP):** segmentos que pertencem a classe negativa e foram classificados como positivos.
3. **Falso Negativo (FN):** segmentos que pertencem a classe positiva e foram classificados como negativos. São segmentos que contêm ovos, porém foram erroneamente classificados como negativos.
4. **Verdadeiro Negativo (VN):** segmentos que pertencem a classe negativa e foram corretamente classificados como negativos.

2.5.2.Métricas de Qualidade

As métricas descritas abaixo são popularmente usadas na comparação de modelos de classificação. Cada uma delas busca avaliar um aspecto diferente do modelo.

1. **Acurácia**: quantidade de segmentos tanto da classe positiva da negativa que foram classificados corretamente. Essa métrica deve ser tomada em consideração com cuidado uma vez que a base de

[4] Disponível em <http://crsouza.com/2009/07/13/analise-de-poder-discriminativo-atraves-de-curvas-roc/> acesso junho, 2017

imagens está desbalanceada, existem muito mais segmentos da classe negativa. Dessa forma mesmo no caso em que todas as predições tivessem um valor fixo para classe negativa o valor da acurácia poderia ser alto.

2. **Precisão**: quantidade de segmentos que, dentre todos os segmentos classificados como positivos, são pertencem a classe positiva.

$$Precisão = \frac{VP}{(VP + FP)}$$

3. **Recall**: quantidade de segmentos positivos que foram classificados corretamente. É importante para garantir que a classe positiva que é minoritária também terá uma boa taxa de acerto. Para esse trabalho essa é uma métrica muito importante, uma vez que é necessário garantir que os ovos contidos na lâmina serão identificados.

$$Recall = \frac{VP}{(VP + FN)}$$

4. **F1**: é uma métrica que considera tanto a taxa de precisão quanto o recall. O seu valor tradicional é a média harmônica entre a precisão e o recall multiplicado pela contaste 2.

$$F1 = 2 * \frac{1}{\left(\frac{1}{Recall} + \frac{1}{Precisão}\right)} = 2 * \frac{Precisão * Recall}{(Precisão + Recall)}$$

5. **Area Under the Receiver Operating Characteristic (AUROC)**: para cada segmento apresentado ao classificador esse irá ter uma saída a classe predita para aquele segmento. Para este trabalho, essa saída possui valor entre 0 e 1. Quanto mais próximo a saída estiver do 0 maior a certeza do classificador de que aquele segmento pertence a classe negativa e quanto mais próximo de 1 da classe positiva. A curva ROC funciona através da variação de um limiar de classificação que vai de 0 a 1 e mede a certeza que o método de classificação tem para cada um dos limiares. O eixo vertical contabiliza a quantidade para a classe positiva e o eixo vertical para classe negativa. Quanto mais próximo essa curva estiver do canto superior esquerdo maior é a certeza do classificador sobre as classificações feitas. A área embaixo

da curva ROC mede exatamente esse nível de certeza do método a ser avaliado. A linha tracejada representa um classificador aleatório.

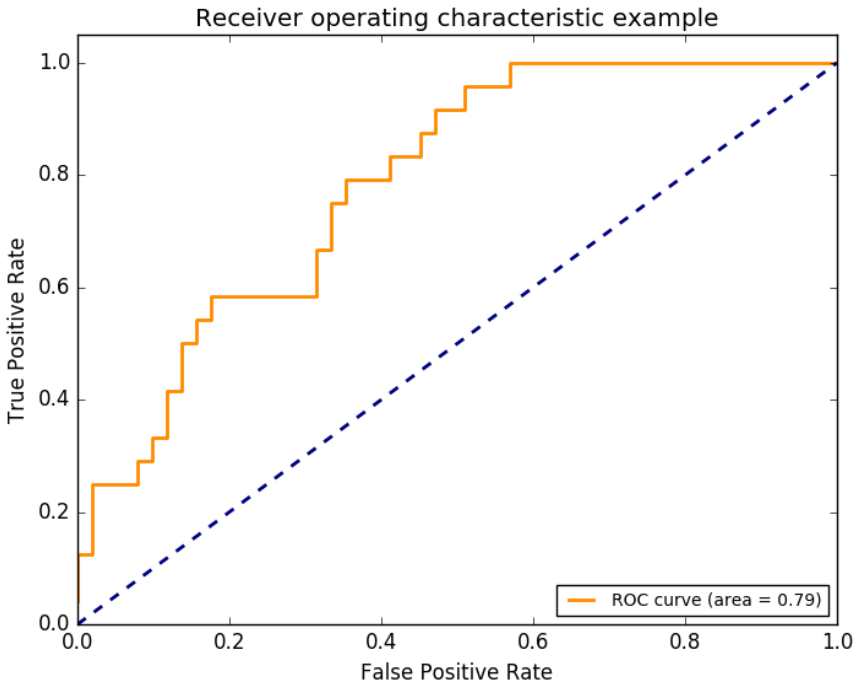


Imagem 9 - Exemplo de curva ROC[5]

[5] Disponível em <http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html> acesso julho, 2017

3. Rede Neural Convolucional

Neste trabalho o foco será dado a um tipo específico dos algoritmos de *deep learning* que são as redes neurais convolucionais. O nome dessa rede se dá pelos filtros de convolução utilizados por esse tipo de técnica. Esses filtros buscas extrair certas características da entrada. Um exemplo de filtro de convolução capaz de dar destaque às bordas presentes em uma figura é mostrado na Imagem 13.

A inspiração para criação dessa arquitetura veio do estudo do funcionamento do córtex visual dos gatos feito 1962 pelos pesquisadores Hubel e Wiesel [15]. Através do estudo da reação cerebral de gatos a diferentes padrões de luz (paradas e em movimento) foi possível observar que o córtex visual desses animais é dividido em sub-regiões. Cada uma dessas sub-regiões é ativada ao sofrer estímulos de diferentes formas, como por exemplo uma borda horizontal ou vertical. Ao final as informações provenientes de todas as sub-regiões são somadas e então o objeto é identificado. Em 1995 os pesquisadores Yann LeCun e Yoshua Bengio [16] utilizaram esse estudo para arquitetar a primeira rede neural convolucional, em inglês *convolutional neural network* (CNN). Esse tipo de rede neural foi desenvolvido para processar dados em múltiplas dimensões, como por exemplo imagens. Uma imagem armazenada no padrão RGB (red, green, blue) possui altura, largura e profundidade, considerando a profundidade como sendo as 3 camadas de cor utilizadas na representação.

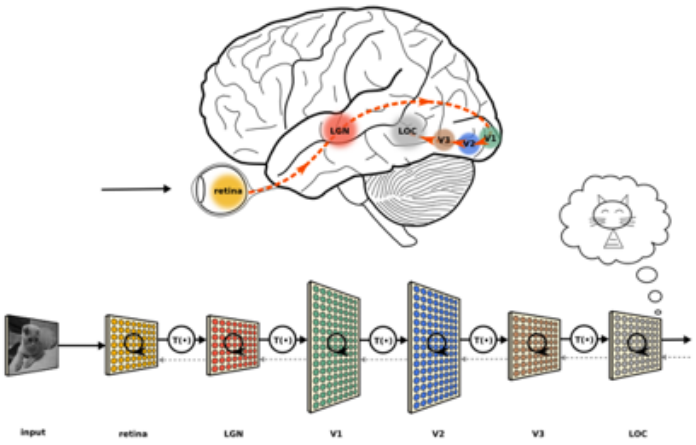


Imagem 10 - Exemplo das sub-regiões do córtex cerebral[6]

[6] Disponível em <https://neuwritesd.org/2015/10/22/deep-neural-networks-help-us-read-your-mind/> acesso julho, 2017

A Imagem 11 mostra a arquitetura geral de uma rede neural de convolução descrita por Alex Krizhevsky et al em 2012 [17]. Essa arquitetura faz uso de dois tipos de unidades que são as de convolução e de *max-pooling*. O funcionamento dessas unidades será explicado com mais detalhes nas subseções a seguir.

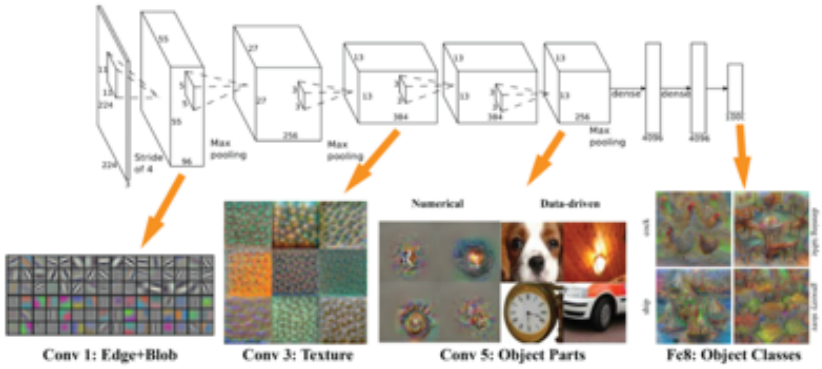


Imagem 11 - Exemplo de arquitetura de uma rede neural convolucional[7]

3.1. Unidade de Convolução

No processo de convolução uma janela com pesos definidos é deslizada por toda a imagem com o objetivo de extrair certas características. Na Imagem 12 por exemplo, um filtro de tamanho 5x5 foi utilizado e posicionado inicialmente no canto superior esquerdo. A entrada possui um tamanho de 32x32. Os valores contidos no filtro são então multiplicados pelos valores dos pixels na posição do filtro, somados e retirado a média, gerando um valor como resultado. O filtro é então movido para a direita e essa etapa é realizada novamente até que toda a imagem tenha sido processada. O resultado desse processo é chamado de mapa de características e possui tamanho 28x28.

[7] Disponível em <http://vision03.csail.mit.edu/cnn_art/index.html> acesso julho, 2017

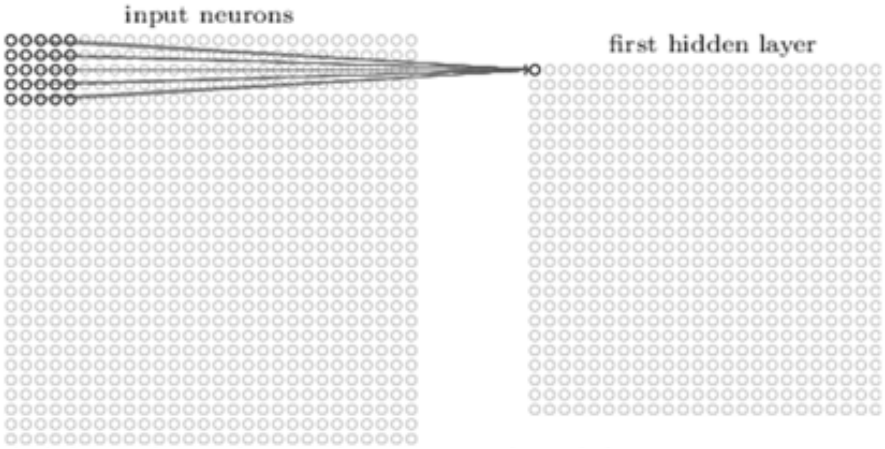


Imagem 12 - Visualização do funcionamento do filtro de convolução[8]

Na Imagem 13 é possível ver com mais facilidade o resultado de um filtro de convolução. O filtro utilizado nesse caso tem como objetivo realçar as bordas presentes na imagem. Vários desses filtros foram desenvolvidos durante os anos, tendo a principal desvantagem o fato de serem fixos, necessitando que o problema seja bem conhecido para se tornar possível a sua utilização. Nas ConvNets os valores dos filtros são obtidos durante o treinamento utilizando os dados reais do problema. A grande vantagem dessa abordagem é que elimina a necessidade da participação de um especialista no problema a ser resolvido, passando essa responsabilidade para a ConvNet. Durante o treinamento os valores dos filtros são ajustados para se adaptar ao problema que se deseja resolver, buscando extrair as melhores características para classificação dos dados de treino.

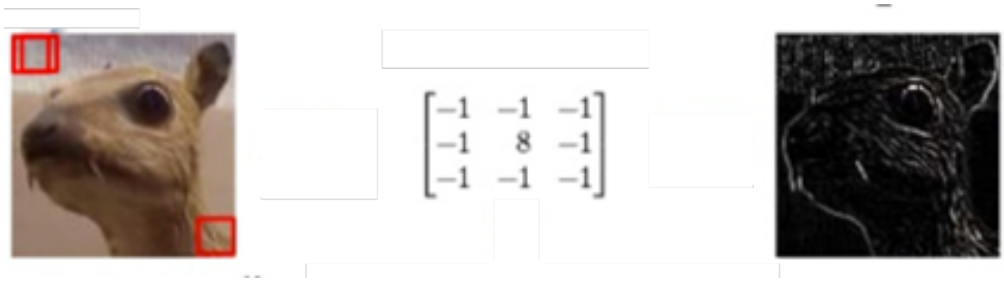


Imagem 13 - Filtro de detecção de bordas[9]

Os filtros descritos até agora processam apenas um nível de profundidade da imagem. Para atingir todas as camadas da entrada basta que múltiplos filtros sejam

[8] Disponível em <http://neuralnetworksanddeeplearning.com/> acesso junho, 2017
[9] Disponível em <https://www.slideshare.net/RukshanBatuwita/deep-learning-towards-general-artificial-intelligence> acesso julho, 2017

empilhados, dessa forma toda a entrada é levada em consideração. O filtro utilizado deve ter pelo menos a mesma profundidade da entrada, porém um número maior pode ser utilizado, permitindo que uma mesma camada seja processada por mais de um filtro no mesmo componente. Um exemplo desse tipo de aplicação é mostrado na Imagem 11. O primeiro filtro possui altura e largura iguais a 11 e profundidade 3. Já a profundidade de saída desse filtro é igual a 96, o que significa que cada camada de entrada é processada por 32 filtros de convolução.

3.2. Unidade de *Pooling*

A unidade de *pooling*, também conhecida como *down sampling*, pode ser utilizada após as camadas de convolução para reduzir a dimensão espacial da entrada. Uma operação similar pôde ser observada no estudo do córtex visual de gatos [15]. Na Imagem 14 um exemplo simples de *max-pooling* de tamanho 2x2 é demonstrado. Assim como o filtro de convolução a aplicação desse filtro envolve o seu deslizamento sobre toda a entrada. A cada passo a janela é deslizada na mesma quantidade da sua largura, de forma que não existam pontos de interseção. A ideia é escolher um candidato de cada vizinhança como representante que no caso do *max-pooling* é sempre o maior valor do grupo. Existem outros filtros para realização do *pooling* como por exemplo média e l2-norm.

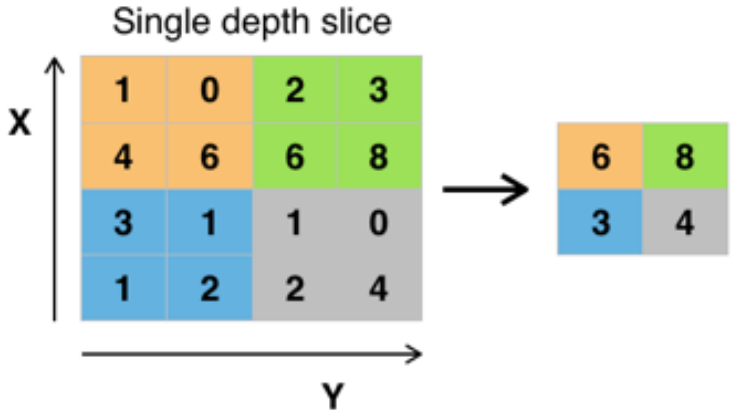


Imagem 14 - Filtro de *max-pooling* de tamanho 2x2[10]

[10] Disponível em <http://neuralnetworksanddeeplearning.com/> acesso junho, 2017

3.3. Unidade de Classificação

Usualmente a última camada de uma rede neural convolucional é um classificador totalmente conectado. Esse classificador irá receber como entrada o vetor de características extraídas pelas camadas anteriores e não a imagem original. Existe uma vasta gama de opções que podem ser utilizadas nesta etapa, como por exemplo a rede MLP cujo modelo foi descrito na seção 2.4.1.

4. Implementação

Esse trabalho tem por objetivo avaliar a performance da ConvNet e compara-la à performance da MLP, logo a implementação desses dois sistemas foi necessária. A linguagem escolhida para essa atividade foi Python[11], mais especificamente a versão 2.7. Python é uma linguagem de programação interpretada de alto nível que possui uma sintaxe limpa e pode ser facilmente aprendida. Uma das vantagens dessa linguagem é que ela libera o programador da necessidade de se preocupar com alocação de memória, sendo tudo feito de forma automática pelo interpretador da linguagem. Uma desvantagem do seu uso é que essa linguagem possui uma performance muito inferior a outras linguagens de mais baixo nível como C ou C++.

Uma forma de diminuir esse problema de performance é a utilização de bibliotecas de computação intensiva. Para esse trabalho a biblioteca chamada *TensorFlow*[12] foi utilizada. O *TensorFlow* é uma biblioteca de código aberto desenvolvida pelo *Google Brain Team*. Essa funciona como uma interface entre Python e C++. Todo o código pode ser escrito utilizando a sintaxe Python, porém toda a execução dos cálculos é realizada em código implementado em C++. Dessa forma é possível ter a facilita de escrita em Python com o poder computacional do C++, adicionando apenas o overhead de comunicação entre as duas linguagens.

No desenvolvimento da MLP a biblioteca Keras[13] também foi utilizada. Essa biblioteca tem como base de funcionamento o TensorFlow, adicionando apenas mais uma camada de abstração para o usuário. A MLP já é implementada pelo Keras, sendo apenas necessário definir alguns parâmetros como por exemplo quantidade de neurônios em cada camada, função de ativação e taxa de aprendizagem. Todos esses parâmetros utilizados serão descritos na seção 5.3. O código aqui desenvolvido pode ser obtido através do GitHub[14].

[11] http://www.python.org/
[12] http://www.tensorflow.org
[13] http://www.keras.io/
[14] https://github.com/victorgutemberg/convolutional

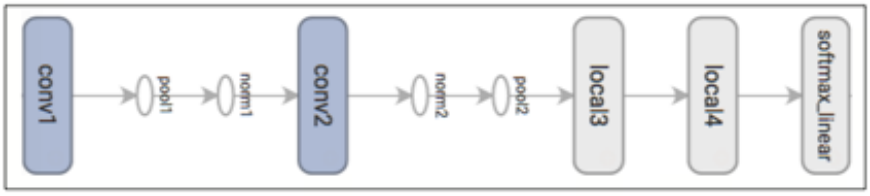


Imagem 15 - Arquitetura proposta por Alex Krizhevsky [4][15]

A arquitetura para ConvNet escolhida foi a proposta no tutorial do TensorFlow para a classificação do problema CIFAR10[16], que possui imagens de 10 classes diferentes tais como carro, avião, cachorro, gato, etc. As imagens classificadas pelo CIFAR10 não possuem relação com esse trabalho, será apenas utilizada a arquitetura da ConvNet desenvolvida para solução do problema. A Imagem 15 apresenta as camadas presentes nessa rede que foi proposta por Alex Krizhevsky [4]. A primeira camada de convolução irá processar a entrada inicial dada ao classificador. Essa camada tem tamanho 5x5x3 de entrada e sua profundidade de saída é 64. Já a segunda camada de convolução possui tamanho de 5x5x64 de entrada e profundidade de saída também 64. As duas camadas de *pooling* são do tipo *max-pooling* e possuem tamanho 3x3. As camadas de normalização são descritas em mais detalhes por Krizhevsky et al [18] e obedecem a formula abaixo.

$$b_{x,y}^{i} = a_{x,y}^{i} \Big/ (k + \alpha \sum_{\mathrm{j}=\max(0,\mathrm{i}-{}^{n}/_{2})}^{\min(N-1,i+{}^{n}/_{2})} (a_{x,y}^{i})^{2})^{\beta}$$

Todo o código utilizado nesta etapa é de livre acesso e pode ser obtido na página do TensorFlow[17].

[15] Disponível em <https://www.tensorflow.org/tutorials/deep_cnn> acesso julho, 2017
[16] https://www.cs.toronto.edu/~kriz/cifar.html
[17] https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10

5. Experimentos e Análise

Afim de examinar com maior detalhe o desempenho dos classificadores alguns experimentos foram definidos. Os experimentos irão avaliar o impacto de diferentes atributos dos classificadores o que possibilitará um maior entendimento do problema e possivelmente a determinação de melhores parâmetros para cada classificador. Será dado um peso maior ao parâmetro de *recall* pois para o problema a ser tratado é importante que o máximo de ovos seja identificado.

5.1. Experimento 1

O primeiro experimento irá avaliar o impacto do *sampling* na para criação da base de dados de treinamento. Inicialmente o número de segmentos da classe positiva é de 2392 enquanto o da classe negativa é 15649. Por causa desse grande desbalanceamento pode acontecer de que os métodos aprendam de forma satisfatória a classificar a classe negativa, porém apresentem uma má performance na classificação dos segmentos positivos. Nesse experimento será realizado o treinamento das duas redes inicialmente com a base original desbalanceada e posteriormente com a mesma base após o balanceamento descritos na seção 2.2. Os parâmetros iniciais da ConvNet estão descritos no experimento 2 e da MLP no experimento 3.

	Original	**K-médias**	**SMOTE**	**Cópia Aleatória**
Acurácia	**0.9767**	0.8184	0.9763	0.9690
Precisão	**0.9214**	0.4192	0.8966	0.8635
Recall	0.9013	**0.9582**	0.9281	0.9097
F1	0.9112	0.5832	**0.9121**	0.8860
AUROC	**0.9922**	0.9565	0.9919	0.9870

Tabela 1 - Resultados da ConvNet à base de dados com diferentes métodos de *sampling*

O método de k-médias obteve o melhor valor de *recall*, porém sua precisão foi muito baixa comparada aos outros métodos. O segundo melhor *recall* foi obtido com o SMOTE, cujas métricas foram bastante similares as maiores obtidas, com exceção da precisão.

	Original	K-médias	Cópia Aleatória	SMOTE
Acurácia	**0.9355**	0.8493	0.8818	0.9091
Precisão	0.7756	0.4627	0.5368	**0.9624**
Recall	0.7224	**0.8495**	0.7926	0.7926
F1	**0.7481**	0.5991	0.6401	0.6981
AUROC	**0.9440**	0.9266	0.9156	0.9388

Tabela 2 - Resultados da MLP à base de dados com diferentes métodos de *sampling*

No caso da MLP o melhor resultado de *recall* também foi obtido pelo k-médias, porém esse método ficou bastante atrás nas outras métricas avaliadas. O segundo melhor *recall* foi alcançado pelo SMOTE, que também conseguiu a melhor taxa de precisão. A base de dados sem nenhum tipo de *sampling* conseguiu três das melhores métricas para acurácia, F1 e AUROC.

Apesar da base de dados sem nenhum método de *sampling* aplicado ter obtido bons resultados, a escolha da base para os próximos experimentos será a do SMOTE. Isso se da pelo fato dessa base ter alcançado o segundo melhor *recall* que para o problema a ser tratado é uma métrica muito importante.

5.2. Experimento 2

Esse experimento irá focar em parâmetros apenas da ConvNet. Inicialmente os filtros de *max-pooling* terão tamanho fixo de 3x3 e os filtros de convolução terão seus tamanhos alterados para 3x3, 5x5 e 7x7.

	3x3	5x5	7x7
Acurácia	**0.9834**	0.9763	0.9738
Precisão	**0.9470**	0.8966	0.9040
Recall	0.9264	**0.9281**	0.8980

F1	**0.9366**	0.9121	0.9010
AUROC	**0.9952**	0.9919	0.9916

Tabela 3 - Variação do tamanho dos filtros convolucionais na ConvNet

O filtro de convolução com tamanho 3x3 obteve melhor resultado que os outros filtros em todos as métricas coletadas, com exceção do recall em que o tamanho 5x5 obteve um resultado ligeiramente maior. É possível observar uma piora dos resultados com o aumento do tamanho dos filtros, o que é um bom indicativo de que o problema aqui tratado possui características locais pequenas.

Após os experimentos com a unidade de convolução, essa terá seu tamanho fixado em 3x3 por ter obtido o melhor resultado e as camadas de *max-pooling* terão seus tamanhos alterados para 5x5.

	3x3	**5x5**
Acurácia	**0.9834**	0.9794
Precisão	**0.9470**	0.9160
Recall	0.9264	**0.9298**
F1	**0.9366**	0.9228
AUROC	**0.9952**	0.9950

Tabela 4 - Variação do tamanho dos filtros *pooling* na ConvNet

Neste caso o filtro de menor tamanho (3x3) também alcançou as melhores métricas, com a única exceção do *recall* que foi bastante próximo ao máximo obtido.

5.3. Experimento 3

Esse experimento irá focar em parâmetros apenas da MLP. Os segmentos de imagem possuem tamanho 28x28x3 o que da um tamanho total de 2352 pixels, que será o tamanho da camada de entrada da rede. Inicialmente será utilizada apenas uma camada escondida de tamanho 1176 que representa 50% do tamanho da camada de entrada. Serão testadas as funções sigmoide, tangencial hiperbólica e *Rectified Linear Unit (ReLU).* A unidade de saída terá a função de ativação sigmoide.

	Sigmoide	Tanh	ReLU
Acurácia	0.9395	0.9346	0.8960
Precisão	0.8083	0.7750	0.6183
Recall	0.7124	0.7140	0.5635
F1	0.7573	0.7433	0.5897
AUROC	0.9535	0.9424	0.7544

Tabela 5 - Resultado da MLP com diferentes funções de ativação na camada escondida

A função de ativação sigmoide obteve o melhor resultado em todas as métricas, com exceção do *recall* em que a tangente hiperbólica obteve um resultado ligeiramente melhor. A função ReLU foi pior em todos os aspectos quando comparado as outras duas funções de ativação. A função sigmoide será utilizada daqui em diante para os próximos experimentos.

Após a definição da função de ativação, o número de neurônios na camada escondida será avaliado. Os tamanhos a serem testados serão baseados no tamanho da camada de entrada vezes 0.25, 0.5 e 0.75. Também serão testados alguns tamanhos arbitrários de 50 e 100 com o objetivo de ver se tamanhos menores são suficientes para resolver o problema.

	50	100	588	1176	1764
Acurácia	0.9078	0.9173	0.9222	**0.9224**	0.9190
Precisão	0.6161	0.6628	0.6939	**0.6950**	0.6870
Recall	**0.8077**	0.7659	0.7391	0.7391	0.7157
F1	0.6990	0.7106	0.7158	**0.7164**	0.7010
AUROC	0.9438	**0.9444**	0.9438	0.9366	0.9331

Tabela 6 - Resultado da MLP com variação do número de neurônios na primeira camada escondida

A rede com 1176 neurônios na camada escondida obteve vantagem em todas as métricas com exceção do *recall* e AUROC. A melhor taxa de *recall* foi alcançada pela rede com 50 nós na camada escondida, porém essa rede teve resultados muito abaixo nas outras métricas quando comparado com todas as opções de redes.

O próximo experimento irá avaliar a adição de mais uma camada escondida à rede. Assim como feito na primeira camada serão testados os tamanhos 50, 100 e os múltiplos de 0.25, 0.5 e 0.75, porém agora considerando o tamanho da primeira camada escondida.

	50	**100**	**294**
Acurácia	0.9308	0.9299	**0.9335**
Precisão	0.7314	0.7196	**0.7411**
Recall	0.7559	**0.7726**	0.7659
F1	0.7434	0.7452	**0.7533**
AUROC	0.9481	0.9450	**0.9488**

Tabela 7 - Resultado da MLP com variação do número de neurônios na primeira camada escondida

Os tamanhos 588 e 822 foram omitidos pois classificaram todos os segmentos como sendo da classe negativas. Dentro os tamanhos testados para a segunda camada escondida o valor 294 obteve destaque. A única métrica em que não alcançou o melhor resultado foi o *recall*, porém apresentou um resultado similar ao melhor atingido. Comparando os valores da Tabela 6 e Tabela 7 é possível observar que a adição de uma camada escondida a mais possibilitou a melhora de todos as métricas analisadas.

A arquitetura final obtida através dos experimentos é uma MLP cuja função de ativação dos neurônios escondidos e de saída é a sigmoide. Essa rede possui duas camadas escondidas sendo a primeira de tamanho 1176 e a segunda 294.

5.4. Análise

Nesta seção será feita uma comparação entre o desempenho obtido pela rede ConvNet e MLP.

	ConvNet	**MLP**
Acurácia	**0.9834**	0.9335
Precisão	**0.9470**	0.7411

Recall	**0.9264**	0.7659
F1	**0.9366**	0.7533
AUROC	**0.9952**	0.9488

Tabela 8 – Resultado da melhor arquitetura obtida para as redes ConvNet e MLP

Através da análise da Tabela 8 é possível observar que o desempenho da rede ConvNet foi bastante superior ao da MLP em todas as métricas calculadas. A maior diferença foi na taxa de precisão em que a ConvNet obteve vantagem de 20.59%.

Essa grande vantagem obtida pelas redes convolucionais pode ser atribuída ao fato de que o classificador final irá classificar apenas um vetor de características extraídas do dado inicial. As ConvNets, durante o treinamento, buscam obter o melhor conjunto de características a serem extraídas para uma determinada base de exemplos. No caso das imagens poderia ser realizado um trabalho manual de descoberta de bons filtros para pré-processamento, o que poderia fazer a MLP gerar resultados tão bons quanto as ConvNets. Esse trabalho, porém, teria um alto custo e exigiria um bom conhecimento do problema [9].

As imagens abaixo são exemplos da aplicação dos filtros de convolução da primeira camada após a realização do treinamento. Essa camada possui 64 filtros de saída, o que gera o grid de imagens mostrado à direita de cada segmento. É possível ver na Imagem 16 que os contornos do ovo estão bem destacados do resto da imagem, principalmente nos resultados da primeira linha. Esses contornos serão ainda processados pelos outros filtros gerando o vetor de características final. Esse reforço de características específicas do problema torna o processo de classificação mais simples, resultando na melhor performance das ConvNets.

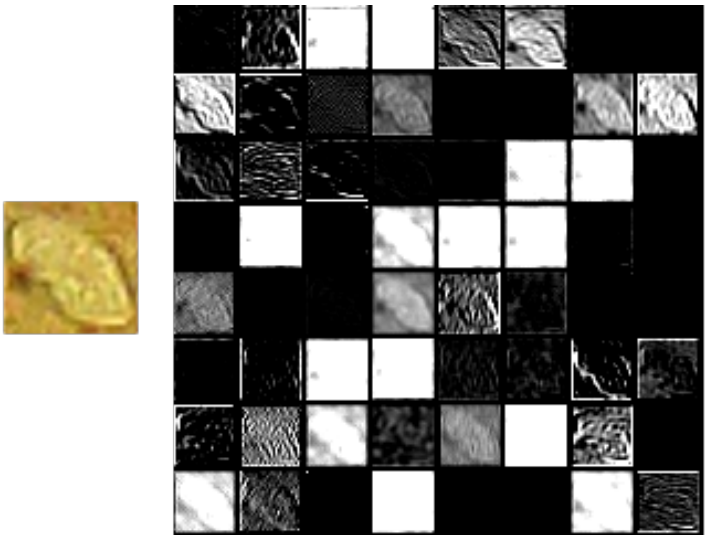


Imagem 16 - Segmento positivos após a aplicação dos filtros de convolução

A Imagem 17 apresenta um segmento que se assemelha a um segmento positivo. É possível observar que apesar da semelhança o resultado da aplicação dos filtros é bastante diferente, sem a definição de nenhum contorno.

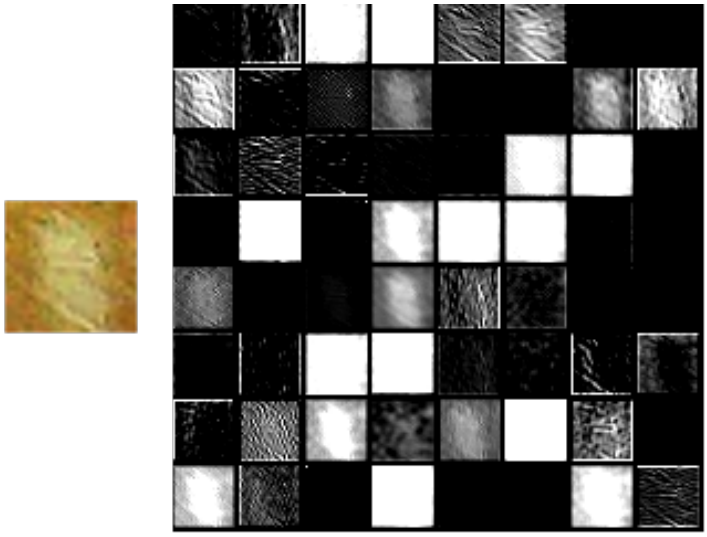


Imagem 17 - Segmento negativo (similar à positivo) após a aplicação dos filtros de convolução

Já a Imagem 18 contém um segmento que é apenas background, logo o resultado esperado são imagens lisas de saída que foi o obtido pela rede.

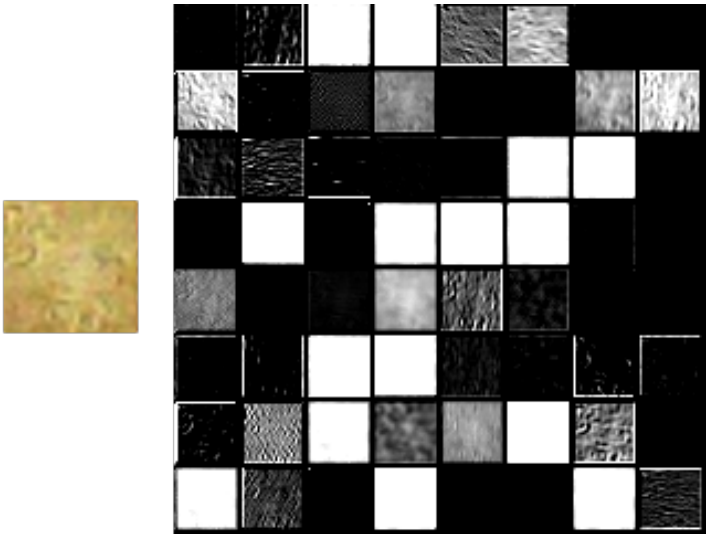


Imagem 18 - Segmento negativo (background) após a aplicação dos filtros de convolução

6. Conclusões e Trabalhos Futuros

O processo completo de diagnóstico não pôde ser realizado devido à baixa performance obtida pelos métodos de segmentação. As imagens das lâminas completas apresentam grande variâncias entre si em relação a iluminação. Além disso a baixa resolução em que as imagens foram capturadas podem ter influenciado nessa etapa. A segmentação das imagens, porém, é uma etapa extra desse trabalho, já que o objetivo era a avaliação das ConvNets.

As métricas coletadas para os classificadores também podem ter sido afetadas pela não padronização das imagens que compõem o banco. Com o isolamento da luminosidade ambiente e utilização de uma câmera com maior resolução seria possível eliminar algumas variáveis do problema.

A rede MLP apresentou bons resultados na resolução do problema chegando a taxa de *recall* máximo de 76.59%. Apesar desse bom resultado essa rede dificilmente poderia ser utilizada numa aplicação real de diagnóstico. Muitos dos ovos do esquistossomo presentes na imagem não seriam detectados, o que poderia levar à não detecção da doença.

Os resultados obtidos pela ConvNet proposta foram bastante satisfatórios, em que a menor métrica obtida foi o *recall* com um valor de 92.64%. Apesar do valor alto, numa aplicação real, alguns cuidados ainda teriam que ser tomados como por exemplo a validação por um especialista da área. Ainda assim o diagnóstico poderia ser acelerado uma vez que, caso ovos do esquistossomo fossem detectados, o trabalho de validação não exigiria a leitura completa da lâmina.

6.1. Trabalhos Futuros

A busca por um método de segmentação eficiente das imagens ainda é um problema em aberto deixado pelo trabalho. A utilização de filtros de pré-processamento e talvez a tentativa de outras técnicas de segmentação poderia resultar em uma boa performance. Com a solução desse problema será possível o diagnóstico completo de uma lâmina de fezes para detecção de ovos do esquistossomo.

A ConvNet obteve já obteve um resultado muito bom, superando as expetativas para o trabalho. A busca por parâmetros melhores pode ainda ser feita para melhorar as métricas obtidas, principalmente o *recall* que é uma das métricas mais importantes e obteve o resultado mais baixo. Também pode ser feita a avaliação dessa rede para a detecção de outras doenças a fim de analisar se sua performance seria tão satisfatória como para esquistossomose.

7. Bibliografia

A. de Souza Andrade Filho, A. C. de Queiroz, M. G. dos Reis, R. M. 1] Amaral e R. M. Brito, “NEUROESQUISTOSSOMOSE,” *Revista Brasileira de Neurologia e Psiquiatria,* vol. 19, 20 5 2016.

K. P. d. N. Hertta Hellen Sousa Marculino, R. A. Pacheco e M. G. V. 2] Sampaio, “ESQUISTOSSOMOSE: UMA QUESTÃO DE SAÚDE PÚBLICA,” *Mostra Interdisciplinar do curso de Enfermagem,* vol. 2, nº 1, 2017.

Portal da Saúde. Ministério da Saúde, “Descrição da Doença,” 17 05 3] 2017. [Online]. Available: http://portalsaude.saude.gov.br/index.php/o-ministerio/principal/leia-mais-o-ministerio/656-secretaria-svs/vigilancia-de-a-a-z/esquistossomose/11240-descricao-da-doenca.

A. Krizhevsky, I. Sutskever e G. E. Hinton, “Imagenet classification with 4] deep convolutional neural networks,” em *Advances in neural information processing systems*, 2012.

A. C. A. Firmo, “Utilização de um Particle Swarm Optmization para 5] otimização do método de diagnóstico da Esquistossomose Mansônica no litoral de Pernambuco,” 2010.

P. Viola e M. J. Jones, “Robust real-time face detection,” *International 6] journal of computer vision,* vol. 57, nº 2, pp. 137-154, 1 5 2004.

J. MacQueen, “Some methods for classification and analysis of 7] multivariate observations,” *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability,* vol. 1, nº 14, pp. 281-297, 21 6 1967.

N. V. Chawla, K. W. Bowyer, L. O. Hall e W. P. Kegelmeyer, “SMOTE: 8] synthetic minority over-sampling technique,” *Journal of artificial intelligence research,* vol. 16, pp. 321-357, 2002.

Y. LeCun, Y. Bengio e G. Hinton, “Deep learning,” *Nature,* vol. 521, nº 9] 7553, pp. 436-444, 28 5 2015.

R. Hecht-Nielsen, “Theory of the backpropagation neural network.,” 10] *Neural Networks,* vol. 1, nº Supplement-1, pp. 445-448, 1 1988.

D. E. Rumelhart, G. E. Hinton e R. J. Williams, “Learning internal

11] representations by error propagation," 1985.

F. Rosenblatt, "The perceptron: A probabilistic model for information
12] storage and organization in the brain.," *Psychological review,* vol. 65, nº 6, p. 386, 11 1958.

L. I. Kuncheva, Combining pattern classifiers: methods and algorithms,
13] John Wiley & Sons, 2004.

L. Deng e D. Yu, "Deep learning: methods and applications," *Foundations
14] and Trends® in Signal Processing,* vol. 7, nº 3-4, pp. 197-387, 30 6 2014.

D. H. Hubel e T. N. Wiesel, "Receptive fields, binocular interaction and
15] functional architecture in the cat's visual cortex.," *The Journal of physiology,* vol. 160, nº 1, pp. 106-154, 1962.

Y. LeCun e Y. Bengio, "Convolutional networks for images, speech, and
16] time series," *The handbook of brain theory and neural networks,* vol. 3361, nº 10, p. 1995, 1995.

A. Krizhevsky e G. Hinton, "Learning multiple layers of features from tiny
17] images," vol. 2, 8 4 2009.

A. Krizhevsky, I. Sutskever, G. Hinton, F. Pereira, C. Burges, L. Bottou e
18] K. Weinberger, "Advances in neural information processing system 25," pp. 1097-1105, 2012.

8. Apêndix

Esse apêndix irá descrever a ordem que os scripts disponíveis através do link https://github.com/victorgutemberg/convolutional devem ser utilizados.

Para a construção da base de dados três scripts serão utilizados. extract_positives.py, get_false_positives.py e get_false_positives.py. Para o primeiro script o único parâmetro necessário é o arquivo com a lista de imagens e suas marcações. Esse arquivo está localizado dentro da pasta Treinamento com o nome positivas.dat. O segundo script requer, além do arquivo de marcação, um arquivo xml contendo o resultado do treinamento utilizando o método de cascades. O arquivo está disponível na pasta Treinamento com o nome cascades.xml. Para executar o script através do terminal use os seguintes comandos:

```
python extract_positives.py –info ../Treinamento/positivas.py
python get_false_positives.py –info ../Treinamento/positivas.py –cascade ../Treinamento/cascade.xml
```

Os dois scripts acima irão gerar arquivos contendo a localização dos diversos segmentos a serem recortados. Por fim o script create_datasets.py irá fazer o recorte das imagens e salva-los em um arquivo binário. O armazenamento é feito de forma que o primeiro byte representa a classe do segmento e os 28*28*3 bytes seguintes são os dados da imagem, sendo primeiro todos bytes da camada vermelha, depois verde e depois azul.

```
python create_datasets.py
```

Com os dados devidamente extraídos é necessário agora gerar as bases de dados aplicando os métodos de *sampling*. Para realizar tal tarefa é necessário executar o script python sampling.py que irá gerar cinco arquivos de saída. Um dos arquivos é utilizado no teste e quatro outros arquivos, um para cada método de *sampling* utilizado.

```
python sampling.py
```

Para executar testes utilizando a rede MLP basta utilizar o arquivo MLP.py. Antes de executar é necessário a definição dos parâmetros a serem utilizados. Para escolher qual método de *sampling* será utilizado basta mudar a variável SAMPLING_NAME para entre as opções *no*, *kmeans*, *smote* ou *copy*. Para mudar a quantidade de neurônios na camada escondida a linha abaixo deve ser alterada. Neste exemplo a rede possui 2351 neurônios de entrada e 1176 neurônios na camada escondida, com função de ativação sigmoide.

```
classifier.add(Dense(1176, activation='sigmoid', input_dim=2352))
```

Para adicionar mais uma camada escondida basta inserir abaixo da linha demonstrada em cima a linha abaixo. Neste caso será adicionado mais uma camada escondida contendo 294 neurônios e função de ativação sigmoid.

```
classifier.add(Dense(294, activation='sigmoid'))
```

Após essas definições basta executar o script com o comando abaixo. Ao final do treinamento os dados de testes são exibidos na tela.

```
python MLP.py
```

Os scripts para execução da ConvNet estão disponíveis através do link https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10. Para definir os parâmetros utilizados será necessária uma pequena alteração no código. Na função *inference* contida no arquivo cifar10.py é feita a definição da arquitetura utilizada. As camadas estão nomeadas em conv1, pool1, norm1, conv2, norm2 e pool2. O tamanho do filtro de convolução pode ser alterado na linha contento:

```
kernel = _variable_with_weight_decay('weights',
                                     shape=[5, 5, 3, 64],
                                     stddev=5e-2,
                                     wd=0.0)
```

O parâmetro *shape* é um vetor contendo 4 números. Os dois primeiros números definem o tamanho do filtro convolucional, seguido pela profundidade de

saída e por final a profundidade de entrada. Somente os dois primeiros números foram variados nesse trabalho.

Para variar o tamanho do filtro de *pooling* basta modificar a linha abaixo. O tamanho do filtro é definido pelo segundo e terceiro número do parâmetro ksize.

```
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                       padding='SAME', name='pool1')
```

Após a definição dos parâmetros, o treinamento da rede é realizado através do script cifar10_train.py ou cifar10_multi_gpu_train.py. O primeiro script irá executar somente utilizando a CPU, caso o computador utilizado possua GPU é recomendado o uso do segundo script. Como parâmetro para os dois scripts é necessário passar a pasta onde está localizado o arquivo de treinamento obtido nos passos anteriores.

```
python cifar10_train.py --train_dir localização_do_arquivo_de_treinamento
ou
python cifar10_ multi_gpu_train.py --train_dir
localização_do_arquivo_de_treinamento --num_gpus número_de_gpus_disponível
```

O treinamento será armazenado de tempos em tempos na pasta "*train*". Para ver os resultados do modelo treinado até o momento basta executar o script cifar10_eval.py. Um parâmetro necessário para esse script é o diretório onde está o arquivo de testes, também obtido nas etapas anteriores.

```
python cifar10_eval.py --eval_dir localização_do_arquivo_de_teste
```