UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

MARCIO ROBERTO SIMÕES JÚNIOR

Redes neurais convolucionais e máquinas de vetores de suporte para classificação de imagens médicas

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. Dr. João Batista Florindo.

Campinas
2019

1. Introdução

As redes neurais artificiais são algoritmos que vêm se destacando dentro da área de aprendizado de máquinas, área esta que explodiu na última década. Vê-se hoje cada vez mais a interação computador-humano no dia-a-dia por meio por exemplo de *softwares* de detecção de spam, sistemas de recomendação, reconhecimento de voz, escrita e de imagens, o qual será melhor abordado aqui.

O reconhecimento de imagens nos dias atuais é feito principalmente por meio de redes neurais artificiais. Variadas arquiteturas são usadas a fim de lidar com os mais diferentes tipos e volumes de dados. Será abordado aqui o que são e como as redes neurais convolucionais funcionam a partir da referência [1] que foi utilizada para esse estudo, e também, diferentes técnicas a fim de realizar a classificação de imagens médicas.

2. Redes Neurais

2.1. História

A ideia de ensinar um computador a aprender não é nova, as primeiras redes neurais existem desde a década de 1950. Porém, apesar de arquiteturas cada vez mais complexas de redes neurais terem sido estudadas nesse meio tempo, durante vários anos e até bem recentemente ainda faltava muito para que os primeiros modelos funcionassem e para que esses algoritmos impactassem a vida do usuário final como hoje tanto impacta. Alguns elementos importantes faltavam para que isso acontecesse, o primeiro é óbvio, os computadores ainda eram pouco acessíveis e não se imaginava que eles tomariam tanto conta da vida cotidiana. Outros dois elementos vieram quase que ao mesmo tempo: o *Big Data* e as placas gráficas (GPUs).

Esses algoritmos dependem de uma etapa de treinamento para funcionar eficientemente e para isso necessita de um grande volume de dados, gerado em variedade e em velocidade cada vez maiores e hoje o mundo está certamente cada vez mais cercado de tudo isso vivendo na chamada era do *Big Data*. Mas ainda faltava algo, como lidar com toda essa quantidade de dados? Com o avanço da processamento paralelo por meio das GPUs, o aprendizado de máquina e o uso das redes neurais explodiu, permitindo as mais diferentes aplicações.

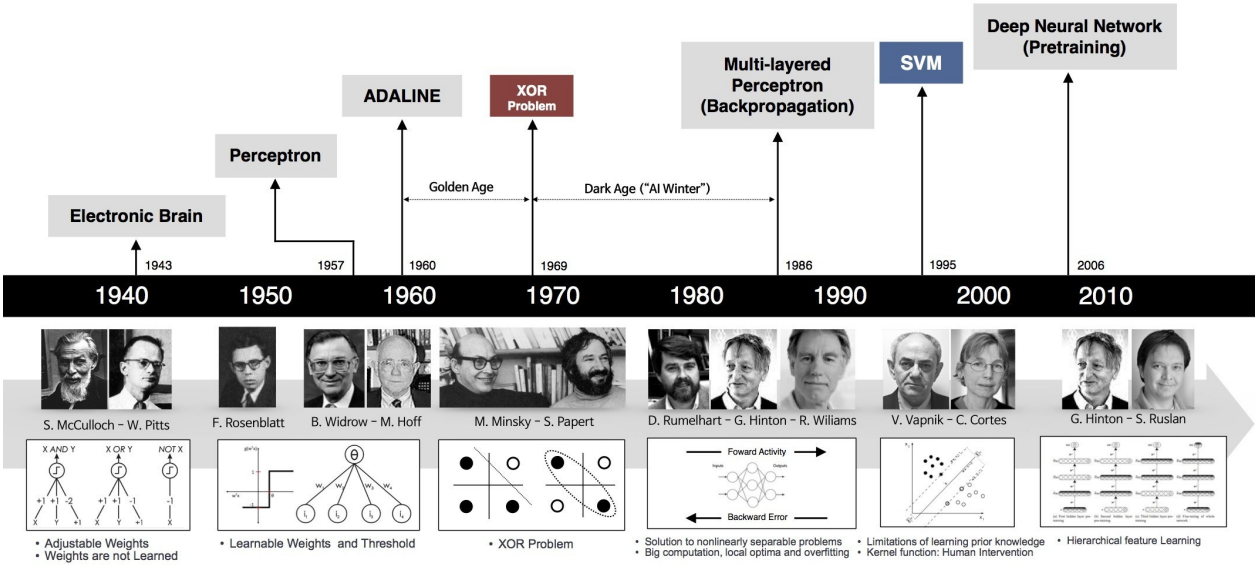

Figura 1: Linha do tempo das redes neurais

Acima pode-se ver a linha do tempo de evolução dos algoritmos de redes neurais, que vai desde o surgimento dos operadores lógicos, *perceptron* – o primeiro modelo a utilizar a ideia de um neurônio matemático – até ao

que se chama hoje de *deep learning* ou aprendizado profundo – que explora cada vez mais a ideia de múltiplas camadas de uma rede neural.

2.2. O Perceptron

As redes neurais artificiais vieram com a ideia de imitar o funcionamento do nosso cérebro no processamento de informação. Na biologia, um neurônio é a unidade básica de um cérebro humano e sua principal função é conduzir impulsos nervosos. Nosso cérebro possui bilhões de neurônios e uma caracteristica importante é que esses neurônios estão conectados entre si, formando também centenas de bilhões de conexões ou em outras palavras uma rede neural. Por essas conexões passam impulsos nervosos e cada neurônio tem a capacidade de controlar a intensidade do impulso que ele irá passar adiante. Assim, o processamento de uma informação (e liberação de estímulos) é realizada em consequência de uma série de combinações de impulsos.

Pensando nessa estrutura biológica, Warren McCulloch e Walter Pitts propuseram em 1943 a estrutura de um neurônio matemático. O neurônio matemático, primeiramente, recebe um impulso x_i de entrada. Um neurônio possui por si a capacidade de controlar a intensidade do impulso que ele passará para frente. No neurônio matemático esse controle é feito por meio dos pesos sinápticos w_i. Quanto maior o peso, mais intenso será o impulso e quanto menor, menos intenso será.

E finalmente chega-se então no Perceptron, desenvolvido por Frank Rosenblatt entre as décadas de 1950 e 1960 e baseado na estrutura do neurônio matemático de Warren McCulloch e Walter Pitts. É o modelo mais básico de uma rede neural, em que dadas as entradas $x_1, x_2, ..., x_n$ é obtida uma única saída.

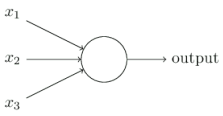

Figura 2: Modelo do perceptron

No exemplo anterior, existem três entradas x_1, x_2, x_3. Rosenblatt introduziu no modelo a ideia dos pesos a fim de expressar a importância de cada entrada para a saída. A saída 0 ou 1 (verdadeiro ou falso) é dada a partir da soma $\sum w_i x_i$, sendo que dado um limiar $-b$, que será chamado mais para frente de *bias*, ocorrerá a ativação ou não do neurônio. Em outras palavras:

$$y = \begin{cases} 1, & \text{se } \sum w_i x_i \geq -b \\ 0, & \text{se } \sum w_i x_i < -b \end{cases}$$

O Perceptron é um dos modelos mais básicos de rede neural. Arquiteturas combinadas e mais complexas desse modelo são capazes de pesar cada vez mais evidências e portanto de tomar melhores decisões de saída. Ao lado são apresentadas algumas das principais arquiteturas de redes neurais que existem hoje. Outra característica que torna essas estruturas mais eficientes é o número de camadas que ela possui, o que justamente dá caminho para o estudo do aprendizado profundo.

Um aplicação importante dos Perceptrons é o seu uso para simular os circuitos lógicos AND, OR e NAND, o que faz com que sejam universais para a computação. Por exemplo, com a rede abaixo é possível simular o circuito lógico NAND, uma vez que seja:

- $x_1 = 0, x_2 = 0$ temos que $-2 \cdot 0 + (-2) \cdot 0 + 3 = 3 \geq 0$ e portanto $y = 1$
- $x_1 = 1, x_2 = 0$ temos que $-2 \cdot 1 + (-2) \cdot 0 + 3 = 1 \geq 0$ e portanto $y = 1$

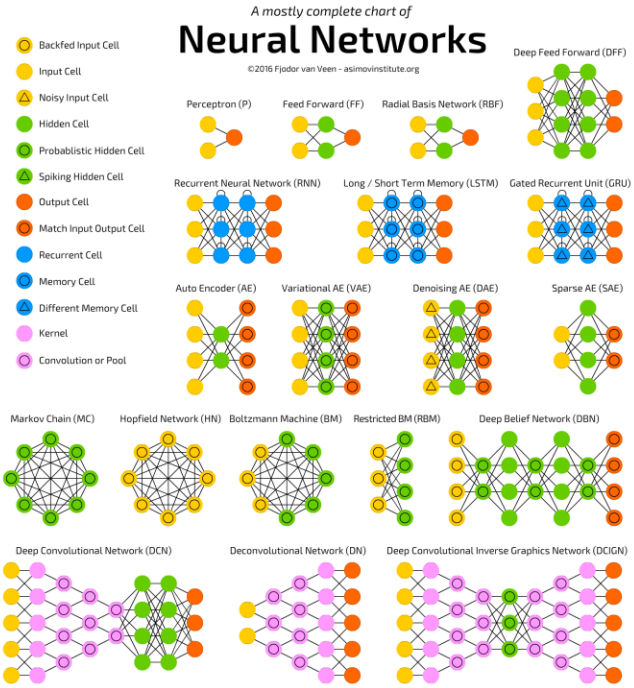

Figura 3: Diferentes arquiteturas de redes neurais

- $x_1 = 0, x_2 = 1$ temos que $-2 \cdot 0 + (-2) \cdot 1 + 3 = 1 \geq 0$ e portanto $y = 1$
- $x_1 = 1, x_2 = 1$ temos que $-2 \cdot 1 + (-2) \cdot 1 + 3 = -1 < 0$ e portanto $y = 0$

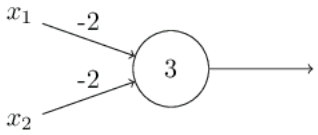

Figura 4: Perceptron simulando circuito NAND.

2.3. Pesos, *bias* e a função de ativação

Mas como então são determinados os pesos e os *bias*? O processo de treinamento de uma rede neural faz com que a rede aprenda os valores ideais de pesos e *biases* com base em entradas e saídas já conhecidas. É por esse motivo que uma grande quantidade de dados é essencial para melhor eficiência de uma rede neural.

Mas ainda falta um elemento importante que faz com que as redes neurais executem tarefas mais complexas e não se comporte apenas como uma regressão linear: as funções de ativação. Essas funções são de grande importância, pois possibilita que pequenas mudanças nos pesos e *biases* cause apenas uma pequena mudança na saída. Assim, aplicando a função de ativação σ a seguinte saída é obtida:

$$y = \sigma(w_i x_i + b),$$

Essas funções são transformações não lineares aplicadas às saídas de cada neurônio. As principais funções de ativação utilizadas hoje são:

Binária:

$$f(x) = \begin{cases} 1, & \text{se } x \geq 0 \\ 0, & \text{se } x < 0 \end{cases}$$

Sigmoide:

$$f(x) = \frac{1}{1 + e^{-x}}$$

2.4. Rede neural para reconhecimento de dígitos

Será construído agora um modelo de rede neural básico para realizar o reconhecimento de dígitos. Para isso, será utilizado o *dataset* MNIST [2] e a seguinte arquitetura:

Para o aprendizado dos pesos e *biases*, será introduzida uma função de custo, cujo o objetivo é quantificar o quão bem a rede está em relação prever as saídas já conhecidas da etapa de treinamento:

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - y^*\|^2,$$

em que y^* é o vetor com as saídas já conhecidas dado x de entrada. Esse custo é também chamado de erro quadrático médio e o objetivo aqui é que $C(w, b)$ se aproxime de zero de modo que $y(x)$ esteja próximo das saídas conhecidas, ou seja, de modo que a rede faça as melhores previsões. Dada então a função objetivo, utiliza-se o método do gradiente descendente para se encontrar pesos e *biases* ideais e inicialmente atribuem-se valores aleatórios para os mesmos.

O método do gradiente consiste basicamente de, uma vez que o gradiente de uma função é sabido apontar para a direção de maior crescimento dessa função, então utilizar-se o gradiente como direção de descida de uma busca linear. Ou seja, para cada iteração de treinamento, os pesos e *biases* são atualizados da seguinte forma:

$$w_i \to w_i' = w_i - \eta \frac{\partial C}{\partial w_i}$$

$$b \to b = b' - \eta \frac{\partial C}{\partial b},$$

em que η é o chamado coeficiente de aprendizado.

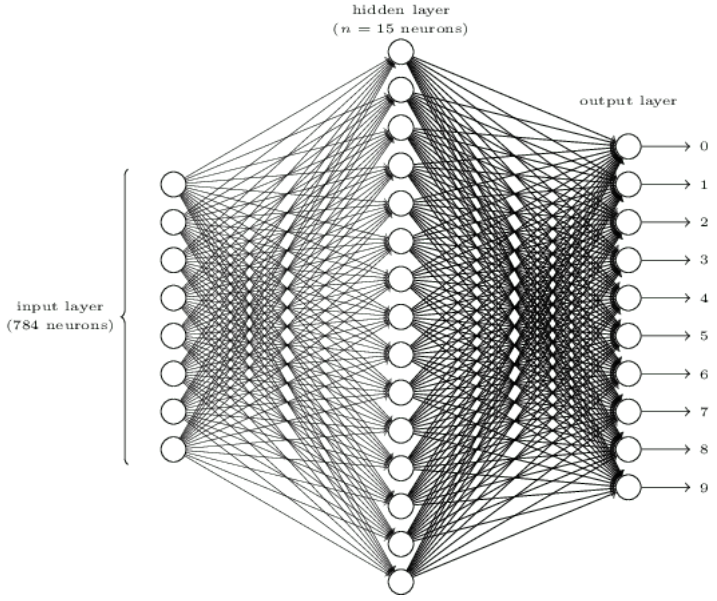

Figura 5: Arquitetura para a rede de conhecimento de dígitos.

2.5. Algoritmo do Backpropagation

Porém, para esse passo ainda falta um conceito importante e que com certeza é a essência do treinamento de redes neurais, que é o algoritmo do *backpropation*. O *backpropation* é um dos algoritmos-chave de uma rede neural. Sem ele o processo de treinamento seria uma tarefa bastante complicada dada a disponibilidade computacional que existe hoje.

Imagine qual seria o custo computacional de se calcular o gradiente da função custo para cada neurônio envolvido e em cada iteração? Para contornar isso foi desenvolvido o algoritmo do *backpropagation*, que basicamente retropropaga os valores das derivadas usando Programação Dinâmica a fim de calcular os gradientes em cada passo e evitando a complicação do cálculo de muitas derivadas.

2.6. Resultado final

O algoritmo da rede neural para reconhecimento de dígitos foi implementado no Jupyter Notebook utilizando a linguagem Python e pode ser encontrado neste link.

Para esse exemplo, foi utilizado $\eta = 3.0$. η é o que se chama de um hiperparâmetro e seu valor ideal varia de acordo com a aplicação. Os valores iniciais dos pesos e *biases* foram escolhidos aleatoriamente segundo uma distribuição Gaussiana de média 0 e desvio 1, uma das distribuições mais importantes da Estatística e que aparece muito na natureza.

Além disso, do conjunto de 60000 imagens do dataset MNIST, 50000 foram utilizadas para o treinamento da rede e 10000 foram utilizadas para validação. O treinamento também foi separado em épocas, que nada mais são do que as passagens completas do conjunto de dados, que devem ser realizadas nos chamados *mini-batchs*. Estes consistem na divisão dos dados em mini-lotes de modo a não usar todos os dados em todas as épocas. Para o exemplo dos dígitos, foram executadas 30 épocas e o conjunto de dados foi dividido em mini-batchs de tamanho 10.

Nesse exemplo, após o treinamento, a rede neural conseguiu prever 94,86% dos dados corretamente.

3. Redes Neurais Convolucionais

3.1. Arquitetura

As redes neurais convolucionais são ótimos classificadores de imagens, pois diferentemente da que foi construído anteriormente, a arquitetura desse tipo de rede considera a estrutura espacial de uma imagem. E, portanto, acaba sendo uma das arquiteturas mais utilizadas para classificar imagens.

Uma rede neural convolucional é capaz de receber uma imagem como entrada e atribuir importância (intensidade dada pelos pesos) a diferentes características de uma imagem. De acordo com sua profundidade é capaz de aprender aspectos bastantes gerais, mostrando-se muito eficiente. Essa rede é inspirada no comportamento

do cérebro humano ao lidar com o campo visual, em que cada neurônio costuma responder a estímulos de regiões restritas da visão através do campo receptivo.

Abaixo está um exemplo de entrada de uma rede neural convolucional, chamada de Campo Receptivo:

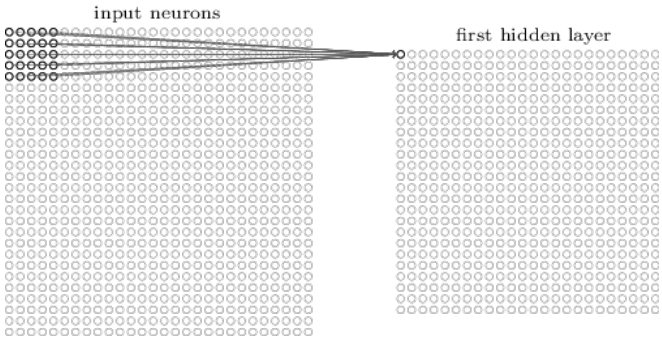

Figura 6: Campo Receptivo de uma rede convolucional.

Como mostra a imagem, um conjunto de neurônios de entrada do Campo Receptivo, chamado de Campo Receptivo Local, é interpretado por cada neurônio da próxima camada (que pode também ser dividida em vários canais, como mostra a Figura 8). E o mesmo processo é realizado, respectivamente, com os próximos campos receptivos locais que são dados apenas se deslocando para os lados.

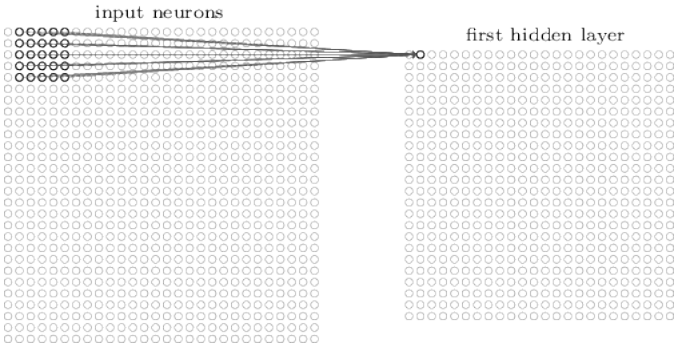

Figura 7: Campo Receptivo Local.

O grande diferencial dessa rede é principalmente o fato de que os pesos e *biases* atribuídos aos campos receptivos locais são compartilhados. É justamente isso que faz com que cada camada aprenda diferentes características da imagem e faz com que o processamento da rede seja bastante eficiente, possibilitando assim trabalhar com muitas camadas. Ou seja, a saída da camada oculta exemplificada acima é tal que:

$$y_{j,k} = \sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m}x_{j+l,k+m}\right)$$

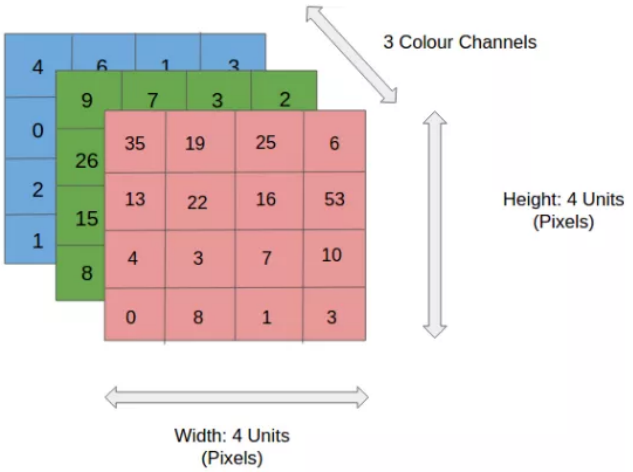

Figura 8: Canais de uma rede convolucional.

Existem ainda outros dois tipos de camadas. A Camada de *Pooling* que recebe a saída de uma outra camada e funciona exatamente como a Camada Receptiva, porém possuí o objetivo de abstrair ainda mais caracteristicas gerais da imagem e de condensar os neurônios. Por fim, tem-se a camada de saída da rede na qual usualmente todos os neurônios estão conectados entre si.

3.2. Rede Convolucional para reconhecimento de digitos

Usa-se agora uma rede neural convolucional para resolver o mesmo problema de reconhecimento de dígitos de antes. Para isso, a seguinte arquitetura será usada:

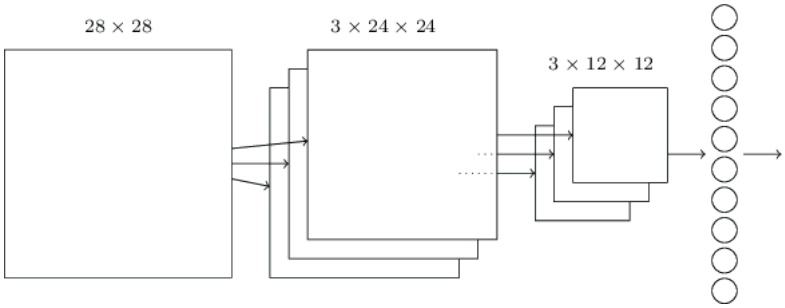

Figura 9: Arquitetura da rede convolucional de conhecimento de dígitos.

Para a implementação do algoritmo, também se usou o Jupyter Notebook e a linguagem Python junto com a biblioteca Theano, na qual já se encontra a arquitetura convolucional pronta. Para o treinamento, foram realizadas 60 épocas, utilizando-se $\eta = 0.1$ e *mini-batchs* de tamanho 10, obtendo-se assim uma incrível taxa de acerto de 97,70%. O código pode ser encontrado neste link.

Nesse mesmo código, implementa-se também uma outra arquitetura um pouco mais profunda, que obteve uma taxa de 98.81%. Ou seja, os resultados são cada vez melhores explorando-se as redes neurais convolucionais.

3.3. *Transfer Learning*

Poucas vezes um conjunto de dados é suficientemente grande para treinar uma rede convolucional. Logo, dificilmente uma rede desse tipo é treinada inteira a partir do zero. Uma técnica que acaba, portanto, sendo muito usada a fim de realizar treinamentos mais rápidos e eficientes é o *Transfer Learning*.

No *Transfer Learning* uma parte da rede é treinada a partir de um pré-treinamento. Isso é possível a partir da capacidade de uma rede convolucional de abstrair características generalistas de uma imagem em cada camada. Para esse pré-treinamento pode-se usar, por exemplo, um outro *dataset* com um conjunto de imagens maior (como a ImageNet) ou pode-se fixar uma parte dos pesos e treinar-se apenas uma outra parte antecipadamente. Assim, o treinamento é inicializado com parte dos pesos e *biases* já pré-treinados e não aleatórios.

Nesse código aqui, implementa-se o *Transfer Learning* na linguagem Python e utilizando-se a biblioteca Pytorch, seguindo o tutorial da própria ferramenta [3], que já possui um modelo pronto. Foram implementados os dois caminhos citados acima e o objetivo era treinar uma rede capaz de classificar imagens de formigas e abelhas a partir de um *dataset* bem pequeno com 120 imagens. Foi obtida, respectivamente, uma taxa de acerto de 94,77% e tempo de 17m41s e 96,07% e 45m54s. Esses códigos foram executados em uma CPU. Em uma GPU os tempos seriam menores ainda.

4. Maquinas de vetores de suporte e classificadores

4.1. Outra maneira de reconhecer imagens

Uma outra maneira de se realizar o reconhecimento de imagens, quando se quer grupá-las em classes já conhecidas, é por meio do uso das máquinas de vetores de suporte (SVM). A ideia aqui é ao invés de se utilizar uma rede neural para abstrair características de um conjunto de images, certos descritores quantitativos desse mesmo conjunto são extraídos e técnicas já conhecidas de *machine learning* são utilizadas para o treinamento de um modelo de classificador.

Esses descritores podem estar relacionados, por exemplo, com a cor, a textura ou a forma da imagem. Essas características são as principais e as mais utilizadas para classificação de uma imagem e pode-se ainda combinar esses descritores de forma a aumentar a eficiência da classificação. Alguns resultados quantitativos que podem ser obtidos a partir dessas informações são:

- **Cor:** Estatísticas do canal de cor (média, desvio padrão, etc.) e histograma de cores
- **Forma:** Momentos Hu [4], Momentos Zernike [5], etc.
- **Textura:** Descritor de Haralick [6] , Padrões Binários Locais [7], etc.
- **Outros:** Histograma de gradientes orientados, estatísticas de adjuvância de limite, etc.

4.2. Classificando imagens médicas

Essa ideia será utilizada para realizar a classificação de imagens médicas. Para tanto, será utilizado um *dataset* que possuí 1300 imagens pré-classificadas em 3 classes diferentes. Os descritores que serão utilizados serão: Cor, quantificada através do histograma de cores; Textura, quantificada através da Textura Heralick e a Forma da imagem, quantificada pelos Momentos de Hu.

O código de implementação será divido em duas etapas, na primeira etapa serão extraídos os quantificadores do conjunto de imagens. Esses quantificadores serão salvos em forma de matriz em um arquivo no formato HDF5. Na segunda etapa, serão aplicados vários modelos de classificação por máquinas de vetores de suporte conhecidos em *machine learning* sob esses quantificadores. O conjunto total de dados será dividido meio a meio para treinamento e validação a fim de obter o melhor modelo de previsão de quantificadores dada uma entrada.

Para a implementação, utiliza-se mais uma vez o Jupyter Notebook e a linguagem Python, junto com a biblioteca Scikit-learn, que já possui vários modelos de *machine learning* implementados e a biblioteca OpenCV-Python, utilizada para a extração dos quantificadores. O código completo se encontra aqui. Os classificadores utilizados foram: Regressão Logística (LR), Análise Discriminante Linear (LDA), K-ésimo Vizinho mais Próximo (KNN), Árvore de Decisão (CART), Florestas Aleatórias (RF), Gaussian Naive Bayes (NB) e SVC. O desempenho de cada um pode ser conferido abaixo:

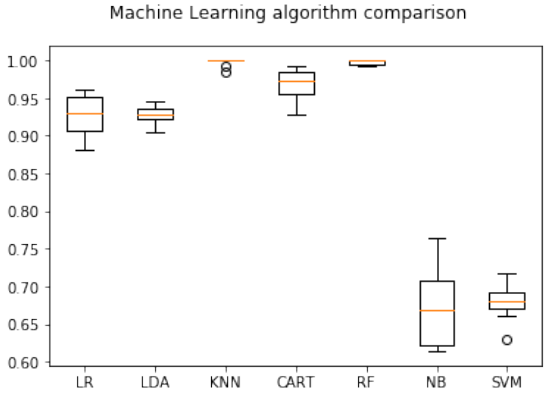

Figura 10: Comparação dos modelos de classificação.

Observa-se então que o modelo de Florestas Aleatórias (RF) teve melhor desempenho (99,76% de acerto). Fez-se então, no fim do código, a implementação direta desse método como teste, a fim de realizar o reconhecimento de 6 imagens do conjunto de imagens (selecionadas previamente) e o modelo proposto reconheceu corretamente todas elas.

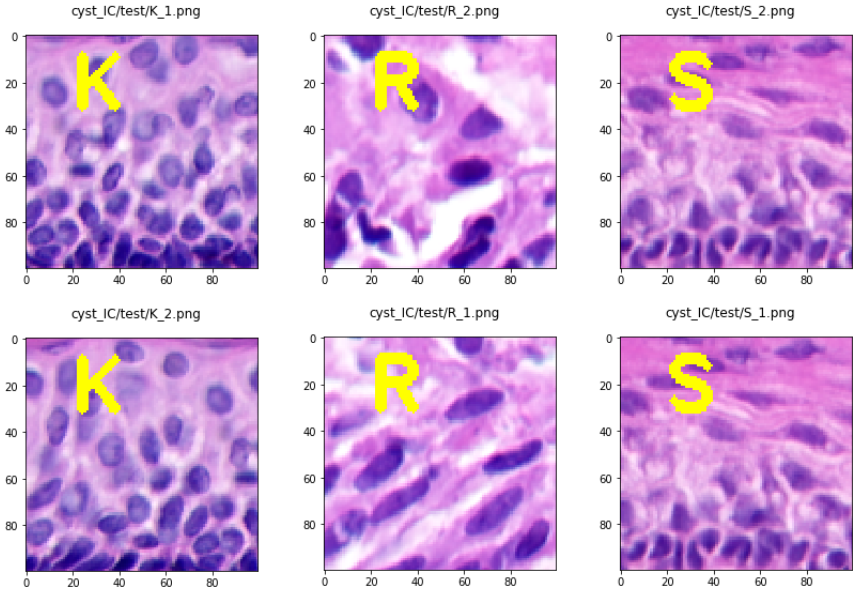

Figura 11: Testes em imagens médicas.

5. Conclusão

Durante esse projeto, foram estudados diferentes algoritmos de aprendizado que realizam o reconhecimento de imagens. Primeiramente, foi explorado o funcionamento das redes neurais, que surgiram com a ideia de realizar tarefas mais complexas do que as realizadas por algoritmos de regressão.

Para o objetivo principal deste trabalho, que era realizar o reconhecimento de imagens em si, foram exploradas as redes neurais convolucionais, um tipo de arquitetura de rede neurais voltada para o reconhecimento de imagens. Notou-se que um dos desafios desses algoritmos de aprendizado é o processamento e volume de dados. Para isso, foram exploradas algumas técnicas que ajudam a minimizar esses desafios, como o *Transfer Learning*.

E por fim, foi explorada uma outra técnica capaz de usar as máquinas de vetores de suporte junto a técnicas já conhecidas de *Machine Learning*, formando assim um classificador para realizar a discriminação de imagens médicas, e observou-se que essa técnica também é capaz de obter bons resultados.

Referências

[1] M. Nielsen, Neural networks and deep learning.
URL `http://neuralnetworksanddeeplearning.com/`

[2] C. C. Yann LeCun, C. Burges, Mnist handwritten digit database.
URL `http://yann.lecun.com/exdb/mnist/`

[3] S. Chilamkurthy, Transfer learning for computer vision tutorial.
URL `https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html`

[4] Hu moments.
URL `https://en.wikipedia.org/wiki/Image_moment`

[5] Zernike moments.
URL `https://en.wikipedia.org/wiki/Zernike_polynomials`

[6] Haralick texture.
URL `http://haralick.org/journals/TexturalFeatures.pdf`

[7] Local binary patterns.
URL `https://en.wikipedia.org/wiki/Local_binary_patterns`