Universidade Federal de Santa Catarina

A Identificação de Dígitos Manuscritos

INE5430 - Inteligência Artificial

Alunos
Andrei Donati
Ígor Assis Rocha Yamamoto
Luis Felipe Pelison

Professor
Mauro Roisenberg

Novembro de 2017

Sumário

1 Introdução

O presente trabalho faz parte da avaliação do módulo sobre aprendizado de máquina da disciplina de Inteligência Artificial e tem como objetivo apresentar a implementação de uma rede neural artificial que realiza a identificação de dígitos manuscritos.

A rede neural a ser implementada é uma *Multi-Layer Perceptron* (MLP) que utiliza o algoritmo de *backpropagation* para realizar o treinamento dos pesos da rede. O modelo implementado é capaz de realizar o reconhecimento de padrões de modo a atribuir corretamente uma classe para um conjunto de padrões de entrada. Quando a rede neural está treinada, ela é usada para classificar novos padrões.

Neste trabalho, um conjunto de dados deve ser usado para treinar e testar uma rede neural, que classifica dígitos manuscritos. Os dados para treinamento, validação e teste foram obtidos a partir de um subconjunto de dados da base de dados MNIST [1]. Este trabalho também apresenta como contribuição, além da implementação de uma rede neural artificial, a criação de uma interface gráfica para testes da classificação de dígitos manuscritos.

2 Ferramentas de Implementação

A implementação da rede neural artificial utilizou *Python 3* como linguagem de programação. Para a rápida prototipação e testes da rede foi utilizado o *Jupyter Notebook* [2]. Para o tratamento de dados e computação matricial eficiente foi utilizado a biblioteca *Numpy* [3]. Para a construção, treinamento e testes da rede foi utilizado a biblioteca *Keras* [4]. Para o cálculo de métricas de desempenho e visualização de resultados pela matriz de confusão foram utilizadas as bibliotecas *scikit-learn* [5], *matplotlib* [6] e *seaborn* [7]. Para a implementação da interface gráfica foi utilizada a biblioteca *PyQt4* [8].

3 Tratamento dos Dados

O conjunto de dados fornecido atende às seguintes especificações: os parâmetros de entrada para a rede são os valores de cada pixel, supondo que os números foram previamente normalizados e centralizados e colocados em uma matriz de 20x20 (400 pixeis em escalas de cinza). A saída é o valor correto do dígito, sendo que o dígito 0 corresponde ao valor 10 (durante a implementação, por conveniência atribuiu-se o valor 0 à classe do dígito 0). A Figura 1 ilustra uma amostra dos dados de entrada.

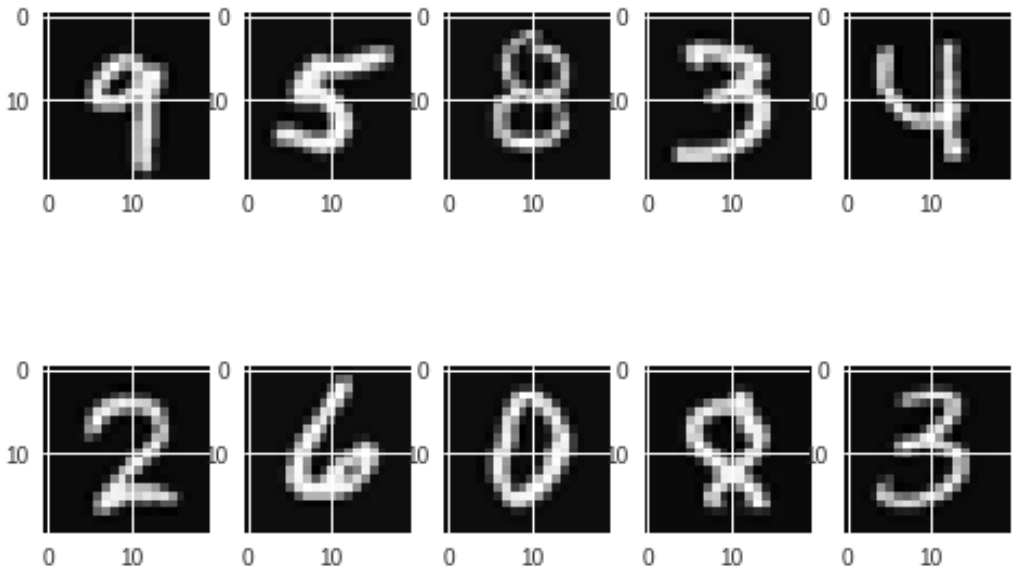

Figura 1: Exemplo de Dados de Entrada

Os dados foram carregados através da biblioteca *numpy* e foram feitas algumas visualizações dos dados para entendimento do conjunto. Observou-se que os dados já estavam centralizados e o uso de uma função de normalização, como o *fit_transform* da classe *StandardScaler* da biblioteca *scikit-leran*, só tende a piorar a qualidade do conjunto de dados.

Em relação aos dados de saída, a normalização é feita utilizando o *one hot encoder*, que transforma o valor do target em um vetor de zeros e valor 1 no indicie correspondente ao valor do target.

4 Arquitetura da Rede Neural

A rede neural implementada trata-se de uma *Multi-Layer Perceptron* (MLP). Esta configura-se na estrutura ilustrada na Figura 2. A arquitetura proposta pra a rede possui três camadas: entrada, escondida (*hidden layer*) e saída. A camada de entrada possui 400 neurônios já que deve receber os 20x20 pixeis da imagem do dígito manuscrito. A *hidden layer*, após uma série de tentativas por experimentação, possui 205 neurônios (uma média entre os neurônios de entrada e saída). Enquanto a camada de saída possui 10 neurônios, com cada neurônio representando uma das 10 classes de dígitos.

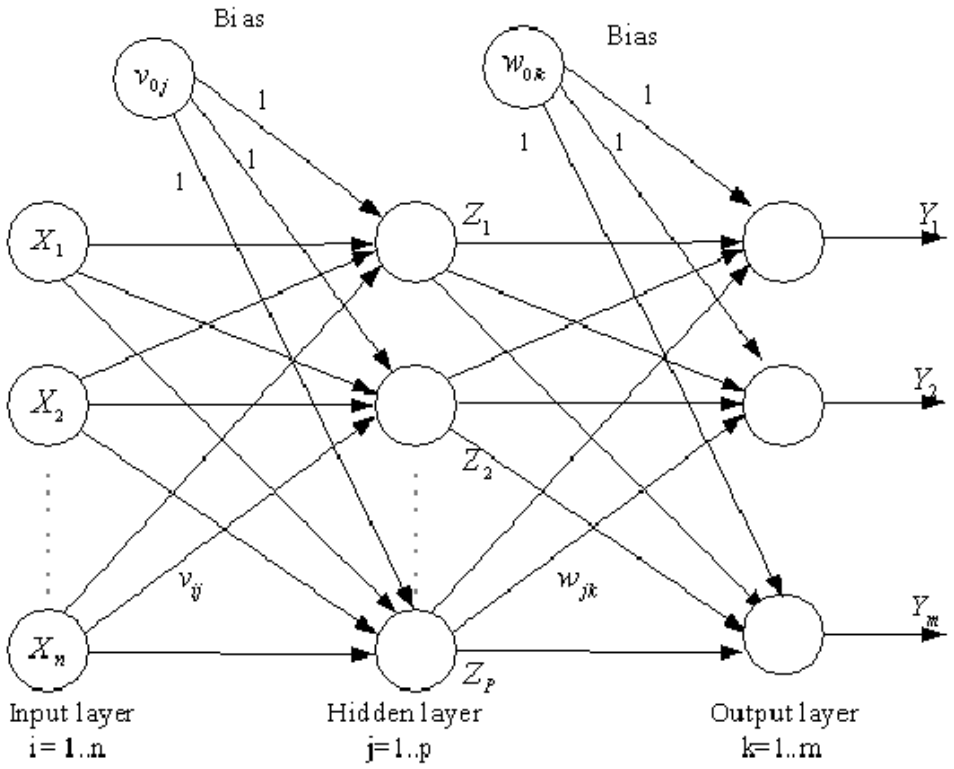

Figura 2: Estrutura de uma Rede Neural MLP

A função de ativação escolhida para a *hidden layer* foi a ReLU (*rectified linear unit*), que pode ser vista na Figura 3. A função possui algumas vantagens em relação a outras, como sigmoide. O gradiente da função é constante e não é zerada para valores grande do argumento, além disso tem a característica de promover matrizes esparsas por implementar a função *max*.

Quanto aos parâmetros de otimização da rede neural, optou-se por realizar um comparativo entre dois otimizadores: SGD (*Stochastic gradient descent optimizer*), que é uma aproximação estocástica para o método do gradiente descendente e resolve o problema de otimização de forma iterativa; e Adam, um otimizador eficiente computacionalmente e que foi o selecionado por apresentar melhores resultados. Como taxa de aprendizado, utilizou-se o valor de $0,01$. Com a escolha do otimizador SGD, ainda foi possível sintonizar os parâmetros de momento e *decay*. Enquanto que a escolha do otimizador Adam, permitiu a sintonia de parâmetros de aprendizado alfa e beta.

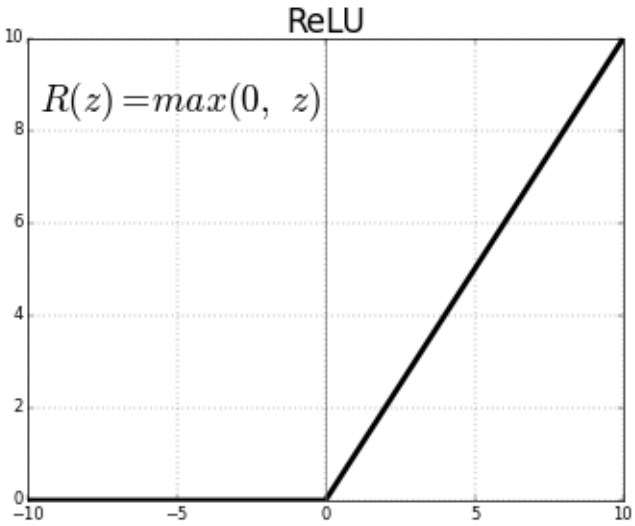

Figura 3: Função de Ativação ReLU

5 Treinamento da Rede Neural

Antes de iniciar o treinamento da rede, o conjunto de dados foi separado para fins de treinamento e validação. Determinou-se a separação de 15% do conjunto de dados (750 dados do total de 5000), a separação foi feita de forma aleatória para melhor distribuição dos dados.

A sessão de treinamento foi realizada diversas vezes para se obter a melhor sintonia nos valores de *batches* e épocas (*epochs*). Por final, escolheu-se fazer o treinamento utilizando o otimizador Adam, com 1000 épocas e com o número de amostras por atualização do gradiente de 150. A execução do treinamento foi realizada em uma CPU dual-core i5 com outros programas em execução, obtendo o tempo de execução de 164.72 segundos. Além disso a utilização da biblioteca *Keras*, que pode ser utilizada em cima do *tensorflow*, permitiu maior eficiência computacional com a utilização da GPU.

6 Resultados

Para mensurar os resultados do treinamento realizado na rede, foi feita a classificação primeiramente de todo o conjunto de dados e, em seguida, a classificação apenas dos dados separados para testes. Além disso, foi implementada uma interface gráfica onde o usuário pode criar novos dados a serem testados pela rede, que será detalhada na próxima secção.

Após diversas versões da rede para diferentes configurações de neurônios

na camada intermediária e ajuste de parâmetros de otimização, o desempenho da rede pode ser considerado muito satisfatório. Com parâmetros modestos e poucos neurônios, a rede facilmente ultrapassava os 80% de acurácia na classificação dos dígitos manuscritos. Conforme os parâmetros e número de neurônios foram sendo aumentados, percebeu-se uma estabilização no desempenho da rede, de forma que o aumento do número de neurônios, por exemplo, não compensava a melhora no resultado das classificações em comparação com o tempo médio de execução do treinamento.

Ao final, a sintonia pelo otimizador Adam atingiu a marca de $94,93\%$ de acurácia para os dados de teste e $99,24\%$ para todo o conjunto de dados. Enquanto o otimizador SGD obteve $93,73\%$ para dados de teste e $98,30\%$ para o conjunto total. Mais números do melhor resultado obtido podem ser vistos nas Figuras 4, 5, 6 e 7, que mostram a matriz de confusão para a predição dos dígitos e outras métricas utilizadas para validar a implementação.

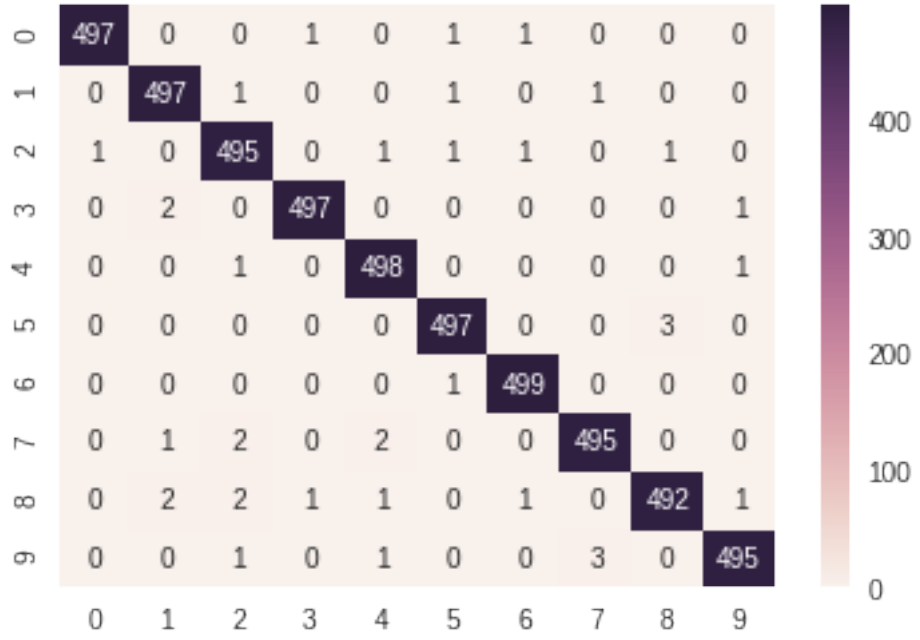

Figura 4: Matriz de Confusão para a Predição de Todo o Conjunto de Dados

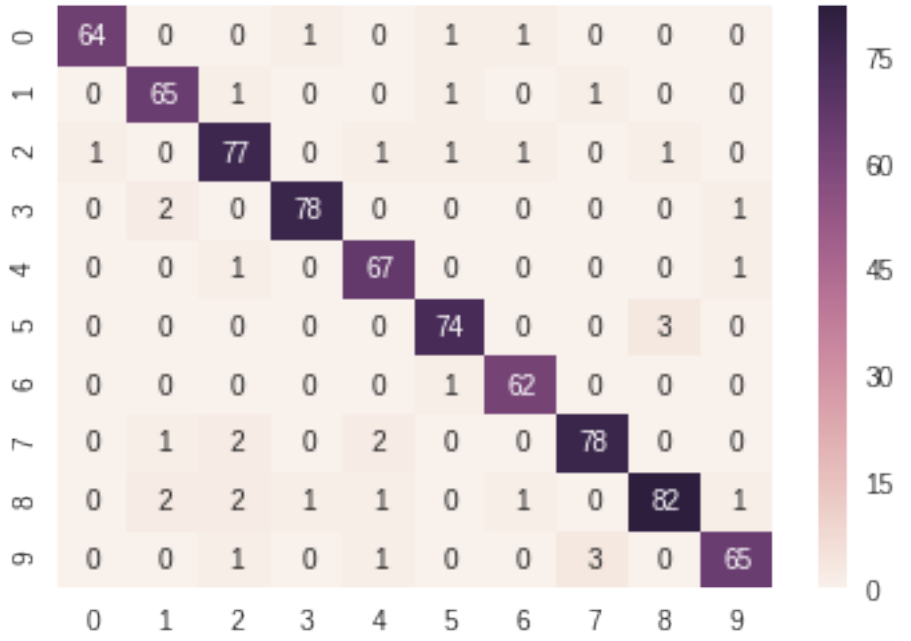

Figura 5: Matriz de Confusão para a Predição dos Dados Separados para Teste

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	500
1.0	1.00	1.00	1.00	500
2.0	0.99	0.98	0.98	500
3.0	0.99	0.99	0.99	500
4.0	0.99	0.99	0.99	500
5.0	0.99	0.99	0.99	500
6.0	0.99	0.99	0.99	500
7.0	0.99	0.99	0.99	500
8.0	0.98	0.99	0.99	500
9.0	0.99	0.99	0.99	500
avg / total	0.99	0.99	0.99	5000

Figura 6: Métricas para a Predição de Todo o Conjunto de Dados

	precision	recall	f1-score	support
0	0.98	0.96	0.97	67
1	0.93	0.96	0.94	68
2	0.92	0.94	0.93	82
3	0.97	0.96	0.97	81
4	0.93	0.97	0.95	69
5	0.95	0.96	0.95	77
6	0.95	0.98	0.97	63
7	0.95	0.94	0.95	83
8	0.95	0.91	0.93	90
9	0.96	0.93	0.94	70
avg / total	0.95	0.95	0.95	750

Figura 7: Métricas para a Predição dos Dados de Teste

Apesar do desempenho extremamente satisfatório, a rede neural apresentou alguns erros de classificação como era esperado, já que esta nunca atingirá uma generalização 100% correta. A Figura 8 mostra alguns casos onde a rede neural falhou em classificar corretamente os dígitos.

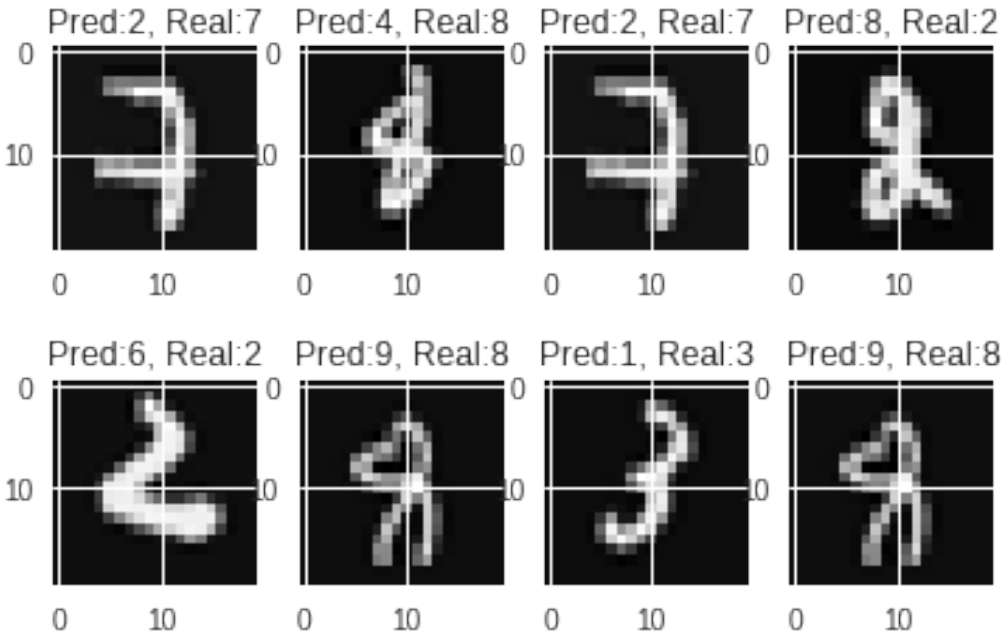

Figura 8: Erros de Classificação da Rede Neural

7 Implementação de uma Interface Gráfica

Com o intuito de validar a implementação da rede neural para a exposição a dados novos, nunca analisados a priori pela rede, foi implementada uma interface gráfica em Python que permite que o usuário crie seus próprios dígitos manuscritos para testar em cima da rede neural apresentada nas secções anteriores.

A interface gráfica foi baseada em projeto de software aberto disponível no *github* [9]. A Figura **??** ilustra o painel principal da interface gráfica. Ela permite que o usuário escolha uma cor dentro da palheta de cores, assim como configure a espessura do pincel e a suavidade das linhas do desenho. Ao final do desenho do dígito, o usuário deve usar o atalho "Ctrl + P"para chamar a função de predição, que utiliza o modelo treinado da rede neural para fazer a classificação da imagem, que antes deve ser convertida para a escala de cinza e reduzida para o tamanho de 20x20 pixeis.

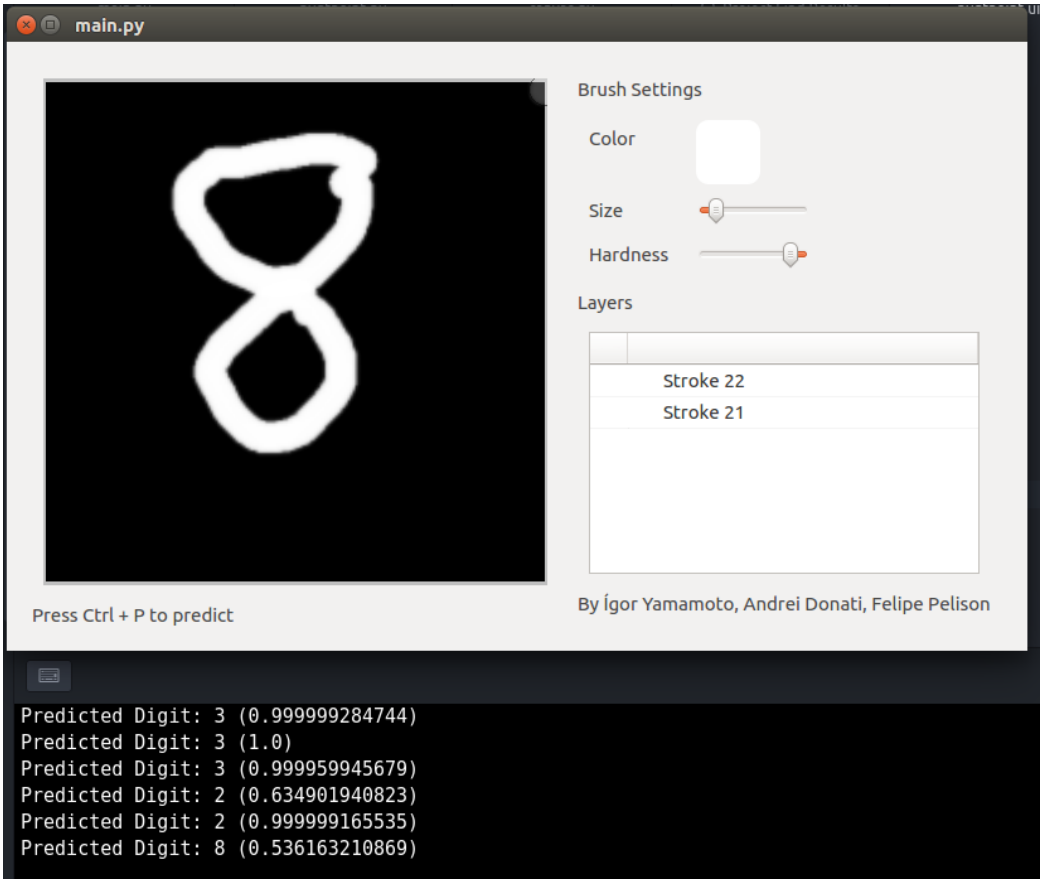

Figura 9: Interface Gráfica Desenvolvida para Validação da Rede Neural Implementada

A interface gráfica implementada é um bom ferramental para a validação da rede neural. A primeira versão (disponível em https://github.com/igoryamamoto/ine5430-neural-networks-mnist) já apresenta bons resultados, apesar de falhar em diversos casos pelo fato do quadro de desenho permitir que o usuário faça o desenho de forma descentralizada ou muito maior/menor que os desenhos encontrados na base de dados MNIST. Ainda assim, alguns resultados interessantes podem ser obtidos, conforme mostra a Figura 10

Figura 10: Classificação do Dígito 3 pela Interface Gráfica Sujeita a Ruídos no Desenho

8 Conclusão

O trabalho de implementação realizado serviu muito bem aos propósitos de aprendizado da disciplina e de incentivo ao interesse em realizar pesquisa sobre a temática aprendizado de máquina.

A respeito dos resultados da implementação da rede neural, verificou-se o grande po

Para trabalhos e pesquisas futuras, a rede neural pode ser aperfeiçoada e expandida para a classificação de outros caracteres, como letras do alfabeto. Outras implementações podem ser feitas com o uso de redes neurais profundas, em especial redes neurais convolucionais, já que as ferramentas utilizadas neste trabalho permitem sua fácil implementação de maneira muito similar a MLP implementada. A interface com o usuário também pode ser expandida e melhorada de modo a permitir a realização de novas baterias de treinamento com base nos novos dados que podem ser coletados.

Referências

[1] http://yann.lecun.com/exdb/mnist/ - Acessado em 23 de Novembro de 2017.

[2] http://jupyter.org/ - Acessado em 23 de Novembro de 2017.

[3] http://www.numpy.org/ - Acessado em 23 de Novembro de 2017.

[4] https://keras.io/ - Acessado em 23 de Novembro de 2017.

[5] http://scikit-learn.org/stable/ - Acessado em 24 de Novembro de 2017.

[6] https://matplotlib.org/ - Acessado em 24 de Novembro de 2017

[7] https://seaborn.pydata.org/ - Acessado em 24 de Novembro de 2017.

[8] https://www.riverbankcomputing.com/software/pyqt/download - Acessado em 24 de Novembro de 2017.

[9] https://github.com/dpebly/pyqt-paint - Acessado em 24 de Novembro de 2017.