Universidade Federal São João del Rei
Departamento de Ciência da Computação

Trabalho Prático 4: Documentação

Inteligência Artificial

Alunos: Elias de Paula Pereira,
Mariane Rodrigues Costa e
Julio Cesar da Silva Rodrigues

Professor: Diego Roberto Colombo Dias

Julho
2022

Conteúdo

1 Introdução

Neste documento, encontram-se os detalhes considerados mais relevantes no desenvolvimento de dois tipos de redes neurais (DNN e CNN) como modelos de aprendizado de máquina para o reconhecimento de dígitos manuscritos. A implementação foi realizada utilizando linguagem Python com auxílio dos recursos do *framework* **TensorFlow** e os *scripts* presentes na especifcação do trabalho prático, e está disponível publicamente no seguinte endereço `https://colab.research.google.com/github/juliorodrigues07/manuscript_digit_recognition/blob/master/cnn_digit.ipynb` no formato *notebook*, e também no repositório `https://github.com/juliorodrigues07/manuscript_digit_recognition`.

Antes de prosseguir, vale a pena citar que a DNN foi implementada apenas para fins de comparação e resolução das questões presentes no trabalho prático. Portanto, os testes realizados foram executados majoritariamente com a CNN, variando os hiperparâmetros de forma extensa, com o objetivo de encontrar uma composição que apresentasse a métrica de acurácia mais alta possível.

2 Decisões de Projeto

Para iniciarmos a análise do aprendizado do modelo, escolhemos uma abordagem bastante ingênua, mas bem simples. Foram executados testes variando os seguintes hiperparâmetros de forma independente:

- Tamanho do lote (*batch*)
- Número de épocas (*epoch*)
- Otimizador
- Função de ativação

Os experimentos iniciais foram realizados aplicando a variação de cada hiperparâmetro de forma separada, ou seja, testamos vários otimizadores enquanto os outros hiperparâmetros não foram modificados por exemplo, observando a acurácia obtida por cada um destes. Em seguida, combinamos os hiperparâmetros que obtiveram maior acurácia em testes independentes para compor o modelo de aprendizado final a ser apresentado.

2.1 Otimizadores

Dentre os otimizadores, os que apresentaram maior desempenho foram *Adam*, *Adamax* e *RMSprop*, enquanto outros como *SGD* e *Adadelta* apresentaram as menores acurácias. Vale citar brevemente que selecionamos como hiperparâmetro para analisar as taxas de perda do modelo a função *categorical_crossentropy*, devido à presença de vários rótulos na classe da base de dados.

2.2 Funções de Ativação

Dentre as funções de ativação, as que apresentaram maiores acurácias foram *ReLU*, *sigmoid* e *tanh*. Vale citar que apenas modificamos as funções de ativação para as camadas ocultas e de entrada, portanto a função de ativação para a camada de saída foi fixada para a *softmax* neste modelo. Como principal motivo para tal decisão, podemos justificar pela presença de vários rótulos distintos na classe (dígitos de 0 a 9), tornando a interpretação de uma distribuição de probabilidades desta função de ativação adequada na realização das predições.

2.3 Tamanho dos *Batches*

Para analisar as implicações que alterações no tamanho dos *batches* produzem na acurácia final, testamos proporções distintas, correspondentes à 0,1%, 1%, 10%, 50% e 100% do total de instâncias presentes na base de dados. Entretanto, observamos que a acurácia aferida descrescia de forma exponencial, e por isso, optamos por utilizar tamanhos de *batches* menores no modelo final, já que o tempo de execução não apresentou variações tão significativas. Por exemplo, para o tamanho de *batch* fixado em 0,1% (60), o tempo de execução decorrido foi de 7 minutos e 29 segundos, enquanto para 10% (6000), o tempo observado foi de 5 minutos e 30 segundos, apresentando uma queda significativa na acurácia aferida.

É importante citar que quando fixamos o tamanho de *batch* correspondente à totalidade do conjunto de dados (60000), não foi possível obter resultados devido à ultrapassagem do limite disponível de memória no ambiente de execução da ferramenta (Google Colab).

2.4 Número de Épocas

Embora tenhamos observado o padrão de aumento da acurácia obtida pelo modelo de aprendizado à medida que este é treinado ao longo de uma

quantidade elevada de épocas, decidimos por não explorar quantidades cuja ordem de grandeza supera as dezenas, devido ao acréscimo significativo no tempo de execução de treinamento do modelo.

3 Implementação

Discutindo agora os detalhes da implementação, o código possui a seguinte estrutura básica que define as etapas de formulação do modelo:

- Definição dos hiperparâmetros
- Pré-Processamento da base de dados
- DNN
- CNN
- Carregamento do modelo
- Classificação de exemplos reais

Para selecionar os hiperparâmetros do modelo final, utilizamos a estratégia citada anteriomente, a qual nos forneceu a combinação apresentanda na Tabela 1:

Tabela 1: Hiperparâmetros do modelo

Função de Ativação	Otimizador	Tamanho dos *Batches*	Número de Épocas
ReLU	Adam	60	4

Grande parte destas etapas foram construídas exatamente como se encontravam na especificação do trabalho prático, preenchendo apenas os campos correspondentes aos hiperparâmetros. Como novas funcionalidades relevantes, podemos citar a construção da matriz de confusão para checagem da distribuição dos dados, assim como as tendências de predições realizadas pelo modelo, se aplicáveis (classificando uma quantidade elevada do dígito 7 como dígito 1, por exemplo).

Além disso, foi implementado também um trecho de código com o intuito de testar o modelo de aprendizado construído com imagens reais de dígitos manuscritos. Embora os processos de manipulação e pré-processamento do conjunto pequeno de imagens tenha ocorrido sem maiores conflitos, as predições realizadas apresentadas não condizeram com as expectativas geradas pela acurácia aferida do modelo, apresentando um padrão de classificação errôneo em ambas redes neurais. Os detalhes desta etapa de desenvolvimento serão discutidos na seção seguinte.

4 Análise de Resultados

Com a combinação de hiperparâmetros citada anteriormente, conseguimos construir uma rede neural convolucional cuja acurácia aferida atingiu **99,15%**, com uma taxa de perda de **2,66%** (97,97% e 7,75% para a DNN, respectivamente). Os gráficos que correspondem às curvas de aprendizado dos modelos são apresentados nas Figuras 1 e 2, relacionando o número de épocas com a acurácia obtida.

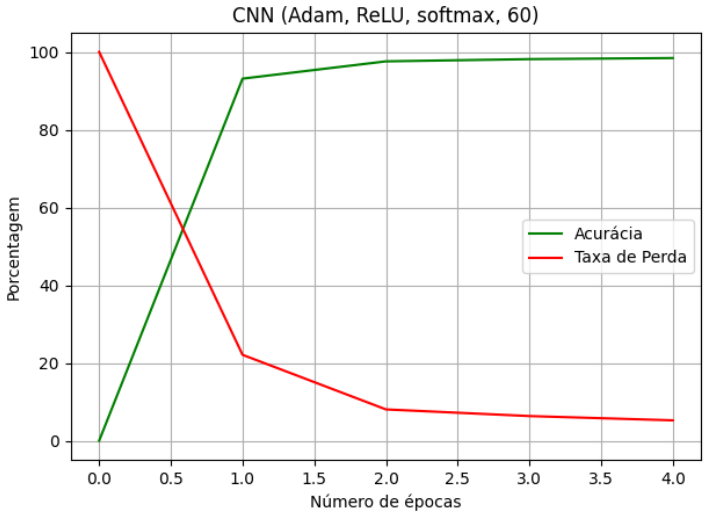

Figura 1: Curva de aprendizado da CNN

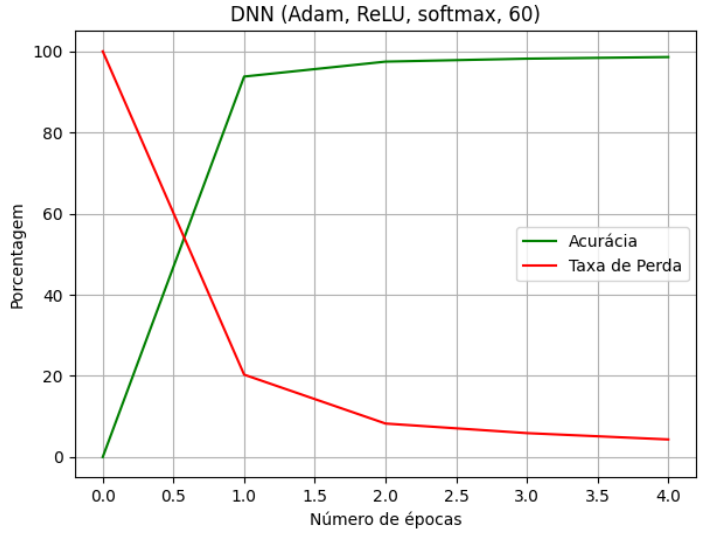

Figura 2: Curva de aprendizado da DNN

A matriz de confusão obtida pelo modelo é exibida na Figura 3.

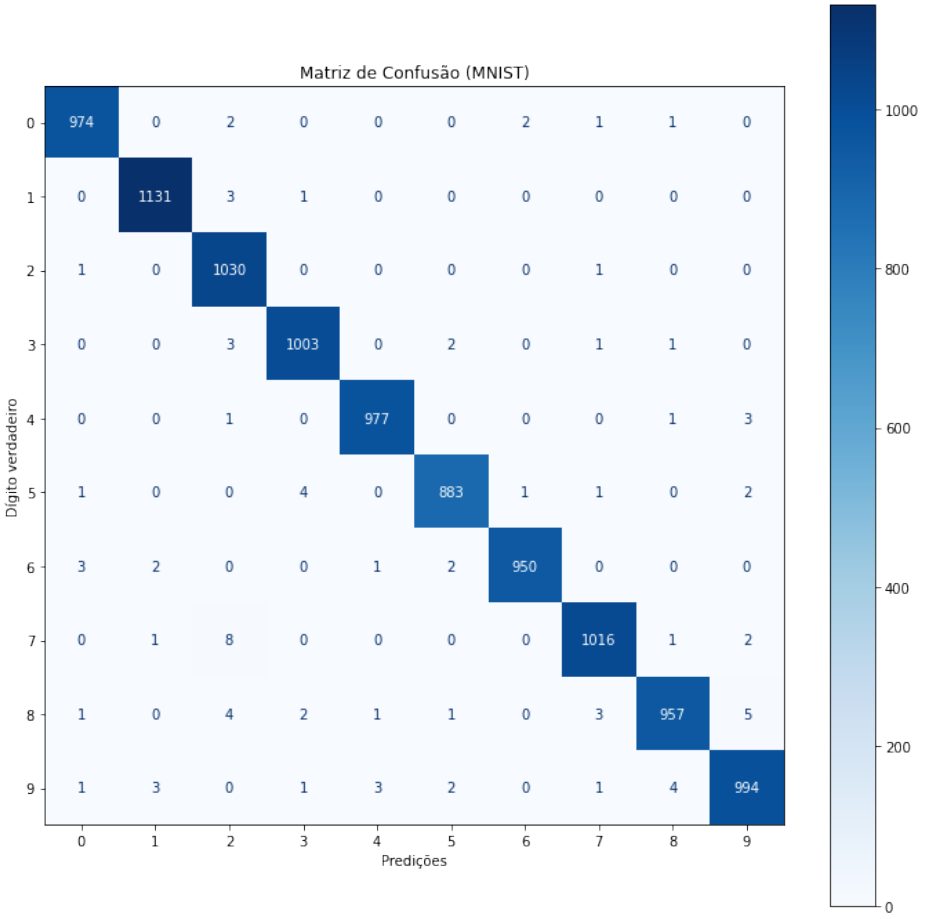

Figura 3: Matriz de confusão da CNN

Nós tentamos formular nosso próprio conjunto de teste, escrevendo dígitos à mão e manipulando suas imagens para adequá-las à entrada das redes neurais, para que estas pudessem executar as predições. No entanto, as predições observadas foram bastante distantes das esperadas, dada a acurácia que as redes apresentaram e o padrão observado em suas predições errôneas.

Em um conjunto pequeno com apenas 13 imagens (Figura 4), a CNN classificou a maioria como o dígito 3, embora a ocorrência do mesmo fosse única no conjunto. Na DNN, verificou-se o mesmo padrão, mas desta vez com o dígito 5, cuja ocorrência é verdadeira apenas em duas imagens. Os possíveis problema iniciais que pensamos que podem estar afetando as predições seriam os contornos dos dígitos, que não se apresentam de forma contígua e uniforme como na base de dados MNIST, além de ocuparem uma porção ligeiramente

menor na imagem de dimensão 28 x 28. Estes são fatores de padronização que podem estar levando a classificação incorreta por parte das redes neurais, mas as causas também podem ser outras que ainda não conseguimos analisar.

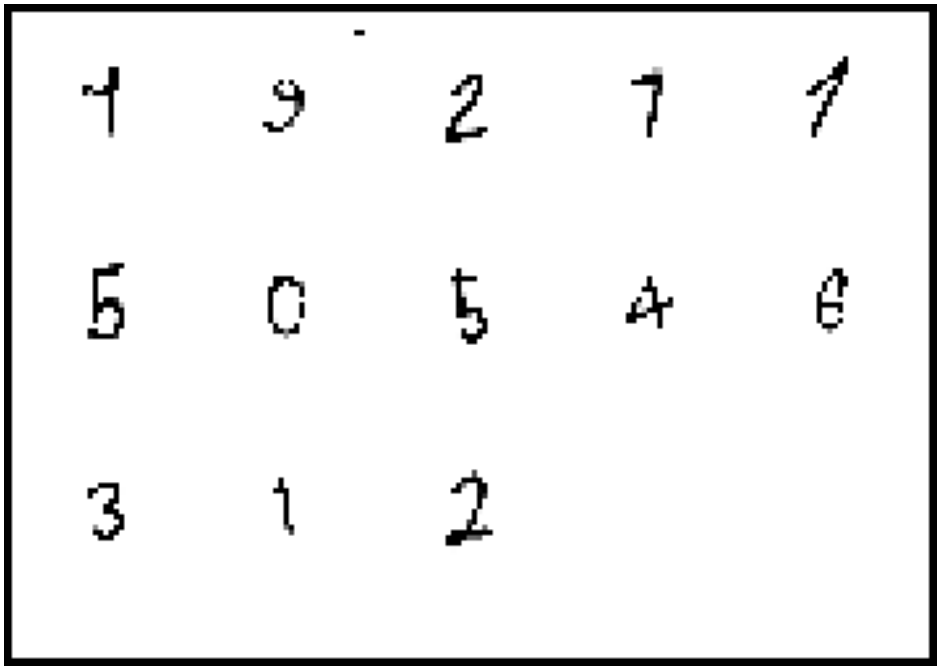

Figura 4: Conjunto de teste real

Embora a CNN tenha apresentado uma acurácia superior à observada na execução da DNN, consideramos que o custo de treinar uma rede neural convolucional não se mostra viável na aplicação para um problema de nicho simples como este. As diferenças entre as taxas de acurácia e perda entre ambas as redes é significativamente pequena, enquanto a disparidade entre seus tempos de execução é extremamamente alta, o que coloca a CNN em desvantagem na escolha de um modelo de aprendizado de máquina para a solução de tal problema.

5 Considerações Finais

Por fim, neste trabalho conseguimos explorar de forma bastante superficial, os conceitos por trás do subcampo da IA, o aprendizado de máquina e como construir modelos inteligentes que podem atuar de forma independente, analisando os mais diversos dados para nos fornecer soluções possíveis de serem obtidas por estes meios para alguns problemas do mundo real.

Referências

- Keras API reference: https://keras.io/api/
- Python & NumPy utilities: https://keras.io/api/utils/python_utils/#to_categorical-function
- Layer activation functions: https://keras.io/api/layers/activations/
- Dense layer: https://keras.io/api/layers/core_layers/dense/
- Conv2D layer: https://keras.io/api/layers/convolution_layers/convolution2d/
- MaxPooling2D layer: https://keras.io/api/layers/pooling_layers/max_pooling2d/
- Optimizers: https://keras.io/api/optimizers/
- Losses: https://keras.io/api/losses/
- Ultimate Guide To Loss functions In Tensorflow Keras API With Python Implementation: https://analyticsindiamag.com/ultimate-guide-to-loss-functior
- Plotting Confusion matrix for tensorflow model: https://www.kaggle.com/code/mdmashurshalehin/plotting-confusion-matrix-for-tensorflow-model/notebook
- CNN — Introduction to Pooling Layer: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/
- Neural Network Optimization Algorithms: https://towardsdatascience.com/neural-network-optimization-algorithms-1a44c282f61d
- Image Thresholding: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html