Relatório Final

Machine Learning

Susana Bouchardet e Rebeca Bordini

September 17,2018

1 Introdução

O problema de reconhecimento de dígitos é uma problema tradicional para validação de algoritmos de machine learning. O dataset mais tradicional para esse problema é o MNIST [3], um dataset de digitos escritos a mão.

Originalmente esse dataset foi criado com o objetivo de criar um sistema de reconhecimento de digitos para reconhecer os números do *zip-code* em cartas. Mas ele se tornou popular para validar modelos e algoritmos de reconhecimento de imagem.

Nesse trabalho o objetivo é, utilizando esse dataset, experimentar diferentes algoritmos e técnicas para solucionar o problema. Para esse curso queremos encontrar uma solução com maior acurácia possiível.

2 Descrição do problema

Para compreender melhor, vamos entender a descrição formal do problema. O problema proposto é um problema de aprendizado supervisionado, ou seja, temos dados já classificados e desejamos aprender uma maneira de classificar novos dados de maneira automática.

Para compreender melhor, vamos entender qual o *Input* do modelo (x) e o *Output* (y).

Input: No caso desse problema, os *inputs* são imagens de 28×28 pixels, com um canal.

Quando falamos de imagem, canal são as cores usadas para formar a imagem. Imagens coloridas tem normalmente 3 canais: Vermelho, Verde e Azul. No caso dos dados de *input*, temos apenas um canal por se tratar de uma imagem em preto e branco.

Cada imagem pode ser descrita como uma matriz de 28 linhas e 28 colunas, onde o valor em cada célula representa a intensidade do pixel. A intensidade de cada pixel é um valor inteiro de 0 a 255, onde 0 representa a cor preta e 255 a cor branca.

Output: Dado o *input* descrito acima, o modelo deve retornar qual número esta desenhado na imagem. Como cada imagem contém apenas um dígito, o problema é um problema de classificação com 10 classes: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9.

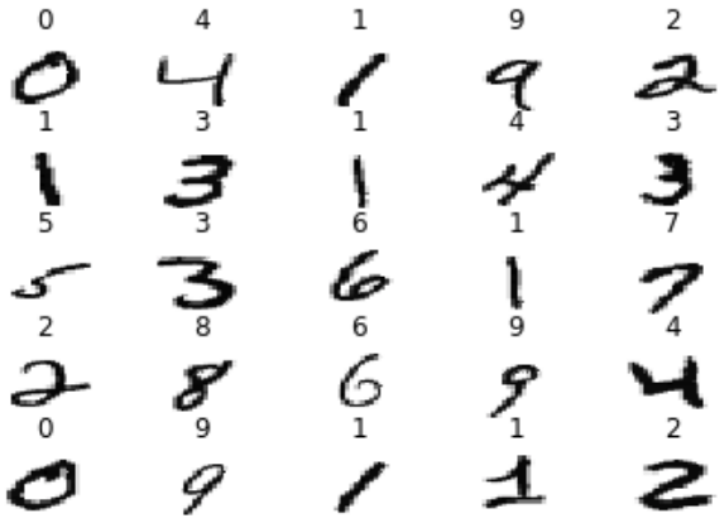

Figure 1: Exemplos dos dados disponíveis no dataset

A figura 1 mostra os dados disponíveis no dataset. Cada imagem é uma represnetação grafica dos *inputs*. Sobre cada imagem existe o número que representa a classe associada a imagem, e que também é o *output* esperado do modelo.

3 Métrica de avaliação

Varios artigos [2, 4, 6, 5] utilizam a acurácia como métrica para avaliar o resultado de um modelo para solucionar esse problema. Por esse motivo decidimos então utilizar a acurácia como métrica.

A acurácia é uma métrica para classificadores que mede a taxa de acerto de um classificador. Seja $\hat{y_i}$ o valor predito pelo classificador e y_i o valor real do i-ésimo exemplo, podemos descrever o calculo da acurácia como

$$acc(y,\hat{y}) = \frac{\#\{\hat{y}_i | \hat{y}_i = y_i\}}{\#\{y_i\}}. \tag{1}$$

Pela equação 1 é possível notar que o valor de *acc* varia entre 0 e 1, de maneira que é 0 se errar todas as classificações e 1 se acertar todas as classificações.

4 *Baseline* e *state-of-the-art*

A métrica para medir a performance do modelo, nesse caso, é a acuracia. Vamos usar dois valores como referência para julgar os modelos desenvolvidos nesse trabalho: um *baseline* e a acurácia do *state-of-the-art*.

4.1 *Baseline*

O *baseline* é a acurácia do classificador mais simples para resolver esse problema. Nesse caso, foi desenvolvido um classifidor randomico que classifica as entradas aleatóriamente de acordo com a distribuição das classes no dataset de treinamento.

De acordo com a figura 2, a quantidade de exemplos por classe é bem proxima. Baseado nisso, o classificador aleatório tem a mesma probabilida de sortear qualquer classe. Como o problema envolve 10 classes, então cada classe tem a probabilidade $\frac{1}{10}$ de ser sorteada pelo classificador aleatório.

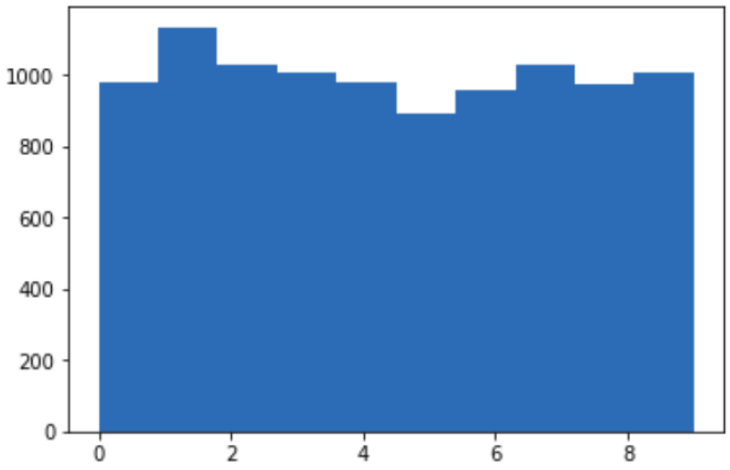

Figure 2: Distribuição dos dados de treinamento por classe

Para esse classificador obtemos um valor de acurácia de 0.1009

4.2 *State-of-the-art (SOTA)*

O *SOTA* atual é um comitê de modelos com 35 modelos [1]. Esses modelos são Redes Neurais Convolucionais (CNN).

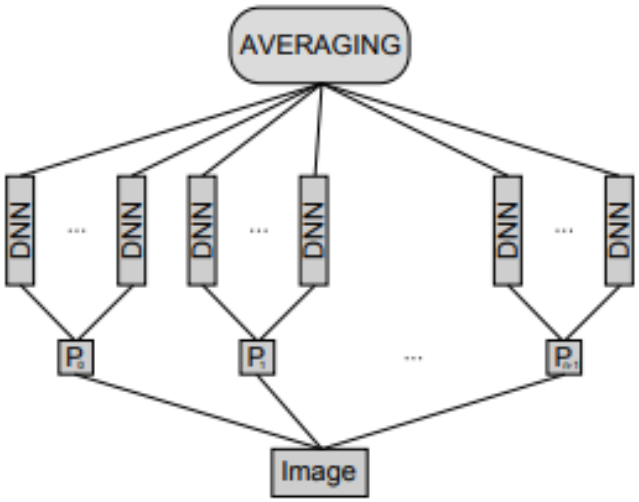

Figure 3: Imagem retirada de [1] que mostra a arquitetura da solução proposta

Na figura 3, o *input* é o nó inferior. Esse nó é distribuido para $P_0, P_1, \ldots eP_{n+1}$, que representam a etapa de pré-processamento. Cada etapa de pré-processamento alimenta uma ou mais redes neurais (DNN). Por fim, cada rede neural gera sua classificação, mas a classificação final é resultado da média dos resultados das redes neurais.

Com essa técnica, foram criadas 35 redes neurais convolucionais, e esse comitê de modelos obteve 0.9977 de a acurácia.

5 Modelos

Nessa secção iremos descrever os modelos tentados, como os algoritmos funcionam e seus respectivos resultados.

5.1 Árvore de decisão

O primeiro modelo testado foi uma árvore de decisão. Como o projeto foi desenvolvido ao longo da disciplina de machine learning, aplicamos parte do que aprendemos em seu desenvolvimento. O primeiro contúdo dado foi árvore de decisção.

A árvore de decisção é um argoritmo de machine learning que tem como objetivo encontrar as regras que podem dividir melhor os dados de treino de acordo com suas classes.

O dataset é disponibilisado em duas partes: treino e validação. Do dataset de treino, nós separamos 80% do dado para de fato ser usado no treino e 20% para ser usado como teste.

	Acurácia
Treino	1
Teste	0.8705
Validação	0.874

Table 1: Métricas obtidas no modelo gerado pela Árvore de decisão

O modelo gerado por esse algoritmo obteve a acurácia mostrada na tabela 1. É possível notar pela acurrácia obtida pelo daod de treino que o modelo aprendeu exatamente a classificação do treino. Isso demonstra que o algoritmo aprendeu o dataset, não a tarefe.

5.2 DNN

Outro algoritmo testado foi uma rede neural profunda (DNN). Nesse caso a arquitetura testada foi com apenas duas camadas ocultas, a primeira com 50 neurônios e a segunda com 25 neurônios. Todas as camadas eram totalmente conectadas.

Assim como nos modelos anteriores, temos os dataset separado em treino, teste e validação. A cada época de treinamento avaliamos a acurácia no dataset de test.

	Acurácia
Treino	0.603
Teste	0.553
Validação	0.521

Table 2: Métricas obtidas no modelo gerado pela DNN

5.3 CNN

A Rede Neural Convolucional (CNN) é um tipo de rede neural com camadas convolucionais. Uma camada convolucional aplica operações de convolução na entrada. A convolução é uma operação que percorre a imagem executando o produto interno de uma matrix de convolução (ou kernel) nas regiões da imagem. O resultado da convolução é um *feature map*.

$$C_{3\times3} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2}$$

A figura 4 mostra um exemplo de convolução aplicada a uma imagem do datase. No Exemplo o filtro aplicado é descrito na matrix 2. Esse kernel é utilizado para evidenciar as bordas da imagem.

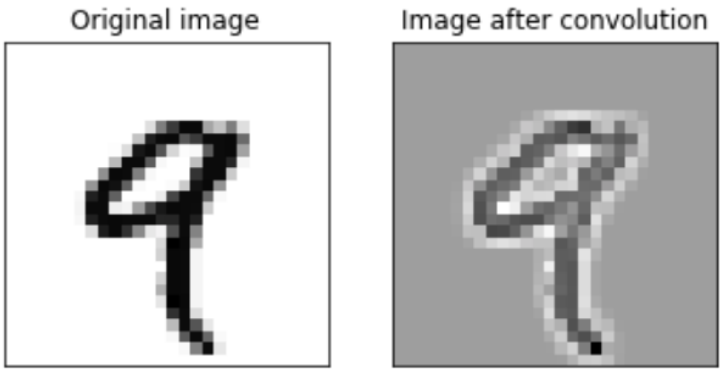

Figure 4: Exemplo de convolução aplicada a uma imagem do dataset

	Acurácia
Treino	0.912
Teste	0.904
Validação	0.902

Table 3: Métricas obtidas no modelo gerado pela CNN

A tabela 3 mostra como foi o desenpenho da CNN no dataset de treino, teste e validação.

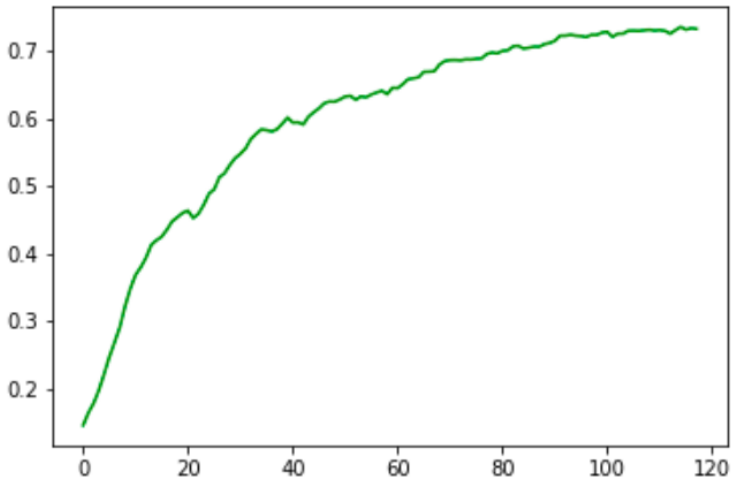

Figure 5: Grafico da evolução da acurácia durante o treinamento

A figuta 5 mostra a evolução da acurácia da rede nos dados de teste ao longo das épocas. A etapa de treinamento é interrompida em 120 épocas após atingir 0.904 de acurácia.

References

[1] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[2] Abien M Fred Agarap. An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification. 2017.

[3] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[4] Karim M Mahmoud. Handwritten Digits Recognition using Deep Convolutional Neural Network: An Experimental Study using EBlearn. pages 3–6, 2014.

[5] Jose M Saavedra. Handwritten Digit Recognition Based on Pooling SVM-Classifiers Using Orientation and Concavity Based Features. Technical report, 2014.

[6] Anuj Sharma. A combined static and dynamic feature extraction technique to recognize handwritten digits. *Vietnam Journal of Computer Science*, 2(3):133–142, 2015.