Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

IA048 – Aprendizado de Máquina 13 de maio de 2024

Docentes: Levy Boccato & Romis Attux

Discente:

– Gabriel Toffanetto França da Rocha – 289320

Atividade 3 – Redes Neurais

Sumário

1 Apresentação dos dados

A base de dados BloodMNIST é formada por um total de 15380 imagens de células sanguíneas divdidas em 8 classes, como mostrado na Figura 1. Essas amostras se dividem em conjunto de treinamento, com 11959 representantes e teste, com 3421 itens. As imagens estão disponíveis em diferentes resoluções, sendo *a priori*, escolhida a resolução de 28x28 para trazer mais eficiência computacional, principalmente para o caso das redes densas. Porém, propõem-se o teste de amostras de maior resolução para a rede convolucional já treinada com as amostras 28x28, para validação da robustez ao tamanho da entrada.

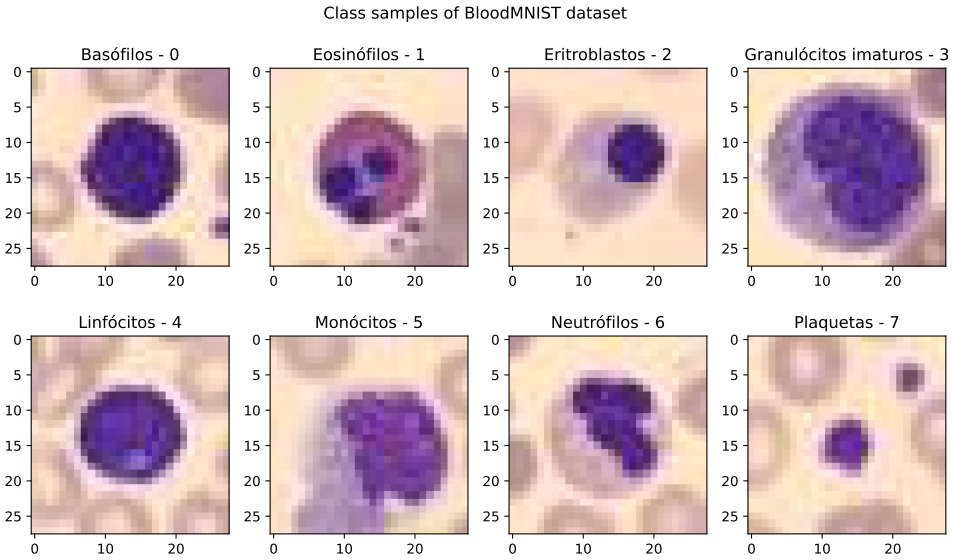


Figura 1: Amostras das classes do *dataset* BloodMNIST.

O *dataset* BloodMNIST já fornece de forma separada os dados utilizados para treinamento, e os que serão utilizados para teste. A Figura 2 mostra a distribuição de amostrar para cada classe, para os conjuntos de treinamento, em azul, e de teste, em verde. Observa-se que existem classes majoritárias, como a 1, 3, 6 e 7, que apresentam muito mais amostras que as demais. O perfil de amostras por classe é similar nos conjuntos de teste e treinamento, o que mostra que o impacto no mapeamento pelo desbalanceamento das classes irá afetar de forma igual ambos os *datasets*. **Tal disparidade pode ser tanto causada por um viés na coleta de dados, quanto também pela ocorrência real de mais representantes de uma classe do que das demais, o que modela de forma realista os dados.**

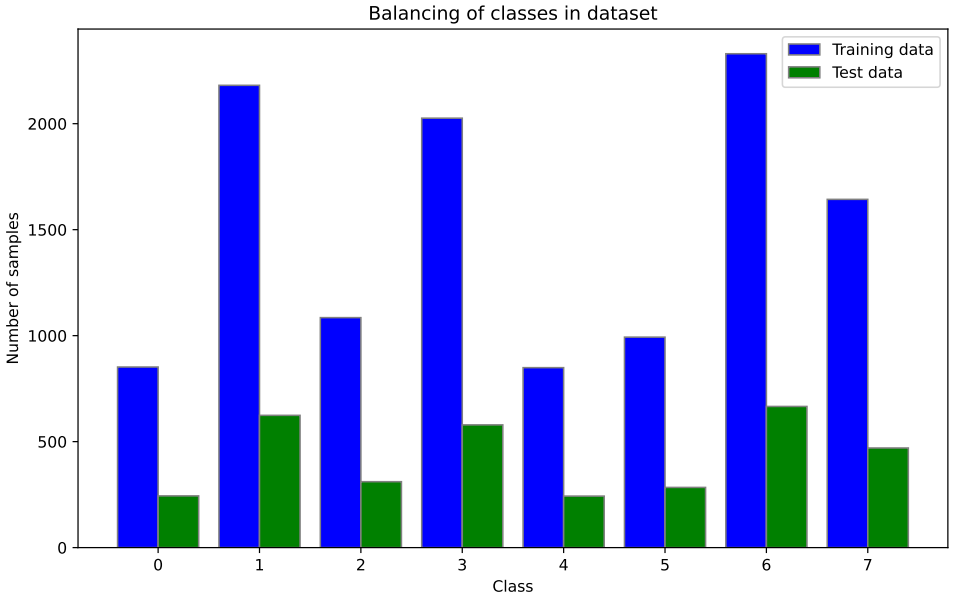


Figura 2: Balanço das classes nos *datasets* de treinamento e teste.

Para realizar o treinamento utilizando da ferramenta de validação cruzada, foi escolhida a validação do tipo *holdout*, por meio do particionamento do conjunto de dados de treinamento, obtendo um novo conjunto de dados de validação. O novo conjunto de treinamento é formado pelos primeiros 70% do conjunto original de treinamento, enquanto o de validação, os 30% restantes do final do *dataset* original. Para garantir a veracidade da validação, quer-se que ambos os conjuntos tenham a mesma representação de cada classe, o que pode se afirmar positivo de acordo com a Figura 3.

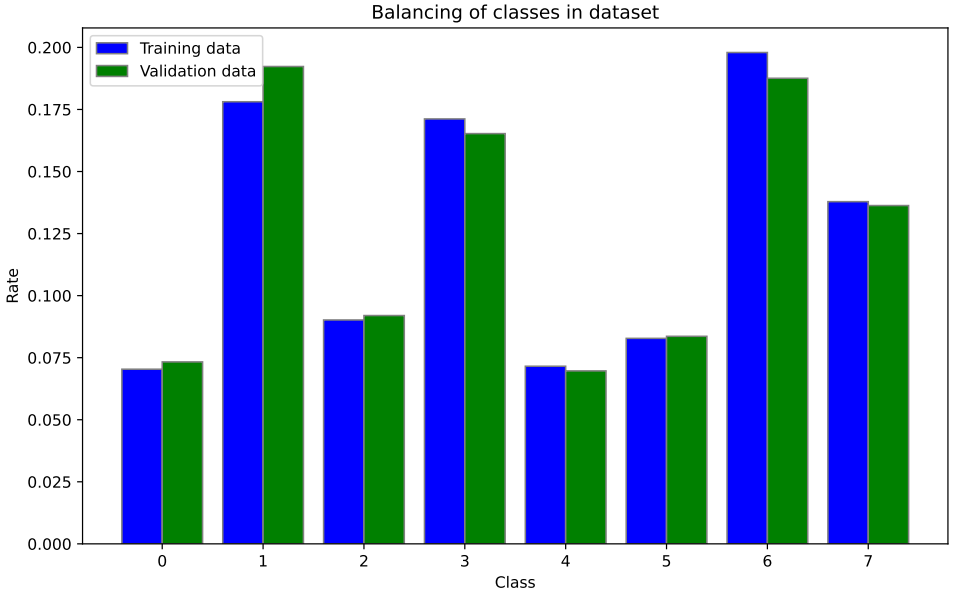


Figura 3: Balanço das classes nos conjuntos de dados de treinamento e validação cruzada do tipo *holdout*.

2 MLP

A implementação da rede MLP com uma camada intermediária se deu por meio do uso do *framework* TensorFlow, possuindo uma camada de entrada que sequencia os pixels da imagem em um vetor, uma camada de neurônios intermediária, e uma camada de saída com função de ativação *softmax* para geração do vetor *one-hot encoding* das probabilidades da entrada pertencer à cada uma das 8 classes.

2.1 Busca do melhor modelo

Para realizar a busca do melhor modelo da MLP, considerando que existem uma grande possibilidade de hiper-parâmetros, foi realizada uma busca exaustiva simplificada por etapas, baseada no funcionamento dos *wrappers*, onde dada uma configuração inicial de parâmetros, baseado no comumente visto na literatura (GÉRON, 2019). As variáveis selecionadas para a busca foram: Número de neurônios da camada intermediária, função de ativação dos neurônios da camada intermediária, taxa de *dropout* dos neurônios da camada intermediária e otimizador para ajuste dos pesos. Demais hiper-parâmetros como tamanho do *batch* e passo do algoritmo de otimização foram mantidos *default*.

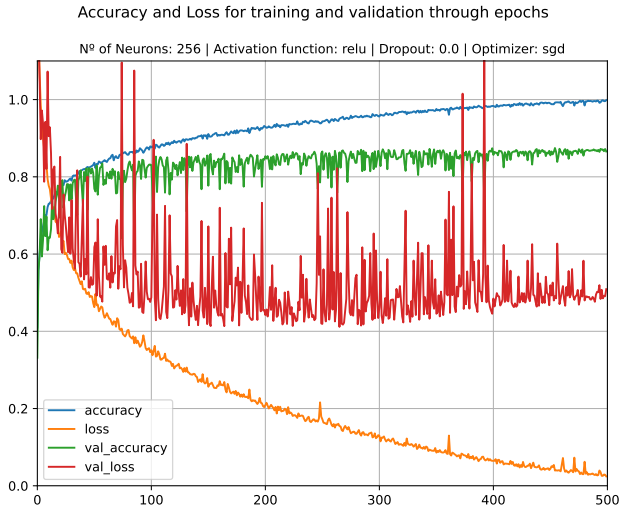
A busca por etapas se deu da seguinte forma: Dada a condição inicial, uma MLP de 256 neurônios na camada intermediária com função de ativação ReLU, sem *dropout* e ajustada por *Stochastic Gradient Descendent* (SGD), testou-se a rede alterando primeiramente o número de neurônios. Uma vez conhecido o valor que obteve maior acurácia, testou-se as funções de ativações canditadas para esse número de neurônios, buscando a combinação que desse a melhor acurácia. O processo se repete para o *dropout* e o algoritmo de otimização, onde ao final se obtém a combinação que agrega o melhor número de neurônios visto, melhor função de ativação para tal conjunto de neurônios, melhor taxa de *dropout* e o melhor otimizador.

Hiper-parâmetros variados durante a busca:

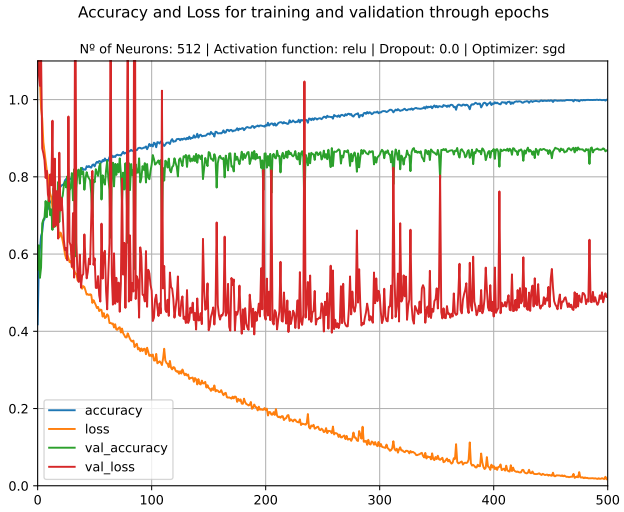
- Número de neurônios: [`256`; `512`; `1024`; `2048`; `4096`]
- Função de ativação: [`ReLU`; `Sigmoide`]
- *Dropout*: [`0,0`; `0,25`; `0,5`]
- Otimizador: [`SGD`; `ADAM`]

Para todos os casos, foram treinadas 500 épocas, com *mini-batch* de 32 amostras, utilizando uma função de *callback* para salvar os parâmetros da melhor época, evitando assim preocupações com *overfitting*.

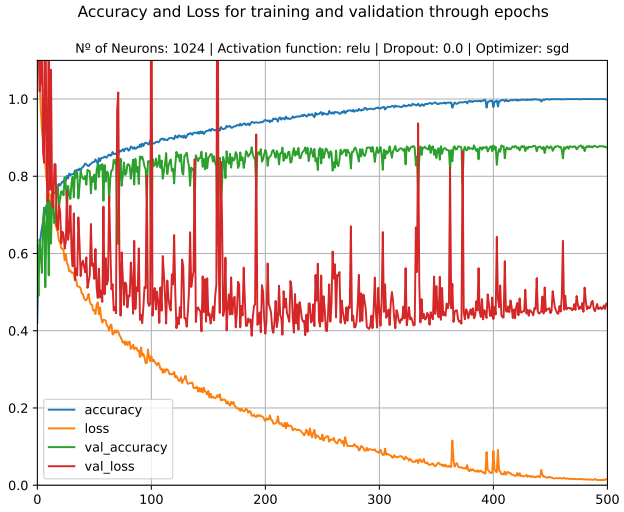
2.1.1 Número de neurônios



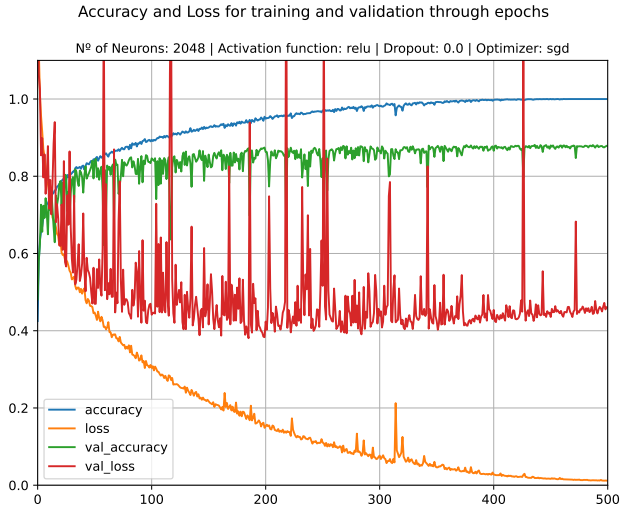
(a) 256 neurônios.



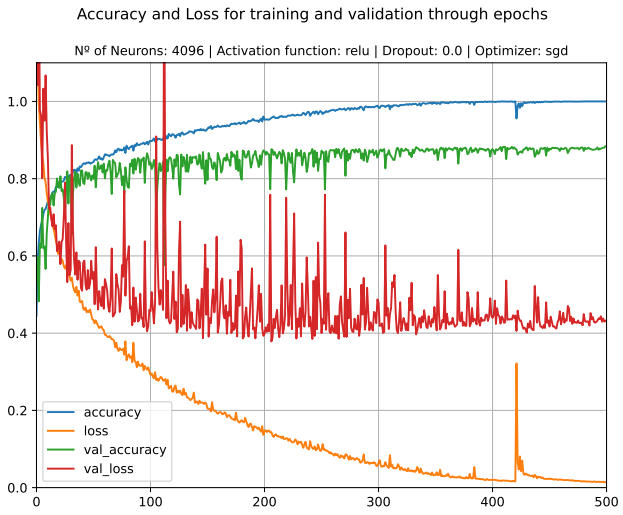
(b) 512 neurônios.



(c) 1024 neurônios.



(d) 2048 neurônios.



(e) 4096 neurônios.

Figura 4: Evolução da perda e acurácia de treinamento e validação com as épocas de treinamento para cada número de neurônios avaliados.

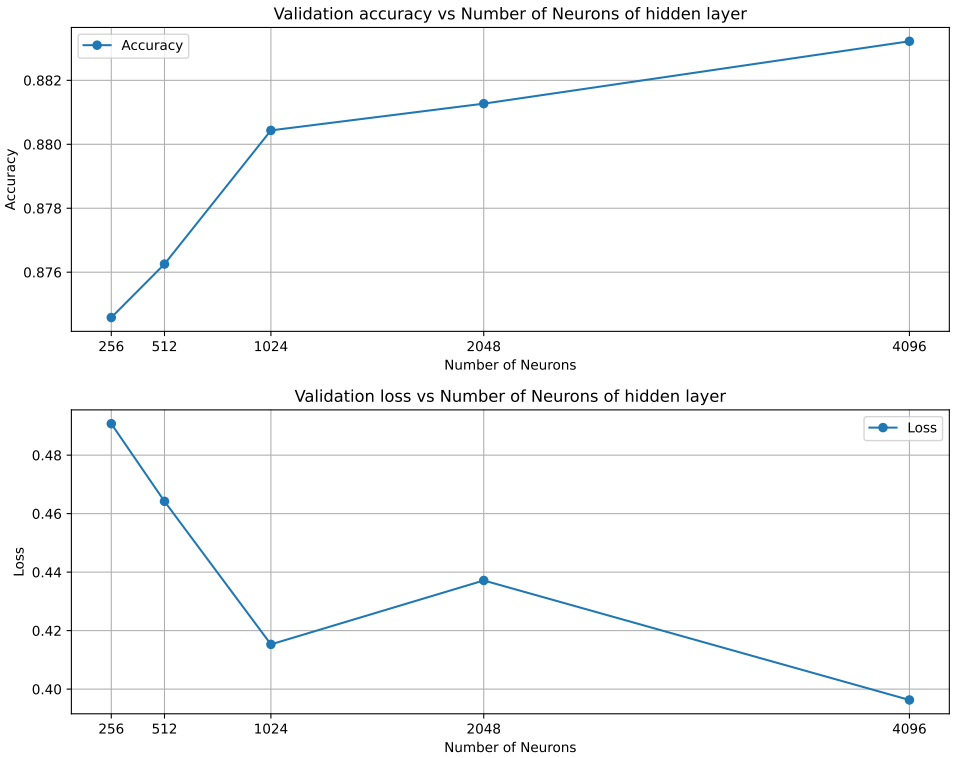
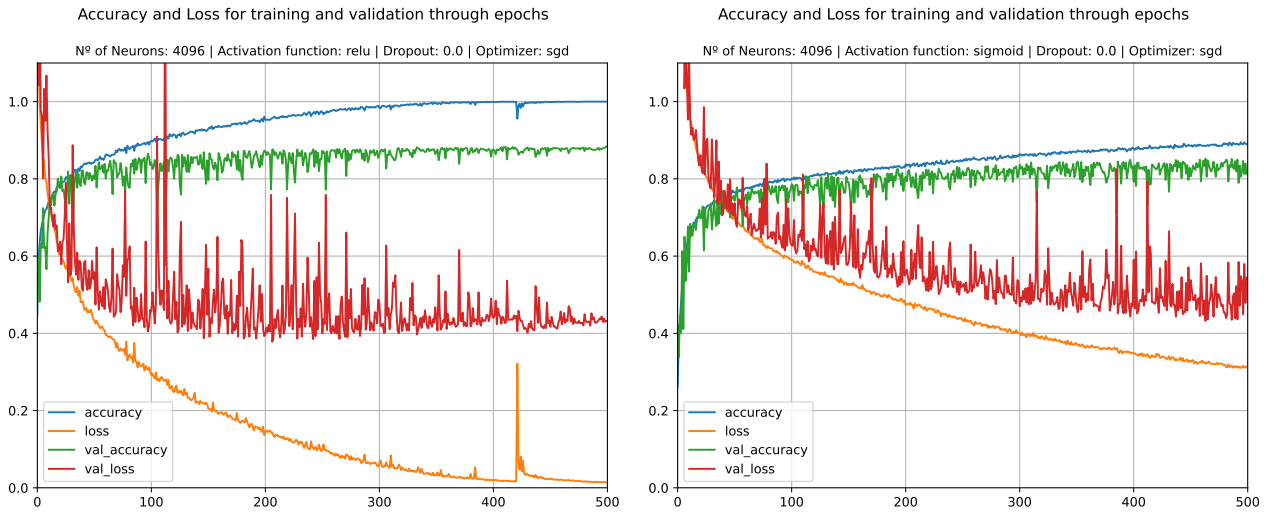


Figura 5: Acurácia e perda com a variação do número de neurônios da camada intermediária.

2.1.2 Função de ativação



(a) ReLU. (b) Sigmoide.

Figura 6: Evolução da perda e acurácia de treinamento e validação com as épocas de treinamento para cada função de ativação candidata.

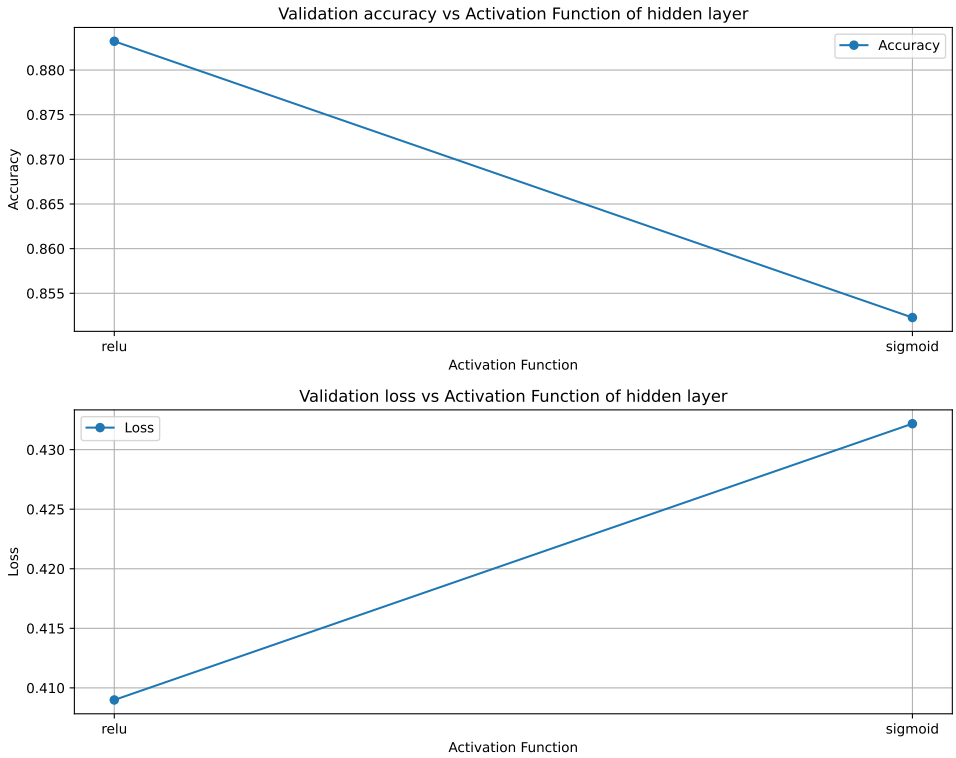
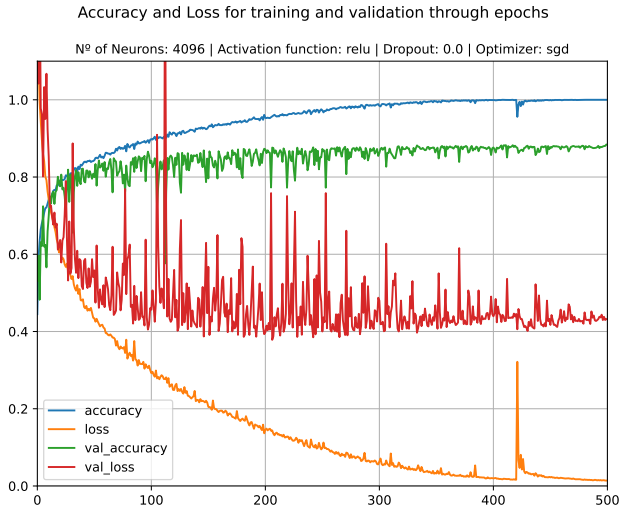
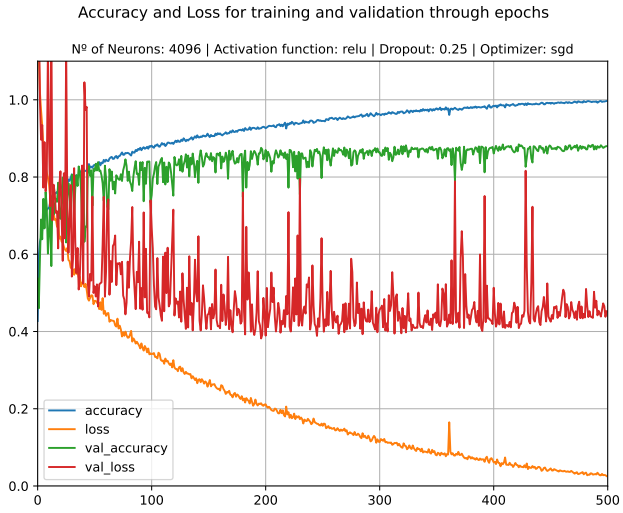


Figura 7: Acurácia e perda com a variação da função de ativação dos neurônios da camada intermediária.

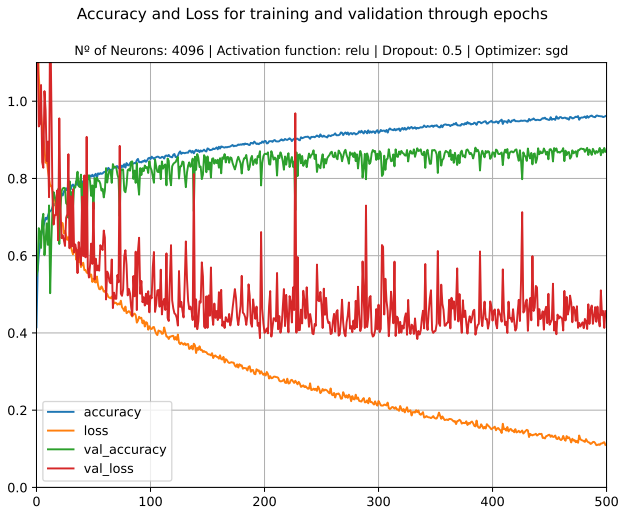
2.1.3 *Dropout*



(a) 0,0.



(b) 0,25.



(c) 0,5.

Figura 8: Evolução da perda e acurácia de treinamento e validação com as épocas de treinamento para diferentes taxas de *dropout*.

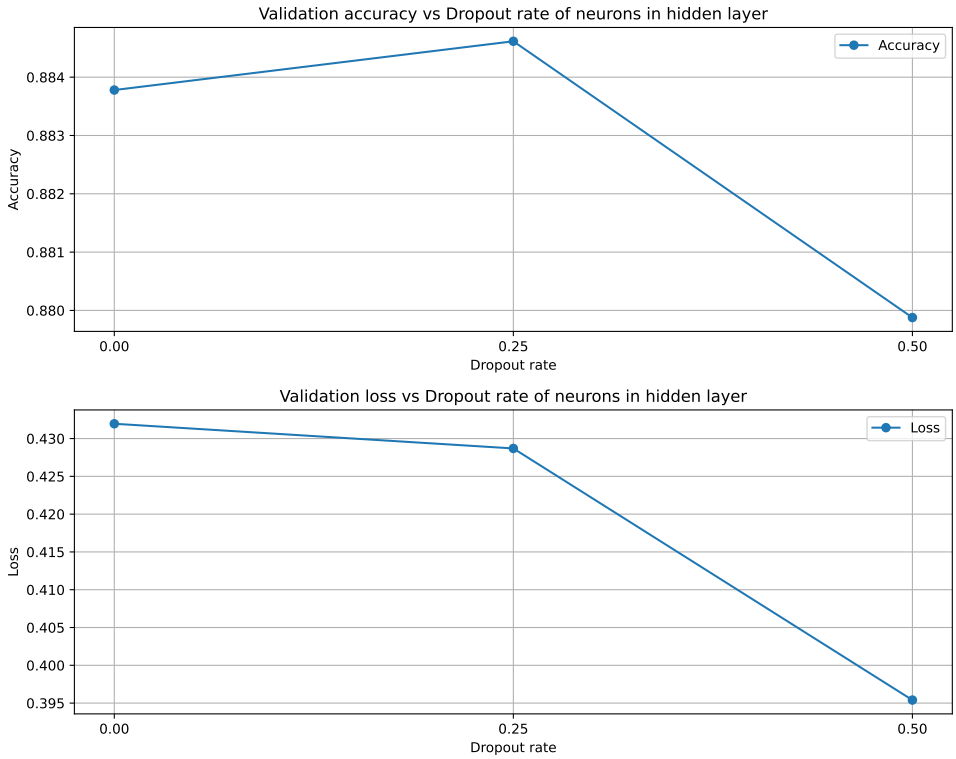
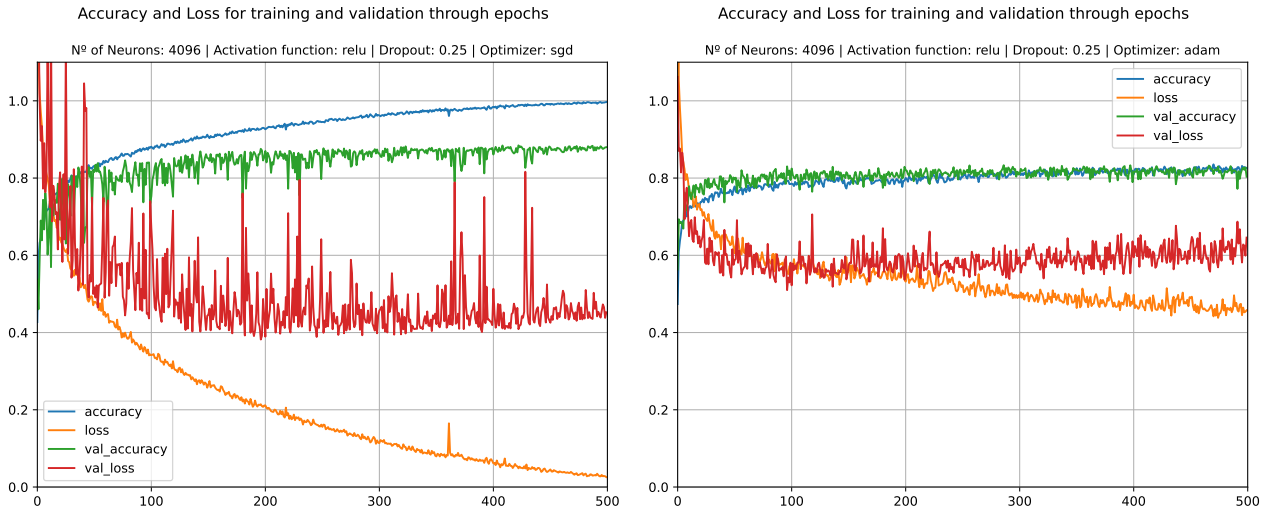


Figura 9: Acurácia e perda com a variação da taxe de *dropout* dos neurônios da camada intermediária.

2.1.4 Otimizador



(a) ReLU. (b) Sigmoide.

Figura 10: Evolução da perda e acurácia de treinamento e validação com as épocas de treinamento para os otimizadores analisados.

Citar (WILSON et al., 2018)

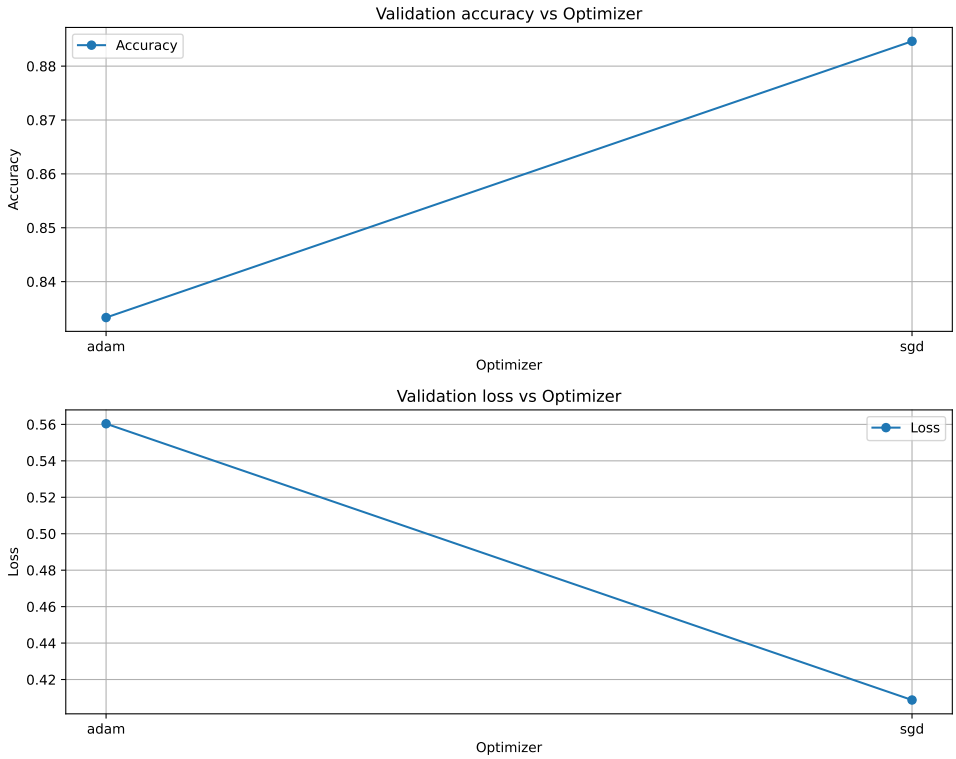


Figura 11: Acurácia e perda com a variação do algoritmo de otimização.

2.2 Análise do melhor modelo

$$Accuracy = 0.8705 \tag{1}$$

$$BA = 0.8437 \tag{2}$$

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	177	3	0	43	4	17	0	0
1	1	613	0	3	1	0	6	0
2	4	1	264	16	6	3	11	6
3	34	26	3	447	5	32	32	0
4	13	0	11	31	182	0	6	0
5	10	2	1	48	3	215	5	0
6	0	22	5	22	2	2	611	2
7	0	0	1	0	0	0	0	469

Tabela 1: Matriz de confusão do classificador baseado em MLP.

Classe	Precisão	*Recall*
0	0.7254	0.7406
1	0.9824	0.9190
2	0.8489	0.9263
3	0.7720	0.7328
4	0.7490	0.8966
5	0.7570	0.7993
6	0.9174	0.9106
7	0.9979	0.9832

Tabela 2: Precisão e *recall* por classe do classificador baseado em MLP.

2.2.1 Análise dos erros

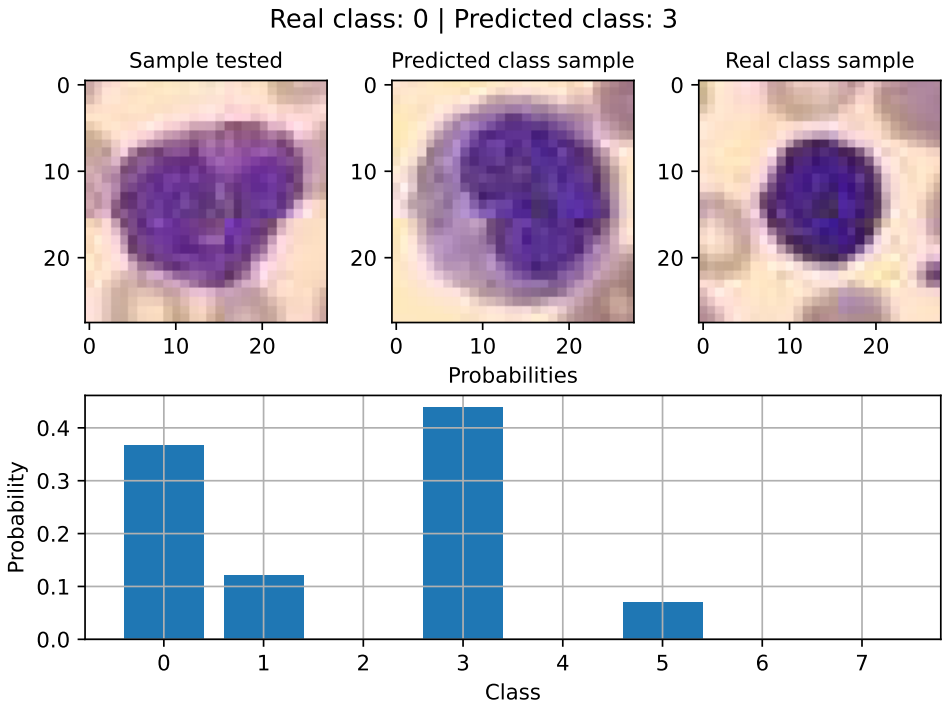


Figura 12: Análise de um caso de erro de classificação.

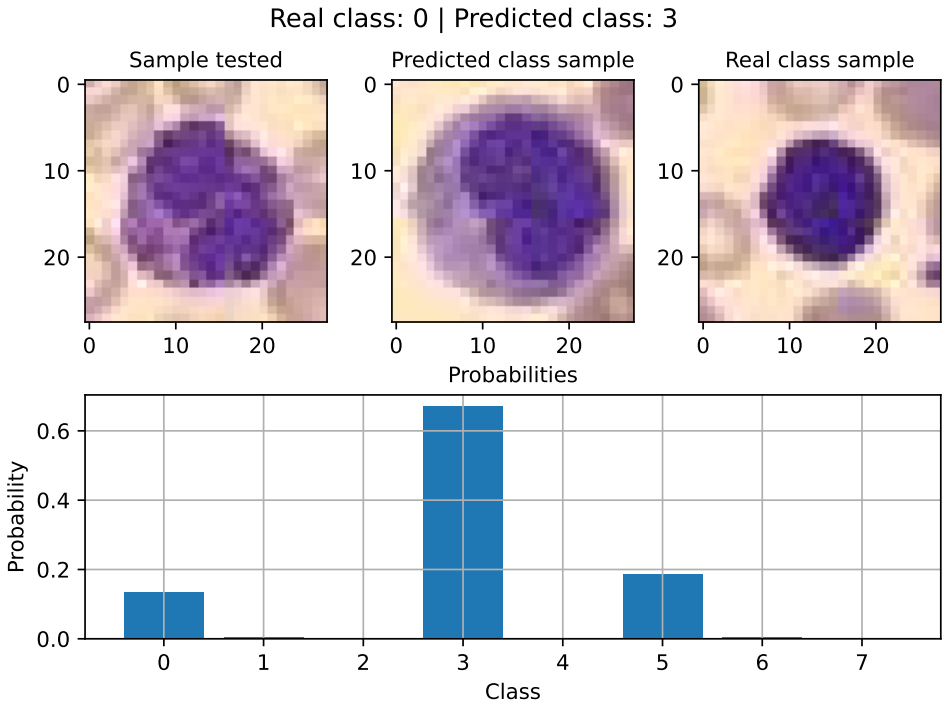


Figura 13: Análise de um caso de erro de classificação.

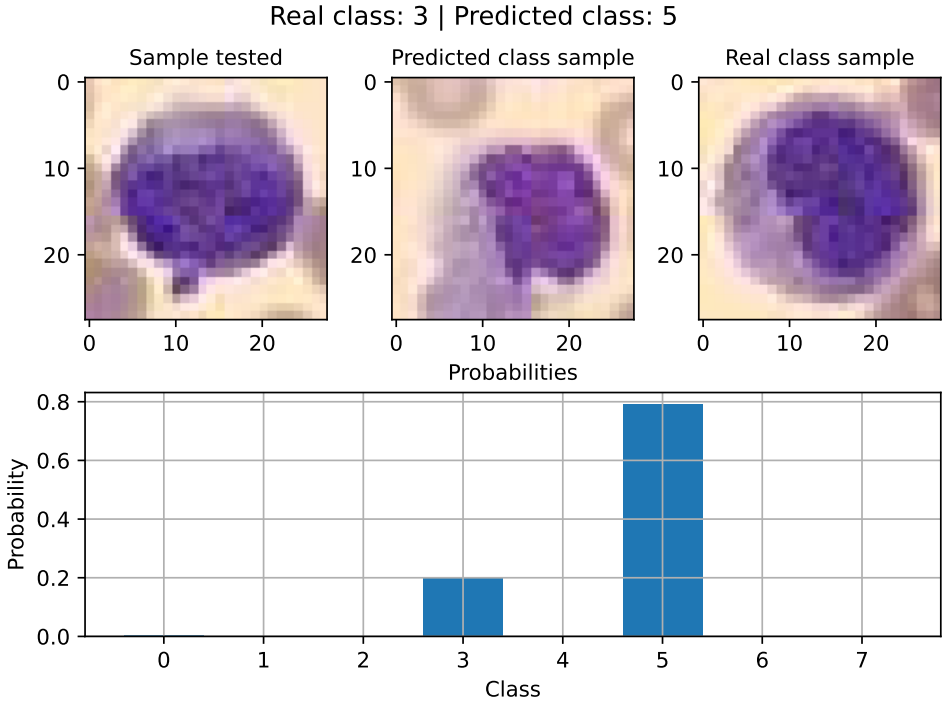


Figura 14: Análise de um caso de erro de classificação.

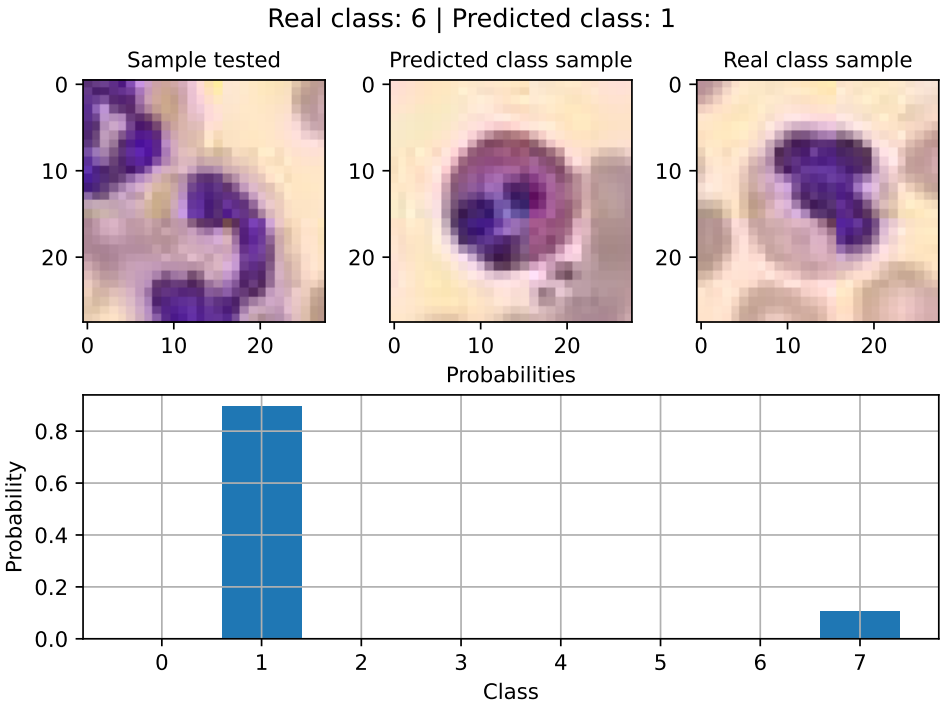


Figura 15: Análise de um caso de erro de classificação.

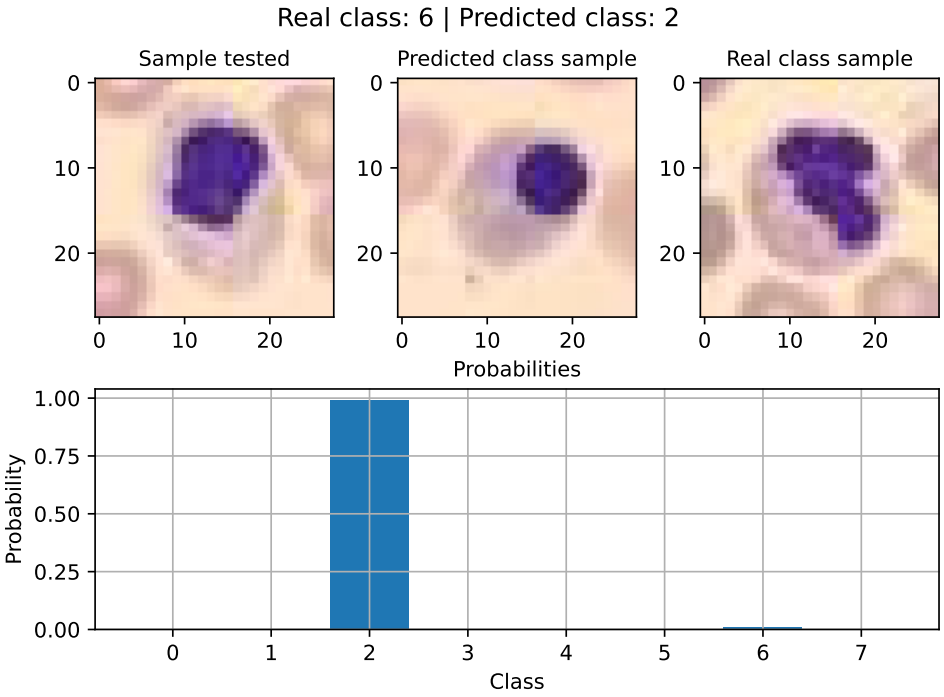


Figura 16: Análise de um caso de erro de classificação.

3 CNN Simples

4 CNN Profunda

Referências

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, 2019. ISBN 9781492032618.

WILSON, A. C. et al. *The Marginal Value of Adaptive Gradient Methods in Machine Learning*. [S.l.]: arXiv: 1705.08292, 2018.

Anexos

Códigos fonte

Todos os códigos fonte e arquivos de dados utilizados para a elaboração deste documento podem ser encontrados no repositório do GitHub no link: github.com/toffanetto/ia048.