EFC 03 - Redes Neurais

Luís Antônio Almeida Lima Vieira (221045) | Nathan Shen Baldon (242448)

10 de novembro de 2022

1 Introdução

O objetivo desta atividade é utilizar técnicas de redes neurais para o problema de reconhecimento de células sanguíneas periféricas, utilizando a base de dados BloodMNIST (ACEVEDO et al., 2020) (YANG et al., 2021). Esta base de dados possui 17092 imagens microscópicas coloridas com dimensão 28x28 pixels e 3 canais de cor. A figura 1 mostra exemplos de imagens de cada classe e a tabela 1 mostra os rótulos de cada classe.

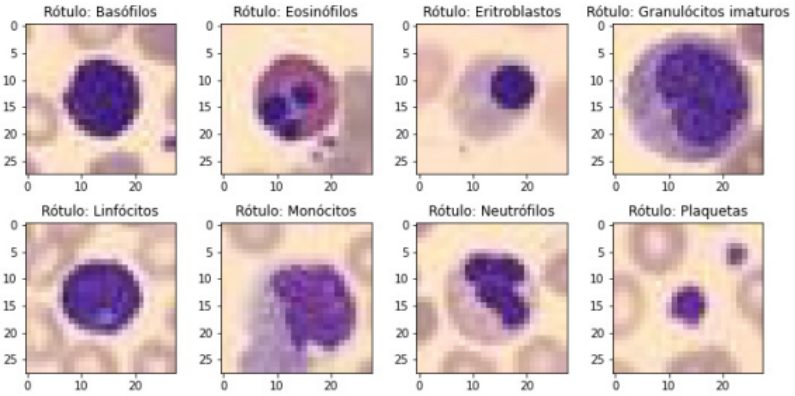


Figura 1: Exemplos de imagens de cada rótulo.

Id	Rótulo
0	Basófilos
1	Eosinófilos
2	Eritroblastos
3	Granulócitos imaturos
4	Linfócitos
5	Monócitos
6	Neutrófilos
7	Plaquetas

Tabela 1: Correspondência entre os identificadores numéricos das classes e o tipo de célula sanguínea..

2 Preparação e análise do conjunto de dados

Inicialmente, importou-se a base de dados *bloodmnist* (ACEVEDO et al., 2020) (YANG et al., 2021) e foram separados os dados e seus respectivos rótulos (com *.imgs* para as imagens e *.labels* para os rótulos). A separação utilizada para os conjuntos de treino, teste e validação,

foi a separação padrão já feita na base de dados, com as 11959 primeiras imagens e rótulos separadas para treinamento, as 1712 imagens e rótulos seguintes separadas para validação e as últimas 3421 imagens e rótulos separadas para teste. Todas as imagens são 28x28, compostas por três canais, por serem coloridas.

Foi feita uma análise simples do balanceamento das classes nos conjuntos disponíveis. Com isso, foi possível observar que nos conjuntos de treinamento, validação e teste, a classe Basófilos (0) representa 7.12% dos dados, a classe Eosinófilos (1) representa 18.24% dos dados, a classe Eritroblastos (2) representa 9.07% dos dados, a classe Granulócitos imaturos (3) representa 16.94% dos dados, a classe Linfócitos (4) representa 7.1% dos dados, a classe Monócitos (5) representa 8.30% dos dados, a classe Neutrófilos (6) representa 19.48% dos dados e a classe Plaquetas (7) representa 13.74% dos dados. Com isso, é possível notar que não há um desbalanceamento muito grande entre as classes, sendo as classes 1, 3, 6 e 7 mais frequentes nos conjuntos.

Antes de passar os dados para as redes neurais, foi feito um pré processamento simples dividindo todas as imagens do conjunto de dados por 255, de forma a manter a intensidade dos *pixels* das imagens em valores reais de 0 a 1.

3 MLP - uma camada intermediária

Para toda a atividade utilizou-se a *Keras API* do *TensorFlow*. Para construção das MLPs, utilizou-se a função "*keras.Sequential()*", tal que as camadas foram adicionadas utilizando-se "*model.add(keras.layers.Flatten())*"(para entrada) e *model.add(keras.layers.Dense())*"(camadas intermediária e de saída).

Para a MLP de uma camada intermediária, foram adicionadas três camadas no total: uma camada de entrada (*Flatten*), uma camada intermediária (*Dense*) e uma camada de saída com 8 saídas (*Dense*). A camada de entrada (*Flatten*) tem a função de alterar o formato da entrada para um vetor de uma direção. Como as imagens tem a forma [28, 28, 3] (imagens de 28x28 *pixels* e com 3 canais de cor), a camada de entrada tem $28 \cdot 28 \cdot 3 = 2352$ saídas. Já a camada de saída foi de 8 neurônios, cada um representando uma possível classe e, devido ao cenário ser multi-classe, utilizou-se a ***softmax*** como função de ativação. Por fim, para camada intermediária, foi necessário escolher o número de neurônios e a função de ativação, escolha realizada utilizando o *Keras Tuner*.

Além desse hiperparâmetro (número de neurônios), também buscou-se os melhores valores de *learning rate* e o melhor número de épocas. Antes, no entanto, outras escolhas foram feitas para o modelo, entre elas a função custo, a função de ativação da camada intermediária, o otimizador, o balanceamento de classes, a métrica otimizada e a forma de regularização.

- **função custo**: utilizou-se a *SparseCategoricalCrossentropy*, por ser a mais recomendada para o cenário em que há mais de duas classes e em que as classes são representadas por números inteiros, como é o caso (KERAS, s.d.).

- **função de ativação**: ReLU, porque apresentou bom desempenho nos testes iniciais, por ter processamento mais rápido e por sua popularidade. Testou-se também a ELU, mas não houve melhora no desempenho.

- **otimizador**: Adam, por ter processamento mais rápido que o gradiente descendente e por ter passo adaptativo, de forma a facilitar a escolha do *learning rate* e acelerar a busca de hiperparâmetros ótimos (GÉRON, 2019).

- **balanceamento de classes**: foi feito utilizando-se a função *-class_ weight.compute_ - class_ weight(class_ weight='balanced')* do *Scikit Learn* para definição dos pesos. Houve uma pequena melhora, mas não tão significativa porque as classes não são muito desbalanceadas e porque redes neurais têm mais pesos que podem ser ajustados de forma a compensar os desbalanceamentos.
- **métrica otimizada**: acurácia, por ser o fator desempenho que se deseja analisar.
- **regularização**: utilizou-se *early stopping* na busca de hiperparâmetros, a fim de se evitar *overfitting*.

3.1 *Keras Tuner*

Para a busca de hiperparâmetros, utilizou-se o *Keras Tuner*. Antes disso, entretanto, tentativas iniciais de otimização dos hiperparâmetros foram realizadas manualmente, de forma que esse processo influenciou na escolha dos valores dos hiperparâmetros testados no *Keras Tuner*. Foram testadas as seguintes opções de hiperparâmetros:

- **número de neurônios**: entre 100 e 2000.
- **passo de adaptação (*learning rate*)**: entre $5 \cdot 10^{-4}$ e $3 \cdot 10^{-2}$, valores baseados no livro-texto (GÉRON, 2019) e por ser a faixa que apresentou melhores resultados nos testes iniciais.
- **número de épocas para treinamento**: seus valores não foram diretamente testados no *Keras Tuner*, mas a escolha foi feita durante validação e na escolha dos hiperparâmetros, observando-se a variação da acurácia e ocorrência de *early stopping*.

Para a otimização com o *Keras Tuner*, buscou-se otimizar a acurácia na validação, e utilizou-se o *Bayesian Optimizer*, o qual reduz o tempo de processamento, por fazer uma busca em um espaço menor de hiperparâmetros (motivo pelo qual não utilizou-se o *GridSearch* ou o *Random Search*).

Encontrou-se os seguintes hiperparâmetros ótimos, com acurácia máxima na validação de 0.8604: **1900** neurônios e *learning rate* de **0.0005**.

3.2 Treinamento e Teste

Com os hiperparâmetros escolhidos, configurou-se a MLP de uma camada intermediária da Figura 2, com 1900 neurônios na camada intermediária e função de ativação ReLU. Assim, foi realizado seu treinamento com 100 épocas e *learning rate* de 0.0005.

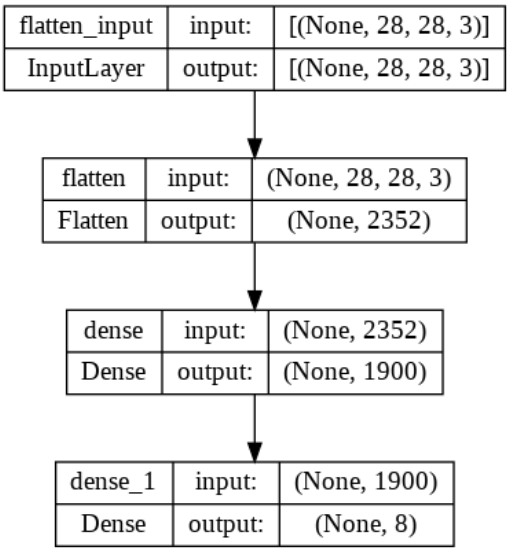


Figura 2: Modelo da MLP de uma camada intermediária.

No treinamento, ainda utilizou-se o conjunto de validação para observação da acurácia e perda em função do número de épocas (Figura 3). Note que isso foi feito de forma a se analisar melhor o efeito do número de épocas na acurácia e perda, além de se verificar se o *learning rate* estava adequado e se houve *overfitting*. Conforme se observa, até a época 100, a acurácia da validação ainda continua crescendo junto com a de treinamento, mas já apresenta taxa de crescimento reduzida e também um pequeno distanciamento da curva do treinamento, indicando um possível início de *overfitting*. Mais tarde, foi realizado também o treinamento utilizando o conjunto de treinamento completo (treinamento+validação) e não houveram melhoras significativas na acurácia.

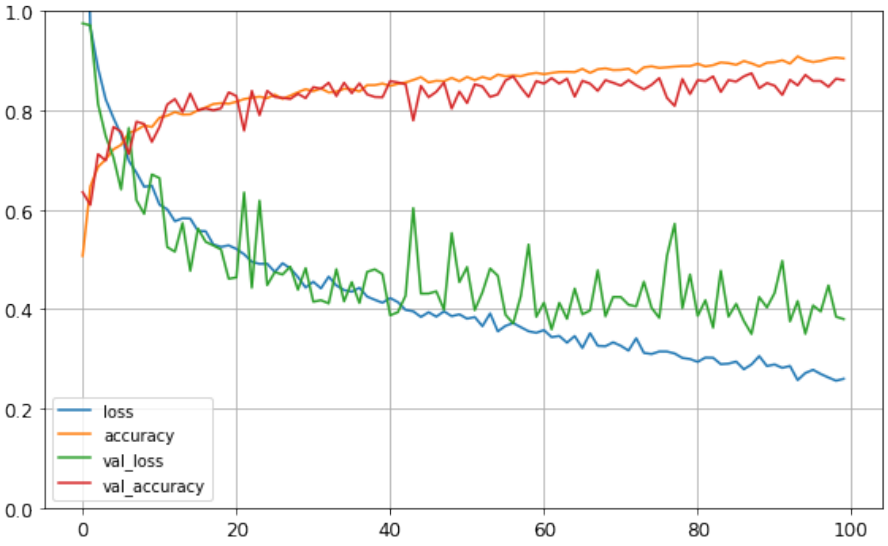


Figura 3: Treinamento da MLP de uma camada intermediária - acurácia e perda (treinamento e validação) em função do número de épocas.

3.3 Análise de Desempenho

Após o treinamento, foi feita a classificação do conjunto de teste, tal que analisou-se o desempenho do modelo observando-se a acurácia global e a matriz de confusão.

3.3.1 Acurácia Global

A acurácia global para os dados de teste foi obtida pela função *evaluate* com os dados de teste (x teste, y teste). Foi obtida uma acurácia global de **0.8556** para os dados de teste. Demais métricas para os dados de teste estão na figura 4. Essas métricas foram obtidas após a previsão dos valores nos dados de teste com o uso do *classification report* do *Scikit Learning*.

	precision	recall	f1-score	support
0	0.67	0.80	0.73	244
1	0.95	0.96	0.95	624
2	0.79	0.90	0.84	311
3	0.81	0.58	0.68	579
4	0.91	0.79	0.84	243
5	0.72	0.80	0.76	284
6	0.86	0.95	0.90	666
7	1.00	0.99	0.99	470
accuracy			0.86	3421
macro avg	0.84	0.85	0.84	3421
weighted avg	0.86	0.86	0.85	3421

Figura 4: Demais métricas de desempenho de teste para a MLP de uma camada intermediária.

3.3.2 Matriz de confusão

Com o modelo ótimo feito, foi feita a previsão para os dados de teste e obtida a matriz de confusão a partir da função *Confusion Matrix* do *Scikit Learning*.

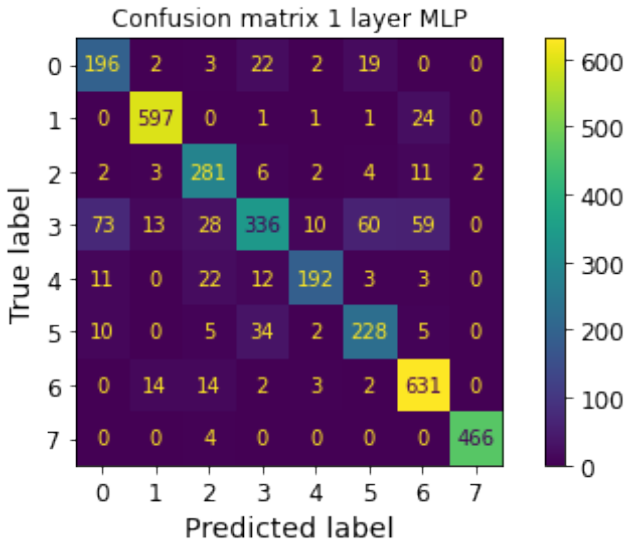


Figura 5: Matriz de confusão para os dados de teste para a MLP de uma camada intermediária.

É possível observar que a grande maioria dos dados de teste estão na diagonal principal, o que significa que a maioria dos dados foram previstos nas classes corretas.

4 MLP - duas camadas intermediárias

Para a MLP de duas camadas, foi feito praticamente o mesmo procedimento da MLP de uma camada, mas, adicionalmente, foi feita a escolha do número de neurônios da segunda camada intermediária. Pelas mesmas razões expostas na MLP de uma camada, escolheu-se para MLP de duas camadas intermediárias: *SparseCategoricalCrossentropy* como função custo; ReLU como função de ativação em ambas as camadas intermediárias; Adam como otimizador; realização do balanceamento das classes; uso *early stopping* na busca de hiperparâmetros.

4.1 *Keras Tuner*

Para a busca de hiperparâmetros, fez-se o ajuste dos mesmos parâmetros da MLP de uma camada, mas também testou-se o uso de *dropout* (regularização, pois há maior possibilidade de *overfitting* por haver mais neurônios). Os hiperparâmetros foram testados entre as seguintes opções:

- **número de neurônios da primeira camada intermediária**: entre 200 e 2000.
- **número de neurônios da segunda camada intermediária**: entre 200 e 2000.
- **passo de adaptação (*learning rate*)**: entre $1 \cdot 10^{-4}$ e $1 \cdot 10^{-2}$, valores baseados no item anterior e por ser a faixa que apresentou melhores resultados nos testes iniciais.
- **número de épocas para treinamento**: conforme MLP de uma camada intermediária.
- **regularização**: adicionalmente, testou-se se a acurácia da validação era maior utilizando-se ou não *dropout* na segunda camada intermediária.

Mais uma vez, utilizou-se o *Bayesian Optimizer* e, após a busca, encontrou-se os seguintes hiperparâmetros ótimos, com acurácia máxima na validação de **0.8966**: **1800** neurônios na primeira camada intermediária, **600** na segunda, *learning rate* de **0.0001210** e uso de *dropout*. É interessante perceber que, na combinação de hiperparâmetros ótima encontrada, a primeira camada intermediária tem uma quantidade de neurônios muito maior do que a segunda, exatamente conforme recomendado pelo livro-texto (GÉRON, 2019).

4.2 Treinamento e Teste

Com os hiperparâmetros escolhidos, construiu-se a MLP de duas camadas intermediárias da Figura 6, com 1800 neurônios na primeira camada intermediária, 600 na segunda, com *dropout* na segunda camada e função de ativação ReLU em ambas. Assim, foi realizado seu treinamento com 100 épocas e *learning rate* de 0.0001210.

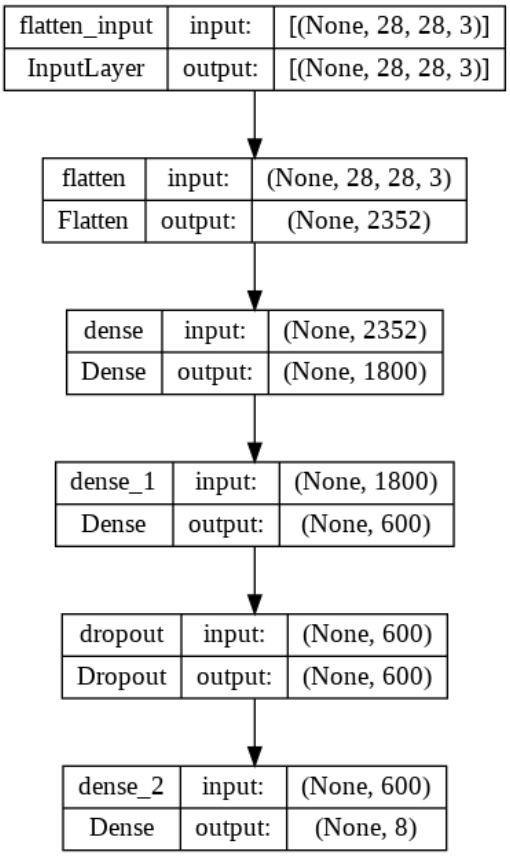


Figura 6: Modelo da MLP de duas camadas intermediárias.

No treinamento, ainda utilizou-se o conjunto de validação para observação da acurácia e perda em função do número de épocas (Figura 7). Note que isso foi feito de forma a se analisar melhor o efeito do número de épocas na acurácia e perda, além de se verificar se o *learning rate* estava adequado e se houve *overfitting*.

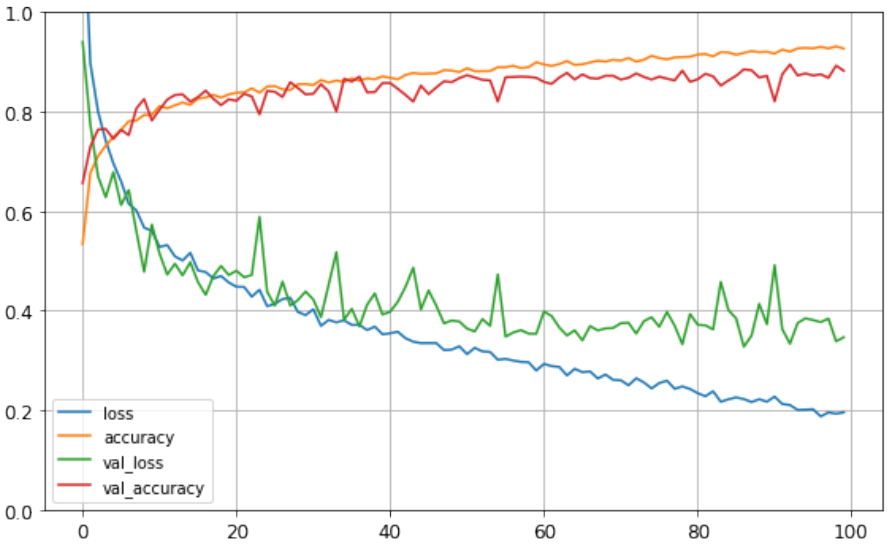


Figura 7: Treinamento da MLP de duas camadas intermediárias - acurácia e perda (treinamento e validação) em função do número de épocas.

4.3 Análise de Desempenho

Após o treinamento, foi feita a classificação do conjunto de teste, tal que analisou-se o desempenho do modelo observando-se a acurácia global e a matriz de confusão.

4.3.1 Acurácia Global

A acurácia global para os dados de teste foi obtida pela função *evaluate* com os dados de teste (x teste, y teste). Foi obtida uma acurácia global de **0.8758** para os dados de teste. Note que, como esperado, a MLP de duas camadas obteve uma acurácia maior que a de uma camada intermediária.

Demais métricas para os dados de teste estão na figura 8. Essas métricas foram obtidas após a previsão dos valores nos dados de teste com o uso do *classification report* do *Scikit Learning*.

	precision	recall	f1-score	support
0	0.79	0.68	0.73	244
1	0.96	0.96	0.96	624
2	0.83	0.92	0.87	311
3	0.76	0.79	0.77	579
4	0.90	0.75	0.82	243
5	0.78	0.75	0.77	284
6	0.91	0.94	0.92	666
7	1.00	0.99	0.99	470
accuracy			0.88	3421
macro avg	0.86	0.85	0.85	3421
weighted avg	0.88	0.88	0.87	3421

Figura 8: Demais métricas de desempenho de teste para a MLP de duas camadas intermediárias.

4.3.2 Matriz de confusão

Com o modelo ótimo feito, foi feita a previsão para os dados de teste e obtida a matriz de confusão a partir da função *Confusion Matrix* do *Scikit Learning*.

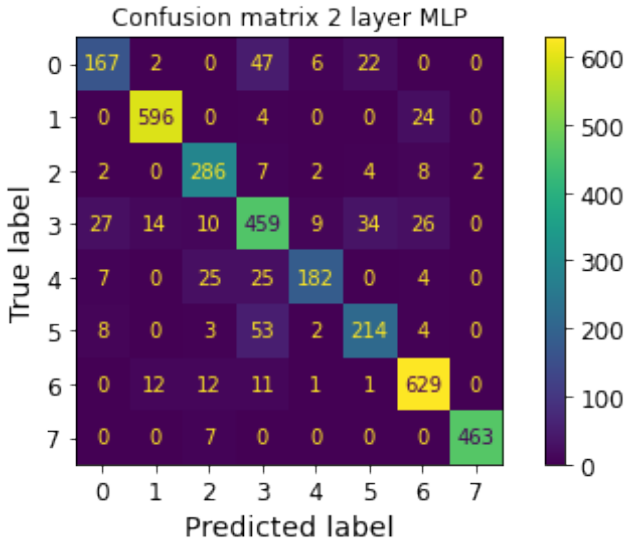


Figura 9: Matriz de confusão para os dados de teste para a MLP de duas camadas intermediárias.

Mais uma vez, é possível observar que a grande maioria dos dados de teste estão na diagonal principal, o que significa que a maioria dos dados foram previstos nas classes corretas.

5 CNN simples

Neste item, foi montada uma rede neural convolucional (CNN) simples a partir da *Keras API* do *TensorFlow*. A CNN simples foi construída contendo uma camada convolucional com função de ativação, uma camada de *pooling* e uma camada se saída do tipo *softmax*. Foram utilizadas para a construção da CNN as funções *Sequential*, *Conv2D*, *MaxPool2D*, *Flatten*, *Dense*, da *Keras API*.

5.1 Acurácia de validação em função do número de *kernels*

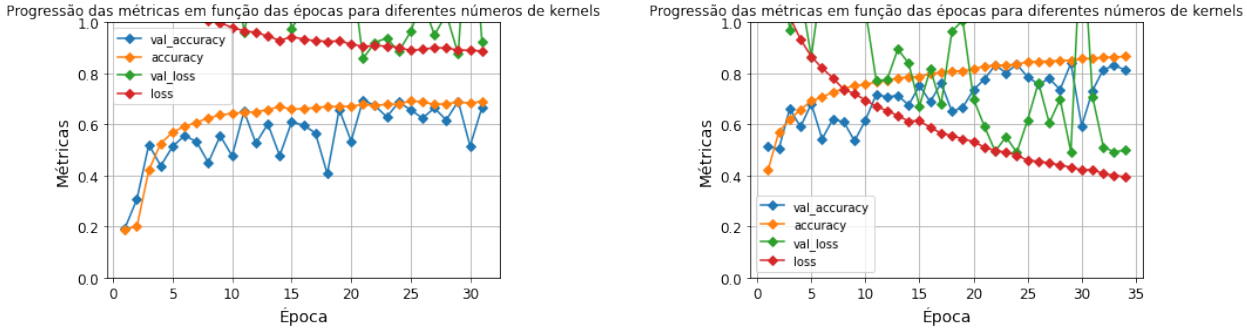
Primeiramente, foi feita uma avaliação da progressão da acurácia junto aos dados de validação em função do número de *kernels* utilizados na camada convolucional. Para isso, todos os outros parâmetros foram fixados de forma arbitrária, apenas para observar a evolução das métricas com a variação do número de *kernels* (filtros). O tamanho do *kernel* de convolução foi fixado em 3x3, o *stride* de convolução foi fixado como 1, foi adotado um *zero padding*, uma inicialização dos pesos definida como *he uniform* e a função de ativação utilizada foi a *ReLu*. Os demais parâmetros de *Conv2D* foram fixados nos seus valores padrões. Para a camada de *pooling*, o tamanho da janela foi fixado em 2x2, o *stride* foi fixado em 1 e não foi adotado *zero padding*.

Antes da camada densa *softmax*, foi utilizada uma camada *Flatten*, para transformar os dados que saíram da camada convolucional em um formato de uma dimensão, para serem compatíveis com a camada densa.

Para a compilação do modelo, foi utilizada a função *compile*, com o otimizador *SGD*, função custo *SparseCategoricalCrossentropy* e a acurácia como métrica. A função custo *SparseCategoricalCrossentropy* foi escolhida por ser feita para problemas com mais de uma classe, com rótulos no formato inteiro.

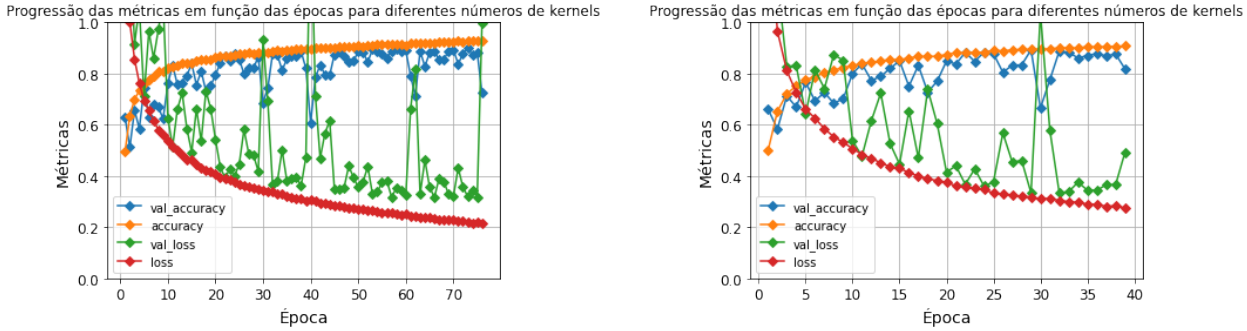
O modelo foi treinado com a função *fit* da *Keras API*, com um *batch size* fixado em 50 e com 100 épocas. Além disso, foi adicionado um *early stopping*, para detectar *overfitting* e evitar processamentos desnecessários. Para o treinamento, a variável que representa o número de *kernels* de convolução foi variada com os valores 1, 5, 10, 15, 20, 30, 40, 50, 100, 150, 200, 250, 350, 500. Os modelos ao longo do treinamento foram salvos pela função *history*, e com isso, foram feitos os gráficos das métricas em função das épocas para cada valor de número de *kernels*, além do gráfico da evolução da melhor acurácia em função do número de *kernels* de convolução. As figuras 10 a 16 mostram a progressão das métricas em função das épocas para cada número de *kernels* de convolução. As figuras 17 e 18 mostram a progressão da acurácia de validação e da função custo de validação em função do número de *kernels*, junto com as melhores épocas obtidas.

É possível perceber, pelas figuras 17 e 18, que para poucos *kernels*, há valores ruins de acurácia para os dados de validação. Em torno de 100 *kernels* foi obtida a melhor acurácia de validação e o melhor valor da função custo. A partir de 100 *kernels*, o valor da acurácia de validação tende a ficar próximo da acurácia obtida para 100 *kernels*. Com mais *kernels*, mais informação é retirada de cada imagem, porém há um limite, sendo que para muitos *kernels* os resultados tendem a piorar, podendo ocorrer *overfitting*.



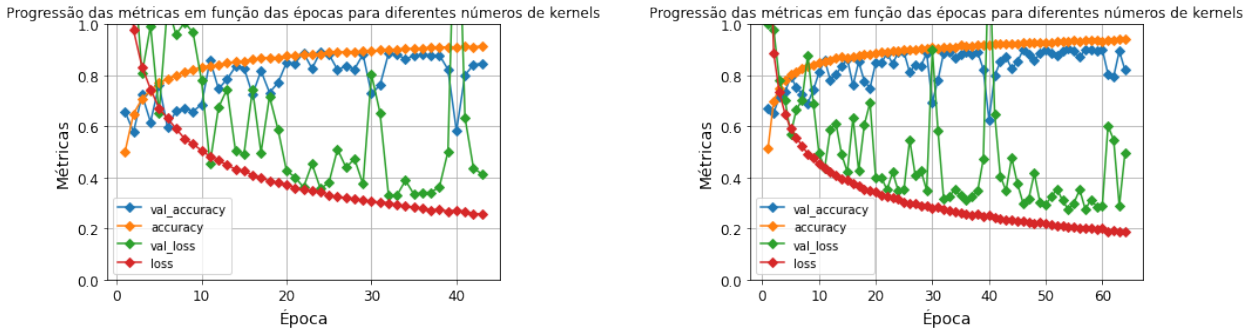
(a) (b)

Figura 10: Em (a), a progressão das métricas através das épocas para 1 *kernel* e, em (b), a progressão das métricas através das épocas para 5 *kernels*.



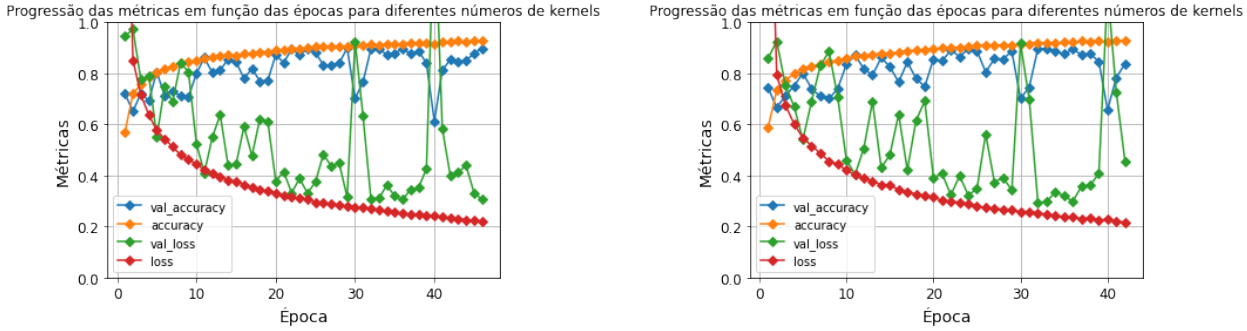
(a) (b)

Figura 11: Em (a), a progressão das métricas através das épocas para 10 *kernels* e, em (b), a progressão das métricas através das épocas para 15 *kernels*.



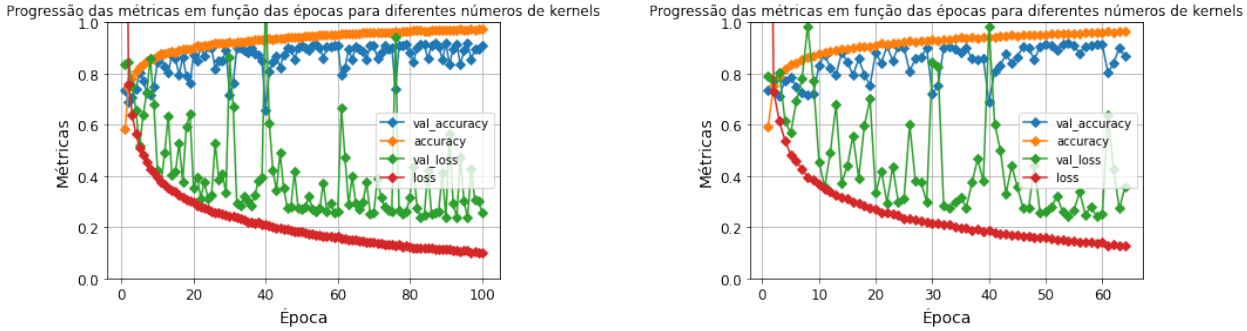
(a) (b)

Figura 12: Em (a), a progressão das métricas através das épocas para 20 *kernels* e, em (b), a progressão das métricas através das épocas para 30 *kernels*.



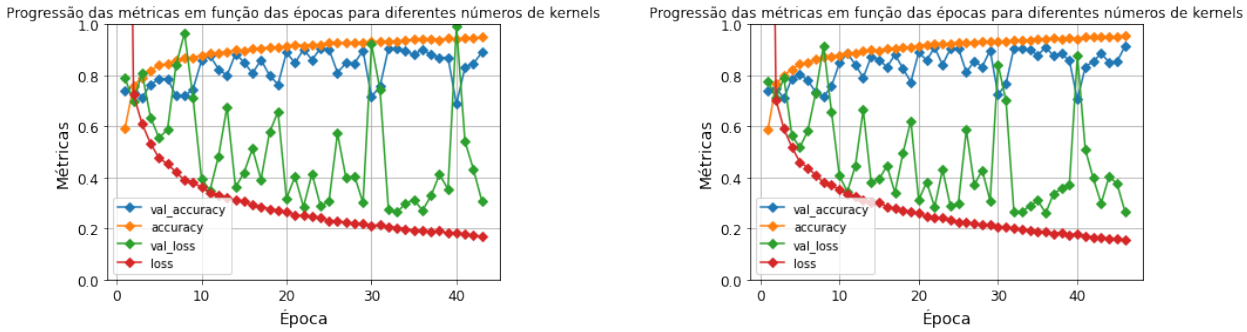
(a) (b)

Figura 13: Em (a), a progressão das métricas através das épocas para 40 *kernels* e, em (b), a progressão das métricas através das épocas para 50 *kernels*.



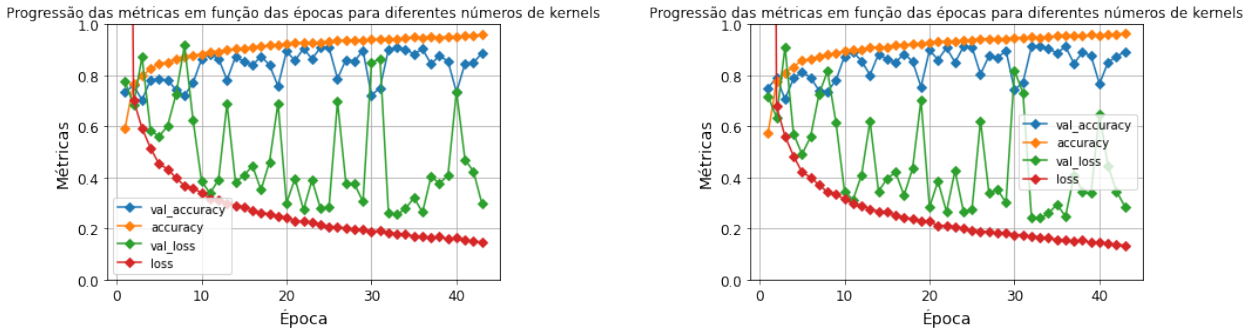
(a) (b)

Figura 14: Em (a), a progressão das métricas através das épocas para 100 *kernels* e, em (b), a progressão das métricas através das épocas para 150 *kernels*.



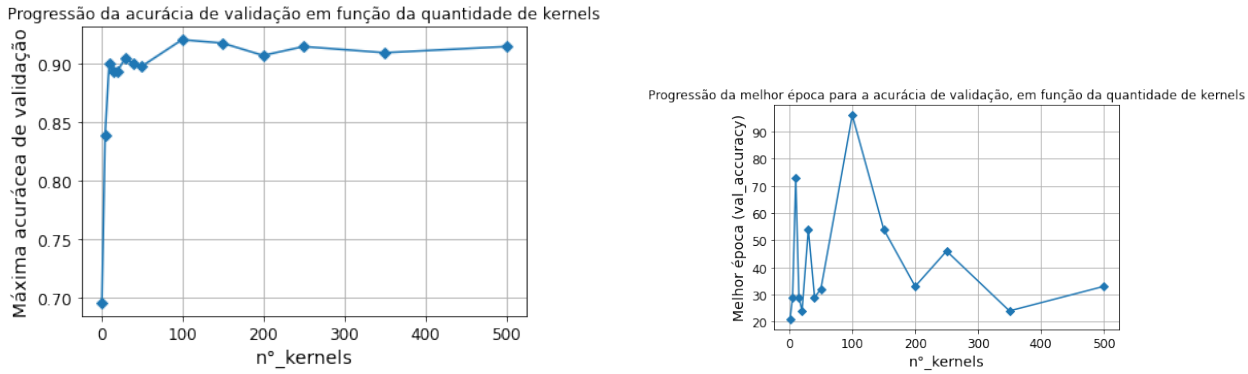
(a) (b)

Figura 15: Em (a), a progressão das métricas através das épocas para 200 *kernels* e, em (b), a progressão das métricas através das épocas para 250 *kernels*.



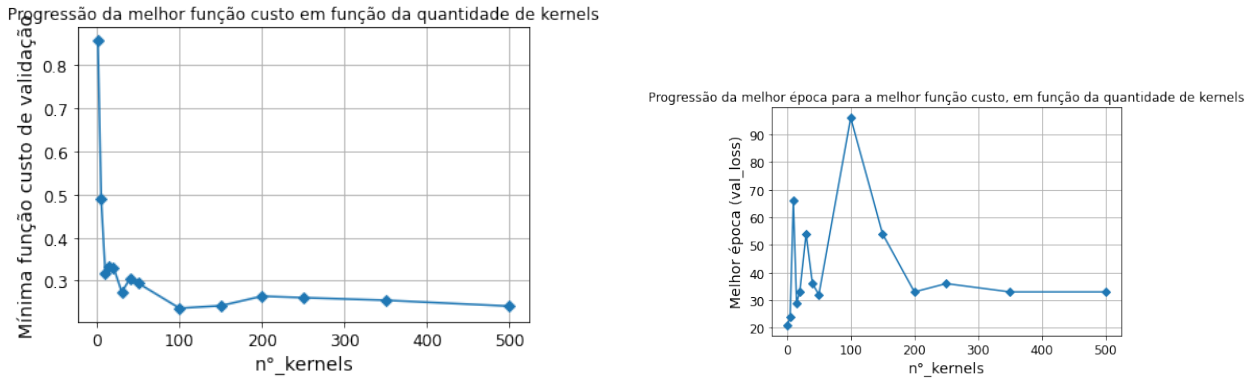
(a) (b)

Figura 16: Em (a), a progressão das métricas através das épocas para 350 *kernels* e, em (b), a progressão das métricas através das épocas para 500 *kernels*.



(a) (b)

Figura 17: Em (a), a progressão da melhor acurácia de validação em função do número de *kernels* e, em (b), a melhor época.



(a) (b)

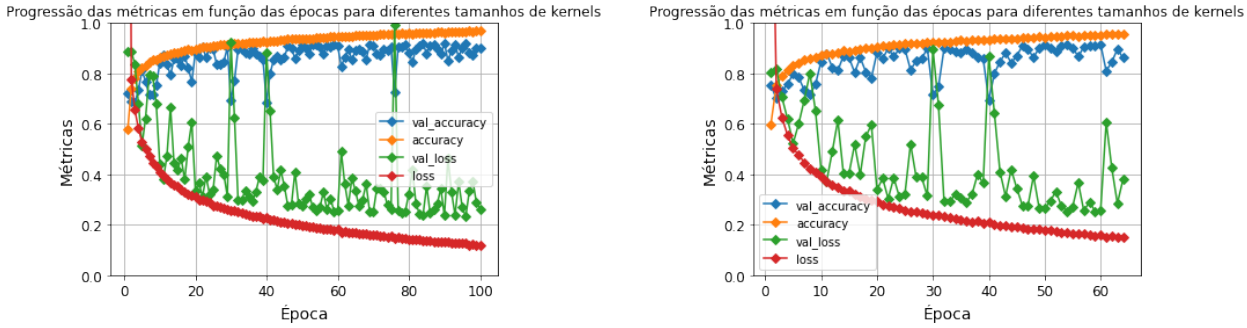
Figura 18: Em (a), a progressão do melhor custo de validação em função do número de *kernels* e, em (b), a melhor época.

5.2 Acurácia de validação em função do tamanho do *kernel*

Neste item, a avaliação da progressão da acurácia de validação foi refeita, porém, agora esta métrica foi avaliada em função do tamanho do *kernel* de convolução. Para isso, os parâme-

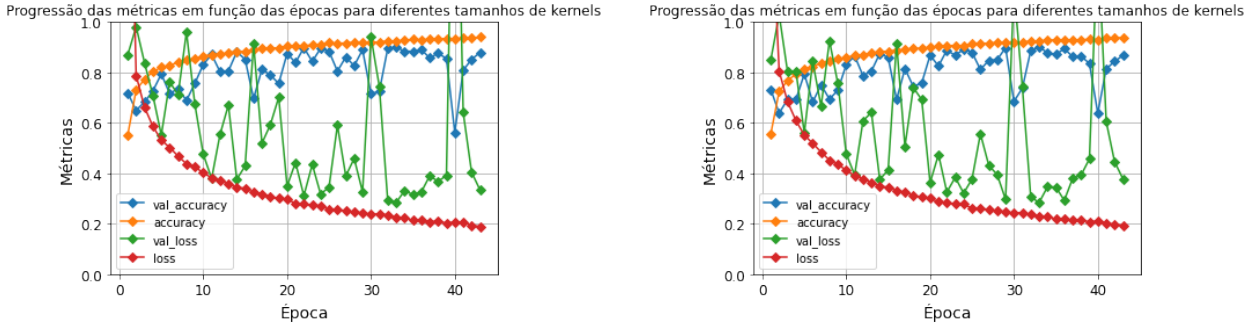
tros fixados no item anterior foram mantidos, o número de *kernels* de convolução foi fixado em 100, que foi o valor ótimo obtido anteriormente, e o treinamento foi feito variando o tamanho do *kernel* de convolução para os valores 2x2, 3x3, 5x5, 7x7, 9x9, 11x11. As figuras 19 a 21 mostram a progressão das métricas em função das épocas para cada tamanho do *kernel* de convolução. As figuras 22 e 23 mostram a progressão da acurácia de validação e da função custo de validação em função do tamanho do *kernel*, junto com as melhores épocas obtidas.

É possível perceber, pelas figuras 22 e 23, que para *kernels*, pequenos, há valores muito bons de acurácia para os dados de validação. Para *kernels* 2x2, foi obtida a melhor acurácia de validação e o melhor valor da função custo. A partir de *kernels* 4x4, o valor da acurácia de validação possui uma piora. É possível observar também nas figuras 19 a 21, que para *kernels* muito grandes, a acurácia de validação possui uma maior oscilação de valores, conforme a progressão das épocas. Para *kernels* pequenos e *strides* pequenos, há um detalhamento maior dos detalhes da imagem. Para este caso, como as imagens possuem dimensão pequena (28x28), *kernels* pequenos são o suficiente para detalhar a imagem e conseguir um melhor desempenho.



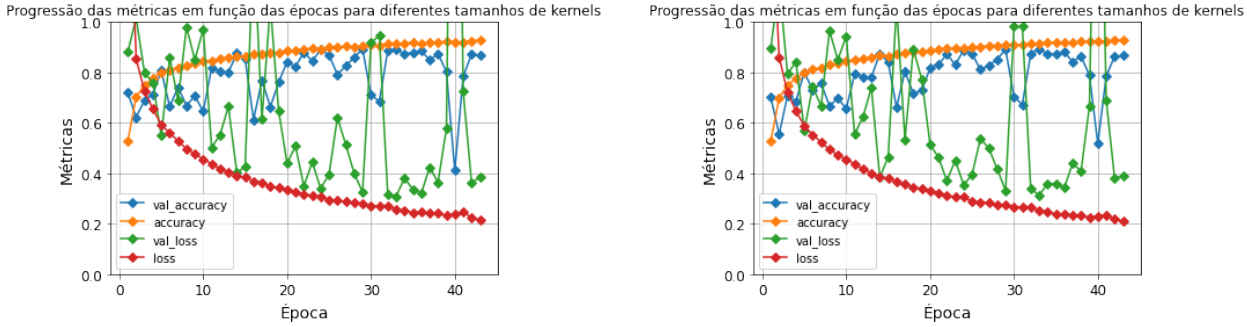
(a) (b)

Figura 19: Em (a), a progressão das métricas através das épocas para *kernel* 2x2 e, em (b), a progressão das métricas através das épocas para *kernels* 3x3.



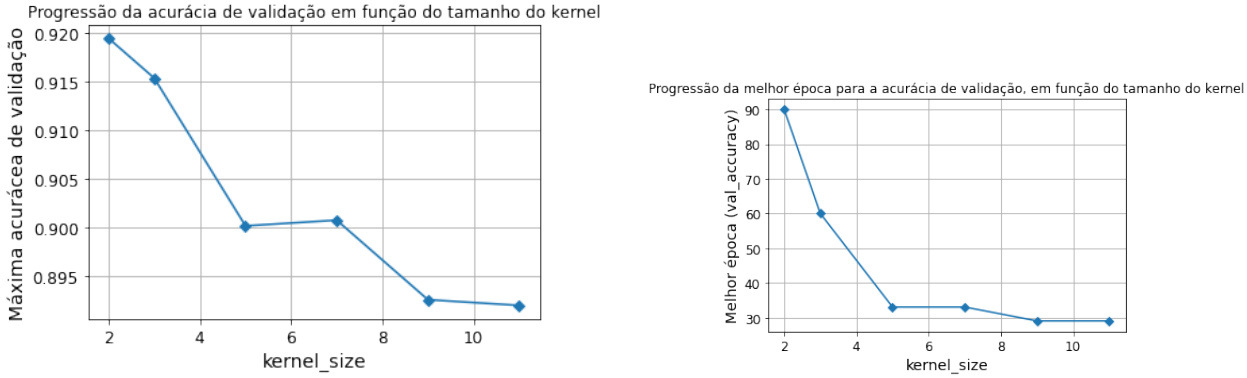
(a) (b)

Figura 20: Em (a), a progressão das métricas através das épocas para *kernel* 5x5 e, em (b), a progressão das métricas através das épocas para *kernels* 7x7.



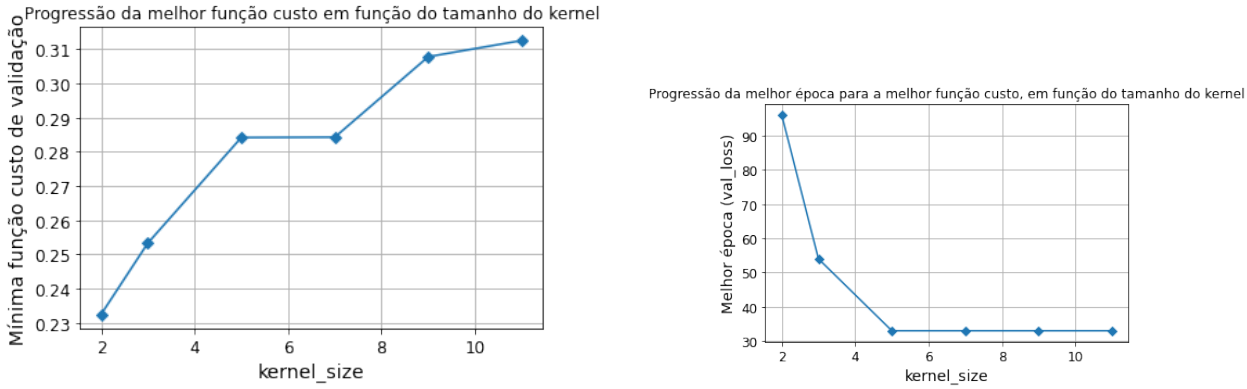
(a) (b)

Figura 21: Em (a), a progressão das métricas através das épocas para *kernel* 9x9 e, em (b), a progressão das métricas através das épocas para *kernels* 11x11.



(a) (b)

Figura 22: Em (a), a progressão da melhor acurácia de validação em função do tamanho do *kernel* e, em (b), a melhor época.



(a) (b)

Figura 23: Em (a), a progressão do melhor custode validação em função do tamanho do *kernel* e, em (b), a melhor época.

5.3 Validação cruzada e escolha dos hiperparâmetros ótimos

Para encontrar os parâmetros ótimos, foi utilizada a biblioteca *KerasTuner*, que permite aplicar algoritmos de otimização nos hiperparâmetros selecionados, de forma à observar os hiperparâmetros ótimos com relação à alguma métrica. Neste caso, a métrica utilizada para a otimização foi a acurácia de validação.

Na primeira busca, o número de filtros de convolução foi variado em torno de 100, que foi o valor ótimo encontrado no item anterior. Foram testados treinamentos com 90, 100 e 110 filtros. O tamanho do *kernel* de convolução foi variado em torno de 2, que foi o valor ideal encontrado no item anterior. Foram testados treinamentos com tamanhos de filtros 1, 2 e 3. O *stride* de convolução foi variado entre os valores 1 e 2. O *Learning Rate* foi variado na faixa de 0.0001 até 0.01 com amostragem definida para o tipo "log", para todos os valores na faixa possuírem probabilidade igual. Os demais hiperparâmetros mantiveram iguais aos utilizados nos itens anteriores. O tipo de otimização utilizado foi a *Bayesian Optimization*, por ser um otimizador que não precisa testar todas as combinações possíveis, pois a escolha dos próximos melhores hiperparâmetros é feita com base nos melhores anteriores. Este tipo de busca por hiperparâmetros é eficiente e mais rápida do que uma busca aleatória ou exaustiva. O otimizador foi ajustado para monitorar a acurácia de validação e com um máximo de 10 tentativas.

O resultado desta primeira busca, encontrou uma acurácia de validação ótima de 0.916, com 90 *kernels* de convolução, filtro de convolução de tamanho 3x3, *Learning Rate* de 0.01 e *stride* 1. Foi utilizado *Earling Stopping* no treinamento com tolerância 10.

Na segunda busca, o números de filtros foi fixado em 90, o tamanho do filtro de convolução foi fixado em 3x3, e o *stride* de convolução fixado em 1. Estes foram os valores ótimos da busca anterior. Nesta nova busca, foi variado o otimizador entre SGD e Adam, o tamanho do filtro de *pooling*, com valores 2, 3 e 4, e o *Learning Rate* com valores 0.1, 0.01 e 0.001. As mesmas configurações foram aplicadas ao otimizador *Bayesian Optimization*. Foi utilizado *Earling Stopping* no treinamento com tolerância 10.

O resultado desta segunda busca, encontrou uma acurácia de validação de 0.9339, com filtro de *pooling* 4x4, otimizador Adam com *Learning Rate* 0.001.

5.4 Treinamento final e resultados para o conjunto de testes

Com os hiperparâmetros ótimos obtidos, foi montado o modelo da CNN simples utilizando: 90 filtros de convolução de tamanho 3x3, com *stride* unitário, com *zero padding*, função de ativação ReLu, inicialização dos pesos *he uniform*, *bias* iniciando em zero, filtros de *pooling* 4x4, com *stride* unitário, sem *padding*, otimizador Adam, com *Learning Rate* de 0.001 e 200 épocas.

Com este modelo final, foram medidas algumas métricas de desempenho no conjunto de teste. O modelo final está na figura 24. As escolhas dos valores do número de filtros de convolução, do tamanho do filtro de convolução, e do *stride* de convolução foram feitas com base nas otimizações de hiperparâmetros. A função de ativação escolhida foi a ReLu, por ser uma função popular e apresentar bons resultados de acurácia. A inicialização dos pesos escolhida foi do tipo *he uniform*, por funcionar muito bem com a função de ativação ReLu. O tamanho do filtro de *pooling* e o tipo do otimizador foram definidos na busca pelos melhores hiperparâmetros, assim como o valor do *Learning Rate*. A função custo foi escolhida por ser a mais adequada para classificação multi classe com classes inteiras. Não foi utilizado *Early Stopping* nem regularização no treinamento final, pois testes isolados com essas inclusões não mostraram melhora nos resultados. Foi utilizado no treinamento um *batch size* de 50 e 200

épocas. O valor das épocas foi definido com a observação da acurácia de validação. Para 200 épocas, o resultado melhorou um pouco com relação a 100 épocas. Não foi observado *overfitting*, pois não houve grande piora no desempenho nas últimas épocas.

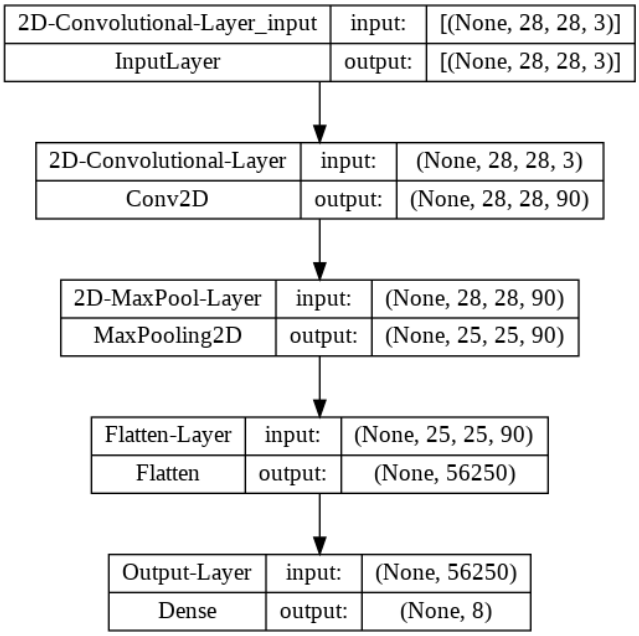


Figura 24: Modelo ótimo obtido para a CNN simples.

5.4.1 Acurácia Global

A acurácia global para os dados de teste foi obtida pela função *evaluate* com os dados de teste (x teste, y teste). Foi obtida uma acurácia global de 0.9129 para os dados de teste. Demais métricas para os dados de teste estão na figura 25. Essas métricas foram obtidas após a previsão dos valores nos dados de teste com o uso do *classification report* do *Scikit Learning*.

```
107/107 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.88      0.84      0.86       244
           1       0.98      0.97      0.98       624
           2       0.94      0.92      0.93       311
           3       0.80      0.83      0.82       579
           4       0.91      0.91      0.91       243
           5       0.78      0.79      0.79       284
           6       0.95      0.95      0.95       666
           7       1.00      0.99      0.99       470

    accuracy                           0.91      3421
   macro avg       0.90      0.90      0.90      3421
weighted avg       0.91      0.91      0.91      3421
```

Figura 25: Demais métricas de desempenho de teste.

5.4.2 Matriz de confusão

Com o modelo ótimo feito, foi feita a previsão para os dados de teste e obtida a matriz de confusão a partir da função *Confusion Matrix* do *Scikit Learning*.

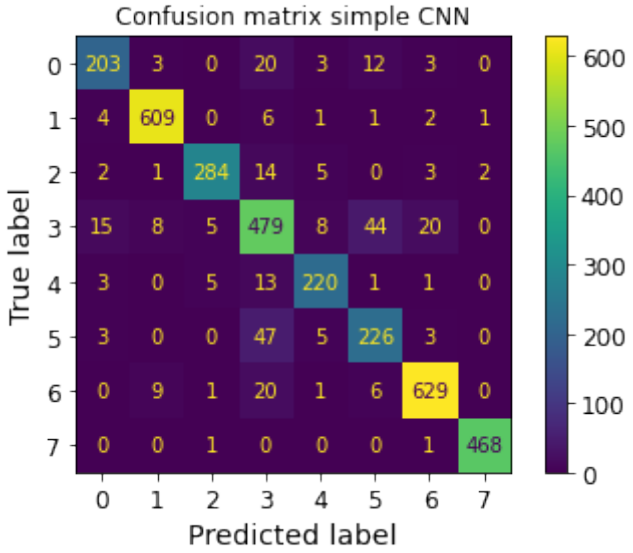


Figura 26: Matriz de confusão para os dados de teste para a CNN simples.

É possível observar que a grande maioria dos dados de teste estão na diagonal principal, o que significa que a maioria dos dados foram previstos nas classes corretas. É possível observar também , que há mais classes classificadas corretamente com relação à MLP de uma camada e à MLP de duas camadas. A classe em que houve mais acertos foi a classe 6 e a classe em que houve menos acertos foi a classe 0.

5.4.3 Cinco padrões de teste que foram classificados incorretamente

Após a previsão realizada pela CNN simples, foram verificados os cinco primeiros padrões de teste classificados incorretamente, observando a classe esperada, a classe prevista, e as probabilidades da previsão. As figuras 27 a 31 mostram as cinco primeiras previsões incorretas, junto com as probabilidades, a classe esperada e a classe prevista.

Figura 27: Primeira amostra do conjunto de teste.

Figura 28: Amostra 25 do conjunto de teste.

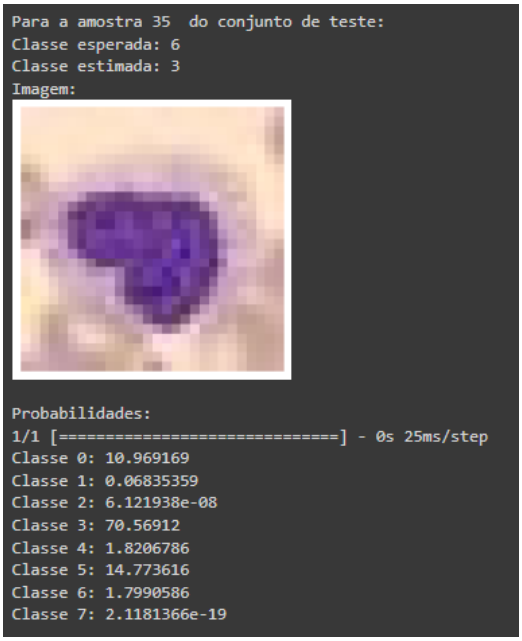


Figura 29: Amostra 35 do conjunto de teste.

Figura 30: Amostra 48 do conjunto de teste.

Figura 31: Amostra 50 do conjunto de teste.

É possível observar que as probabilidades das classes que eram esperadas estão dentre as maiores probabilidades obtidas. Em alguns casos, a probabilidade da classe estimada incorretamente foi muito elevada, em outros, houve uma maior dúvida na escolha ao possuir uma probabilidade não muito alta para a classe incorreta, e uma probabilidade significativa para a classe correta. Isso mostra que o modelo não errou tanto em alguns casos. É possível observar também que os 5 primeiros erros estão entre as 50 amostras.

6 CNN profunda inspirada na ResNet

Foi montada uma CNN mais profunda, inspirada na ResNet. Esta rede foi montada utilizando uma camada convolucional na entrada de 64 filtros 3x3 com *stride* unitário e sem *padding*, seguida de uma camada de *batch normalization* e função de ativação ReLu. A saída deste bloco vai para uma camada de *Max pooling*,com filtros 3x3, *stride* 2, e com *zero padding*. Essas primeiras camadas são inspiradas em modelos de ResNet para imagens 28x28. No modelo das ResNet completa, após essas primeiras camadas, existem diversas camadas com unidade residual. A unidade residual, é o grande diferencial da ResNet, e está mostrada na figura 32.

Para o modelo deste exercício, foram utilizadas três unidades residuais com 64, 128 e 256 filtros. Após as unidades residuais, foi implementada uma camada de *Global Average Pooling*, seguida de uma camada *Flatten* para prepara os dados para entrarem nas camadas densas. Foi feita uma camada densa com 1000 neurônios, e função de ativação ReLu, seguida pela camada densa de 8 neurônios com função de ativação *softmax*.

6.1 Validação cruzada e escolha dos hiperparâmetros ótimos

Os parâmetros da rede foram escolhidos com base nas redes profundas mais comumente encontradas. A principal inspiração para a escolha dos hiperparâmetros foi a ResNet 34 (GÉRON, 2019). Com isso, foi feito o treinamento com o otimizador SGD, função custo *Sparse Categorical Crossentropy*, com 100 épocas e *batch size* de 50. Primeiro foi feito o treinamento sem regularização e *Early Stopping*, e foi observado uma acurácia de validação em torno de 0.92, que é um valor ruim, muito próximo da acurácia da CNN simples. Adicionando o *Early Stopping*, os resultados do treinamento melhoraram muito. O uso de regularizações não melhorou muito os resultados, sendo o *Early Stopping* suficiente para uma boa acurácia de validação.

Foi feito o teste com o otimizador Adam, e os resultados não aumentaram muito. Neste item, devido à complexidade, o treinamento com a variação de hiperparâmetros, foi extremamente custoso e demorado. Devido à isso, foram testados valores mais comuns de serem utilizados, e com testes manuais foi obtido acurácias de validação acima de 0.97.

6.2 Treinamento final e resultados para o conjunto de testes

O modelo final obtido para a CNN profunda está na figura 32. Com isso, foram obtidas as métricas para o conjunto de teste com a função *Evaluate*, e com as previsões no conjunto de teste.

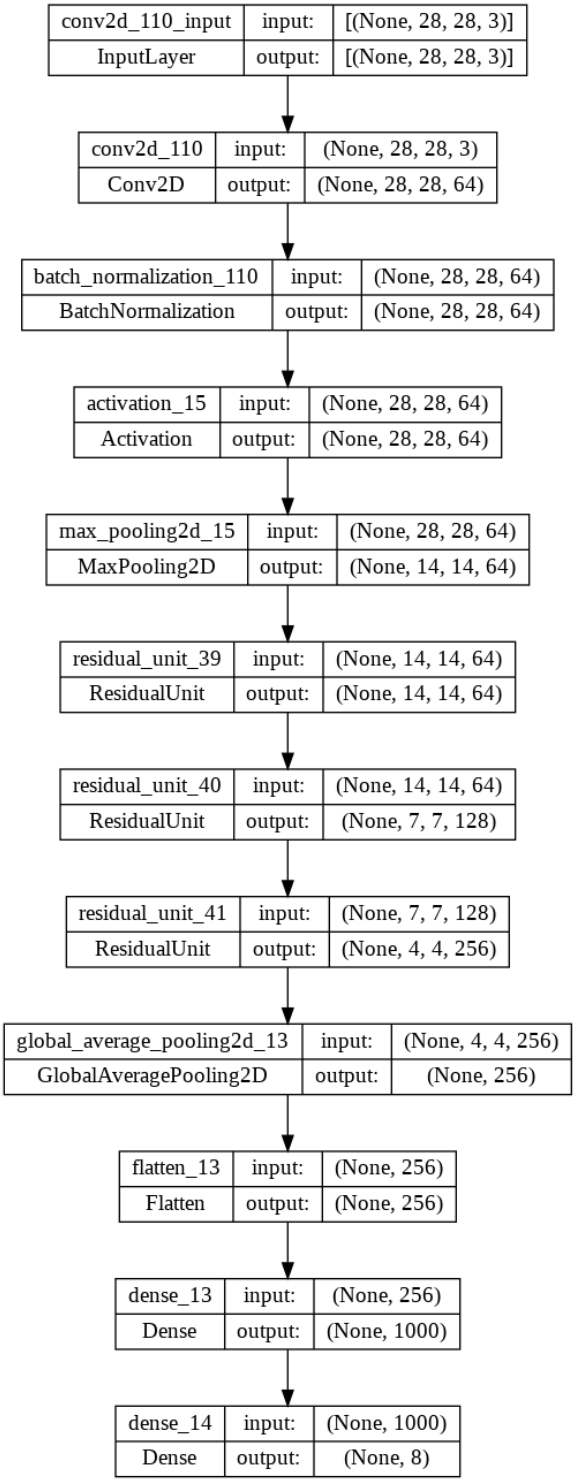


Figura 32: Modelo da ResNet simplificada utilizada.

6.2.1 Acurácia Global

A acurácia global para os dados de teste foi obtida pela função *evaluate* com os dados de teste (x teste, y teste). Foi obtida uma acurácia global de 0.9547 para os dados de teste. Demais métricas para os dados de teste estão na figura 33. Essas métricas foram obtidas após a previsão dos valores nos dados de teste com o uso do *classification report* do *Scikit Learning*.

```
107/107 [==============================] - 1s 4ms/step
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       244
           1       1.00      0.99      0.99       624
           2       0.97      0.95      0.96       311
           3       0.89      0.89      0.89       579
           4       0.95      0.95      0.95       243
           5       0.91      0.89      0.90       284
           6       0.96      0.97      0.97       666
           7       1.00      1.00      1.00       470

    accuracy                           0.95      3421
   macro avg       0.95      0.95      0.95      3421
weighted avg       0.95      0.95      0.95      3421
```

Figura 33: Métricas obtidas com a ResNet simplificada.

6.2.2 Matriz de confusão

Com o modelo ótimo feito, foi feita a previsão para os dados de teste e obtida a matriz de confusão a partir da função *Confusion Matrix* do *Scikit Learning*.

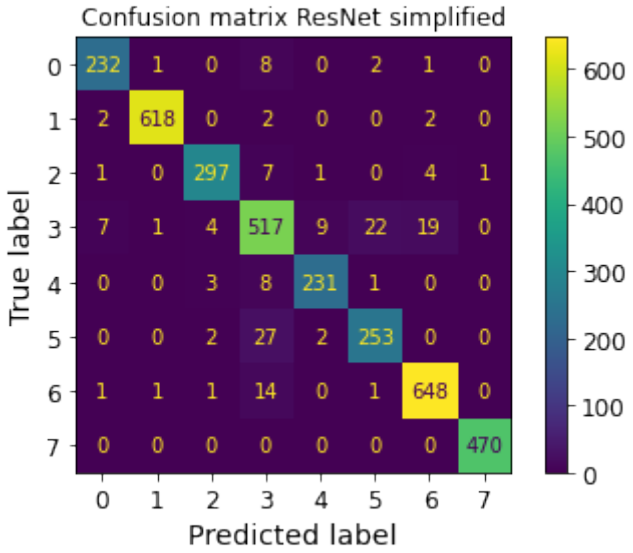


Figura 34: Matriz de confusão para os dados de teste para a CNN profunda.

É possível observar que a grande maioria dos dados de teste estão na diagonal principal, o que significa que a maioria dos dados foram previstos nas classes corretas. É possível observar também, que há mais classes classificadas corretamente com relação à MLP de uma camada, à MLP de duas camadas e a CNN simples. A classe em que houve mais acertos foi a classe 6 e a classe em que houve menos acertos foi a classe 4. É possível observar uma melhora progressiva da matriz de confusão para cada modelo deste exercício, sendo a CNN profunda com os melhores resultados.

6.2.3 Cinco padrões de teste que foram classificados incorretamente

Após a previsão realizada pela CNN simples, foram verificados os cinco primeiros padrões de teste classificados incorretamente, observando a classe esperada, a classe prevista, e as probabilidades da previsão. As figuras 35 a 39 mostram as cinco primeiras previsões incorretas, junto com as probabilidades, a classe esperada e a classe prevista.

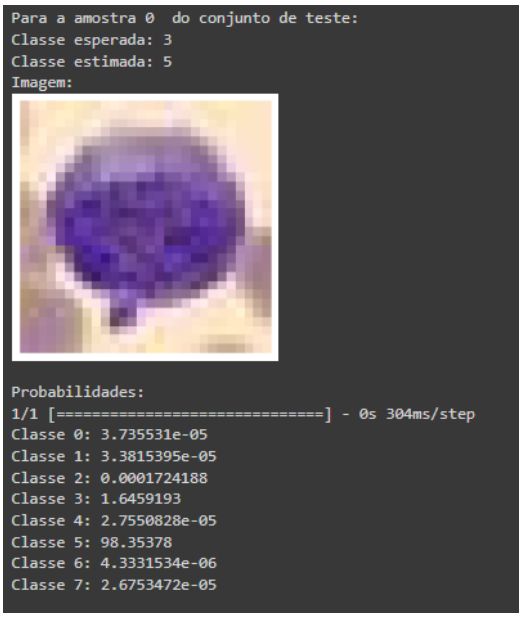


Figura 35: Primeira amostra do conjunto de teste.

Figura 36: Amostra 8 do conjunto de teste.

Figura 37: Amostra 65 do conjunto de teste.

Figura 38: Amostra 102 do conjunto de teste.

Figura 39: Amostra 118 do conjunto de teste.

É possível observar que as probabilidades das classes que eram esperadas estão dentre as maiores probabilidades obtidas. Em alguns casos, a probabilidade da classe estimada incorretamente foi muito elevada, em outros, houve uma maior dúvida na escolha ao possuir uma probabilidade não muito alta para a classe incorreta, e uma probabilidade significativa para a classe correta. Isso mostra que o modelo não errou tanto em alguns casos. É possível observar também que os 5 primeiros erros estão entre as 118 amostras. Isso mostra que há uma menor densidade de amostras previstas incorretamente com relação à CNN simples. Se observar o total de amostras erradas, é possível perceber que há menos amostras classificadas incorretamente com relação ao caso anterior.

Referências

ACEVEDO, Andrea et al. A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. **Data in brief**, Elsevier, v. 30, 2020.

GÉRON, A. **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow**. [S.l.]: Alta Books, 2019. ISBN 9788550809021. Disponível em: <https://books.google.com.br/books?id=Z0mvDwAAQBAJ>.

KERAS. ***Probabilistic losses***. [S.l.: s.n.]. https://keras.io/api/losses/probabilistic_losses/#sparsecategoricalcrossentropy-class [Accessed: 09.11.2022].

YANG, Jiancheng et al. MedMNIST v2: A Large-Scale Lightweight Benchmark for 2D and 3D Biomedical Image Classification. **arXiv preprint arXiv:2110.14795**, 2021.