Aprendizagem Profunda Aplicada – 2022/2023

Assignment 1

Duarte Sousa Cruz
2017264087

Gonçalo Tomaz Arsénio
2017246034

November 3, 2022

1 Intro

Este trabalho visa construir um modelo de uma CNN (Convolutional Neural Network) e fazer a sua avaliação usando dois datasets previamente disponíveis. Tem também como finalidade aumentar a percentagem de classificação sem denegrir muito a sua desempenho, para tal é necessário reajustar alguns parâmetros, modificar a rede e fazer diversos testes para comparar que conjunto de parâmetros maximizam a sua performance.

2 Parte I

Nesta primeira parte do trabalho foi nos fornecido uma *baseline network*, sendo objetivo alterá-la de forma a obter *mCA* de, pelo menos, 75% - 80% no dataset CIFAR-10.

2.1 Arquitetura CNN

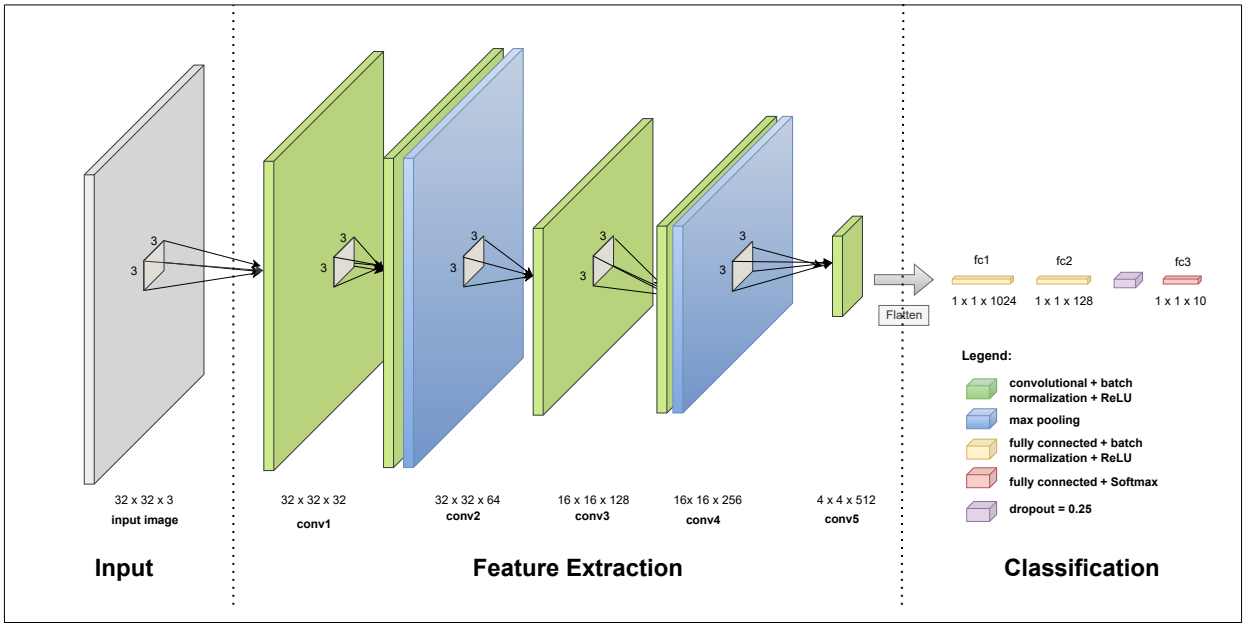

Figure 1: Arquitetura da rede neural convolucional implementada

Tendo como ponto de partida a rede neural fornecida no enunciado, começamos por adicionar mais camadas convolucionais e observando ter um impacto bastante positivo no comportamento da rede, no entanto, não podiam ser adicionadas mais do que cinco camadas convolucionais. De seguida, alteramos a profundidade das camadas, aumentando de camada para camada, e como consequência aumentamos também o número de neurónios das camadas *fully connected*. Apesar da melhoria no desempenho, ainda não tínhamos atingido o patamar pretendido, sendo assim, adicionamos alguns métodos de regularização às nossas camadas, como *batch normalization*, que tornou a nossa rede mais rápida e mais estável através da normalização das camadas, e o *max pooling*, que reduziu o custo computacional ao diminuir o número de parâmetros usados no treino mantendo apenas a informação mais importante.

2.2 Conjunto de Dados

Nesta primeira parte do trabalho foi utilizado o dataset CIFAR-10, que para cada classe, possui 600 imagens RGB, onde 500 são usadas para o treino e 100 imagens são usadas para o teste.

Para obter um melhor desempenho, decidimos duplicar o dataset, aplicando diferentes transformações a cada conjunto para obter uma maior variação de dados para o treino. No primeiro conjunto aplica-se uma rotação e um inversão horizontal, no outro um corte da imagem aleatório e eliminação de aleatória de pequenos pedaços da imagem. No entanto, ambos sofrem uma normalização e é aplicada essa mesma normalização também ao conjunto de teste, garantindo assim que está tudo numa mesma escala sem se perder informação.

2.3 Análise e Comparação de Diferentes combinações

Conjunto	**Epochs**	**Batch Size**	**Learning Rate**	**Optimizer Algorithm**	**Loss Function**
1	25	32	0,0001	Adam	Cross Entropy
2	25	64	0,0001	Adam	Cross Entropy
3	25	32	Plateu	SGD	Cross Entropy
4	25	32	Plateu	Adam	Cross Entropy

Table 1: Conjunto de parâmetros usados

Combinação	**mCA**	**F1-Score**	**Precision**	**Recall**
1	88.20	94.35	96.88	93.75
2	86.87	90.48	91.07	92.86
3	87.48	83.81	87.50	85.71
4	89.09	89.76	95.83	87.50

Table 2: Métricas das combinações utilizadas

2.3.1 Combinação 1

Para a primeira combinação usámos o *learning rate* e número de épocas sugerido no enunciado. Para o algoritmo de otimização e para a função *loss* foi usado o *Adam* e o *Cross Entropy*, respetivamente, que já tinham sido usados nas aulas práticas.

Com esta combinação conseguiu-se obter ótimos resultados ultrapassando os 80% exigidos no enunciado, sendo a segunda combinação com melhor *accuracy* e a que obteve o melhor *F1-score*.

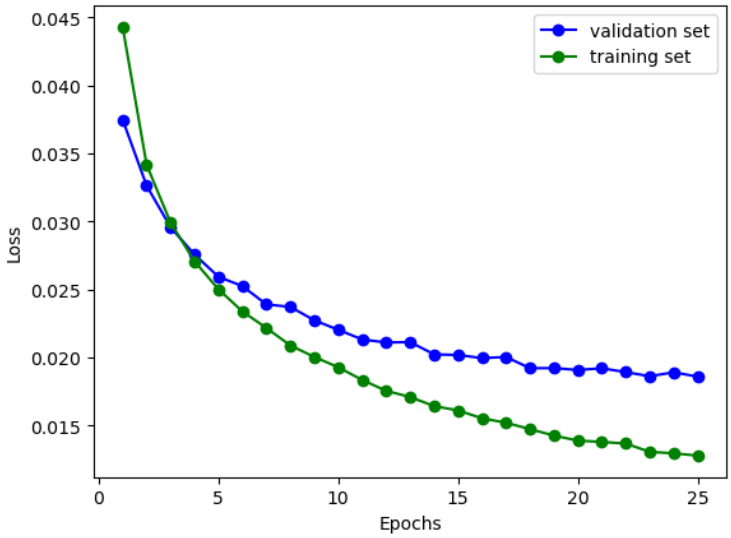

Figure 2: Curva loss conjunto 1

2.3.2 Combinação 2

Comparativamente com a combinação anterior aumentamos o tamanho do *batch size* para 64, número de amostras fornecidas à rede antes de se atualizarem os parâmetros internos desta, com este aumento conseguimos diminuir um pouco o custo computacional do nosso treino, no entanto, causou uma ligeira pioria no desempenho do nosso modelo.

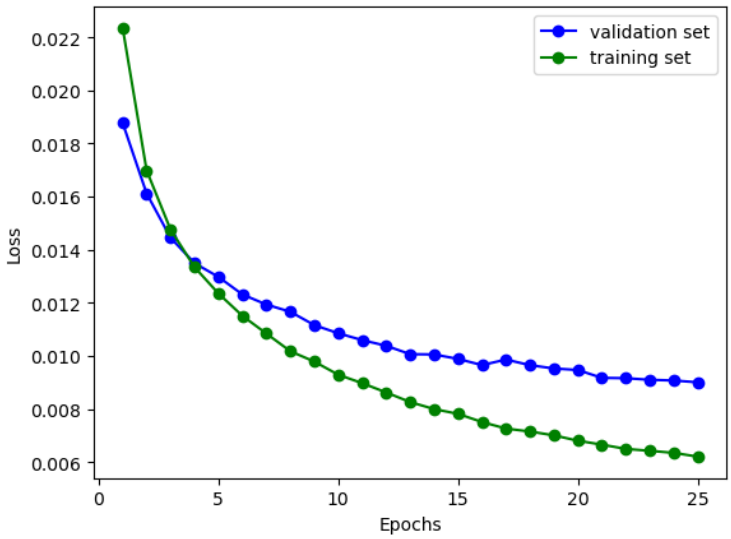

Figure 3: Curva loss conjunto 2

2.3.3 Combinação 3

Nesta combinação usamos um tamanho de 32 no *batch size*, pois foi o aquele com que obtivemos melhores resultados nas combinações anteriores. Como algoritmo de otimização usou-se o *Stochastic Gradient Descent*, este implementa uma descida do gradiente estocástico, que é uma estratégia de otimização que visa minimizar o gradiente de uma determinada função. Fez-se ainda uma última alteração relativamente ao *learning rate*, usando um valor adaptável e não um valor fixo, este

hiperparâmetro é de extrema importância, pois determina quão rápido o modelo se aproxima do mais favorável em resposta ao erro estimado, sendo utilizado o modelo *ReduceLROnPlateau*, que avalia uma certa métrica e, no caso desta não sofrer alterações consideráveis ou estiver estagnada durante um número definido de *epochs*, reduz o *learning rate* por 0,1.

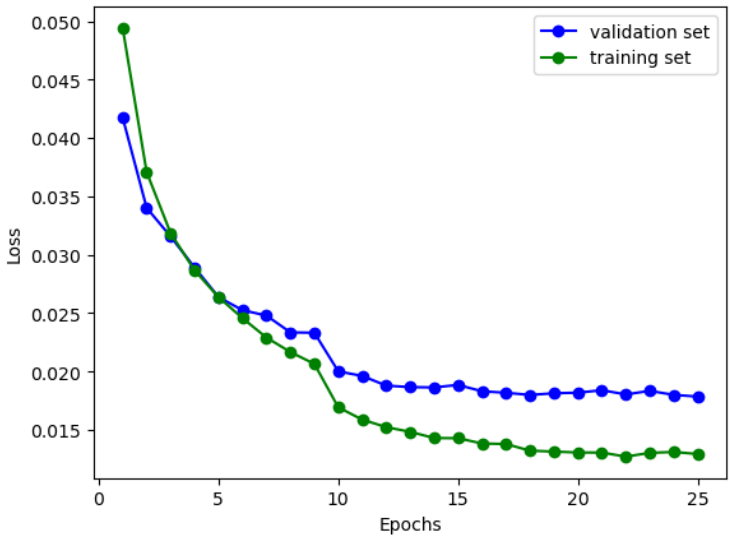

Figure 4: Curva loss conjunto 3

2.3.4 Combinação 4

Por fim chegamos à melhor combinação, sendo esta uma junção dos melhores hiperparâmetros dos conjuntos anteriores.

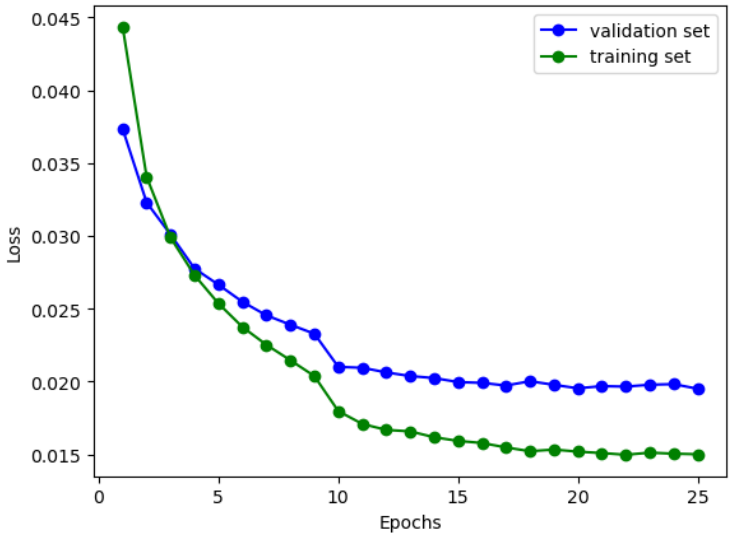

Figure 5: Curva loss conjunto 4

$$\begin{bmatrix} 926 & 3 & 30 & 13 & 4 & 10 & 5 & 5 & 27 & 17 \\ 5 & 964 & 1 & 2 & 1 & 1 & 0 & 2 & 8 & 29 \\ 13 & 1 & 849 & 44 & 34 & 23 & 20 & 12 & 5 & 3 \\ 6 & 3 & 18 & 757 & 22 & 97 & 21 & 13 & 5 & 4 \\ 8 & 0 & 41 & 33 & 883 & 24 & 6 & 24 & 2 & 0 \\ 1 & 1 & 23 & 93 & 12 & 811 & 5 & 15 & 0 & 1 \\ 4 & 0 & 19 & 35 & 16 & 10 & 932 & 4 & 2 & 0 \\ 4 & 0 & 11 & 11 & 22 & 18 & 6 & 922 & 1 & 3 \\ 22 & 5 & 6 & 6 & 5 & 2 & 5 & 0 & 942 & 9 \\ 11 & 23 & 2 & 6 & 1 & 4 & 0 & 3 & 8 & 934 \end{bmatrix}$$

Através da diagonal principal da matriz de confusão da Figura 5, podemos confirmar que a rede acerta a maioria das imagens de teste.

2.3.5 Análise dos gráficos

Com a observação dos gráficos das curvas *loss validation* e da *loss training* conseguimos interpretar se a nossa rede apresenta *overfitting* ou *underfitting* assim como ajustar o número de épocas necessário.

Nos vários testes realizados, todas as combinações apresentam *overfitting* e este pode ser observado na lacuna existente entre as duas curvas, pois o erro de validação começa a estabilizar enquanto o de teste continua a baixar. Também é possível inferir que a utilização de demasiadas *epochs* para além de causar um *overfitting* mais elevado ao nosso modelo, consumirá recursos computacionais desnecessários.

3 Parte II

Nesta segunda parte do trabalho temos como objetivo alterar a rede neural anterior para atingir um *mCA* de pelo menos 80% no dataset EuroSAT. Comparativamente com o CIFAR-10, este contém muito menos imagens e não inclui separação entre imagens de treino e imagens de teste.

3.1 Arquitetura CNN

A arquitura usada para este dataset, foi a mesma que na primeira parte, mudando apenas a imagem de *input* que no caso do EuroSAT passa a ter um tamanho de 64x64x3 em vez de 32x32x3, que era o caso do CIFAR-10. Aplicou-se também no treino deste dataset os hiperparâmetros que tiveram o melhor desempenho na primeira parte do trabalho.

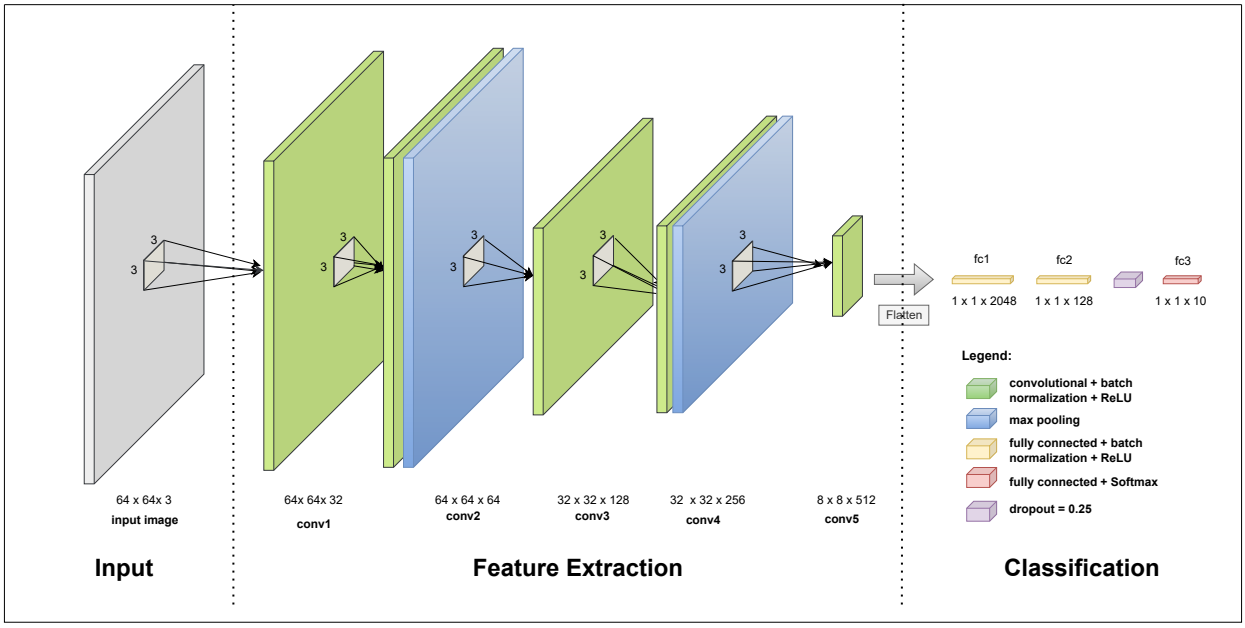

Figure 6: Arquitetura da rede neural convolucional implementada

3.2 Conjunto de Dados

A maior diferença entre este dataset e o anterior, é que neste caso, o dataset não inclui diferentes imagens para treino e para teste, tendo essa separação de ser feita manualmente. Após uma pequena pesquisa, percebemos que o *split ratio* mais comum é de 80/20, tendo assim aplicado esse *ratio* ao nosso datset.

Além disso, aplicamos as mesmas transformações que foram utilizadas no CIFAR-10, tendo duplicado também o tamanho do nosso conjunto de dados.

3.3 Métricas

mCA	F1-Score	Precision	Recall
95.78	91.39	94.10	91.67

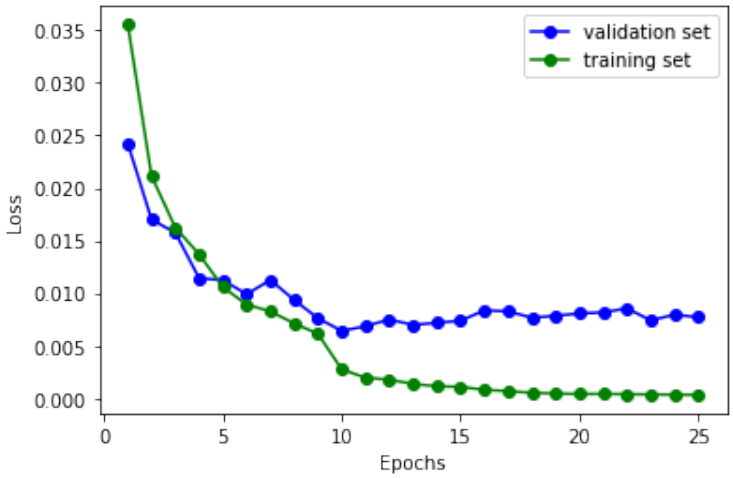

Figure 7: Curva loss EuroSAT

Como era de esperar, sendo este um dataset mais simples que o anterior, a nossa rede apresentou um desempenho muito melhor relativamente ao CIFAR-10, ultrapassando por alguma margem os 90% de *accuracy*.

4 Part III

4.1 Considerando o valor de perdas em cada época, e o número de épocas indefinidas (em execução até que uma condição seja atendida), quando deve parar a etapa de treino?

Ora, após estudo e observação nos testes que efetuámos, chegámos à conclusão que o melhor momento para a fase de treino terminar é quando a *loss* de validação começa a ganhar uma vertente ascendente, o que significa que o modelo começa a entrar em *overfitting*. Podemos também assumir que neste momento o valor da *loss* de treino começa a estagnar, não sofrendo alterações significantes, o que indica que ao longo do resto do treino os pesos nas camadas da CNN também não sofrerão alterações notáveis.

4.2 Pode melhorar ainda mais o desempenho da melhor arquitetura CNN que obteve? Explique a sua resposta.

Existem diversas formas para melhorar o desempenho da arquitetura de uma CNN. Primeiramente, partindo pelas camadas da rede, poderíamos modificar a arquitetura acrescentando, por exemplo, mais camadas convolucionais. Neste trabalho aplicamos apenas 5 camadas convolucionais pois esse era o limite imposto pelo enunciado, porém, com a incrementação de mais camadas deste tipo poderíamos obter certamente um melhor resultado. Outra forma de podermos melhorar a nossa arquitetura CNN seria incrementar ainda o nosso dataset. Aumentar o número de imagens para o treino poderia também ajudar numa melhoria, isto porque estes tipos de redes necessitam de muitos dados para ter o melhor desempenho possível.

4.3 Quão importantes são as camadas convolucionais numa CNN? Explique o seu funcionamento.

As camadas convolucionais são bastante importantes, pois são o que nos permite extrair um mapa de *features* da imagem que temos como *input* na nossa CNN, os valores dessas features são os

pesos a ser "aprendidos" durante o treino da nossa rede. Esse mapa de *features* é obtido ao aplicar um patch, um filtro com determinados pesos, sobre a imagem sistematicamente até percorrer toda essa imagem.

4.4 Quão importante é o valor do *learning rate* no estágio de otimização?

O *learning rate* é um hiperparâmetro que controla o ajuste dos pesos da rede em relação ao gradiente descendente de modo a baixar a função de perda. Se definirmos o valor muito baixo, o progresso do treino será muito lento, pois os ajustes efetuados nos pesos serão muito pequenos. No entanto, se o valor for muito alto pode-se tornar num comportamento divergente devido às atualizações drásticas nos pesos da rede. Este hiperparâmetro tem um grande impacto na convergência para o mínimo local, solução ótima daí ser importantíssimo no estágio de otimização.

4.5 Deve usar funções de ativação lineares ou não lineares? Explique a sua resposta.

Após alguma pesquisa conseguimos perceber que as funções não lineares são as mais usadas na comunidade, pois tornam a rede neural mais adaptável a diversos conjuntos de dados e permite o "empilhamento" de várias camadas de neurónios, criando assim uma *deep neural network*. Uma função de ativação linear apenas é efetiva numa camada de profundidade, tornando-a insignificante.

5 Bibliografia

References

[1] https://towardsdatascience.com/a-comprehensive-guide-to-image-augmentation-using-pytorch-fb162f2444be

[2] https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/

[3] https://pytorch.org/docs/stable/optim.html

[4] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[5] https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/