TÉCNICO LISBOA

Aprendizagem Automática

Relatório dos laboratórios

Autores:

Rodrigo Coimbra: 100078 Rodrigo Pereira: 100080

Novembro 2023

1 Primeiro Problema - Regressão Linear

1.1 Problema

Para este primeiro trabalho, tínhamos como objetivo chegar a uma regressão linear que melhor se aproximava da regressão que gerou os pontos dados pelos professores. Para isso, tínhamos disponíveis um conjunto de 15 dados, tendo cada um 10 *features*.

1.2 Tipos de regressão linear abordados

No desenvolvimento deste projeto várias abordagens foram testadas, tendo cada uma as suas vantagens e desvantagens.

1.2.1 Regressão Linear dos mínimos quadrados

A primeira abordagem testada foi a regressão linear usando o método dos mínimos quadrados.

Inicialmente, ao treinar o modelo com todas as amostras (n=15), obteve-se um valor de $MSE = 0.329$. No entanto, esse resultado não corresponde a um bom modelo de regressão linear, pois estamos a treinar e testar usando os mesmos dados, resultando em *overfitting*. Para melhorar a qualidade do modelo, é fundamental utilizar uma abordagem de *cross-validation*, onde os dados são divididos em conjuntos de treino e teste independentes.

A análise da tabela 1 demonstra um desempenho substancialmente inferior ao referido anteriormente, no entanto oferece uma representação mais precisa da eficácia do modelo para dados novos.

1.2.2 Regressão de Ridge

A segunda regressão testada foi a regressão de *Ridge*. Esta regressão é um método frequentemente utilizado quando a quantidade de dados de treino não é elevada. Isso é obtido através da introdução de um termo de regularização na função, que permite controlar a complexidade do modelo e evitar a ocorrência de *overfitting*.

Como visto no último modelo, é necessário separar os dados em que se testa dos dados onde se treina de forma a obter valores fidedignos. Para isso, usou-se mais uma vez o método de *cross validation*, onde para cada tamanho de *folds* comparámos o seu respetivo valor de *MSE*. Para além disso, de forma a chegar a um valor ideal do termo de regularização, λ, utilizámos um método de *grid search* de forma a iterar por todos os valores entre 0 e 5 com espaçamento entre valores de 0.001.

Assim obteve-se os seguintes gráficos fig. 1, onde a azul podemos ver o valor do *MSE* do modelo linear do *Ridge* em função do λ. Como se pode ver todos os resultados do *Ridge* são consideravelmente consistentes entre diferentes quantidades de *folds* do *cross-validation*, especialmente entre o intervalo [2,3], e tem valores de *MSE* inferiores comparativamente ao modelo linear anterior.

1.2.3 Regressão de Lasso

Por fim, também testamos a regressão de *Lasso*. Este modelo é ideal para quando existem várias *features* nos dados que são irrelevantes para o resultado final, pois este modelo permite *feature removal*. Assim, é um modelo discutido quando os dados a serem avaliados possuem um elevado número de features.

Observando novamente os gráficos da fig 1, mas desta vez focando na função a laranja, nota-se que os seus resultados são bastante inconsistentes quando se trata de *CVs* baixos. No entanto, fazendo o 15 *folds*, ou seja, *Leave one out* obteve-se o melhor resultado entre os 3 modelos.

1.3 Comparação de resultados e conclusão

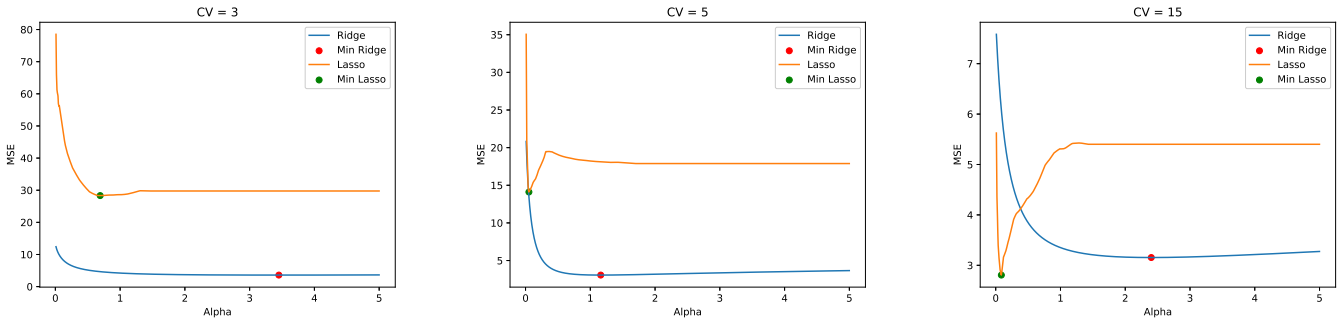
1.3.1 Solução Entregue

CV	Linear MSE	Ridge λ	Ridge MSE	Lasso λ	Lasso MSE
3	10.10	3.451	3.59	0.692	28.33
5	21.71	1.161	3.07	0.052	14.10
15	7.86	2.402	3.15	0.087	2.81

Tab. 1: MSE para os vários modelos

Por fim, ao ponderar todos os modelos que foram testados através da *cross-validation* com diferentes quantidades de *folds* (3, 5 e 15), chegamos à conclusão de que o modelo mais apropriado é o *Ridge*. Embora não seja o modelo que alcançou o menor valor de *MSE*, tendo esse sido obtido pelo modelo do *Lasso*, é importante levar em consideração o desempenho inconsistente deste último modelo em diferentes conjuntos de *cross-validation*. Portanto, partindo do princípio de que não podemos garantir a fiabilidade deste modelo, especialmente considerando que no método *leave one out* só há uma forma de dividir os dados. Isso impossibilita a comparação deste resultado do modelo com outros que utilizam o mesmo número de divisões, o que tornaria possível entender se a quantidade de dados deste conjunto de divisões, onde treina com 14 e testa com 1, podia ser a principal influencia destes resultados elevados.

Após observar o comportamento do modelo escolhido para os diferentes *folds*, notamos que ele mantém uma consistência notável dentro do intervalo dos três valores de λ obtidos. Portanto, decidimos selecionar um valor de λ com base na média dos resultados das três runs. Por fim, fizemos o *fit* do *Ridge* com esse λ para o conjunto completo de pontos e submetemos o conjunto de pontos obtidos para os dados dos professores.



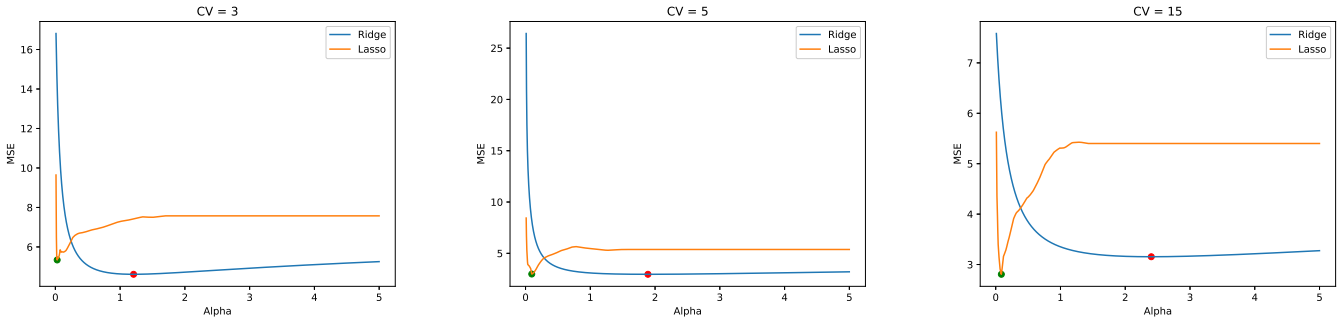
(a) CV = 3 (b) CV = 5 (c) CV = 15

Fig. 1: Gráficos do MSE dos Modelos de Ridge e Lasso em função do valor de λ

1.3.2 Nova Solução

Após as colocações terem saído, foi-nos dito que o melhor modelo teria sido o *Lasso*, o que não correspondeu aos nossos testes realizados, mostrando que houve algum problema na nossa solução. Ao

tentar resolver o problema novamente de forma a que se obtivesse um melhor resultado, decidimos mudar a nossa abordagem ao código, tendo desta vez sido utilizado o gridsearchCV da biblioteca do *sklearn*. Assim obteve-se as seguintes imagens, onde o Lasso apresenta uma performance tão boa e até melhor que o *Ridge*.



(a) CV = 3 (b) CV = 5 (c) CV = 15

Fig. 2

Apesar de não termos compreendido o que aconteceu com a forma como realizamos a entrega,já que muito provavelmente se deu a algum problema de código, isto demonstra um pequeno lapso que prejudicou por completo as nossas conclusões finais. Ao analisarmos agora a tab. 2, observamos que a consistência do *Lasso* é muito melhor que a vista anteriormente e que ao fazer *fit* com uma maior quantidade de dados (a partir do CV 5), este modelo obtém melhores resultados que o *Ridge*.

CV	Ridge λ	Ridge MSE	Lasso λ	Lasso MSE
3	1.208	4.614	0.028	5.340
5	1.890	2.956	0.096	2.981
15	2.402	3.154	0.087	2.807

Tab. 2: MSE para os vários modelos

2 Segundo Problema - Regressão Linear com dois modelos

2.1 Problema

No segundo trabalho foi nos dado um conjunto de dados, criados a partir de duas regressões lineares, e foi-nos pedido para classificarmos os pontos de acordo com o modelo que os criou, de forma a obtermos duas regressões linear idêntica à do problema anterior. A principal dificuldade aqui encontra-se no facto de que a data não está identificada, o que nos força a realizar algum método de *Unsupervised Learning* de forma a que seja possível separar ambos.

2.2 Tipos de classificação usados

2.2.1 K-means

O primeiro método de classificação usado foi o *K-means*, que é um algoritmo de *clustering*. Algoritmos de *clustering* são um tipo de *Unsupervised Learning* que consiste em agrupar diferentes tipos de dados por *clusters*, atribuindo assim *labels* a cada um dos dados observados.

O método de funcionamento consiste em arranjar um número de centróides, escolhido previamente, de forma a que a variância dentro de cada *cluster* seja minimizada.

Após a aplicação do *K-means* para 2 *clusters*, calculou-se o *MSE* para os modelos para ser possível tirar conclusões. Para o cálculo do *MSE*, usamos uma simples regressão linear dos mínimos quadrados o que nos resulta um $MSE = 13.106$.

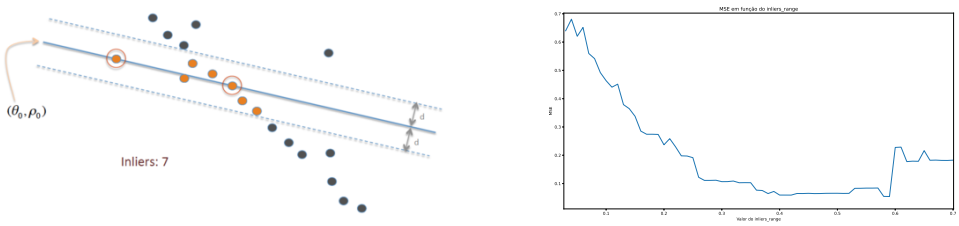
2.2.2 RANSAC

O segundo algoritmo utilizado neste problema foi o *RANSAC*. Neste algoritmo, nós iteramos pelo número total de pontos (n) e para cada ponto visto fazemos um iteração por todos os pontos ainda não vistos($n -$ (índices vistos)). Considerando o ponto da primeira iteração como "x_1" e o da segunda como "x_2", realizamos uma regressão linear entre x_1 e x_2. Tendo esta reta, fazemos uma previsão do y para cada ponto e contamos o número de *inliers*, ou seja, pontos que cumpram a condição 1. A fig. 3a * demonstra a forma como o *RANSAC* funciona a 2D.

Após iterarmos por todos os pontos, escolhemos o resultado com o maior número de *inliers* e consideramos esse como um conjunto de pontos e o resto dos pontos não selecionados vão para outro conjunto. Em seguida, repetimos o processo, mas incrementando o valor do $inliers\_range$ referido na condição 1.

$$|y\_prev - y\_real| <= inliers\_range \tag{1}$$

Por fim, calculámos o *MSE* para o conjunto de treino, da forma instruída pelo corpo docente, ou seja, selecionando o melhor para cada ponto e obtivemos o gráfico da fig. 3b.



(a) Exemplo do RANSAC (b) Evolução do MSE em função do inlier_range

Fig. 3

Como se pode observar obtemos um valor mínimo de $MSE = 0.054$ encontra-se quando o $inlier\_range = 0.59$, o que nos leva a ter dois conjuntos, um contendo 64 pontos e outro com os restantes 36.

2.3 Comparação de resultados e conclusão

2.3.1 Solução Entregue

Modelo	MSE
K-Means	13.106
RANSAC	0.059

Tab. 3: MSE para os vários modelos

Através da Tab. 3, podemos afirmar que de ambos os modelos testados, a performance do *RANSAC* é superior. Uma possível justificação para isto provém que a forma como o *K-means* interpreta os dados, em que o modelo interpreta os dados em função do seu centróide, porém devido a pontos que se encontrem mais distantes, o centróide pode afastar-se do que seria a posição ideal.

*https://en.wikipedia.org/wiki/Random_sample_consensus

2.3.2 Análise pós-submissão

Mais uma vez, haveria possibilidade de melhorar os resultados obtidos. Ao realizar o algoritmo do *RANSAC*, não nos ocorreu a possibilidade onde as retas poderiam se interceptar. Por exemplo, num caso em 2D, os modelos utilizados para adquirir a data de treino poderiam ter declives simétricos, como demonstra a figura 4. Nesse cenário, a aplicação do nosso algoritmo do *RANSAC* levaria a um dilema, pois o ponto verde deveria pertencer à reta laranja, mas para isso o *RANSAC* também consideraria o ponto vermelho, devido ao seu critério de considerar *inliers* todos os pontos dentro do *inlier_range*. Como o ponto verde está mais distante, isso faria com que a reta laranja contivesse ambos os pontos, prejudicando a reta azul, que perderia um ponto favorável. Por outro lado, caso deixássemos o ponto vermelho e verde para a reta azul, isto levaria a uma pior regressão da mesma.

Neste caso, a melhor abordagem seria aplicar o *RANSAC* a ambas as retas. Os pontos que ambas as retas considerassem *inliers* seriam escolhidos com base na sua distância às duas retas, com o ponto mais próximo determinando a qual reta pertenceria. Isso permitiria uma seleção mais precisa e justa dos pontos.

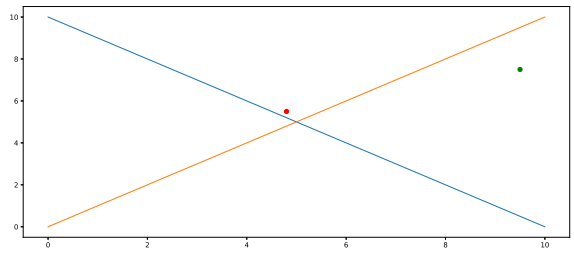


Fig. 4

3 Terceiro Problema - Análise de imagens médicas 2D

3.1 Problema

Para o terceiro problema foi-nos pedido para realizarmos uma classificação de imagens no formato *RGB*, de tamanho 28x28, e identificarmos a condição de pele a que pertencem. Neste caso temos apenas duas condições distintas para classificar, embora seja importante destacar os dados não estarem equilibrados.

3.1.1 Dados desequilibrados

Devido à desproporção existente no conjunto de dados, onde 86% dos dados eram da 1ª classe e apenas 14% da 2ª, foi necessário fazermos um aumento da quantidade de dados. No entanto esse aumento só pode ocorrer no conjunto de treino, portanto antes disso separámos os dados para um conjunto de teste(e validação, no caso da *CNN*), mas garantindo que a proporção de dados nesses conjuntos se mantém igual.

Depois disso adicionámos à segunda classe a rotação de 90º, 180º e 270º de cada uma das suas imagens ao conjunto de treino. Como mesmo assim não chegou para equilibrar os dados, imagens da classe minoritária foram escolhidas aleatoriamente e duplicadas até ficarmos com um conjunto com a mesma proporção.

Também testamos a técnica *SMOTE* que cria dados novos através de processos sintéticos de imagens já existentes, ao invés do *Random Oversampling* explicado em cima. No entanto, após analisar as imagens obtidas, estas encontravam-se com algumas irregularidades o que tornava os dados utilizados "irreais", isto é, ficavam alguns pixeis fora do que seria esperado, parecendo algo completamente diferente das doenças de pele pretendidas.

3.1.2 Normalização

O conjunto de dados fornecido consistia em três canais *RGB*, cada um representando a intensidade da cor com valores de 8 bits, variando de 0 a 255. No entanto, ao aplicar classificadores a esses dados, a ampla

discrepância entre os valores pode resultar em desafios significativos de classificação. Para superar esse problema, normalizamos todos os dados de entrada dividindo-os por 255.

3.2 Tipos de classificadores usados

3.2.1 SVC

O primeiro classificador que utilizamos foi o *SVC*. O objetivo neste classificador, é encontrar um hiperplano que melhor separe as duas classes que temos em um espaço multidimensional. Para obtermos os melhores parâmetros recorremos à função do *gridsearchCV*, onde os seguintes parâmetros foram testados:

- C: $[0.1, 1, 10, 100]$,
- kernel: $['poly', 'rbf']$,
- gamma: $[0.001, 0.01, 0.1, 1]$

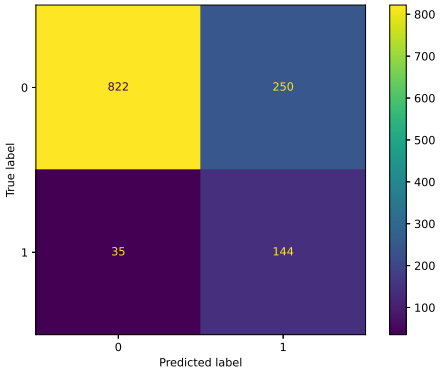


Fig. 5: SVC

No final obtivemos os melhores resultado com os seguintes parâmetros: $C = 10, kernel = "rbf"$ *e* $gamma = 0.01$.

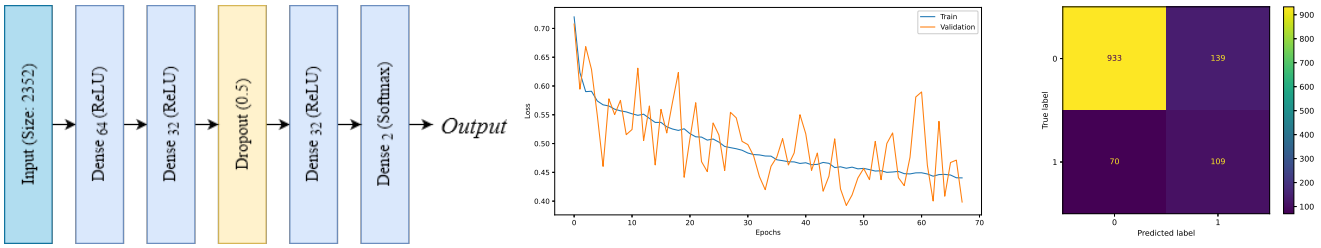
Depois de criarmos o modelo com esses parâmetros, criamos a *confusion matrix* para o conjunto de teste, como visto na fig 5, e calculámos o seu respetivo *balanced accuracy*, que pode ser observado na tab. 4.

3.2.2 Redes neuronais

De seguida decidimos testar modelos de redes neuronais, treinados de forma a identificar a classe das imagens.

Para além da *data augmentation* feita anteriormente, aplicamos também a mesma rotação a todas imagens e não somente às imagens em que estavam numa menor quantidade, pois as redes neuronais escalam bem com uma maior quantidade de dados. No entanto, no final para voltar a manter o equilíbrio, voltamos a fazer *Random Oversampling*.

Depois disso começamos por avaliar a *MLP*, onde primeiro apenas usámos *Dense Layers* e em seguida testamos também com uma *layer* de regularização chamada de *Dropout layer* que melhorou ligeiramente os resultados. Assim, a rede com melhores resultados pode ser vista na fig. 6a. No entanto, mesmo variando quantidade de *layers* e os seus valores, esta rede não demonstrou grandes resultados, tendo obtido um *balanced accuracy* máximo de 0.740 com a confusion matrix da fig. 6c.



(a) MLP (b) Train VS Validation loss (c) Confusion Matrix

Fig. 6: MLP

Partimos assim para a próxima etapa, decidindo então usar uma *CNN* que possui um funcionamento idêntico mas adiciona as *Convolution Layer* e *Pooling Layer*. A *Convolution Layer* realiza múltiplas convoluções de forma a identificar padrões nas imagens, enquanto que a *Pooling layer* reduz o tamanho dos *feature maps* obtidos pela *Convolution Layer*. Também utilizamos uma *Dropout Layer*, que descarta uma parte dos neurónios, introduzindo assim uma forma de regularização e evitando portanto, o *overfitting*.

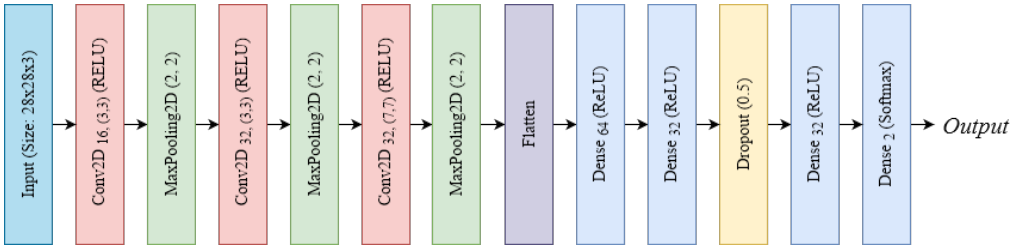
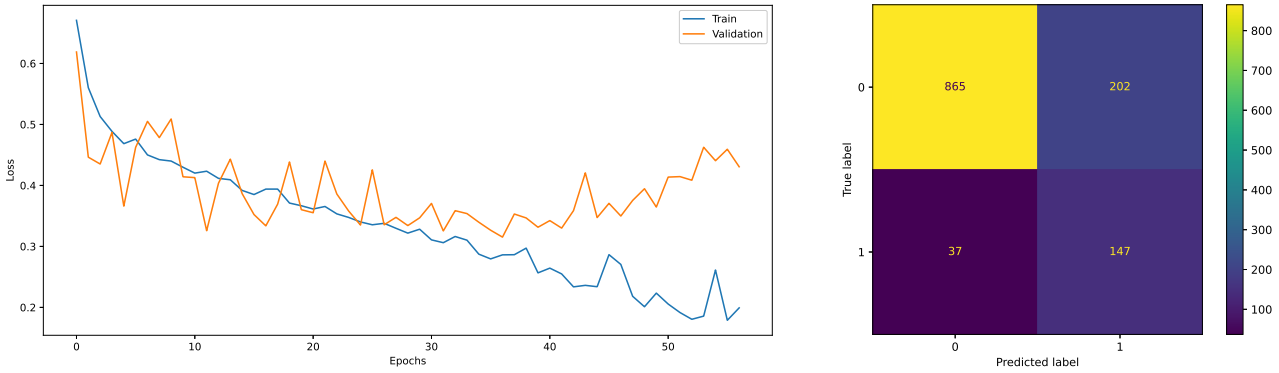


Fig. 7

A nossa rede é composta por uma *Convolution Layer* de 16 filtros com um *kernel size* de 3x3, seguido de uma *MaxPooling*, repete-se isto mais duas vezes aumentando os filtros para 32 e o *kernel size* para 3x3 e 7x7. Após o *flatten*, coloca-se duas *Dense Layers* uma com 64 neurónios e outra com 32, realiza-se um *Dropout* de 0.5 e por fim uma *Dense layer* com 32 neurónios e uma *output layer* com 2 saídas que têm como função de ativação uma *softmax*, como se pode ver na fig. 7. Visto virem dois valores no output por cada imagem, e nós querermos só um, nós escolhemos o maior valor, pois este representa a classe que a *CNN* tem maior confiança de ser a correta. Todas as *layers* excluindo a última usam *ReLU* como função de ativação. Para realizar o *compile*, utilizamos o otimizador Adam com uma *Learning Rate* de 0.001.

Realizamos o *fit* para diferentes *epochs* e diferentes *batch sizes*, obtivemos como melhor resultado um *batch size* de 1000 e 100 *epochs*. Adicionamos também um *early stop*, pois permite-nos recuperar o melhor valor de *Loss* para o conjunto de validação obtido durante o treino. Isto permite-nos terminar o *fit* antes da última *epoch* caso não veja melhorias por 20 *epochs*, ou seja, caso em que o modelo está a começar a dar *overfit*. Isto pode ser visto na fig. 8a, onde a *Loss* do conjunto de treino continua a diminuir a partir da *epoch* 38, no entanto, a *Loss* do conjunto de validação começa a aumentar, indicando claramente o início do *overfitting* aos dados de treino e, portanto, graças ao *early* stop guardámos o valor da *epoch* 38. Já no 8b, podemos observar os resultados para o conjunto de teste, onde obtivemos um *balanced accuracy* de 0.805.



(a) Train VS Validation loss

(b) Confusion Matrix

Fig. 8: CNN

3.3 Comparação de resultados e conclusão

3.3.1 Solução entregue

	SVC	MLP	CNN
BAcc	0.786	0.740	0.805

Tab. 4: Balanced accuracy dos vários modelos

Como se pode observar nos resultados obtidos, a *CNN* obtida possui um *BAcc* superior aos restantes, o que indica uma melhor solução para o problema.

Depois da entrega, obtivemos resultados bastante satisfatórios com este modelo, não tendo, portanto, nenhuma solução que pudesse melhorar consideravelmente os resultados.

4 Quarto Problema - Análise de imagens médicas 2D com dois conjuntos de dados diferentes

4.1 Problema

Para o quarto problema foi-nos pedido para realizarmos uma classificação de imagens em formato *RGB*, de tamanho 28x28, entre 6 classes. No entanto, dessas 6 classes, 3 classes pertenciam ao conjunto dermatológico (classes 0, 1 e 2), enquanto que as restantes pertenciam ao conjunto de células sanguíneas(classes 3, 4 e 5).

Como no problema anterior, este conjunto de dados encontrava-se altamente desbalanceado, tendo a seguinte proporção:

0	1	2	3	4	5
50.45%	8.37%	1.09%	21.69%	9.31%	9.09%

Para solucionar isso, aplicamos os mesmos métodos de rotação de imagens e *Over Sampling* mencionados na secção 3.1.1. Com isso, obtivemos um conjunto de imagens equilibrado no final. O mesmo princípio foi aplicado à normalização dos dados, onde se efetuou o mesmo tratamento de dados referidos na secção 3.1.2.

4.2 Tipos de classificadores usados

4.2.1 SVC

Tal como no problema anterior, voltámos a testar o *SVC* neste conjunto de dados, uma vez que o seu *BAcc* estava relativamente próximo da melhor solução do último trabalho.

Ao executar novamente o *GridSearchCV* com os mesmos parâmetros listados anteriormente, obtivemos que desta os vez os melhores parâmetros são: $C = 100, kernel =' rbf' egamma = 0.01$.

Ao criar a *confusion matrix* deste modelo para os dados finais obtemos a fig. 11a. A partir desta, podemos calcular o *BAcc* que, neste modelo, é igual a 0.812.

4.2.2 Random Forest

Para este problema, também utilizamos um modelo de classificação chamado *Random Forest*. Para determinar os parâmetros a serem utilizados neste modelo, recorremos ao mesmo método utilizado para o *SVC*. Os parâmetros que foram considerados são os seguintes:

- n_estimators: $[100, 200, 300]$
- max_depth: $[None, 10, 20, 30]$
- min_samples_split: $[2, 5, 10]$
- min_samples_leaf: $[1, 2, 4]$
- max_features: $['sqrt', 'log2']$

Após a execução do *GridSearchCV* obtivemos os seguintes resultados para os parâmetros: $n\_estimators = 200, max\_depth = None, min\_samples\_split = 2, min\_samples\_leaf = 1, max\_features =' sqrt'$.

Testando este modelo com esses parâmetros no conjunto de teste, obtivemos a *confusion matrix* da fig. 11b, da qual calculamos um $BAcc = 0.727$.

4.2.3 CNN

Por fim, testámos um modelo de CNN. Tal como referido na secção 3.2.2, voltamos a fazer o mesmo *data augmentation*, que fizemos especificamente para a rede neuronal anterior.

Testando para a mesma rede obtida no problema anterior, obtivemos resultados superiores a qualquer uma das soluções discutidas anteriormente.

No entanto, este modelo ainda podia ser melhorado para este problema e depois de afinado os valores de cada *layer*, obteve-se a rede da fig. 9.

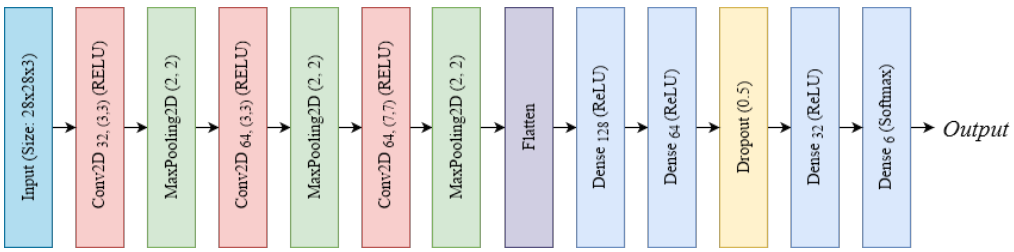
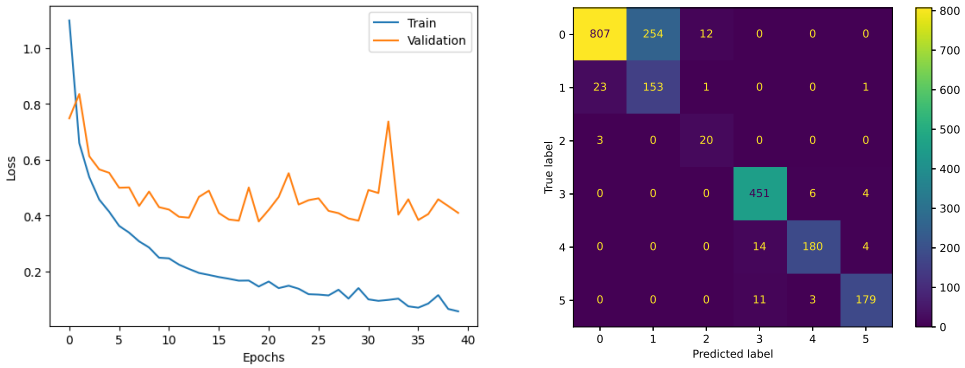


Fig. 9

O funcionamento permanece idêntico ao explicado anteriormente na secção 3.2.2, tendo-se mantido o número de *epochs*, *batch size* e *early stopping*. Os resultados obtidos estão representados na fig. 10.

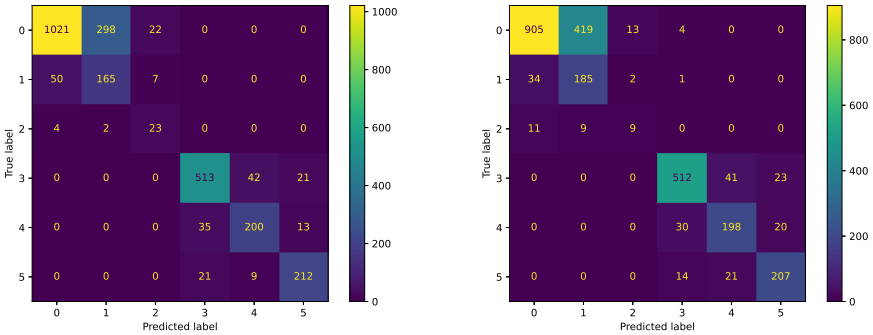


(a) Train VS Validation loss (b) Confusion Matrix

Fig. 10: CNN

4.3 Comparação de resultados e conclusão

4.3.1 Solução entregue



(a) Confusion Matrix do SVC (b) Confusion Matrix da Random Forest

Fig. 11

	SVC	Ran. Forest	CNN
BAcc	0.812	0.727	0.883

Tab. 5: Balanced accuracy dos vários modelos

Analisando os resultados da *BAcc* entre todos os modelos(Tab. 5), observamos que a solução que obteve os melhores resultados foi a CNN, sendo essa a nossa submissão final.

4.3.2 Análise pós-submissão

Apesar de termos obtido resultados favoráveis quando os resultados foram divulgados, existe um modelo que, embora não tenha sido criado por nós devido a limitações de tempo, poderia oferecer melhores resultados. Como pode ser observado na fig. 10b, as previsões em que a nossa *CNN* obtém piores resultados é a prever a classe 0, onde a mesma prevê incorretamente a classe 1 várias vezes.

Para abordar essa questão, propomos uma estratégia que envolve treinar a *CNN* de forma mais intensiva para um conjunto menor de classes. Isso é possível porque, ao rever o problema (sec. 4.1), temos 2 *datasets* distintos, facilmente distinguíveis pela nossa *CNN*. Isso é evidenciado pelo facto de que a maioria dos erros de previsão ocorre dentro do mesmo *dataset*, com apenas um caso em que a *CNN* prevê erroneamente uma imagem da classe 1 como sendo da classe 5, como se pode observar na fig. 10b.

Assim, a nossa abordagem envolveria uma *CNN* que teria como objetivo prever a que *dataset* a imagem pertence e, em seguida, teríamos duas *CNNs*, cada uma dedicada a um *dataset*, que seria especializada em prever a classe. Isso reduziria o número de classes a prever por cada CNN de seis para apenas três.