TÉCNICO LISBOA

Projeto de Aprendizagem Automática

Instituto Superior Técnico
Aprendizagem Automática
Mestrado Bolonha em Engenharia Aeroespacial

Grupo 134:
Diogo Delgado, 92676
Mariana Lima, 92707

Docente:
Prof. António Farinhas

Ano Letivo:
2021-2022

Lisboa,
15 de novembro de 2021

Objetivos: Num mundo cada vez mais digital, de elevado poder computacional e de inteligência artificial, é cada vez mais necessário processar e trabalhar sobre grandes quantidades de dados. A ciência de dados e a aprendizagem automática estão na vanguarda do desenvolvimento do reconhecimento de imagem, de veículos autónomos, de decisões nos setores financeiros e energéticos, de redes sociais e até de desenvolvimentos na medicina. Algoritmos de Regressão e Classificação, classificados como algoritmos de aprendizagem supervisionada, revelam-se técnicas fundamentais de aprendizagem automática. A maior diferença entre eles consiste no facto de os algoritmos de Regressão serem utilizados para prever valores contínuos, ao passo que os de Classificação têm como objetivo prever ou classificar valores discretos. Assim, o presente relatório, elaborado o âmbito da Unidade Curricular de Aprendizagem Automática, encontra-se dividido em duas partes distintas, referentes ao estudo e à implementação de técnicas de Regressão e de Classificação.

Parte 1
Regressão

Técnicas de Regressão apresentam-se como uma das mais importantes em aprendizagem automática e estatística. Existem vários métodos de Regressão, divididos em Regressão Linear e Não-Linear. Regressão consiste no processo de encontrar correlações entre variáveis independentes e dependentes. Os algoritmos de Regressão têm como objetivo o desenvolvimento de modelos, baseados em dados conhecidos, capazes de prever novos dados, dos quais não se têm conhecimento. Estes modelos de previsão são descritos como problemas matemáticos de aproximação de uma função de mapeamento das variáveis de entrada para as de saída. Neste relatório, incidir-se-á sobretudo na análise e implementação de modelos de regressão lineares, onde o valor previsto consiste numa combinação linear das várias *features*. Assim, matematicamente, o valor previsto é dado por

$$\hat{y}(\omega, x) = \omega_0 + \omega_1 x_1 + ... + \omega_p x_p, \tag{1.1}$$

onde $\omega = (\omega_0, ..., \omega_p)$ representa os coeficientes de ordem 0 a p, respetivamente.

1.1. Primeiro problema de regressão

Neste primeiro problema de regressão, tem-se como objetivo treinar modelos de previsão tais que $\hat{y} = f(x)$, para um determinado conjunto de dados de treino $\tau_r = \{(x^{(1)}, y^{(1)}), ..., (x^{(100)}, y^{(100)})\}$, com $x^{(i)} \in \mathbb{R}^{20}$ e

$y^{(i)} \in \mathbb{R}$. Deste modo, treinam-se vários modelos de regressão, de forma a escolher aquele que melhor se adapta ao conjunto de dados e, posteriormente, efetuam-se as previsões $\hat{y}^{(i)}$ correspondentes aos vetores de *features*, $x^{(i)}$, associados ao conjunto de dados de teste.

Primeiramente, inicia-se a abordagem visualizando as estatísticas do conjunto de dados de treino, através da utilização da função `pandas.DataFrame.describe`. Esta função retorna estatísticas tais como o número de observações não nulas, a média, o máximo e o mínimos dos valores presentes no conjunto, bem como o desvio padrão das observações. De seguida, standardiza-se o conjunto de dados `Xtrain` e `Xtest`, com recurso à função `sklearn.preprocessing.StandardScaler`. Esta, tem como objetivo standardizar as *features*, removendo a média e alterando a escala para variância unitária, ou seja, o valor standardizado de uma amostra x é dado por

$$z = \frac{x - u}{s}, \tag{1.2}$$

onde u corresponde à média das amostras de treino e s ao desvio padrão das mesmas. A standardização assume que o conjunto de dados pode ser aproximado por uma distribuição Gaussiana bem definida, e é bastante útil ou até recomendada em alguns algoritmos de aprendizagem automática, uma vez que este conjunto de dados pode apresentar diferentes escalas. Note-se que, o código entregue correspondente a este primeiro problema apresenta um erro na linha referente à standardização do conjunto de test, uma vez que deveria ter sido utilizado o `StandardScaler()` com os parâmetros definidos no conjunto de treino, isto é, a linha `Xtest_std = StandardScaler().fit_transform(Xtest)` deveria ser substituída por `Xtest_std = scaler.transform(Xtest)`, com `scaler = StandardScaler()`. Este erro apenas altera a previsão do $Ytest$ final, não comprometendo os resultados mostrados adiante.

Os modelos de regressão utilizados para treinar o conjunto de dados apresentam-se como modelos lineares e encontram-se no módulo `linear_model` da biblioteca `sklearn`. Os modelos treinados foram então: `LinearRegression`, `Lasso`, `Ridge` e `BayesianRidge`.

O modelo `LinearRegression` ajusta um modelo linear com coeficientes $\omega = (\omega_1, ..., \omega_p)$ para minimizar a soma dos quadrados da diferença entre os valores observados no conjunto de dados e os valores previstos pela aproximação linear. Matematicamente, pode ser descrito pela fórmula

$$\min_w \|Xw - y\|_2^2. \tag{1.3}$$

No que respeita o modelo de regressão `BayesianRidge` tem a vantagem de se adaptar aos dados utilizados e, para obter um modelo totalmente probabilístico, assume-se a previsão como uma distribuição Gaussiana em torno de $X\omega$, tendo-se

$$p(y \mid X, w, \alpha) = \mathcal{N}(y \mid Xw, \alpha) \text{ onde } p(\omega \mid \lambda) = \mathcal{N}(\omega \mid 0, \lambda^{-1} I_p). \tag{1.4}$$

Por sua vez, o modelo de regressão `Lasso` estima coeficientes esparsos, sendo útil devido à sua tendência para valorizar soluções com menos coeficientes não nulos, reduzindo o número de *features* das quais a solução é dependente. Matematicamente, consiste num modelo linear, com a adição de um termo de regularização. A função que se pretende minimizar é

$$\min_w \frac{1}{2n_{\text{samples}}} \|Xw - y\|_2^2 + \alpha \|w\|_1, \tag{1.5}$$

onde α é um hiperparâmetro constante e $\|\omega\|_1$ a norma-l_1 do vetor de coeficientes. Recorre-se à função `LassoCV`, responsável por incorporar o modelo `Lasso` com `Cross Validation`, com o objetivo de obter o hiperparâmetro α ótimo.

Relativamente ao modelo de regressão `Ridge`, este impõe uma penalização no tamanho dos coeficientes. Os coeficientes do modelo minimizam

$$\min_w \|Xw - y\|_2^2 + \alpha \|w\|_2^2, \tag{1.6}$$

onde $\alpha \geq 0$ é o hiperparâmetro que controla a redução. Quanto maior este valor, maior a redução e, por isso, os coeficientes tornam-se mais robustos à existência de colinearidade. Neste problema, recorre-se à função `RidgeCV`, que incorpora o modelo `Ridge` com `Cross Validation`, de forma a determinar o valor

ótimo para o hiperparâmetro α. Para isso, foi necessário enviar como entrada desta função um vetor de possíveis hiperparâmetros, neste caso `np.arange(1e-7, 0.005, 1e-7)`.

De forma a validar os modelos de regressão, opta-se por utilizar o algoritmo `Leave One Out Cross Validation`. Este apresenta-se como um caso particular de `Cross Validation`, onde o número de divisões é igual ao número de amostras no conjunto de treino. Assim, este algoritmo é aplicado uma vez para cada amostra, utilizando as restantes como conjunto de treino, enquanto que a selecionada funciona como uma única amostra no conjunto de teste. O critério de escolha do modelo de regressão a adotar baseia-se no valor do erro quadrático médio (*mean squared error*, MSE) associado a cada um. Recorrendo à função `cross_val_score` do módulo `model_selection` do `sklearn`, obtêm-se os MSEs associados a cada modelo, apresentados na Tabela 1.1, juntamente com os hiperparâmetros ótimos determinados.

Tabela 1.1: Hiperparâmetros e MSEs obtidos para cada modelo de regressão treinado no primeiro problema de regressão.

Modelo	`LinearRegression`	`Lasso`	`Ridge`	`BayesianRidge`
Hiperparâmetro	-	$\alpha = 0.001863$	$\alpha = 0.033291$	-
MSE	0.015574	0.015395	0.015571	0.015574

Deste modo, o modelo de regressão escolhido foi a regressão `Lasso`, uma vez que, dos quatro estudados, é o que apresenta o valor de MSE mais diminuto.

Finalmente, após a escolha do modelo de regressão, de forma a verificar que não existe sobre-ajuste aos dados de treino, dividem-se os mesmos em dados de treino (70%) e dados de teste (30%), com recurso à função `train_test_split` do módulo `model_selection` do `sklearn`. Obtiveram-se os MSEs de 0.0083 e 0.0185 relativos aos dados de treino e de teste, respetivamente. Estes valores são bastante diminutos e semelhantes, confirmando a não existência de sobre-ajuste. O problema termina com a previsão final do vetor `Ytest`, recorrendo, para isso, à função `model.predict(Xtest)`, com o conjunto de dados de testes na entrada e `model` a corresponder a `LassoCV(cv=LeaveOneOut()).fit(Xtrain_std,Ytrain)`.

1.2. Segundo problema de regressão

Este segundo problema de regressão é bastante similar ao primeiro, apresentando os mesmos objetivos, sendo que principal diferença entre eles consiste na existência de *outliers*, isto é, pontos do conjunto de dados de treino que diferem tanto dos demais que a sua presença no treino do modelo escolhido pode pôr em causa os resultados obtidos.

Estes dados podem levar a desvios significativos nas estatísticas e na distribuição do conjunto de dados levando a uma representação errónea dos dados de treino e suas relações. Como tal, é necessária a sua remoção do conjunto de dados de treino antes da criação de um modelo. Este passo irá originar previsões mais corretas quando comparadas com as de um modelo que se tente ajustar a valores discrepantes.

1.2.1 Remoção de *outliers*

A presença de *outliers* está limitada a 10% do conjunto de dados de treino, pelo que a abordagem utilizada consiste no seguinte: assumindo que o conjunto de dados de treino apresenta uma distribuição linear, defini-se como modelo de regressão a `LinearRegression`. Por sua vez, recorre-se novamente à função `cross_val_score` com `Leave One Out Cross Validation` de forma a obter o vetor de erros quadráticos médios associados a cada amostra, sendo, posteriormente, removida a amostra que apresentava maior erro. Este passo repete-se até que cerca de 20% dos pontos iniciais sejam removidos, de forma a comparar o efeito da remoção de amostras válidas com o efeito da remoção de *outliers*. Este processo iterativo é realizado com o intuito de guardar as diferenças entre os MSEs globais a cada iteração. É ainda importante referir que a cada iteração se efetua um ajuste do modelo, de modo a excluir amostras removidas em iterações anteriores.

Como a partir da remoção de 10% dos pontos já não existem *outliers*, calcula-se a média das diferenças dos MSEs globais a partir da décima iteração, de modo a averiguar que influencia apresenta a remoção de uma amostra válida do conjunto de dados de treino neste valor.

Como critério de decisão sobre que diferença constitui um *outlier*, utiliza-se uma margem de 50% sobre o valor médio das diferenças a partir da décima iteração. Deste modo, removeram-se oito *outliers*, com índices do vetor inicial {85, 78, 92, 33, 52, 2, 80, 62}, pela ordem de remoção. Nas Figuras 1.1 e 1.2 é possível visualizar o MSE correspondente aos 8 *ouliers* removidos, sendo os dois primeiros, correspondentes aos índices 85 e 78, os mais evidentes.

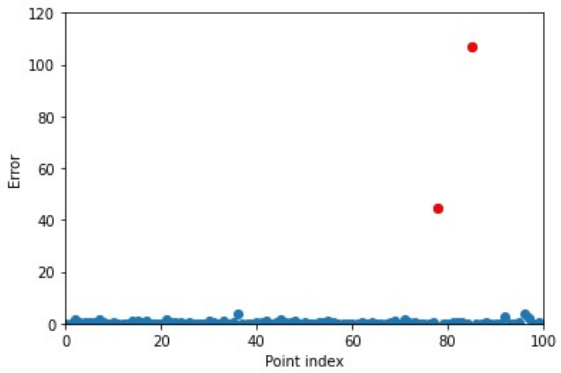


Figura 1.1: Identificação dos dois primeiros *outliers*.

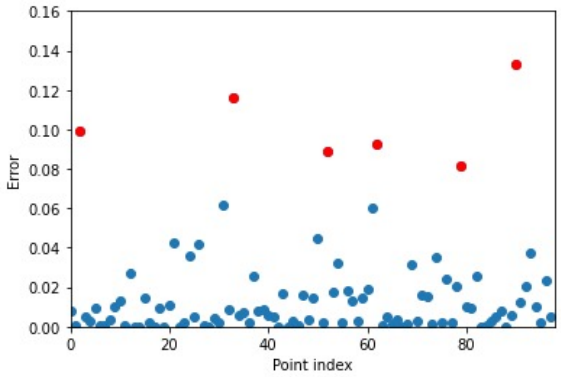


Figura 1.2: Identificação dos seis últimos *outliers*.

1.2.2 Treino do modelo

Após a remoção dos *outliers*, a escolha do modelo de regressão a adotar segue a mesma lógica do primeiro problema. Desta vez, foi implementado mais um modelo de regressão para além dos apresentados no primeiro problema, sendo este o `OrthogonalMatchingPursuit (OMP)`. Este modelo consiste em aproximar o ajuste de um modelo linear com os constrangimentos impostos no número de coeficientes não nulos. O método `OMP` aproxima o vetor de solução ótimo com um número fixo de elementos não nulos. Tem-se, portanto

$$\arg\min_{w} \|y - Xw\|_2^2 \text{ tal que } \|w\|_0 \leq n_{\text{nonzero\_coefs}}. \tag{1.7}$$

De modo a obter o valor ótimo do hiperparâmetro do modelo (número de coeficientes não nulos na solução) recorre-se à função `OrthogonalMatchingPursuitCV` que incorpora o modelo `OMP` com `Leave One Out Cross Validation` de modo a retornar o valor ótimo para o hiperparâmetro.

Tal como na secção 1.1, com recurso à função `cross_val_score` e escolhendo como método de `Cross Validation` o `Leave One Out`, obtiveram-se os MSEs associados a cada modelo, presentes na Tabela 1.2, juntamente com os hiperparâmetros ótimos determinados.

Note-se que o hiperparâmetro ótimo que o algoritmo escolhe para o modelo `Ridge` é o mais pequeno (bastante próximo de zero) do intervalo de hiperparâmetros possíveis, fazendo com que este seja equivalente ao modelo `LinearRegression`.

Assim, o modelo de regressão escolhido foi a regressão `OMP`, uma vez que, dos cinco estudados, é o que apresenta o menor valor de MSE.

Tabela 1.2: Hiperparâmetros e MSEs obtidos para cada modelo de regressão treinado no segundo problema de regressão.

Modelo	`LinearRegression`	`Lasso`	`Ridge`	`BayesianRidge`	`OMP`
Hiperp.	-	$\alpha = 0.001626$	$\alpha = 1 \times 10^7$	-	$n = 19$
MSE	0.008999	0.009051	0.008999	0.009000	0.008763

Por fim, tal como para o primeiro problema, dividem-se os mesmos em dados de treino (70%) e dados de teste (30%), de forma a confirmar a não existência de sobre-ajuste aos dados de treino. Obtiveram-se os MSEs de 0.050 e 0.0133 para os dados de treino e teste, respetivamente. Uma vez mais, estes valores são bastante diminutos e semelhantes, confirmando a não existência de sobre-ajuste.

Note-se que na submissão do código relativo a este problema de regressão, a previsão do vetor `Ytest` final não foi realizada da maneira correta, uma vez que o modelo utilizado para a previsão do mesmo, foi ajustado somente para 70% dos dados de treino. De modo a ajustar o modelo a 100% dos dados de treino, deveria ser acrescentada a linha de código `model.fit(XtrainOutliers,YtrainOutliers)` antes da previsão.

Parte 2
Classificação

A Classificação consiste no processo de obtenção de uma função capaz de separar o conjunto de dados em classes distintas, baseado em diferentes parâmetros. O objetivo dos algoritmos de Classificação prende-se com encontrar a função de mapeamento capaz de mapear o valor de entrada para um valor discreto na saída. Nesta fase do projeto, pretende-se classificar um conjunto de imagens faciais, adaptadas do conjunto de dados bem conhecido UTKFace, primeiramente em duas classes de acordo com o género (masculino e feminino) e, numa segunda abordagem, em quatro diferentes classes, com base na etnia (caucasiana, africana, asiática e indiana). Deste modo, testaram-se diversas abordagens, tais como a máquina de vetor de suporte (*Support Vector Classifier*, `SVC`), a rede neuronal *perceptron* multicamada (*Multilayer Perceptron*, `MLP`) e a rede neuronal convolucional (*Convolutional Neural Network*, `CNN`).

2.1. Primeiro problema de classificação

Este primeiro problema de Classificação consiste numa classificação binária, onde o objetivo se prende com treinar modelos que sejam capazes de prever o género de uma pessoa a partir de imagens da cara da mesma. Assim, uma imagem pode ser classificada como '0' (género masculino) ou '1' (género feminino).

Numa primeira instância, verifica-se que os dados de treino disponibilizados apresentam cerca de 42.8% de pessoas do género masculino e 58.2% de pessoas do género feminino. Assim, dada a proximidade dos 50/50%, considera-se que os dados se encontram balanceados. De seguida, normalizam-se os dados recorrendo à função `true_divide` presente na biblioteca `numpy`. Posteriormente, utiliza-se a função `reshape`, desta mesma bibliteca, de forma a que os dados fiquem com uma dimensão de 50 por 50 pixeis e que seja possível a visualização correta das imagens, no caso da implementação da rede neuronal convolucional.

Numa segunda fase, procede-se à divisão dos dados disponíveis em dados de treino (85%) e dados de validação (15%), com recurso à função `train_test_split`, mencionada anteriormente, de modo a ser possível a comparação da performance entre diferentes modelos.

Por forma a ter mais imagens para treinar o modelo, implementam-se técnicas de aumento de dados. Estas técnicas permitem gerar várias versões do conjunto de dados sem ter de se proceder à recolha de novas amostras, aumentando artificialmente o tamanho do conjunto de dados de treino. Exemplos destas técnicas são a adição de ruído, virar na horizontal e na vertical, a rotação e o zoom. No presente caso, utiliza-se a técnica de virar na horizontal, com recurso à função `fliplr` da biblioteca `numpy`, pois é que a melhor se adequa às imagens do conjunto. Nas Figuras 2.3 e 2.4, pode observar-se um exemplo desta técnica. Assim, foi possível duplicar o tamanho do conjunto de dados de treino. A utilização de várias combinações destas técnicas poderia ter sido benéfica, contudo, com o aumento sucessivo do conjunto de dados de treino, isto seria bastante dispendioso computacionalmente.

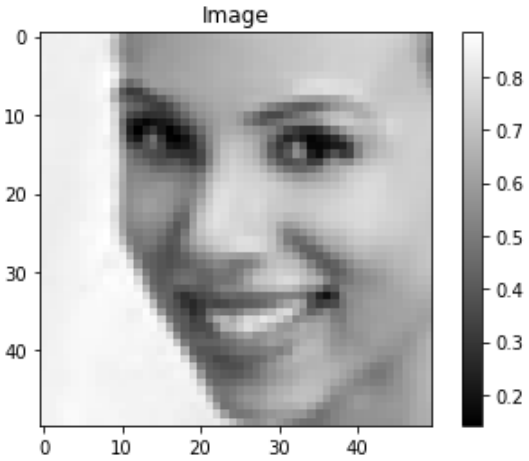


Figura 2.3: Imagem original.

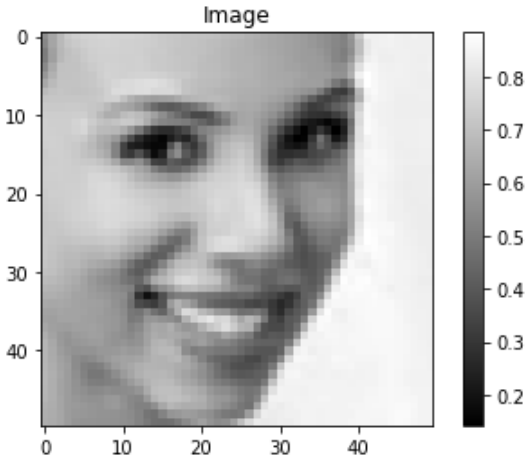


Figura 2.4: Imagem após a utilização da técnica de aumento de dados - virar na horizontal.

Após o tratamento de dados descrito, implementaram-se as diversas técnicas mencionadas. Note-se que os métodos `SVC` e `MLP` requerem a utilização de funções da biblioteca `sklearn` e os seus hiperparâmetros são determinados com auxílio da função `GridSearchCV`. Assim, os parâmetros a otimizar são α no caso da rede neuronal `MLP` e C, bem como γ no caso do `SVC`.

Seguidamente, foi ainda implementada a codificação *One Hot Encoding*, que consiste na representação de variáveis categóricas como vetores binários, nos valores de saída quer do conjunto de treino, quer do conjunto de validação. Na Tabela 2.3, é possível verificar os hiperparâmetros ótimos determinados, bem como o *Balanced Accuracy* associados a cada modelo de classificação treinado.

Tabela 2.3: Hiperparâmetros e *Balanced Accuracy* (BACC) obtidos para cada modelo de classificação treinado no primeiro problema de classificação.

Modelo	`MLP`	`SVC`	`CNN`
Hiperparâmetro	$\alpha = 0.0004$	$\gamma = 0.0007$, $C = 92$	-
BACC	0.8753	0.8997	0.9157

Analisando a Tabela 2.3, observa-se que o modelo com melhor BACC é a rede neuronal convolucional, pelo que se implementa esta mesma rede no decorrer do problema.

2.1.1 Rede Neuronal Convolucional (CNN)

Na presente secção, aprofunda-se o estudo e a implementação da rede neuronal convolucional, que segue a estrutura exemplificada na Figura 2.5.

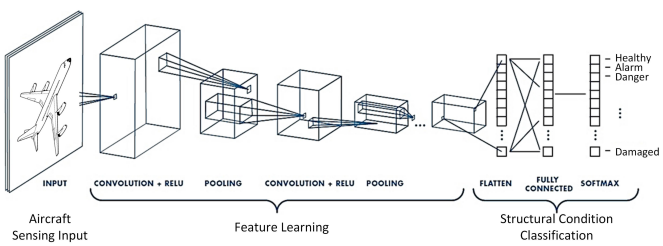


Figura 2.5: Ilustração da estrutura de uma rede neuronal convolucional.

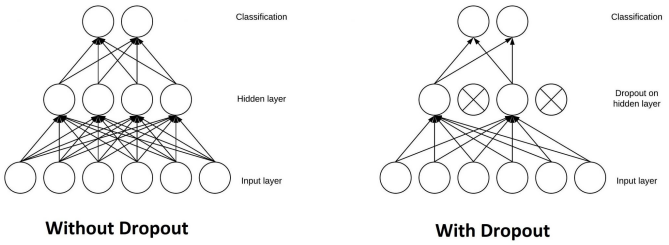


Figura 2.6: Ilustração do funcionamento do *dropout*.

Num primeiro momento, inspirado numa rede convolucional criada para o conjunto de dados *MNIST Handwritten Digit Classification*, foi criada uma rede convolucional com apenas uma camada convolucional

(`Conv2D`) com 2 filtros de tamanho diminuto (3,3), seguida de uma camada *max pooling* (`MaxPooling2D`). De seguida, foi criada uma camada `Flatten` para fornecer *features* ao classificador. Como o objetivo consiste em classificar as imagens em duas classes, é necessária uma camada `Dense` com dois nós. Contudo, antes desta camada foi introduzida uma outra camada `Dense`, desta vez com 32 nós. Todas estas camadas utilizam ativação *'relu'*, por ser uma boa prática, com exceção da última camada `Dense`, que necessita de ativação *'softmax'*. Posteriormente, de modo a otimizar a rede foi adicionada, antes da camada `Flatten`, uma segunda camada `Conv2D` com o dobro dos filtros da primeira, seguida de uma nova camada `MaxPooling2D`.

Ademais, mostrou-se benéfica a criação de camadas `Dropout` de 0.15 após cada camada `MaxPooling2D`. Estas camadas são muito úteis pois funcionam como uma máscara que anula a contribuição de alguns neurónios para a camada seguinte, prevenindo assim a existência de sobre-ajuste ao conjunto de dados de treino. Na Figura 2.6 é possível observar o efeito causado por uma camada de `Dropout`.

Após a criação destas camadas, é necessário compilar o modelo escolhendo um otimizador e uma função de custo (*loss function*), neste caso, a *loss function* escolhida foi a *Categorical Cross Entropy*.

A escolha do número de épocas de treino que se devem utilizar é difícil. Uma forma de otimizar esta escolha é a utilização de `EarlyStopping`, permitindo uma escolha de épocas arbitrariamente grande (neste caso 200 épocas), que terminará o treino do modelo quando uma quantidade que esteja a ser monitorizada deixe de melhorar. Neste modelo, a quantidade a ser monitorizada é a *validation loss*, pretendendo-se minimizar este valor. Definindo um valor de `patience` de 20 épocas, o modelo finaliza o seu treino após 20 épocas sem redução da *validation loss*, restaurando as características do modelo na época em este parâmetro foi mínimo.

A rede acima descrita é apenas um exemplo de uma de muitas redes neuronais convolucionais implementadas. Na Tabela 2.4, é possível comparar a performance de algumas variações da rede descrita.

Tabela 2.4: *Validation Accuracy* e BACC para diferentes redes neuronais convolucionais, referentes ao primeiro problema de classificação.

Nº de camadas	**Aumento de dados**	**Otimizador**	***Validation Accuracy***	**BACC**
2	`fliplr`	*'adam'*	0.9176	0.9157
2	-	*'adam'*	0.9011	0.9005
2	`flipud`	*'adam'*	0.8795	0.8823
2	`fliplr`	`SGD(lr=0.01, momentum=0.9)`	0.8836	0.8809
3	`fliplr`	*'adam'*	0.9042	0.9056

Pela análise da Tabela 2.4, constata-se que:

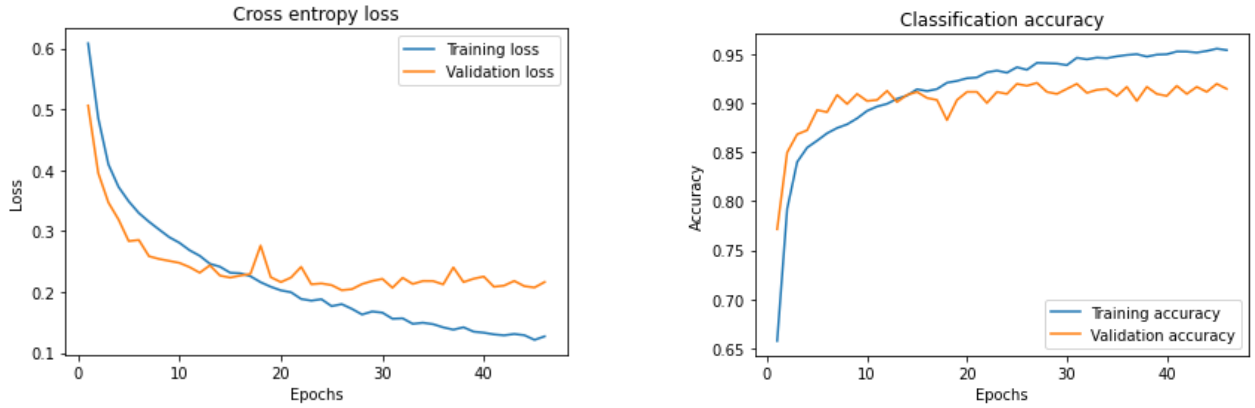
- Pela comparação da primeira com a última linha, o aumento do número de camadas não é benéfico para os resultados obtidos;
- Pela comparação da primeira com a quarta linha, o otimizador mais adequado é o *'adam'* quando comparado com o SGD (com *learning rate*=0.01 e momento=0.9);
- Pela comparação entre a primeira e a segunda linha, utilizando a técnica de aumento de dados de virar à esquerda/direita, os resultados melhoram consideravelmente, dado que esta técnica se adequa ao conjunto de dados pois todas as imagens apresentam pessoas a olhar de frente para a câmara. Comparando a segunda linha com a terceira, observa-se que a utilização da técnica de virar para cima/baixo é prejudicial para o modelo, algo que já era expectável uma vez que não se adequa ao conjunto de dados.

Assim, conclui-se ainda que a melhor rede neuronal estudada corresponde à primeira linha da tabela, constituída por duas camadas, utilizando a técnica de aumento de dados de virar à esquerda/direita e recorrendo ao otimizador *'adam'*.

Após a implementação desta rede CNN, obtiveram-se os gráficos da Figura 2.7, que ilustram a evolução da *cross entropy loss* e da *classification accuracy* para o conjunto de dados de treino e de validação.

Da análise da Figura 2.10a, verifica-se que a *cross entropy loss* associada ao conjunto de dados de treino diminui progressivamente com o aumento do número de épocas. Por sua vez, a *cross entropy loss* associada ao conjunto de dados de validação desce até uma determinada época, a partir da qual inicia a subida. A

partir deste ponto, considera-se que existe sobre-ajuste da rede ao conjunto de dados de treino e, por essa razão, deve-se recuperar os valores correspondentes à época que apresenta *cross entropy loss* associada aos dados de validação mínima.



(a) *Cross Entropy Loss.* **(b)** *Classification Accuracy.*

Figura 2.7: *Cross Entropy Loss* e *Validation Accuracy*, para os dados de treino e de validação, com rede neuronal convolucional otimizada.

No que respeita a Figura 2.10b, observa-se uma elevada subida da *classification accuracy* logo nas primeiras épocas, quer para o conjunto de dados de treino, quer de validação. Com o aumento do número de épocas, a *classification accuracy* associada ao conjunto de dados de treino tende para 100%, ao passo que a associada ao conjunto de dados de validação tende a estagnar num valor inferior.

2.2. Segundo problema de classificação

Este segundo problema de classificação é bastante similar ao primeiro, sendo que a principal diferença incide sobre o número de classes em que os dados se dividem. Neste caso, existem quatro classes baseadas na etnia das pessoas das imagens (caucasiana, africana, asiática e indiana). Assim, uma imagem pode ser classificada como '0' (etnia caucasiana), '1' (etnia africana), '2' (etnia asiática) ou '3' (etnia indiana).

Num primeiro momento , verifica-se que as imagens de treino disponibilizadas apresentam uma grande representatividade de pessoas de etnia caucasiana (60.8%), estando as outras etnias em minoria no conjunto, 18.2% e 16.4% para as etnias asiática e indiana, respetivamente, e finalmente, a etnia africana com a menor representatividade (apenas 4.6%). Considera-se então que os dados não se encontram balanceados sendo, por isso, importante a utilização de classes ponderadas no modelo. A principal diferença de implementação dos modelos do primeiro para o segundo problema deve-se precisamente à utilização de classes ponderadas.

Em diante, a resolução deste problema segue a mesma metodologia do primeiro, à exceção da necessidade de implementação das classes ponderadas. Esta ponderação faz-se com base no número de imagens pertencentes a cada classe. Exemplificando a ponderação atribuída à etnia caucasiana, obtém-se $\omega_0 = (N_{\text{caucasiano}} + N_{\text{africano}} + N_{\text{asian}} + N_{\text{indian}})/(4 \times N_{\text{caucasiano}})$. Na Tabela 2.5 é possível encontrar a comparação da performance dos diferentes modelos testados.

Pela análise desta Tabela, observa-se que o modelo com melhor BACC é, uma vez mais, a rede neuronal convolucional, pelo que se implementa esta mesma rede no decorrer do problema. A implementação da rede neuronal é equiparável à descrita no primeiro problema, excetuando a duplicação no número de filtros em cada camada de convolução e alterando ainda o `kernel_size` de 3 para 5, o `Dropout` de 0.15 para 0.2 e a `patience` de 20 para 15. Naturalmente, foi também necessário alterar o último `Dense`, de forma a obter quatro saídas, ao invés de duas, uma vez que, neste segundo problema, existem quatro classes. Os restantes parâmetros característicos da rede mantêm-se constantes, nomeadamente o `EarlyStopping`, a *loss function* e o número de épocas. Neste problema, a implementação das ponderações das classes é feita através do argumento `class_weight` da função `model.fit` do módulo `keras` da biblioteca `Tensorflow`, com `model=Sequential()`, do módulo `keras.models`.

Tabela 2.5: Hiperparâmetros e *Balanced Accuracy* (BACC) obtidos para cada modelo de classificação treinado no primeiro problema de classificação.

Modelo	`MLP`	`SVC`	`CNN`
Hiperparâmetro	$\alpha = 0.002$	$\gamma = 0.001$, $C = 18$	-
BACC	0.6498	0.7153	0.7842

Na Tabela 2.6, é possível observar a *Validation Accuracy* e BACC para diferentes redes neuronais convolucionais.

Tabela 2.6: *Validation Accuracy* e BACC para diferentes redes neuronais convolucionais, referentes ao segundo problema de classificação.

Nº de camadas	**Ponderação de pesos**	***Validation Accuracy***	**BACC**
2	Sim	0.8308	0.7842
2	Não	0.8462	0.7783
3	Sim	0.8063	0.7403

Da análise da Tabela anterior, concluiu-se que a adição de uma terceira camada na rede neuronal não melhora os resultados, uma vez que tanto a *validation accuracy*, como o BACC se mostram inferiores. Comparando a utilização da ponderação de pesos com a sua não utilização, observa-se que o *validation accuracy* é maior na rede que não utiliza ponderação, contudo, o BACC é menor. Neste caso, é mais importante ter em conta os valores do BACC pois, tal como o nome indica, *balanced accuracy*, este valor tem em conta as ponderações das classes no conjunto de dados de treino, conduzindo a resultados mais realistas.

Nas Figuras 2.8 e 2.9 é possível observar as matrizes de confusão, para as redes CNN com e sem ponderação de pesos, respetivamente.

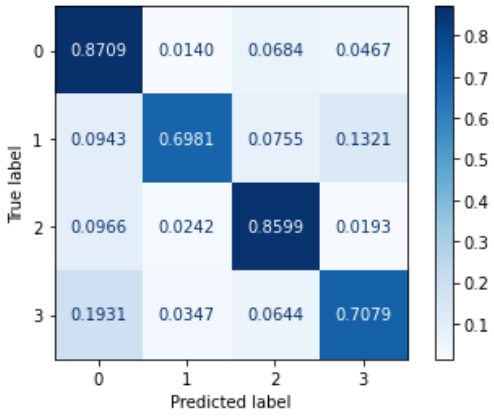


Figura 2.8: Matriz de confusão com ponderação de pesos.

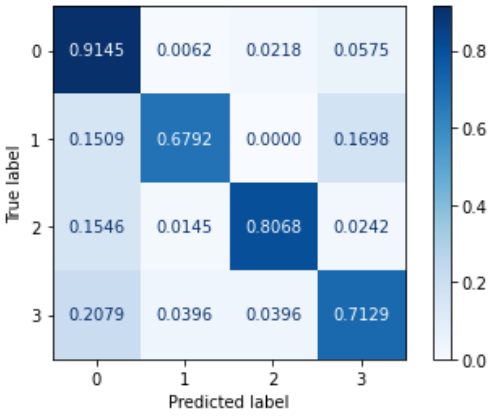
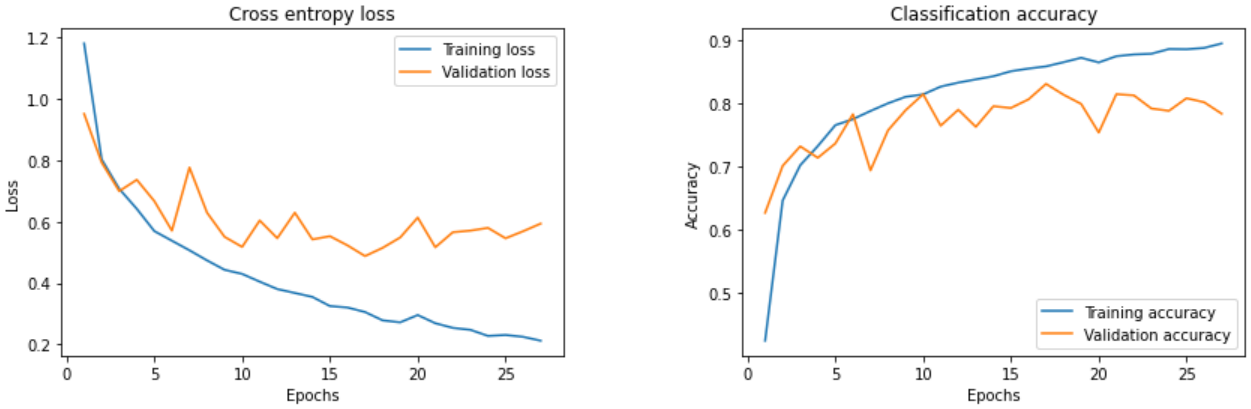


Figura 2.9: Matriz de confusão sem ponderação de pesos.

Analisando as Figuras anteriores, conclui-se a classe mais acertada é a classe referente à etnia caucasiana, seguida da asiática e da indiana, sendo a menos acertada a etnia africana. Isto vai de acordo com o esperado, dada a representatividade de cada etnia no conjunto de dados. Verifica-se que, na generalidade, a percentagem de acerto em classes menos representadas aumenta com a adição de ponderação de pesos, ao contrário do que acontece para a classe mais representada. Dependendo do objetivo, pode ser vantajoso utilizar uma rede CNN com ou sem ponderação destes mesmos pesos.

Novamente, após implementação da rede CNN com melhor BACC, obtêm-se os gráficos da Figura 2.10. A análise destas Figuras é semelhante à análise efetuada para a Figura 2.7. Note-se que, no caso do segundo problema, os valores obtidos para a *cross entropy loss* são superiores aos obtidos no primeiro problema, ao passo que os valores de *classification accuracy* se evidenciam inferiores. Isto reflete a maior dificuldade em

classificar as imagens em quatro classes com base na etnia, quando comparado com a classificação em apenas duas classes baseadas no género.



(a) *Cross Entropy Loss.* **(b)** *Classification Accuracy.*

Figura 2.10: *Cross Entropy Loss* e *Validation Accuracy*, para os dados de treino e de validação, com rede neuronal convolucional otimizada.

Conclusões

A presente secção diz respeito às conclusões gerais do ensaio. Relativamente ao primeiro problema de regressão, concluiu-se que, de forma a minimizar os erros quadráticos médios associados a um modelo de regressão, é necessário ajustar os valores dos hiperparâmetros dos mesmos, caso existam. Comparando os modelos de regressão treinados, depreende-se que o modelo de regressão que melhor se ajustou ao conjunto de dados de treino foi a regressão `Lasso`. Foi utilizado como método de validação o `Leave One Out`, um mecanismo de `Cross Validation`. A escolha deste método prende-se com o facto do mesmo não desperdiçar uma grande quantidade de dados, uma vez que apenas é removida uma amostra do conjunto de treino.

No que concerne o segundo problema de regressão, a principal conclusão consiste no grande impacto que a presença de *outliers* pode causar no desempenho de um modelo. Assim, é de extrema importância o tratamento dos dados antes do ajuste dos modelos aos mesmos. Constatou-se que a identificação do número concreto de *outliers* é bastante desafiante, dada a dificuldade em definir um critério exato a partir do qual se consideram os valores como discrepantes, face aos demais. Assim, o método de remoção de *outliers* pode variar bastante, dependendo do conjunto de dados e da exigência do critério de decisão. Neste problema, após a remoção de oito *outliers*, o modelo de regressão que melhor se adaptou ao conjunto de dados foi o `OrthogonalMatchingPursuit`, apresentado o menor erro quadrático médio entre os cinco modelos estudados.

Por sua vez, em relação aos modelos de classificação, para ambos os problemas, a rede neuronal de convolução apresentou melhores resultados, quando comparada com as restantes, tal como previsto. Verificou-se ainda que a utilização de técnicas de aumento de dados, nomeadamente a técnica de virar horizontalmente, conduz a resultados mais satisfatórios. Contudo, é necessário ter especial atenção ao tipo de técnica a utilizar, uma vez que podem ter o efeito contrário caso não se adequem ao conjunto de dados. Novamente para ambos os problemas, o otimizador '*adam*' revelou-se o que conduzia a melhores resultados, assim como a utilização de duas camadas na rede.

Em suma, concluiu-se que o aumento do número de classes a classificar tem como consequência uma diminuição da *balanced accuracy*. Corrobora-se ainda a noção de que, quando a representatividade das classes no conjunto de dados não é semelhante, é aconselhável a utilização de ponderação pesos, de modo a conduzir a resultados mais realistas.

De forma a melhorar o *balanced accuracy* em ambos os problemas, sugere-se a utilização de imagens com cores, de forma a distinguir melhor as características específicas de cada classe, sobretudo no que respeita as etnias.