Lista 3 - Fundamentos em Redes Neurais e Aprendizagem Estatística

Lorran de Araújo Durães Soares*

2024

Introdução

Este documento refere-se à elaboração da terceira lista de exercícios da disciplina de Fundamentos de Redes Neurais e Aprendizagem Estatística, promovida pelo Laboratório Nacional de Computação Científica (LNCC), ministrada pelo professor Gilson Antonio Giraldi. A lista consiste em três questões, que serão apresentadas neste formato de artigo, detalhando o passo a passo necessário para a resolução e implementação de cada uma delas. Todas as questões foram implementadas na linguagem Python, utilizando notebooks do tipo iPynb e empregando bibliotecas como `Keras` (CHOLLET et al., 2024), `Pandas` (TEAM, 2024b), `Matplotlib` (TEAM, 2024a), `Numpy` (DEVELOPERS, 2024a), `Scikit-learn` (DEVELOPERS, 2024b), `OpenCV` (OpenCV Team, 2024), `Seaborn` (WASKOM et al., 2024) e `PIL` (CONTRIBUTORS, 2024). As implementações de todas as questões serão disponibilizadas ao final da explicação de cada uma delas neste documento.

Para a realização de todas as questões, foi utilizada a base de imagens CIFAR-10, fornecida pelo `Keras`. Esse conjunto de dados é composto por 60.000 imagens coloridas com 32 pixels de largura e 32 pixels de comprimento, distribuídas em 10 classes. Entretanto, foi realizada uma filtragem para trabalhar com a classificação de apenas duas classes de imagens, aviões e carros, reduzindo então o conjunto para 12.000 imagens. Algumas dessas imagens são ilustradas na Figura 1. Além disso, devido à limitações de hardware, nos exercícios 1 e 3, foram utilizados apenas 1.200 das 12.000 imagens disponíveis.

*lorranspbr@gmail.com

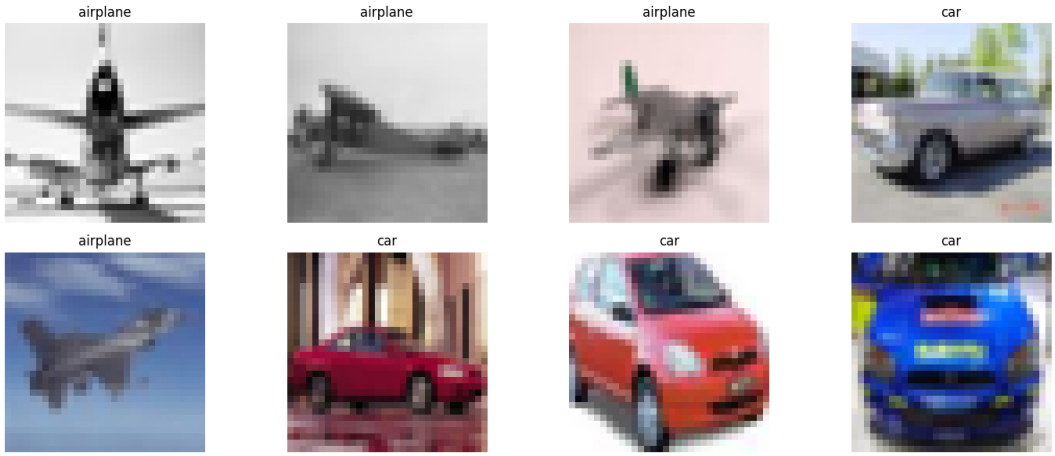

Figura 1 – Amostra do banco de imagens

O banco de imagens utilizado foi importado da classe `datasets.cifar10` da biblioteca `Keras` através da função `load_data()`. A escolha de usar um mesmo banco de dados em todas as questões se objetivou em realizar uma comparação entre as diferentes estratégias de classificação dos dados que cada exercício pede.

Pré-processamento

Para a realização dos exercícios 1 e 3, foi realizado um pré-processamento dos dados, necessário para a execução das operações necessárias nestas questões.

As seguintes operações foram realizadas:

- Conversão para escala de cinza: realizada através da biblioteca `OpenCV`, utilizando a função `cvtColor`;
- Vetorização das imagens: realizada através do método `reshape` da biblioteca `Numpy`;
- Normalização das features: realizada através da ferramenta `StandardScaler` da biblioteca `Scikit-Learn`. O `StandardScaler` ajusta cada feature do conjunto de dados de acordo com a fórmula:

$$z = \frac{x - \mu}{\sigma}$$

Onde:

  - x é o valor original da feature.
  - μ é a média dos valores da feature.
  - σ é o desvio padrão da feature.

Após isso, através da ferramenta `train_test_split` da biblioteca `Scikit-Learn`, o conjunto de dados foi separado com proporção de 70% para treinamento e 30% para teste. Além disso, com o intuito de realizar a comparação sob o mesmo conjunto de dados para todas as questões, foi selecionada a opção `random_state` = 42 na função utilizada para a separação dos dados, fixando a semente de sorteio

dos dados. Na implementação de cada uma das questões foi realizado o cálculo da quantidade de imagens de cada classe no conjunto de treinamento, com o intuito de observar possíveis desbalanceamentos de dados. Como mostra a Figura 2, as duas classes tinham números de dados iguais, não sendo necessário, portanto, a exclusão ou adição de novos dados no conjunto de treinamento para a realização dos exercícios.

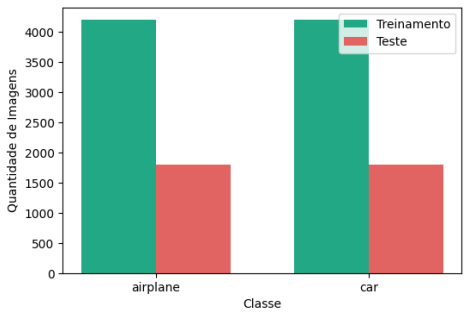

Figura 2 – Histograma da quantidade de imagens por classe para treinamento e teste

Questão 1

Considere um banco de dados de imagens e um problema de classificação. Aplique validação cruzada leave-one-out multi-fold explicada na seção 8.5 de (GIRALDI, 2024), com $K = 4$, e SVM como segue:

(a) SVM Linear não separável com espaço de características obtido através do KPCA.

(b) SVM Kernel não separável com espaço de características obtido através do PCA.

(c) Compare os resultados dos itens (a) e (b).

Para a realização deste exercício, especificamente para o proposto na letra (a), foi inicialmente realizado o cálculo do Kernel Principal Component Analysis (KPCA) através da biblioteca `Scikit-Learn` com a função `KernelPCA`, com o intuito de determinar o número de componentes principais de todas as bases que serão calculadas posteriormente para cada fold. Foi empiricamente escolhido um kernel polinomial para o cálculo da matriz KPCA com todas as componentes, para posteriormente selecionar as principais componentes. O conjunto de dados projetados nas duas principais componentes do KPCA está presente na Figura 3.

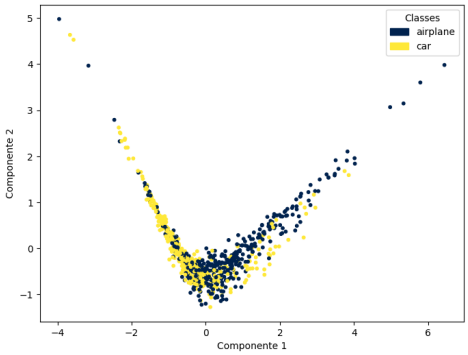

Figura 3 – Projeção do conjunto de treinamento na base obtida pelo KPCA

A função para calcular a base KPCA também retorna os autovalores correspondentes aos autovetores que formam as colunas da base KPCA obtida no espaço de características. Assim, foi possível calcular a variância explicada dividindo cada autovalor pela soma de todos os autovalores, que estão em ordem decrescente. Utilizando a biblioteca `Matplotlib`, foi gerado o gráfico mostrado na Figura 4, que ilustra a soma cumulativa das variâncias explicadas. Esse gráfico permite avaliar o número de componentes da base KPCA necessários para representar uma certa porcentagem da variância total do conjunto de dados.

Para determinar a quantidade ideal de componentes principais, utilizou-se a biblioteca `Numpy` com a função `argmax` para calcular o número de componentes necessários para explicar no mínimo 95% da variância dos dados, com base na variância explicada cumulativa. O resultado indicou que são necessários 435 componentes. A partir disso, a matriz KPCA calculada anteriormente foi reduzida para este número de componentes.

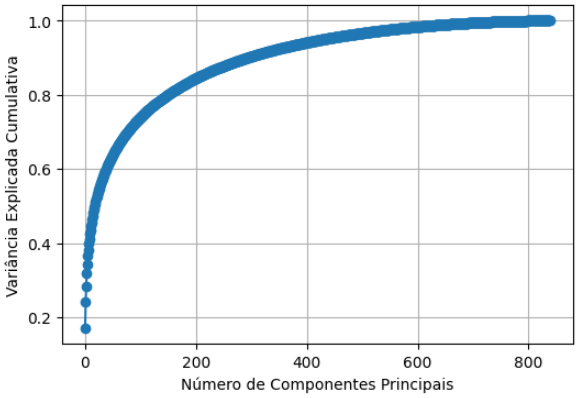

Figura 4 – Gráfico da variância explicada cumulativa por componentes KPCA

Utilizando a técnica de K-Fold Cross-Validation, os dados são divididos em k subconjuntos (ou "folds") de tamanhos aproximadamente iguais. Em cada iteração, um dos k folds é reservado como conjunto de validação, enquanto os $k-1$ folds restantes são utilizados para treinamento, resultando em k modelos distintos. Para este experimento, foi utilizado `K-fold` $= 4$, sendo realizado o PCA para o conjunto de treinamento correspondente de cada fold através da função `pca.tranform()`, reduzindo a dimensionalidade deste conjunto de acordo com o número de componentes encontradas anteriormente, neste caso, 435. Logo após, a função `SVC` da biblioteca `Scikit-Learn` foi empregada para calcular o modelo de Máquinas de Vetores de Suporte (SVM), para o caso não separável, com o objetivo de realizar a classificação dos dados. Esta construção é baseada na minimização da função custo τ decrita abaixo:

$$\tau(\phi, \xi, C) = \frac{1}{2}\|\phi\|^2 + C\sum_{i=1}^{M}\xi_i$$

Onde:

- $\frac{1}{2}\|\phi\|^2$: É o termo de regularização que controla a complexidade do modelo, minimizando o valor dos pesos (norma de ϕ).
- C: É um hiperparâmetro que controla o trade-off entre a margem da fronteira de decisão e os erros de classificação. Valores maiores de C dão mais ênfase à minimização dos erros ξ_i, enquanto valores menores favorecem uma maior margem com a penalização menor dos erros.

- $\sum_{i=1}^{M} \xi_i$: Soma das variáveis de erro (ξ_i), que representam a quantidade pela qual as amostras violam a margem suave. Quanto maior a soma dessas variáveis, maior é o número de amostras classificadas incorretamente ou dentro da margem.
- M: O número total de amostras.

Foi utilizado a constante $C = 1$ para a realização desta questão, definida experimentalmente.

Os resultados de acurácia obtida para os modelos de cada fold, tanto para os conjuntos de validação quanto para os conjuntos de teste, estão apresentados na Tabela 1. Em média, observa-se que o classificador apresentou uma maior acurácia nos conjuntos de teste em comparação com os conjuntos de validação, obtendo, em ambos casos, acurácias superiores à casa dos 78%, chegando a 86% no conjunto de testes.

Conjunto	Fold 1	Fold 2	Fold 3	Fold 4	Média
Validação	0.78	0.78	0.78	0.84	0.795
Teste	0.82	0.86	0.81	0.84	0.8325

Tabela 1 – Acurácia da classificação do SVM Linear com o KPCA

Com o intuito de promover uma visualização gráfica para exemplificar como seria a reta de separação para apenas as duas principais componentes obtidas pelo cálculo do KPCA, foi realizado o treinamento de um novo SVM linear sobre o conjunto de dados projetados apenas nas duas principais componentes do KPCA, obtendo a reta de separação mostrada na Figura 5, juntamente com a projeção do conjunto de teste nesta base. Neste caso, a acurácia média obtida no conjunto de testes foi de 69%, inferior à de 79% obtida em média anteriormente com todas as componentes principais, evidenciando que a redução de dimensionalidade para apenas duas componentes principais impactou negativamente na performance do classificador. Além disso, a margem de separação construída possui um tamanho considerável, onde, nesta região, o classificador teve muita incerteza ao apontar de qual classe eram os dados.

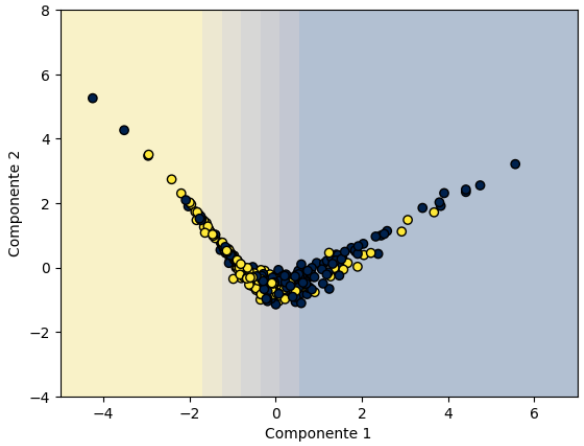

Figura 5 – Reta de decisão do SVM Linear com KPCA

Para a realização do item (b), foi inicialmente realizado o cálculo do Principal Component Analysis (PCA) utilizando a biblioteca `Scikit-Learn` com a função `PCA`, também com o intuito de determinar o número de componentes para todas as bases PCA que serão calculadas em cada fold. Neste primeiro passo, foram calculadas todas as componentes principais, permitindo posteriormente a seleção das mais relevantes para este conjunto de dados. A projeção do conjunto de dados na base PCA está representada na Figura 6.

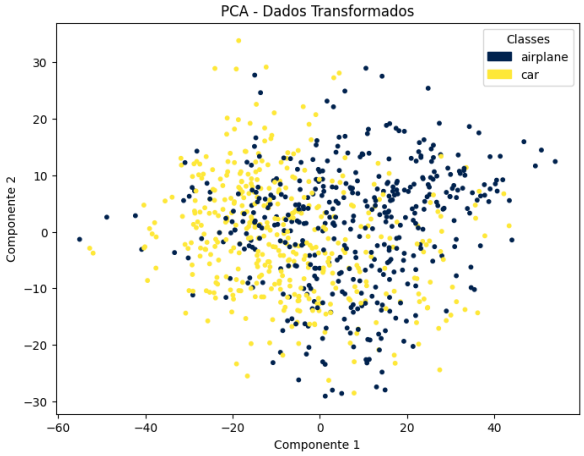

Figura 6 – Projeção do conjunto de treinamento na base PCA

Após o cálculo das componentes principais, foi gerado o gráfico da variância explicada cumulativa, fornecido pelo próprio cálculo do PCA através do método `explained_variance_ratio_`. O resultado obtido está ilustrado na Figura 7. Para selecionar o número adequado de componentes principais, foi utilizada a função `argmax` da biblioteca `Numpy`, determinando o número de componentes necessárias para explicar pelo menos 95% da variância dos dados, resultando em 123 componentes. Com isso, a matriz de dados transformada pelo PCA será reduzida para conter apenas essas componentes.

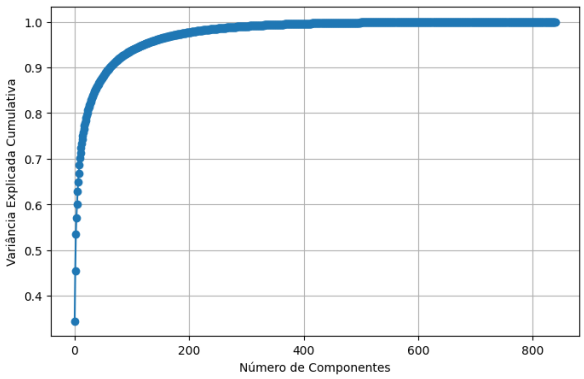

Figura 7 – Gráfico da variância explicada cumulativa para as componentes do PCA

Em seguida, foi aplicada a estratégia `K-fold` com valor 4, com a mesma subdivisão de dados realizada no item (a), para que fosse posteriormente realizada uma comparação entre os modelos. Novamente, foi realizado o cálculo de PCA para o conjunto de treinamento respectivo a cada fold, reduzindo a dimensionalidade para as 123 componentes principais. Utilizando a função `SVC` da biblioteca

`Scikit-Learn`, foi construído um Kernel Support Vector Machine (KSVM), com kernel polinomial definido empiricamente, com o objetivo de realizar a classificação dos dados, obtendo as acurácias mostradas na Tabela 2 para os conjuntos de validação e teste. Novamente, a constante de regularização considerada foi $C = 1$, definida experimentalmente.

Conjunto	Fold 1	Fold 2	Fold 3	Fold 4	Média
Validação	0.81	0.70	0.73	0.75	0.7475
Teste	0.83	0.84	0.77	0.81	0.8125

Tabela 2 – Acurácia da classificação do SVM Polinomial com o PCA

Com o intuito de visualizar o plano de separação utilizando apenas as duas componentes principais obtidas pelo PCA, foi treinado um novo SVM com kernel polinomial sobre o conjunto de treinamento projetado na base PCA reduzida apenas para suas duas componentes principais, gerando a curva de separação presente na Figura 8. Essa figura também mostra a projeção do conjunto de testes nas duas primeiras direções principais. Nesse cenário, a acurácia média alcançada sobre o conjunto de validação foi de 62%, evidenciando mais uma vez que a redução de dimensionalidade para duas componentes impactou negativamente na acurácia do classificador.

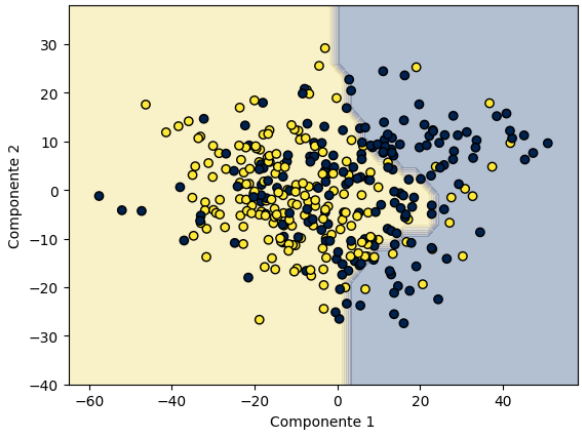

Figura 8 – Curva de decisão do SVM Polinomial com PCA

Observando as Tabelas 1 e 2, podemos notar que, no geral, o modelo proposto no item (a) obteve uma maior acurácia na classificação das imagens, tanto no conjunto de treinamento quanto no de conjunto de validação. Além disso, a matriz de confusão presente na Figura 10, construída sobre o conjunto de testes, mostra que, por exemplo, para o fold igual a 3 no modelo proposto no item (b), houve uma quantidade significativa de predições erradas, onde aviões foram preditos como carros. Este problema não ocorreu no modelo proposto no item (a), como mostrado na Figura 9, o que também evidencia sua performence superior na tarefa em questão.

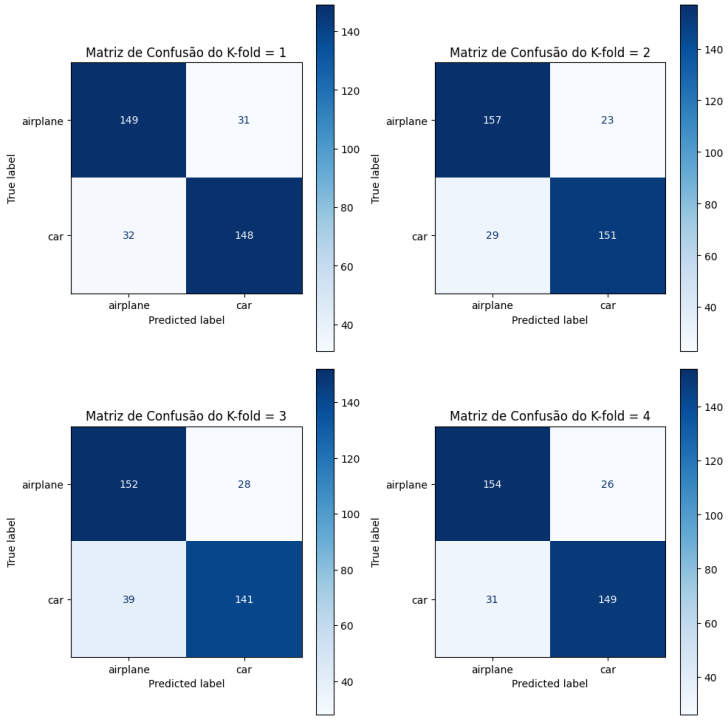

Figura 9 – Matriz de convolução dos modelos do item (a)

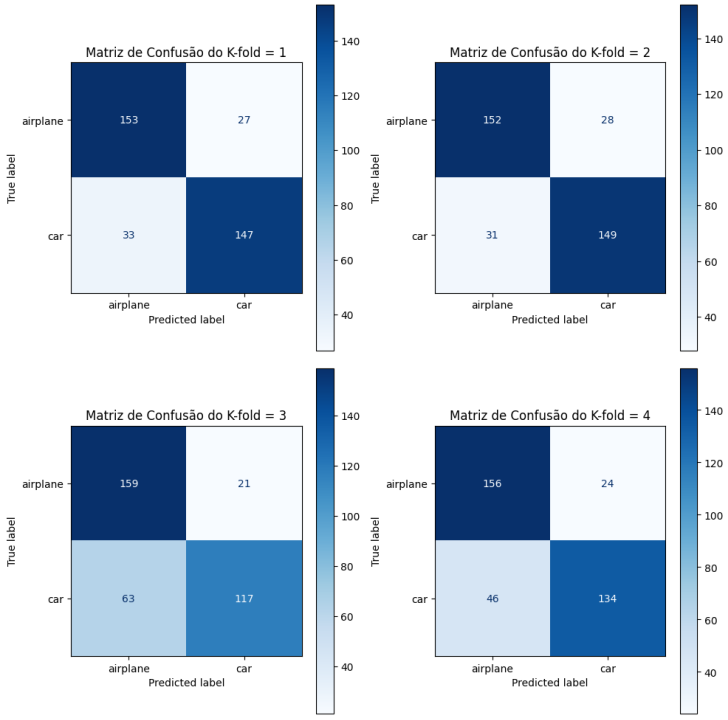

Figura 10 – Matriz de convolução dos modelos do item (b)

Para acessar o código referente à realização deste exercício, clique aqui.

Questão 2

Considere um banco de dados e um problema de classificação. Aplique a validação cruzada multi-fold leave-one-out explicada na seção 8.5 de (GIRALDI, 2024), com $K = 5$, para um modelo de CNN. Utilize as facilidades disponíveis em bibliotecas para a implementação de redes neurais, como Keras, TensorFlow, etc.

(a) Mostre a representação gráfica da evolução das etapas de treinamento e validação (ver Figura 8.8 da monografia do curso).

(b) Realize uma análise estatística do desempenho (seção 8.6) dos cinco modelos aplicados sobre a $\mathbb{D}_{te}$.

Para a realização deste exercício, foram utilizadas todas as 12.000 imagens da base de dados CIFAR-10 referentes às classes aviões e carros.

O modelo da Rede Neural Convolucional (CNN) foi construído empiricamente, com arquitetura inspirada na rede Virtual Geometry Group Network (VGGNet), criada por Simonyan e Zisserman, pesquisadores da Universidade de Oxford, em 2014. A principal inovação da VGGNet foi o uso de blocos de convolução com filtros de tamanho 3x3, onde essa arquitetura ajudou a melhorar a precisão das classificações (SIMONYAN; ZISSERMAN, 2014).

A arquitetura construída com essa inspiração está descrita na Tabela 3.

Camadas	Tipo	Filtros/Neurônios	Tam. do filtro	Padding	F. Ativação
1ª Conv e Pool	Convolucional	32	(3,3)	same	ReLU
	Convolucional	32	(3,3)	same	ReLU
	Max Pooling	-	(2,2)	-	-
2ª Conv e Pool	Convolucional	64	(3,3)	same	ReLu
	Convolucional	64	(3,3)	same	ReLU
	Max Pooling	-	(2,2)	-	-
3ª Conv e Pool	Convolucional	128	(3,3)	same	ReLU
	Convolucional	128	(3,3)	same	ReLU
	Max Pooling	-	(2,2)	-	-
Flatten	Flatten	-	-	-	-
Dense	Dense	512	-	-	ReLU
Saída	Dense	2	-	-	Softmax

Tabela 3 – Arquitetura do modelo da rede CNN

Logo, foram usadas no total 6 camadas do tipo convolucionais, 3 camadas de pooling e uma camada densa Multi Layer Perceptron (MLP), onde:

- A função de ativação Rectified Linear Unit(ReLU) foi utilizada nas MLP e convolucionais da rede, como na arquitetura da VGGNet, dada por:

$$\text{ReLu}(x) = \max(0, x).$$

- A função Softmax foi usada na última camada. Geralmente ela é utilizada para problemas de classificação multiclasse. Ela transforma os valores de entrada em uma distribuição de probabilidade, onde cada valor de saída é interpretado como a probabilidade da entrada pertencer a uma das classes. A função Softmax é definida como:

$$\text{softmax}(\mathbf{z}) = \left(\frac{\exp(z_1)}{\sum_{j=1}^{K} \exp(z_j)}, \frac{\exp(z_2)}{\sum_{j=1}^{K} \exp(z_j)}, \ldots, \frac{\exp(z_K)}{\sum_{j=1}^{K} \exp(z_j)} \right).$$

- Foi utilizado o parâmetro das redes convolucionais padding igual a same, que significa que haverá um preenchimento uniforme nas bordas com zeros, e strides (1,1), de modo que as saídas das camadas convolucionais terão o mesmo tamanho que as entradas delas (CHOLLET et al., 2024).

Foram usadas também algumas outras estratégias na construção da rede, como:

- Batch Normalization: técnica usada para melhorar a estabilidade e a velocidade do treinamento de redes neurais profundas. Ela funciona normalizando a entrada de cada camada para ter média zero e variância unitária. A ideia é normalizar as saídas das camadas para que estejam em uma faixa específica, o que permite que o treinamento seja mais estável e rápido (IOFFE, 2015). Camadas de batch normalization foram aplicadas nas saídas de todas as camadas CNN e MLP.
- Dropout: técnica de regularização usada para prevenir o overfitting em redes neurais. Durante o treinamento, o dropout desativa aleatoriamente uma fração dos neurônios em uma camada, forçando a rede a aprender representações mais robustas e a não depender excessivamente de nenhum neurônio específico. Isso faz com que a rede neural generalize melhor, pois cada unidade não pode confiar excessivamente em outras unidades (SRIVASTAVA et al., 2014). Para este modelo, um dropout de 20% foi aplicado a última camada da primeira, segunda e terceira camada Conv e Pool. Além disso, foi usado um dropout de 25% na camada MLP.

Para função de perda (Loss), foi utilizada a categorical cross-entropy (CCE), que é usada para problemas de classificação multiclasse. Ela mede a divergência entre a distribuição de probabilidade prevista pela rede (saída do softmax) e a verdadeira distribuição de rótulos. A CCE é definida da seguinte forma:

$$\text{CCE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

onde:

- N: é o número de amostras no conjunto de dados.
- C: é o número de classes.
- y_{ij}: é o rótulo verdadeiro para a i-ésima amostra e a j-ésima classe. Em uma codificação one-hot, y_{ij} é 1 se a amostra i pertence à classe j, e 0 caso contrário.
- $\hat{y}_{ij}$: é a probabilidade prevista pelo modelo de que a i-ésima amostra pertence à j-ésima classe. Este valor é a saída do modelo após a aplicação da função softmax.

O otimizador utilizado foi o Stocastic Gradiente Descendent (SGD), considerando learning rate igual a 0.001 e um momentum igual a 0.9. Logo, a atualização dos pesos será dada da seguinte forma (CHOLLET et al., 2024):

$$velocity = velocity - learning\_rate * g$$

$$w = w + velocity$$

onde g é o gradiente da função Loss. Além disso, foi utilizada a regularização L^2 com parâmetro $\lambda = 1e-6$, a qual adiciona um termo de penalização à função de perda original, que ajuda a manter os pesos da rede menores e mais controlados, pois estes são considerados na função de perda. Logo, a função Loss regularizada é dada por:

$$\text{Loss} = CCE + \frac{\lambda}{2}\sum_{i=1}^{n} w_i^2$$

onde λ é o parâmetro de regularização e w_i são os pesos do modelo.

A inicialização dos pesos é realizada através de uma distribuição uniforme, denominada Glorot Uniform. O intervalo do qual os pesos são amostrados é definido por:

$$w \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right)$$

Onde:

- w são os pesos.
- n_{in} é o número de unidades de entrada da camada (ou seja, o número de neurônios na camada anterior).
- n_{out} é o número de unidades de saída da camada (ou seja, o número de neurônios na camada atual).
- $\mathcal{U}(a, b)$ representa uma distribuição uniforme entre a e b.

Já os bias são inicializados com valores iguais a zero.

Para o treinamento do modelo, utilizando a biblioteca `Keras`, foi considerada uma estratégia de early stopping da classe `callbacks` da função `fit` para evitar overfitting. Considerando a métrica de loss da validação como a quantidade a ser monitorada no early stopping, temos que os seguinte parâmetros para este critério de parada foram utilizados:

- *min_delta* = 0.0001: quantidade mínima para ser considerada como melhoria;
- *patience* = 6: número de épocas sem melhorias a quais o treinamento será interrompido;
- *restore_best_weights* = True: restaura para a rede os pesos referentes à epoca com melhor resultado na métrica escolhida.

Finalmente, foi realizado o treinamento cdo modelo usando $K - Fold = 5$, como determinado pela questão. Para isso, foi considerado o batch size tamanho 64 e 50 épocas, que foram definidos empiricamente. A evolução da acurácia e da loss, para o treinamento e a validação, estão presentes nos gráficos das figuras 11, 12, 13,14 e 15 . Através do gráfico, podemos perceber no geral que a evolução

da loss não demonstrou a ocorrência de overfitting, mas pode se notar uma oscilação na acurácia do conjunto de validação do treinamento.

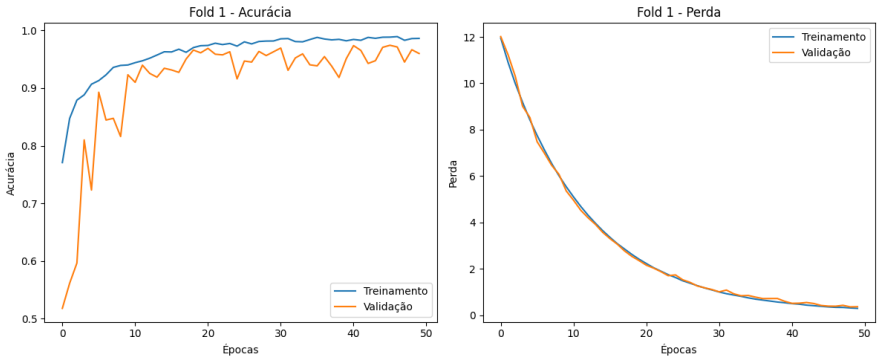

Figura 11 – Evolução da acurácia e da loss sob o conjunto de treinamento e de validação para o fold 1

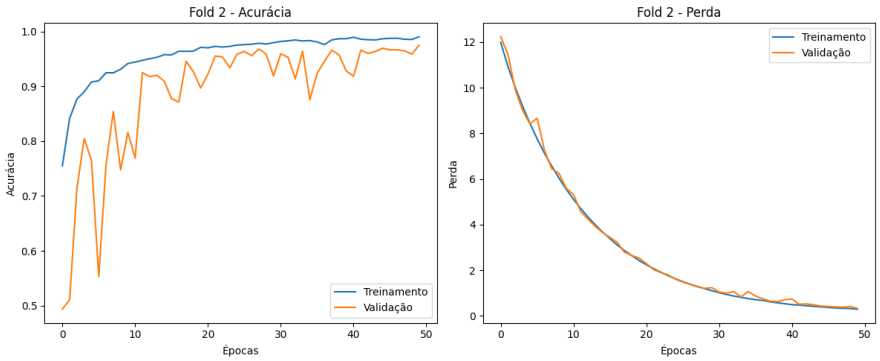

Figura 12 – Evolução da acurácia e da loss sob o conjunto de treinamento e de validação para o fold 2

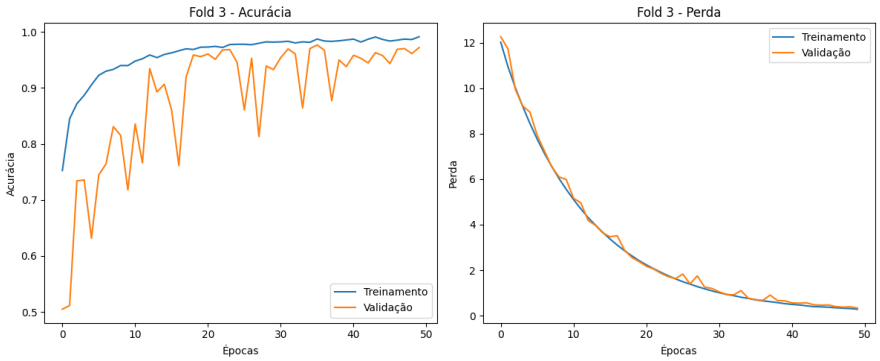

Figura 13 – Evolução da acurácia e da loss sob o conjunto de treinamento e de validação para o fold 3

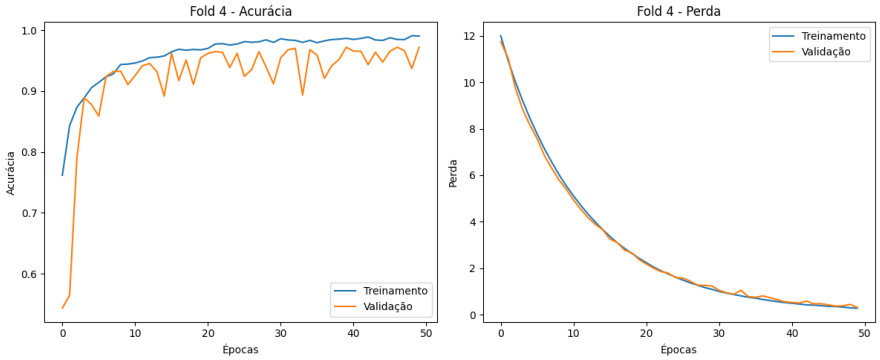

Figura 14 – Evolução da acurácia e da loss sob o conjunto de treinamento e de validação para o fold 4

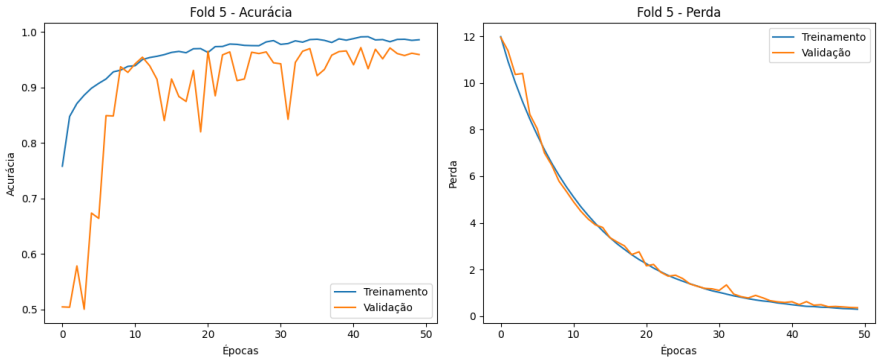

Figura 15 – Evolução da acurácia e da loss sob o conjunto de treinamento e de validação para o fold 5

Os modelos apresentaram uma acurácia final média de 97% sobre o conjunto de validação. Com as ferramentas `confusion_matrix` e `classification_report`, presentes na biblioteca `Scikit-Learn`, podemos então plotar a matriz de confusão relativa ao modelo de cada fold, presente na Figura 16, onde 0 é referente a aviões e 1 a carros, e também calcular as métricas estátisticas descritas na seção 8.6 de (GIRALDI, 2024), com precisão de duas casas decimais, obtendo então os resultados, sobre o conjunto de teste, presentes na tabela 4.

Métricas	K=1	K=2	K=3	K=4	K = 5	Média
Macro Precision of the classifier	0.96	0.97	0.97	0.97	0.98	0.97
Macro Recal of the Classifier	0.96	0.97	0.97	0.97	0.97	0.97
Macro F1 score	0.96	0.97	0.97	0.97	0.97	0.97
Multi-Class Acuracy	0.96	0.97	0.97	0.97	0.97	0.97
Avanged Error Rate	0.04	0.03	0.03	0.03	0.03	0.03

Tabela 4 – Métricas de performance dos modelos aplicados ao conjunto de testes

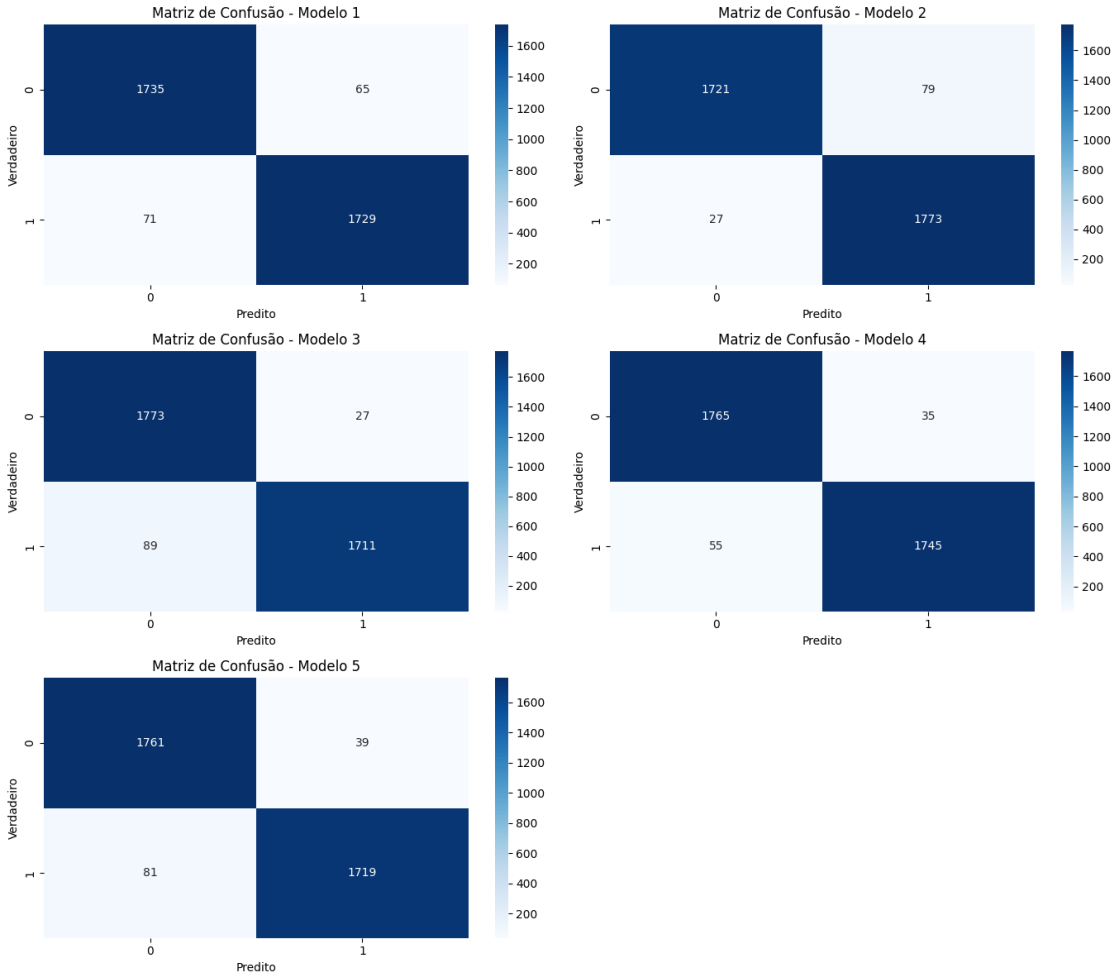

Figura 16 – Matriz de confusão dos modelos sobre o conjunto de testes

Através dessas métricas, podemos observar que, no geral, os modelos se comportaram de maneira bastante semelhante, atingindo valores próximos para as métricas calculadas. A matriz de confusão também evidencia que não houve nenhuma classe com uma quantidade considerável de falsos positivos ou falsos negativos, resultando em uma ótima acurácia média de 97%.

Para acessar o código referente à realização deste exercício, clique aqui.

Questão 3

Considere um banco de dados de imagens e um problema de classificação.

(a) *Aplique a validação cruzada multi-fold leave-one-out explicada na seção 8.5 de (GIRALDI, 2024), com $K = 4$ usando LDA no espaço reduzido de PCA e realize a classificação sobre o conjunto de teste. Analise os resultados.*

(b) *Aplique a validação cruzada multi-fold leave-one-out explicada na seção 8.5 de (GIRALDI, 2024), com $K = 4$ e SVM Kernel não separável com espaço de características obtido através da Análise Discriminante de Componentes Principais (DPCA).*a

(c) *Compare os resultados obtidos nos itens (a) e (b) acima.*

Para a realização da letra (a), foi então calculado um PCA e efetuada a redução de dimensionalidade, para cada fold, de forma análoga à realizada anteriormente na questão 1, na letra (b). Mas, agora, ao invés de usar um Kernel SVM para realizar a classificação dos dados, foi realizado o treinamento do Linear Discriminant Analysis (LDA), com K-fold = 4, com o número de componentes iguais a 1 (pois temos apenas duas classes) sobre o conjunto obtido pela projeção do conjunto de dados na base PCA, através da biblioteca `Scikit-Learn` com a função `lda.fit`. A acurácia obtida na classificação do conjunto de validação e de teste através do LDA está descrita na tabela 5.

Conjunto	Fold 1	Fold 2	Fold 3	Fold 4	Média
Validação	0.82	0.75	0.72	0.77	0.765
Teste	0.77	0.81	0.81	0.81	0.8

Tabela 5 – Acurácia da classificação do LDA com o PCA

Para a realização do item (b) deste exercício, foi realizado o seguinte passo a passo para o cálculo da base Discriminant Principal Components Analysis (DPCA):

1. Usando a projeção dos dados na base PCA calculada na questão anteriormente, foi então calculado o SVM com através da função `SVC`;
2. O método .support_vectors_ permite que o vetor discrimante (pesos) gerados pelo treinamento do SVM possam ser acessados. Foi então calculada a norma desses pesos através da função `np.linalg.norm` do `Numpy`;
3. Estes pesos foram organizados em ordem decrescente através da função `argsort` também do `Numpy`;
4. Através do método `pca.components_`, foi obtida os vetores da base PCA calculada anteriormente e ordenada conforme a orientação em ordem decrescente dos pesos realizada no passo anterior;
5. Foi projetado então o conjunto de dados nesta base obtida, denominada base DPCA.

Finalmente, foi treinado um KSVM com kernel polinomial, onde todo o processo descrito foi realizado com K-fold igual a 4, ou seja, foi treinada uma base DPCA para cada fold, reduzindo a dimensionalidade de acordo com o número de componentes principais encontradas no PCA da letra (a), neste caso, 123 componentes. Foram obtidas assim então as acurácias presentes na Tabela 6 para o conjunto de validação e de teste.

Conjunto	Fold 1	Fold 2	Fold 3	Fold 4	Média
Validação	0.82	0.70	0.73	0.75	0.75
Teste	0.84	0.83	0.77	0.82	0.815

Tabela 6 – Acurácia da classificação do KSVM com o DPCA

Pelo resultado obtido das acurácias, podemos afirmar que o desempenho dos modelos de classificação propostos nos itens (a) e (b) obtiveram performances similares, tendo o modelo construído no item (b) obtido uma acurácia ligeiramente melhor. Além disso, a matriz de confusão presente na Figura 17 e 18 mostra que para o K-fold igual a 3, o item (a) teve um desempenho melhor, tendo o modelo do item (b) para este fold um número considerável de falsos positivos da classe carros.

Este fenômeno também foi observado na questão 1 e item (b), mostrando que o conjunto de dados de treinamento produzidos pelo fold 3, que é o mesmo para ambas as questões, influenciaram negativamente na construção destes modelos em relação aos conjuntos produzidos pelos outros folds.

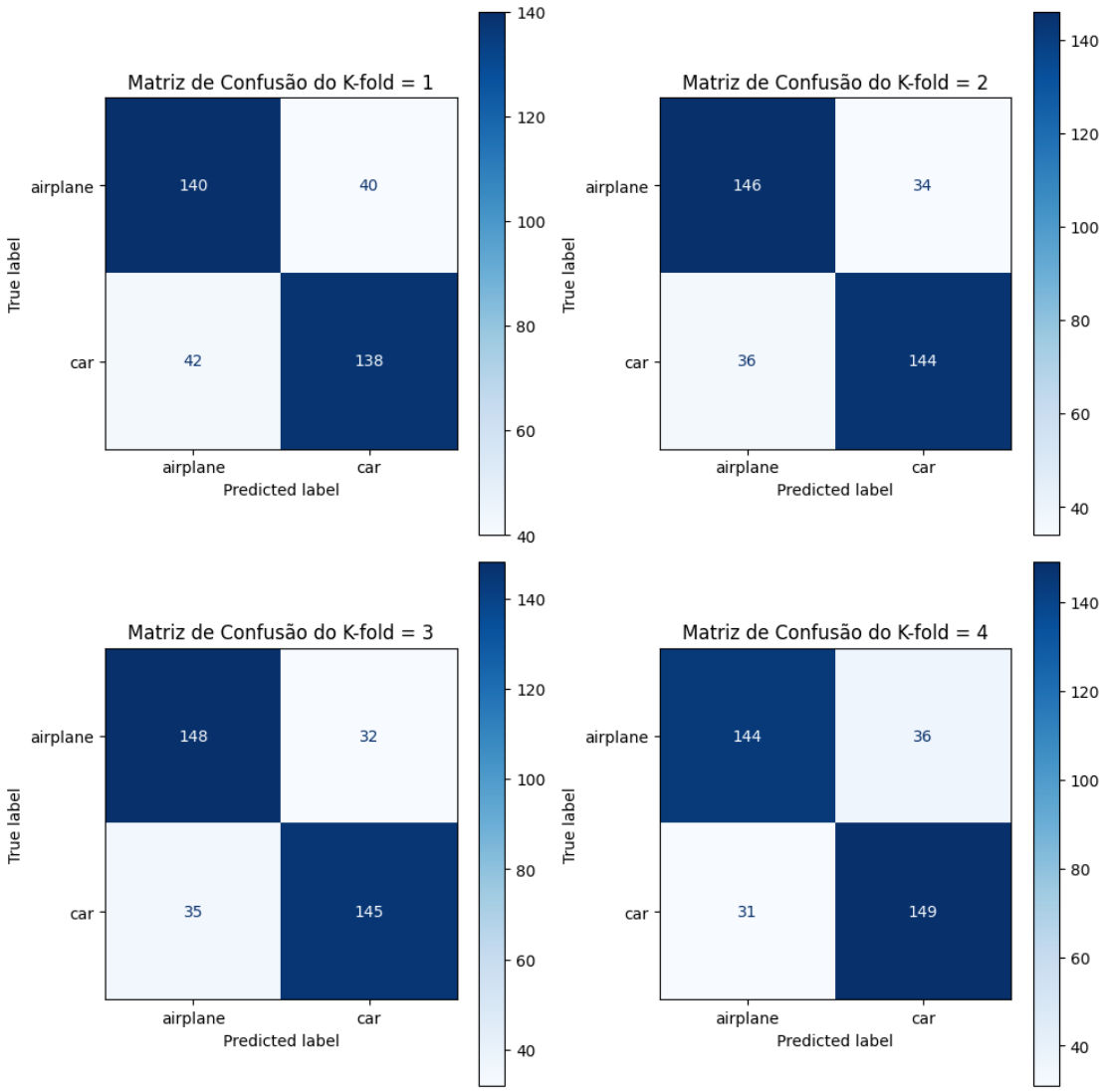

Figura 17 – Matriz de confusão dos modelos do item (a) sobre o conjunto de testes

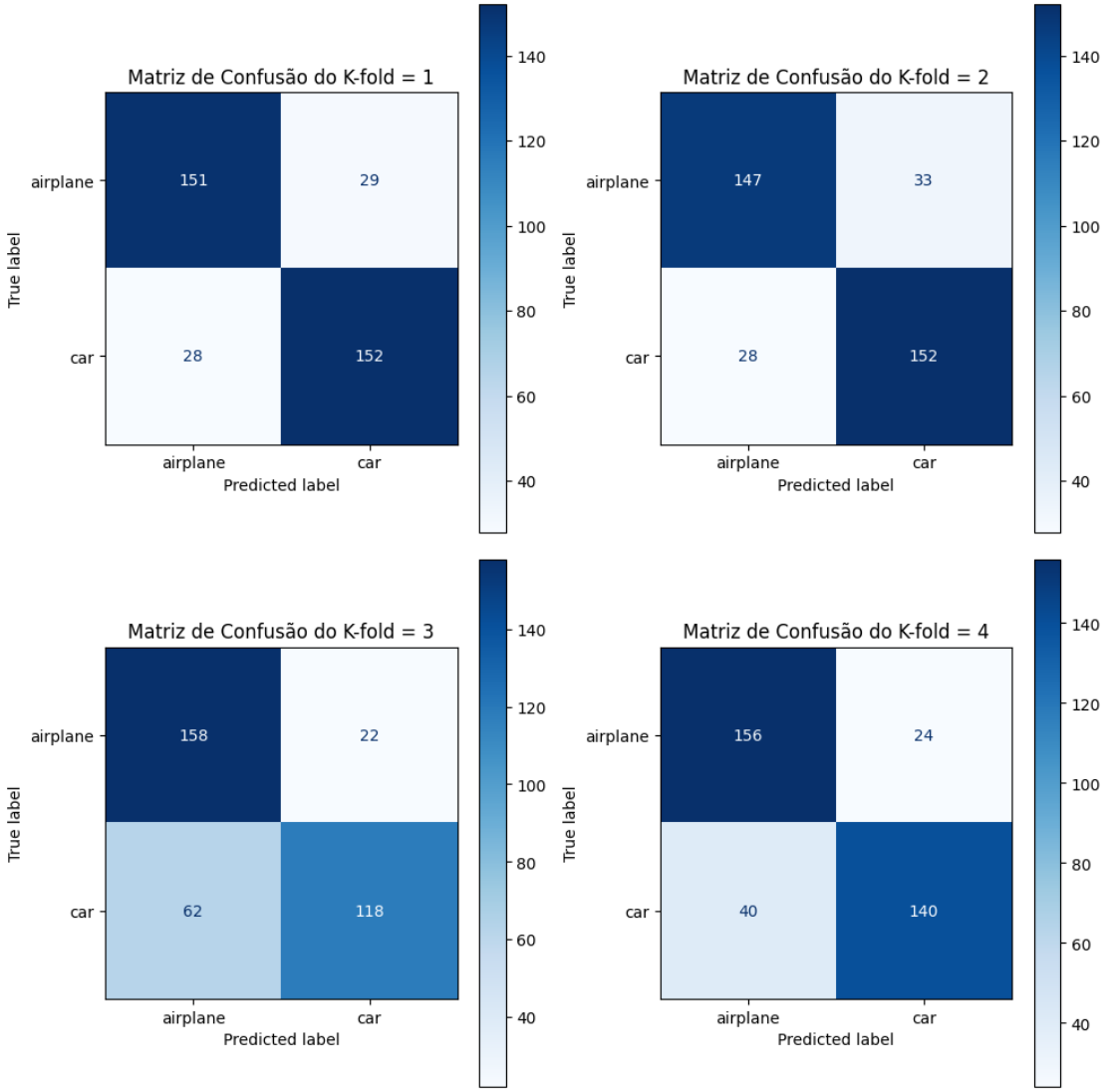

Figura 18 – Matriz de confusão dos modelos do item (b) sobre o conjunto de testes

Para acessar o código referente à realização deste exercício, clique aqui.

Referências

CHOLLET, F. et al. *Keras documentation*. 2024. <https://keras.io/>. Accessed: 2024-08-22. Citado 2 vezes nas páginas 1 e 10.

CONTRIBUTORS, P. *Pillow documentation*. 2024. <https://pillow.readthedocs.io/>. Accessed: 2024-08-22. Citado na página 1.

DEVELOPERS, N. *NumPy documentation*. 2024. <https://numpy.org/doc/>. Accessed: 2024-08-22. Citado na página 1.

DEVELOPERS, S. learn. *Scikit-learn documentation*. 2024. <https://scikit-learn.org/stable/>. Accessed: 2024-08-22. Citado na página 1.

GIRALDI, G. A. Fundamentals of neural networks and statistical learning. 2024. Citado 4 vezes nas páginas 3, 9, 13 e 14.

IOFFE, S. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Citado na página 10.

OpenCV Team. *OpenCV documentation*. 2024. <https://opencv.org/>. Accessed: 2024-08-22. Citado na página 1.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado na página 9.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014. Citado na página 10.

TEAM, M. D. *Matplotlib documentation*. 2024. <https://matplotlib.org/stable/contents.html>. Accessed: 2024-08-22. Citado na página 1.

TEAM, P. D. *Pandas documentation*. 2024. <https://pandas.pydata.org/docs/>. Accessed: 2024-08-22. Citado na página 1.

WASKOM, M. et al. *Seaborn: statistical data visualization*. 2024. <https://seaborn.pydata.org/>. Accessed: 2024-08-22. Citado na página 1.