

UNIVERSIDADE DE SÃO PAULO – USP
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO

SCC0276 — APRENDIZADO DE MÁQUINA

PROJETO 3

Alice Valença De Lorenci - 11200289
Gabriel Soares Gama - 10716511
Marcos Antonio Victor Arce - 10684621

Sumário

Sumário	1
1 Objetivos	2
2 Parte 1	2
2.1 Objetivos	2
2.2 Dificuldade do Problema	2
2.3 Benchmarks	2
2.4 Conjuntos de dados	4
2.4.1 Disponibilidade de dados	4
2.4.2 Limpeza dos dados	4
2.4.3 Distribuição dos dados	6
2.4.4 Pré-Processamento	7
3 Parte 2	9
3.1 Objetivos	9
3.2 Pré-Processamento	9
3.2.1 Skull stripping	9
3.2.2 Bias field correction	10
3.2.3 Extração de features	11
3.2.4 Balanceamento	11
3.3 Métodos utilizados	13
3.3.1 KNN	13
3.3.2 SVM	14
3.3.3 MLP	16
3.4 Conclusão	18
3.5 Código fonte	19
4 Parte 3	20
4.1 Objetivos	20
4.2 MobileNet V2	20
4.2.1 Aspectos fundamentais	20
4.2.2 Arquitetura	24
4.3 Experimentos	25
4.4 Conclusão	28
4.5 Código fonte	29
Referências	30

1 Objetivos

O objetivo desse projeto é aplicar técnicas de aprendizado de máquina a fim de determinar se um paciente possui doença de Alzheimer (AD); comprometimento cognitivo leve (*early* MCI, MCI ou *late* MCI), que é uma etapa transicional entre envelhecimento cognitivo natural e doença de Alzheimer; ou se é cognitivamente normal (CN) [1].

Essa classificação é de interesse prático pois permite a detecção precoce de Alzheimer (detecção de grau de comprometimento cognitivo leve) e a adoção de medidas preventivas.

2 Parte 1

2.1 Objetivos

O objetivo da primeira parte do projeto é descrever as particularidades do problema escolhido quando à dificuldade do problema, limpeza de dados, *benchmarks* (métricas usadas para comparação de modelos) e disponibilidade de dados.

2.2 Dificuldade do Problema

Dado o contexto do problema, existem certas dificuldades em sua execução, como a obtenção das imagens, sua seleção e seu pré-processamento.

Para construir a base de dados, deve-se obter várias imagens MRI (Magnetic Resonance Imaging) cerebrais de múltiplos pacientes com uma diversificação de idade e sob diferentes estágios da doença.

Diante dos dados obtidos, há a necessidade de fazer um pré-processamento complexo para remover aspectos indesejados e isso requer conhecimentos específicos da área médica e da técnica de MRI. Em vista disso, é necessário se basear em diferentes artigos sobre a literatura tais como [2] e [3].

Além disso, construir um modelo que seja adequado para dados reais é uma tarefa complexa. Isso ocorre devido ao fato de que os dados reais podem ser totalmente diferentes dos dados utilizados no treinamento, tal como existem diferentes métodos de geração de imagem MRI e neste projeto, utiliza-se apenas um.

2.3 Benchmarks

Visto que em etapas posteriores do projeto será comparado o desempenho de diferentes métodos de aprendizado de máquina, foi necessário identificar os *benchmarks* utilizados em imagens médicas MRI, isto é, quais métricas são adotadas na comparação de modelos. Tais *benchmarks* foram identificados a partir do artigo [2], em que são sintetizados os resultados de diversas pesquisas, sendo que as métricas mais comumente utilizadas foram:

acurácia (ACC), sensibilidade (SEN), especificidade (SPE) e área sob a curva ROC (AUC). Deste modo, esse conjunto de métricas foi adotado como *benchmark* para o projeto.

Vale ressaltar que as métricas supracitadas expressam as seguintes características do modelo:

- **Acurácia:** Percentual da taxa de acerto, independente da classe

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

- **Sensibilidade:** Chance de ser corretamente classificado, dado que o paciente possua tal condição.

$$SEN = \frac{TP}{TP + FN} \quad (2)$$

- **Especificidade:** Chance de ser corretamente classificado, dado que o paciente não possua tal condição.

$$SPE = \frac{TN}{TN + FP} \quad (3)$$

- **Área sob a curva ROC (AUC):** Expressa a capacidade do modelo de distinguir as classes.

Sendo as siglas TP, TN, FP, FN, verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo, respectivamente.

A título de exemplo, foram compilados na Tabela 1 alguns exemplos do estado da arte na classificação de imagens em AD ou NC, que utilizam as métricas supracitadas. Por mais que os dados utilizados sejam diferentes em cada modelo, pode-se obter uma estimativa do que é possível de ser alcançado.

Autores	Pacientes	AD vs. NC			
		ACC	SEN	SPE	AUC
Sigi [4]	204HC + 180AD	0.79	0.83	0.87	0.78
Suk [5]	101HC + 128sMCI + 76pMCI + 93AD	0.92	0.92	0.95	0.97
Lian [6]	429HC + 465sMCI + 205pMCI + 358AD	0.90	0.82	0.97	0.95
Mingxia [7]	229HC + 226sMCI + 167pMCI + 203AD	0.91	0.88	0.93	0.95

Tabela 1 – Resultados no dataset ADNI. Fonte: Adaptado de [2]

Diferentemente do que foi feito nos artigos da Tabela 1, propõe-se que o modelo classifique a imagem como uma de 5 classes possíveis: NC, AD, EMCI (*early* MCI), MCI, LMCI (*late* MCI). Como as métricas definidas são utilizadas para classificação binária, o modo de avaliação do modelo será adaptado de forma que as métricas sejam calculadas para classe, seguindo a lógica *One-vs-Rest*.

Optou-se por um classificador multiclasse pois, em contraste aos 4 classificadores binários necessários para classificar cada categoria em relação ao NC, ao utilizar apenas um modelo com 5 classes de saída, evita-se o problema de possivelmente enquadrar o paciente em mais de uma classe, gerando dúvida quanto à sua real condição.

2.4 Conjuntos de dados

2.4.1 Disponibilidade de dados

Atualmente, existe grande disponibilidade de imagens de ressonância magnética cerebral em repositórios públicos, como os conjuntos de dados OASIS, IBSR e MICCAI, além do conjunto de dados do Alzheimer's Disease Neuroimaging Initiative (ADNI). Optou-se por utilizar o repositório do ADNI devido à grande disponibilidade de dados e sua popularidade na comunidade científica [2].

O Alzheimer's Disease Neuroimaging Initiative é uma iniciativa multi-institucional que visa possibilitar o compartilhamento de dados médicos para o desenvolvimento de marcadores biológicos para detecção precoce e acompanhamento do desenvolvimento da doença de Alzheimer. Os estudos desenvolvidos por essa iniciativa se dividem em quatro etapas (ADNI-1, ADNI-GO, ADNI-2 e ADNI-3) com focos distintos, em cada etapa foram recrutados participantes da América do Norte que concordaram em realizar uma bateria de exames clínicos e de imagens, permitindo diagnosticar seu grau de demência [8].

2.4.2 Limpeza dos dados

Nesse projeto foram utilizadas as imagens de apenas uma das etapas do estudo, a fim de garantir a padronização da forma de aquisição das imagens MRI e do processo utilizado no diagnóstico dos pacientes. Dentre as etapas, foi escolhida a ADNI-2 por contar com maior número de pacientes e, consequentemente, de dados [9].

A seleção do conjunto de dados foi feita por meio da funcionalidade de busca avançada da *Image & Data Archive* (IDA) da *University of South California*, que hospeda a base de dados da ADNI, permitindo a obtenção de um conjunto de dados limpo. Utilizando tal funcionalidade, restringiu-se a busca a imagens 2D de MRI cerebral, ponderadas em T1, de pacientes com mais de 50 anos, diagnosticados como CN (cognitive normal), EMCI (early stage mild cognitive disease), MCI (mild cognitive disease), LMCI (late stage mild cognitive disease) ou AD (Alzheimer disease). Os filtros de busca utilizados foram reproduzido na Figura 1 a fim de garantir a reprodutibilidade dos resultados do projeto.

2.4.3 Distribuição dos dados

O conjunto de dados utilizado contém imagens de MRI cerebral de 960 pacientes, obtidas no período de 2011 a 2017, totalizando 19520 imagens. Nas Figuras 2 e 3 são apresentadas as distribuições de idade e grau de demência do paciente em cada imagem. Vale ressaltar que cada paciente foi acompanhado por um longo período de tempo.

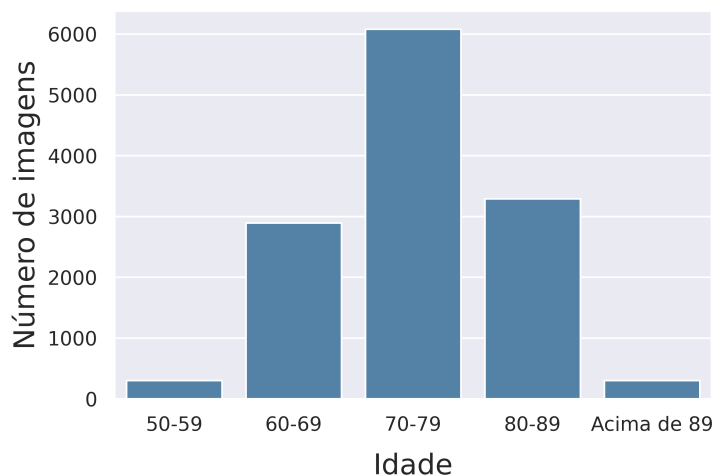


Figura 2 – Distribuição da idade dos pacientes em cada imagem de MRI.

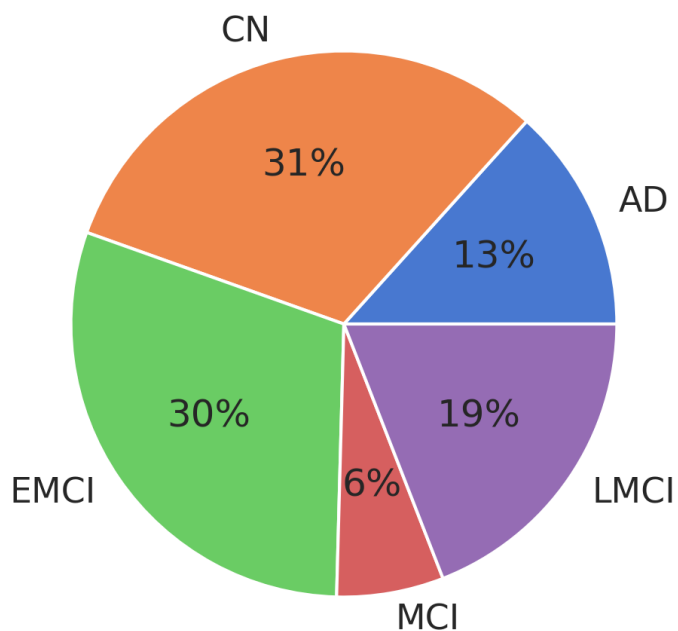


Figura 3 – Distribuição do grau de demência dos pacientes em cada imagem de MRI.

2.4.4 Pré-Processamento

Conforme foi mencionado, as imagens de MRI exigem um pré-processamento específico. Por exemplo, a figura 4 demonstra as etapas realizadas em [2].

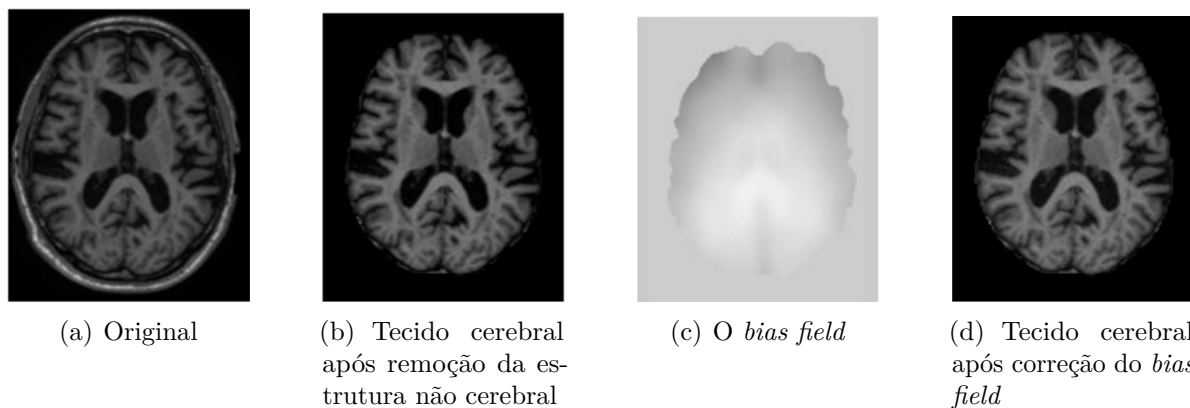


Figura 4 – Etapas do pré-processamento de um MRI. Retirado de [2]

A primeira etapa (Figuras 4(a) e 4(b)) consiste em remover partes da cabeça, olhos, crânio, dentre outras estruturas não-cerebrais que podem aparecer no MRI, a qual pode ser realizada por algoritmos de *skull stripping* [11].

Após isso, é aplicado o método *bias field correction* (Figuras 4(c) e 4(d)), em que é realizada uma correção no contraste da imagem para compensar a não homogeneidade do campo magnético [12]. Essa não homogeneidade promove, de forma geral, uma diferença, na imagem, na intensidade para um mesmo tipo de tecido, que deveria ser constante.

Também em [2], é abordado o fato de que o MRI pode gerar um ruído de alta frequência que segue uma distribuição *Rician* [12]. Objetivando diminuir a intensidade desse ruído no MRI, poderá se aplicar um algoritmo de redução de ruído.

Por último, é feita uma reorientação da imagem (*image registration*), permitindo que todas as imagens estejam no mesmo sistema de coordenadas.

Um processo de pré-processamento similar a esse é feito também em [3], cujo *dataset* é composto por imagens tridimensionais, de modo que também realiza uma normalização da imagem para o espaço padrão definido pelo *Montreal Neurological Institute* (MNI).

Neste projeto em específico, pretende-se como última etapa cortar as imagens de forma a diminuir o fundo, e depois, se necessário, uniformizar a proporção das imagens por preenchimento (*padding*). Ao final, elas serão redimensionadas para um mesmo tamanho.

Devido ao desbalanceamento das classes observado na Figura 3, realizar-se-á o *data augmentation* das classes com menos elementos. Em vista da anatomia do cérebro ser fundamental para a análise, métodos que manipulam o formato dos objetos na imagem (rotação, *zooming*, cortes, inversão, translação) são prejudiciais. Diante disso, serão

utilizados métodos de transformação à nível de *pixel*, como aplicações de ruído Gaussiano [13].

Por fim, visto que um dos objetivos do projeto é a aplicação de métodos de aprendizado de máquina apresentados em aula, deverão ser utilizados descritores para extrair *features* das imagens de forma que elas possam ser classificadas por esses métodos.

3 Parte 2

3.1 Objetivos

O objetivo da segunda parte do projeto é aplicar alguns dos métodos estudados na disciplina ao problema de classificação proposto, descrevendo os resultados obtidos até o momento.

3.2 Pré-Processamento

Conforme identificado na primeira parte do projeto, o pré-processamento de imagens de MRI exige a aplicação de técnicas específicas, elencadas na Seção 2.4.4. Dentre estas foram aplicadas as técnicas de:

- *skull stripping*;
- e *bias field correction*.

Em seguida, visto que trabalhar-se-á com modelos de aprendizado de máquina que exigem dados de entrada estruturados, realizou-se a extração de features utilizando uma rede neural convolucional (CNN) pré-treinada no conjunto de dados ImageNet.

Finalmente, frente ao desbalanceamento observado das classes, que pode ser constatado a partir da Figura 7, foi necessário considerar como tratar o conjunto de dados a fim de obter uma melhor distribuição das classes.

3.2.1 Skull stripping

A fim de remover as estruturas não-cerebrais das imagens de MRI foi utilizada a metodologia desenvolvida por Duarte et al.[14], em que as imagens são segmentadas a fim de extrair o cérebro. Para isso, foi aplicado o código disponibilizado pelos autores no *GitHub*[15].

Vale ressaltar que, no processo de segmentação e extração do cérebro, são realizados outros procedimentos de pré-processamento relevantes:

- remoção de ruído do fundo;
- ajuste de contraste, visto que as imagens de MRI usualmente apresentam baixo contraste, o que pode prejudicar a análise de detalhes.

Finalmente, a extração do cérebro é realizada por meio da detecção das componentes conexas da imagem. Na Figura 5 é possível visualizar o resultado da aplicação da metodologia a uma das imagens do conjunto de dados.

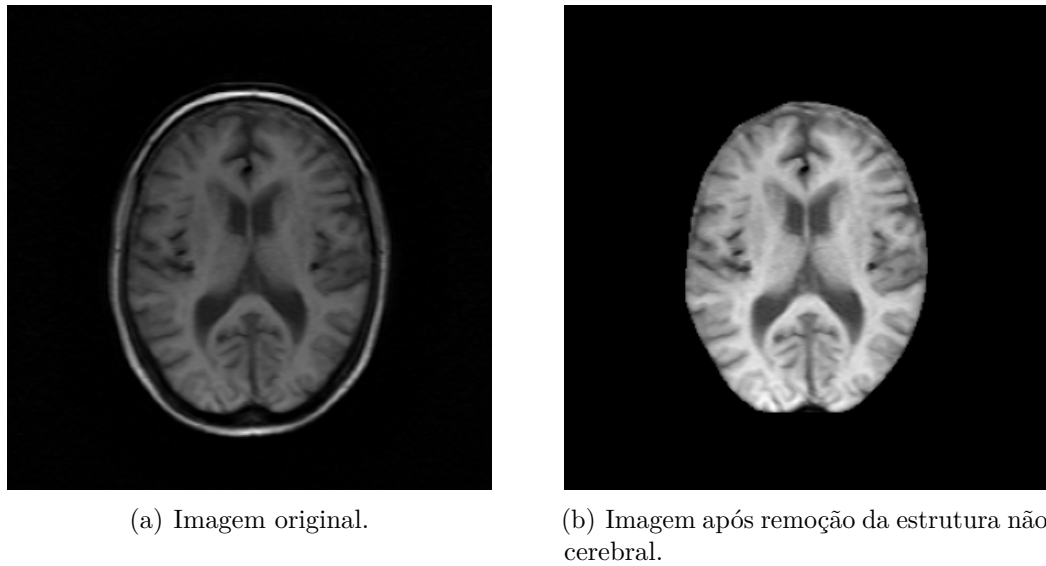


Figura 5 – Aplicação de *skull stripping* em uma imagem de MRI do conjunto de dados.

3.2.2 *Bias field correction*

Uma vez realizado o *skull stripping* aplicamos o algoritmo N4ITK de *bias field correction*[16], em que a presença de componentes de baixas frequências não uniformes, devida à não homogeneidade do campo magnético, é compensada. O método foi aplicado utilizando a implementação da biblioteca Insight Toolkit[17].

Na Figura 6 é possível visualizar o *bias field* estimado para uma imagem de MRI do conjunto de dados e a imagem resultante após a correção.

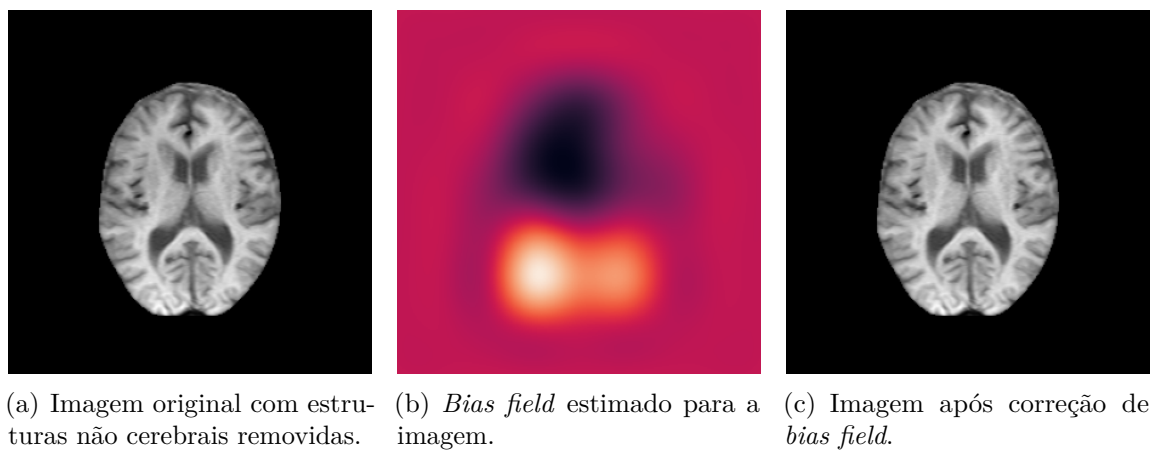


Figura 6 – Aplicação de *bias field correction* a uma imagem de MRI do conjunto de dados.

3.2.3 Extração de features

Visto que objetivamos utilizar modelos de classificação que recebem como entrada dados estruturados, sendo que o conjunto de dados proposto é composto por imagens, foi necessário realizar a extração de *features* das imagens. Para isso, utilizamos uma CNN pré-treinada com o conjunto de dados ImageNet [18], realizando o *forward propagation* das imagens na rede e parando em uma camada pré-determinada: as saídas da rede para essa camada foram utilizadas como *features*.

Em particular, a biblioteca Keras fornece diversas redes pré-treinadas na base ImageNet [19], dentre os quais foi escolhida a rede MobileNetV2, de 3.5M parâmetros e profundidade 105. A escolha foi feita pela rede apresentar acurácia competitiva na base ImageNet (Top-1 Accuracy de 71.3% e Top-5 Accuracy de 90.1%) ao mesmo tempo em que o tempo de inferência da rede é relativamente pequeno (25.9 ms por passo de inferência utilizando uma CPU e 3.8 ms em uma GPU), tornando viável seu uso.

Conforme apresentado no artigo [20] a arquitetura da rede MobileNetV2 é dada pela Tabela 2.

Entrada	Operador	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Tabela 2 – Arquitetura da rede MobileNetV2. Cada linha da tabela representa uma ou mais camadas idênticas, repetidas n vezes. A primeira camada de cada uma dessas sequências de camadas apresenta *stride* s e todas as demais utilizam *stride* 1. As camadas de convolução utilizam *kernels* 3×3 . Já o parâmetro t diz respeito ao bloco *bottleneck*. O número final de canais k pode ser ajustado. Extraído de [20].

Diante da arquitetura apresentada, optou-se por realizar o *forward propagation* até a camada *avgpool 7x7*, obtendo um vetor de *features* de dimensão 1280 para cada imagem.

3.2.4 Balanceamento

A partir dos dados escolhidos, tem-se a distribuição apresentada na Figura 7.

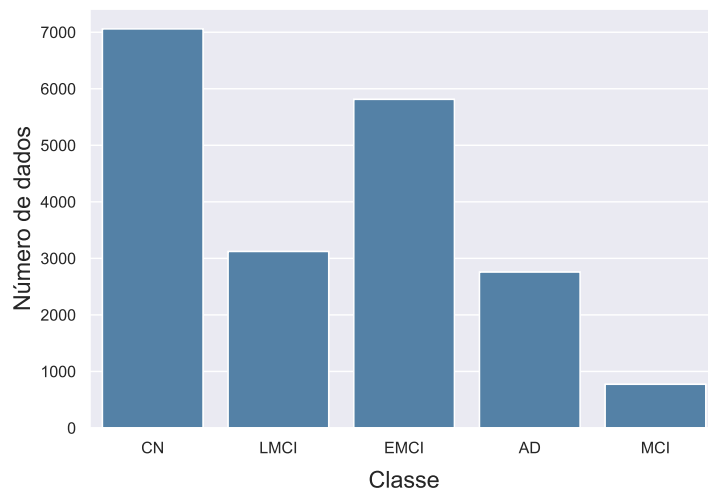


Figura 7 – Distribuição original dos dados

Como pode-se observar, há um grande desbalanceamento entre MCI e as demais classes. Pela complexidade presente nesse tipo de imagem, os métodos de *data augmentation* que podem ser aplicados são restritos, conforme discutido na Seção 2.4.4. Diante disso, julgou-se mais adequado remover a classe MCI, visto que já existem as classes *early* MCI e *late* MCI, assim não haveria grande prejuízo ao suprimir essa classe.

Dessa forma, tem-se a distribuição de dados dada pela Figura 8.

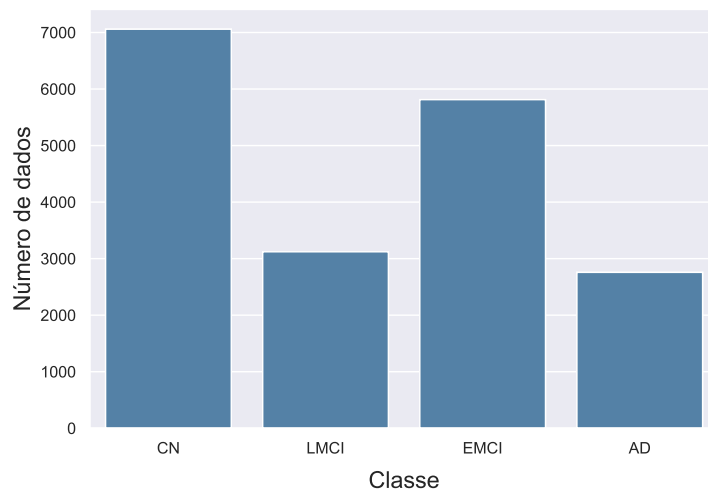


Figura 8 – Distribuição utilizada dos dados.

Visto que ainda assim existe um desbalanceamento marcante entre as classes, durante o treinamento dos modelos *Support Vector Machine* (SVM) e *Multi-Layer Perceptron* (MLP), atribuiu-se pesos às classes na função de perda, utilizando a função `compute_class_weight` da biblioteca *scikit-learn*[21], a fim de compensar o desbalanceamento.

3.3 Métodos utilizados

Para a realização do aprendizado, foram elencados três diferentes métodos expostos em aula: *K-Nearest Neighbours* (KNN), SVM e MLP.

Dentre os métodos vistos na disciplina, escolheu-se o KNN por ser um algoritmo de *instance-based learning*, sendo simples porém com boa performance. Já o SVM foi escolhido devido ao seu embasamento matemático, além de apresentar flexibilidade por meio da escolha do *kernel*. Por fim, escolheu-se o MLP devido à popularidade das redes neurais e pelo fato de preceder as técnicas de *deep learning*, que poderão ser utilizadas na terceira parte desse trabalho.

Para todos os modelos, separou-se 90% dos dados para treinamento e os 10% restantes para teste. Nos dois primeiros métodos (KNN e SVM), foi aplicado o método de *cross validation* com separação dos dados de treinamento em cinco partes, a fim de testar diferentes valores para os parâmetros dos modelos.

3.3.1 KNN

A fim de aplicar o algoritmo KNN, os dados foram normalizados utilizando a padronização *Z-score* (os atributos são centralizados em zero, com desvio padrão unitário). Com o intuito de definir o melhor modelo, foram testados diferentes números de vizinhos k , utilizando o método de *cross validation*. Os resultados obtidos constam na Tabela 3.

k	Média	Desvio Padrão
1	0.5679	0.0045
3	0.5262	0.0034
5	0.5208	0.0074
7	0.5132	0.0040
11	0.4984	0.0085

Tabela 3 – Acurácia do algoritmo KNN para diferentes valores de k .

A partir da tabela, verifica-se que o melhor resultado foi obtido para $k = 1$. Treinando o modelo com todo o conjunto de dados de treinamento, para $k = 1$, obteve-se uma acurácia final de 58,24% nos dados de teste.

Analisando o desempenho desse modelo para cada classe, por meio da matriz de confusão da Figura 9, observa-se que o melhor desempenho foi obtido para as classes majoritárias CN e EMCI. Tal comportamento corresponde ao esperado, visto que para esse modelo não foram aplicados pesos às classes, por tal funcionalidade não ser provida pela biblioteca **sklearn**. É possível também determinar a acurácia, sensibilidade e especificidade do modelo para cada classe, conforme consta na Tabela 4. Os resultados obtidos serão discutidos na Seção 3.4.

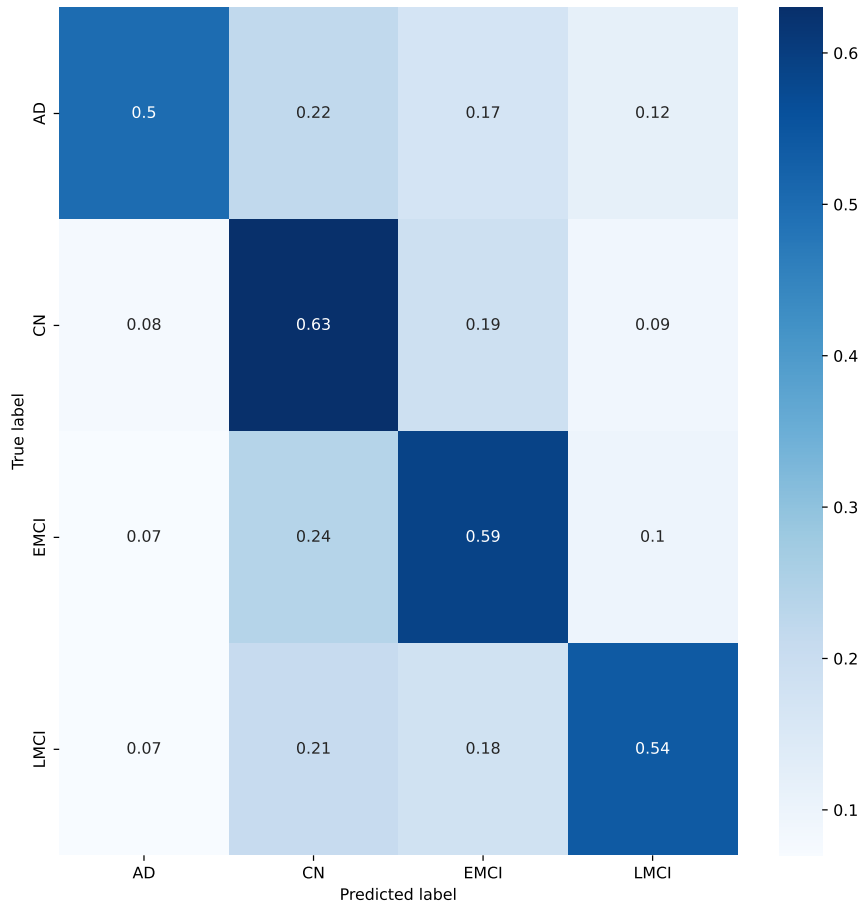


Figura 9 – Matriz de Confusão para o algoritmo KNN ($k = 1$).

Classe	Acurácia	Sensibilidade	Especificidade
AD	0.8624	0.4981	0.9234
CN	0.7243	0.6331	0.7756
EMCI	0.7419	0.5862	0.8167
LMCI	0.8363	0.5389	0.8977

Tabela 4 – Métricas de avaliação de desempenho para o algoritmo KNN ($k = 1$).

3.3.2 SVM

A fim de aplicar o algoritmo SVM, os dados também foram normalizados utilizando a padronização *Z-score*. Para esse algoritmo, foram testados três *kernels*: o linear, o polinomial e o *radial basis function* (RBF).

Para o *kernel* polinomial, testaram-se polinômios de graus 3 e 5. E, para o RBF, primeiramente foram realizados testes variando o coeficiente γ entre 0.1, 1 e 10, e, para o melhor valor de γ , variou-se o parâmetro C de regularização entre 10 e 100.

Para o *kernel* linear, a acurácia foi de $0,3976 \pm 0,0101$. Já para o *kernel* polinomial, foram obtidas as acurácias da Tabela 5. Finalmente, para o *kernel* RBF, as acurácias

são apresentadas na Tabela 6. Visto que o melhor gamma foi 0.1, variou-se o valor de C conforme indicado anteriormente.

Grau	Média	Desvio Padrão
3	0.4832	0.0079
5	0.5187	0.0135

Tabela 5 – Acurácia do SVM com *kernel* polinomial sob diferentes graus.

γ	C	Média	Desvio Padrão
0.1	1	0.4999	0.0079
0.1	10	0.5304	0.0115
0.1	100	0.5317	0.0180
1	1	0.2649	0.0108
10	1	0.2541	0.0086

Tabela 6 – Acurácia do SVM com *kernel* RBF sob diferentes parâmetros.

Dessa forma, vê-se que o modelo que teve um melhor desempenho foi o modelo com *kernel* RBF e com parâmetros $\gamma = 0.1$ e $C = 100$. Treinando-o com todo o conjunto de dados de treino e avaliando seu desempenho no conjunto de teste, foi obtida uma acurácia de 55.19%. Esse modelo foi avaliado utilizando as mesmas métricas aplicadas ao KNN, os resultados constam na Figura 10 e na Tabela 7.

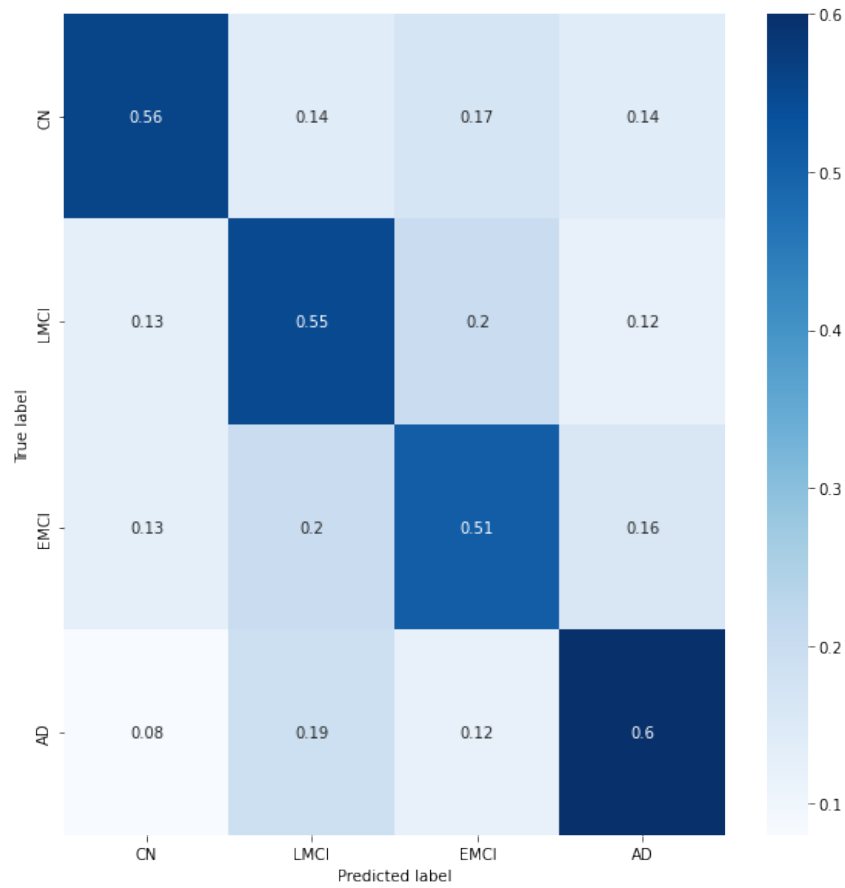


Figura 10 – Matriz de confusão para o algoritmo SVM com *kernel* RBF ($\gamma = 0.1$, $C = 100$).

Classe	Acurácia	Sensitividade	Especificidade
CN	0.808673	0.558824	0.88385
LMCI	0.752551	0.552189	0.82025
EMCI	0.742347	0.505988	0.836105
AD	0.80017	0.600733	0.860465

Tabela 7 – Métricas de avaliação de desempenho para o algoritmo SVM com *kernel* RBF ($\gamma = 0.1$, $C = 100$).

3.3.3 MLP

No caso do *multilayer perceptron*, primeiramente, foi definida a utilização do otimizador Adam[22] e da função de perda *sparse cross entropy*[23]. Em seguida, foram testadas diversas arquiteturas com diferentes configurações de hiper-parâmetros.

Durante o treinamento dos diferentes modelos, percebeu-se que a rede era suscetível a *overfitting*, diante disso, foi reduzido o tamanho do modelo e foram inseridas camadas de *dropout*, além de ser implementado o *early stopping* com 3 épocas de paciência. Conside-

rando tais fatores, foi desenvolvido o modelo final descrito na Figura 11, com *learning rate* de 10^{-4} e *batch size* 32. A acurácia final no conjunto de dados de teste foi de 46.72%.

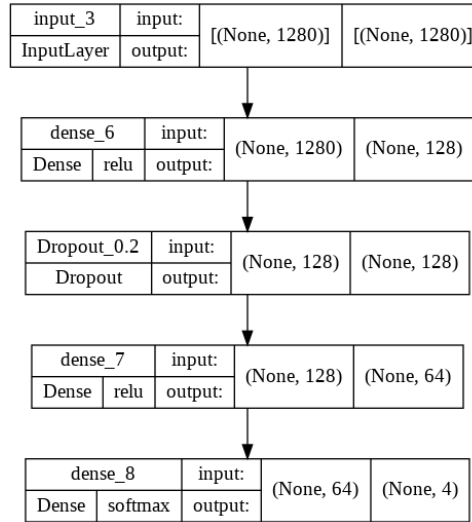


Figura 11 – Arquitetura do MLP.

Como a saída da rede consiste na probabilidade do dado pertencer a cada classe, foi possível avaliar a rede utilizando a curva de ROC (Figura 12), além das métricas utilizadas anteriormente, exibidas na Figura 13 e na Tabela 8,

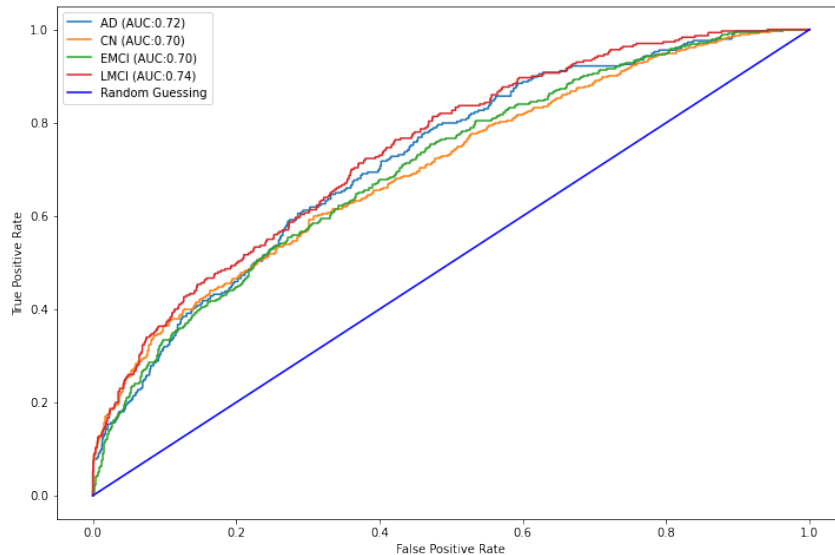


Figura 12 – Curva ROC para o MLP.

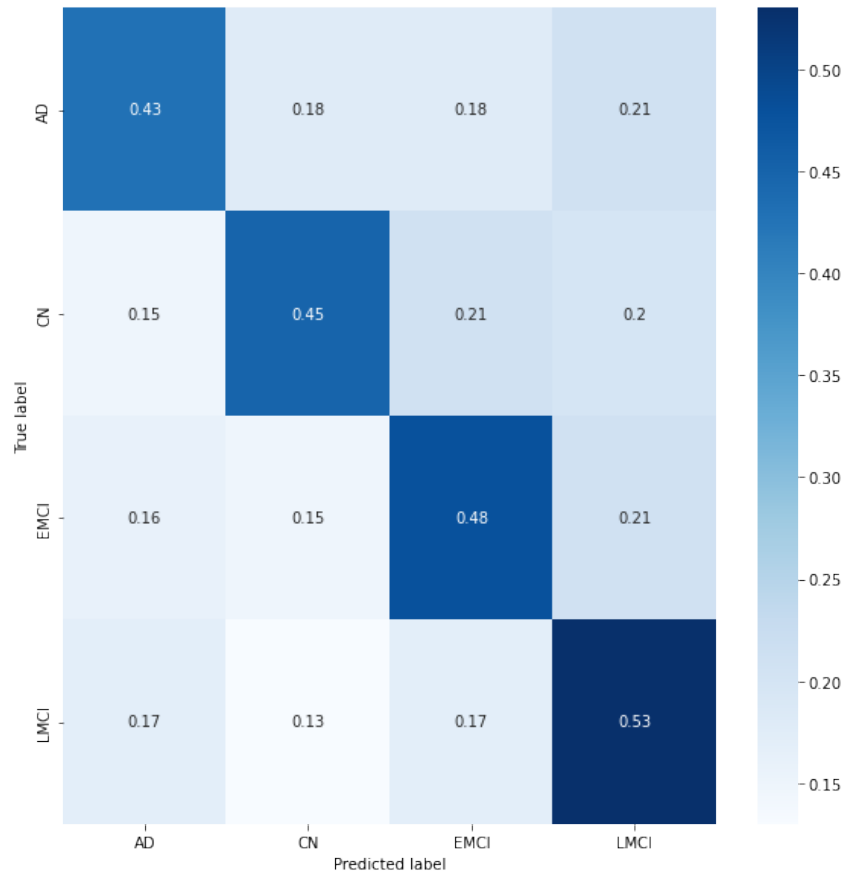


Figura 13 – Matriz de confusão para o MLP.

Classe	Acurácia	Sensibilidade	Especificidade
AD	0.7797	0.4320	0.8444
CN	0.6939	0.4477	0.8463
EMCI	0.7072	0.4770	0.8063
LMCI	0.7536	0.5300	0.7962

Tabela 8 – Métricas de avaliação de desempenho para o MLP.

3.4 Conclusão

Em resumo, partindo dos métodos selecionados (KNN, SVM e MLP), foram testados diferentes modelos variando parâmetros dos algoritmos ou a arquitetura da rede, de modo que, ao final, foram considerados três modelos:

- algoritmo KNN com $k = 1$;
- algoritmo SVM com *kernel* RBF e parâmetros $\gamma = 0.1$ e $C = 100$;
- MLP com arquitetura correspondente ao diagrama da Figura 11, *learning rate* 10^{-4} e *batch size* 32.

Dentre esse modelos, obteve-se como maior acurácia de teste o valor de 58,24%, aplicando o algoritmo KNN para $k = 1$.

Avaliando as matrizes de confusão (Figuras 9, 10 e 13), foi possível supor que a atribuição de pesos às classes teve um impacto positivo, visto que, para os modelos em que foi possível realizar essa compensação (SVM e MLP), observou-se uma maior uniformidade de desempenho entre as diferentes classes.

Ademais, diante dos resultados obtidos, concluí-se que nenhum dos modelos foi satisfatório para a classificação de imagens de MRI. Um dos fatores para a baixa acurácia é o uso, para a extração de *features*, de uma rede neural pré treinada em uma base dados de escopo muito distante das imagens de interesse, de modo que a técnica de transferência de aprendizado não será tão efetiva.

Outrossim, notou-se que, dentre as imagens de MRI de corte axial utilizadas, existem imagens referentes à diferentes planos de *scan* do cérebro, por exemplo, existem imagens correspondente à altura do maxilar ou ao topo do crânio, enquanto que outras imagens apresentam porções mais relevantes do cérebro. Possivelmente, tal variedade de planos afeta o desempenho dos algoritmos existentes, entretanto, a base de dados ADNI não possibilita a identificação destes planos, exceto por inspeção visual.

Destarte, propõe-se para a próxima etapa, o *fine tuning* de modelos convolucionais para a extração de *features* mais relevantes, ou o treinamento completo de uma CNN, também será considerada a substituição da base de dados atual por uma que permita a identificação dos planos de *scan* axial.¹

3.5 Código fonte

Os programas desenvolvidos para aplicação das etapas de pré-processamento e treinamento dos modelos constam no seguinte repositório: <https://github.com/AliceDeLorenci/alzheimer-disease-detection>.

¹ Identificou-se que a base de dados OASIS [24] especifica o plano axial para cada imagem de MRI, entretanto, o pedido de acesso do grupo aos dados não foi aceito.

4 Parte 3

4.1 Objetivos

Na segunda parte deste projeto, foi utilizada uma rede neural do tipo *MobileNetV2*, pré-treinada no conjunto de dados ImageNet, para extrair *features* das imagens de MRI cerebral, obtendo assim um conjunto de dados estruturado que pôde ser utilizado como entrada para demais algoritmos de classificação (KNN, SVM e MLP).

Entretanto, visto que o desempenho obtido para esses algoritmos foi insatisfatório, propôs-se, na terceira parte deste trabalho, o treinamento de uma *MobileNetV2* diretamente sobre as imagens de MRI cerebral, permitindo uma comparação dos resultados das partes 2 e 3 do trabalho. Com esse intuito, foi estudada a arquitetura *MobileNetV2* a partir dos artigos [25, 20], os quais serão sumarizados na Seção 4.2. Em seguida, foi possível treinar a rede com o conjunto de imagens pré-processadas de MRI, sendo que os resultados obtidos serão discutidos na Seção 4.3.

4.2 MobileNetV2

O desenvolvimento e ajuste de modelos de redes neurais profundas visando obter melhores performances, resulta em arquiteturas altamente complexas, demandando cada vez mais recursos computacionais. Por outro lado, a arquitetura *MobileNet*, proposta em sua primeira versão em 2017 e revista em 2019, sob o nome *MobileNetV2*, propõe o uso de um modelo o mais simples possível, mantendo os mesmos níveis de acurácia. Isso a torna adequada sobretudo a aplicações *mobile* e embarcadas, pois reduz significativamente o número de operações realizadas e as exigências de memória [25, 20].

Tal simplicidade aliada a um alto desempenho é possível devido ao uso de estruturas residuais invertidas (*inverted residual blocks*), com conexões de atalho (*shortcut connections*) entre camadas *bottleneck* lineares, de menor dimensionalidade. Os *inverted residual blocks* têm por entrada uma representação comprimida, de baixa dimensionalidade, que é expandida para mais altas dimensões e filtrada utilizando *depthwise separable convolutions*, mais leves computacionalmente do que camadas convolucionais padrão. A representação obtida é então projetada linearmente para um espaço de dimensionalidade mais baixa [20].

Na Seção 4.2.1 serão abordados em maiores detalhes os blocos e operações fundamentais que diferenciam a *MobileNetV2* dos demais *designs* de redes neurais. Em seguida, na Seção 4.2.2 será possível apresentar sua arquitetura.

4.2.1 Aspectos fundamentais

As *depthwise separable convolutions* são um aspecto fundamental no *design* de redes neurais eficientes, trata-se da decomposição de uma convolução padrão em uma *depthwise*

convolution e uma *pointwise convolution*. A *depthwise convolution* aplica um único filtro à cada canal de entrada, realizando a etapa de filtragem, já a *pointwise convolution* combina os canais de saída da *depthwise convolution* [25].

A Figura 14 ilustra a decomposição dos filtros de uma convolução padrão $k \times k \times d_i \times d_j$ (sendo d_i o número de canais de entrada e d_j o número de canais de saída) em uma *depthwise convolution* e uma *pointwise convolution*. Já a Figura 15 ilustra, comparativamente, uma convolução padrão e a organização das camadas para a *depthwise separable convolution* correspondente.

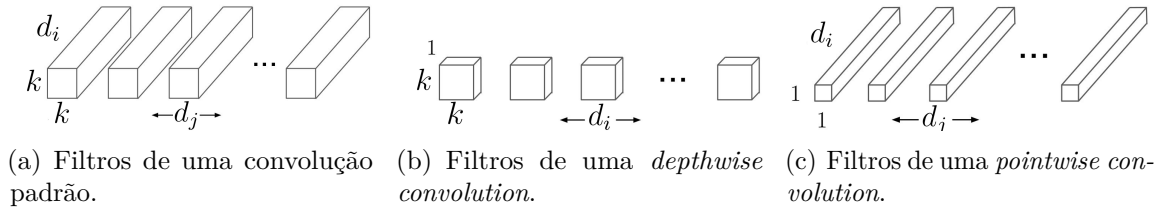


Figura 14 – Decomposição de uma convolução padrão em uma *depthwise convolution* e uma *pointwise convolution* no contexto de *depthwise separable convolutions*. Fonte: Adaptado de [25].

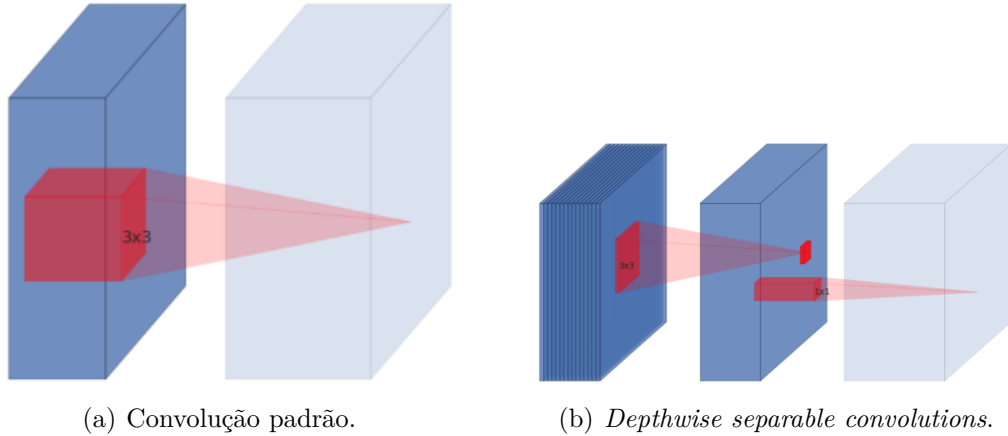


Figura 15 – Decomposição de uma convolução padrão em uma *depthwise convolution* e uma *pointwise convolution* no contexto de *depthwise separable convolutions*. Fonte: [20].

Empiricamente, a *depthwise separable convolution* apresenta performance comparável à uma convolução padrão, todavia, tem custo computacional significativamente inferior. Assumindo um tensor de entrada $h_i \times w_i \times d_i$, um filtro convolucional $k \times k \times d_i \times d_j$, *stride* $s = 1$ e *padding*, o custo computacional de uma convolução padrão é:

$$k \cdot k \cdot h_i \cdot w_i \cdot d_i \cdot d_j$$

Enquanto que a *depthwise* e *pointwise convolutions* têm custo, respectivamente:

$$k \cdot k \cdot d_i \cdot h_i \cdot w_i$$

$$d_i \cdot d_j \cdot h_i \cdot w_i$$

De modo que a *depthwise separable convolutions* têm custo total:

$$h_i \cdot w_i \cdot d_i (k^2 + d_j)$$

Desse modo, obtêm-se uma redução do custo computacional por um fator de:

$$\frac{k^2 \cdot d_j}{k^2 + d_j}$$

Outro componente central à eficiência e ao desempenho da arquitetura *MobileNetV2* são os *inverted residuals blocks* com *linear bottlenecks*. No que segue, os tensores de entrada das camadas serão interpretados como *containers* de $h_i \times w_i$ pixels de d_i dimensões (canais).

Teoriza-se que a informação codificada nos tensores de entrada pode ser embutida em sub-espacos de menor dimensionalidade, à primeira vista, essa compressão poderia ser obtida empregando camadas convolucionais que mapeassem a entrada de dimensão d_i em uma saída de dimensão $d_j < d_i$ (camada *bottleneck*), sendo esta a técnica adotada na *MobileNetV1*. Entretanto, camadas convolucionais introduzem não linearidades por coordenada através da função de ativação, por exemplo, a função ReLU causa, potencialmente, o colapso de alguns canais, levando à perda de informação. Assim, para contornar tal problema, assumindo que a informação codificada nos tensores de entrada possa, de fato, ser codificada em espacos de menor dimensionalidade, são utilizadas camadas *bottleneck* lineares na *MobileNetV2*.

A *depthwise separable convolution* é então modificada para atuar como um *bottleneck*: a *pointwise convolution* é utilizada para reduzir a dimensionalidade do tensor de entrada sem introduzir não-linearidades, conforme ilustrado na Figura 16.

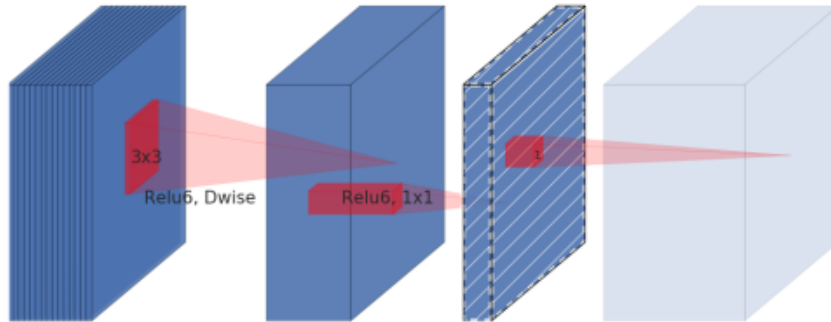


Figura 16 – Ilustração de uma *depthwise separable convolution* com *bottleneck* linear, o preenchimento hachurado indica camadas que não contém não-linearidades. É aplicada uma *depthwise convolution* seguida de uma *pointwise convolution* que não introduz não-linearidades. A última camada, de cor mais clara, indica o início de um próximo bloco. Fonte: [20].

Assim, na arquitetura *MobileNetV2* são utilizados *bottleneck blocks* que consistem em uma entrada, seguida de camadas *bottleneck* e de uma camada de expansão. A partir dos blocos *bottleneck*, são estabelecidas *shortcut connections* que seguem uma lógica contrária a blocos residuais padrão, sendo denominados blocos residuais invertidos (*inverted residual blocks*). Usualmente, em um bloco residual, temos uma entrada, seguida por *bottlenecks*, seguidos por expansões, sendo um *shortcut* é estabelecido entre entrada e saída, por outro lado, na *MobileNetV2*, os *shortcuts* são estabelecidos diretamente entre os *bottlenecks*. Isto é, blocos residuais estabelecem conexões entre camadas com grande número de canais, enquanto que blocos residuais invertidos conectam *bottlenecks* com menor número de canais. Vale ressaltar que a motivação para estabelecer *shortcuts* consiste em melhorar a propagação dos gradientes através de múltiplas camadas. A Figura 17 ilustra a diferença entre estes dois blocos.

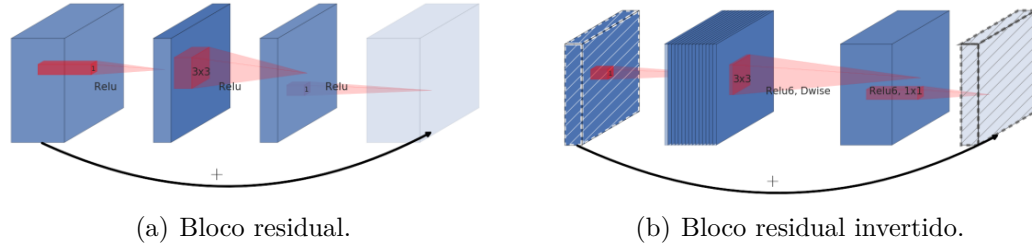


Figura 17 – Comparação entre um bloco residual e um bloco residual invertido. O preenchimento hachurado indica camadas que não contém não-linearidades. Fonte: [20].

Essa modificação, com relação a blocos residuais usuais, inspira-se no fato dos *bottlenecks* codificarem todas as informações presentes nos tensores de entrada, tendo em vista as assunções realizadas, enquanto que as camadas de expansão são meramente um detalhe de implementação que decorre da transformação não-linear dos tensores, além de que a estrutura invertida otimiza o uso de memória.

A estrutura básica de um bloco residual invertido (ou *bottleneck residual block*), como o da Figura 17(b), encontra-se descrita na Tabela 9.

Entrada	Operador	Saída
$h \times w \times d'$	1x1 conv2d, ReLU6	$h \times w \times td'$
$h \times w \times td'$	3x3 dwse s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times td'$
$\frac{h}{s} \times \frac{w}{s} \times td'$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times d''$

Tabela 9 – Bloco residual invertido transformando d' em d'' canais, com passo s e fator de expansão t [20].

Ainda que o modelo base da *MobileNet* seja pequeno e tenha latência baixa, pode ser necessário que o modelo seja menor ainda e mais rápido. Em vista disso, introduz-se um

parâmetro $\alpha \in (0, 1]$ chamado de multiplicador de largura [25], cujo papel é reduzir a rede uniformemente em cada camada.

Dada uma camada e um valor α , a quantidade de canais de entrada d_i torna-se αd_i e a quantidade de canais de saída d_j se torna αd_j . Dessa forma, o custo computacional de uma *depthwise separable convolution* é dada por:

$$k \cdot k \cdot \alpha d_i \cdot h_i \cdot w_i + \alpha d_i \cdot \alpha d_j \cdot h_i \cdot w_i$$

Verifica-se que o multiplicador de largura reduz o custo computacional e o número de parâmetros de forma quadrática por um fator de aproximadamente $1/\alpha^2$.

Analogamente ao multiplicador de largura, tem-se o multiplicador de resolução $\rho \in (0, 1]$ para redução da representação [25]. Aplica-se esse fator na imagem de entrada, e as representações internas de cada camada são reduzidas pelo mesmo fator de forma subsequente.

Agora com a adição do multiplicador de resolução ρ , tem-se que o custo computacional de uma *depthwise separable convolution* é dada por:

$$k \cdot k \cdot \alpha d_i \cdot \rho h_i \cdot \rho w_i + \alpha d_i \cdot \alpha d_j \cdot \rho h_i \cdot \rho w_i$$

reduzindo o custo computacional por aproximadamente $1/\rho^2$, comparado ao custo com a aplicação somente do multiplicador de largura.

4.2.2 Arquitetura

A arquitetura da MobileNetV2 apresenta uma camada inicial de convolução padrão com 32 filtros, seguida por 17 blocos *residual bottleneck* (ilustrado na Figura 18), conforme descrito pela Tabela 10.

Utiliza-se a função de ativação ReLU6, ilustrada na Figura 19, para introduzir não linearidades por causa de sua robustez para computação de baixa precisão, eficiente para aplicativos móveis [25]. Ademais, utiliza-se *dropout* e *batch normalization* no treinamento da rede.

Com exceção da primeira camada, a taxa de expansão (razão entre o número de canais internos ao *residual bottleneck block*, após expansão, e o número de canais da camada *bottleneck* de entrada) é constante ao longo da rede. Conforme experimentos realizados [20], obtém-se curvas de performance quase idênticas para taxas de expansão entre 5 e 10. Outrossim, percebe-se que redes menores possuem performance melhor com taxas de expansão mais baixas, enquanto redes maiores se beneficiam de maiores taxas de expansão.

Para melhorar a performance em modelos pequenos, nas situações em que os multiplicadores α e ρ forem diferentes de 1, aplica-se o multiplicador de largura para todas as camadas exceto a última camada convolucional [25].

Entrada	Operador	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$14^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Tabela 10 – MobileNetV2: Cada linha descreve uma sequência de pelo menos uma camada idêntica repetida n vezes. Todas as camadas na mesma sequência possuem a mesma quantidade c de canais de saída. A primeira camada de cada sequência tem um valor de *stride* s e todas as outras têm *stride* 1. As convoluções espaciais utilizam *kernels* 3×3 . O fator de expansão t é sempre aplicado para o tamanho da entrada como descrito pela tabela 9 [20]

4.3 Experimentos

Novamente, foi utilizada a implementação da *MobileNetV2* disponibilizada pela biblioteca Keras[19]. Os pesos da rede foram inicializados com os valores do modelo pré-treinado no *ImageNet*. No treinamento da rede, foi utilizado o otimizador Adam e *learning rate* 0.0001, sendo que a última camada da arquitetura padrão *MobileNetV2* foi substituída por uma camada densa de tamanho 4, correspondendo ao número de classes.

Foi utilizada uma GTX 1050 TI e, devido a restrições de memória, o modelo foi treinado com resolução das imagens de entrada 224×224 e 96×96 , combinadas, respectivamente, a *batch sizes* de 4 e 16.

Visando evitar o *overfitting*, foi adicionado ruído gaussiano como processo de *data augmentation*, de acordo com o artigo [13]; vale ressaltar que o método foi utilizado para aumentar a variabilidade dos dados e não para realizar balanceamento. Além disso, o modelo foi treinado por no máximo 15 épocas, utilizando *early stopping* com paciência 3.

A rede neural foi treinada sobre os dados pré-processados, conforme descrito na Seção 3.2, e compensando o desbalanceamento pelo uso de pesos, segundo a Seção 3.2.4. Os resultados obtidos constam na Tabela 11.

Observa-se que a performance da rede foi pior nos casos em que utilizou-se *data augmentation*, pode-se supor que isso se deve ao fato das características que determinam o grau de comprometimento cognitivo do paciente serem muito sutis e portanto sensíveis à adição de ruído.

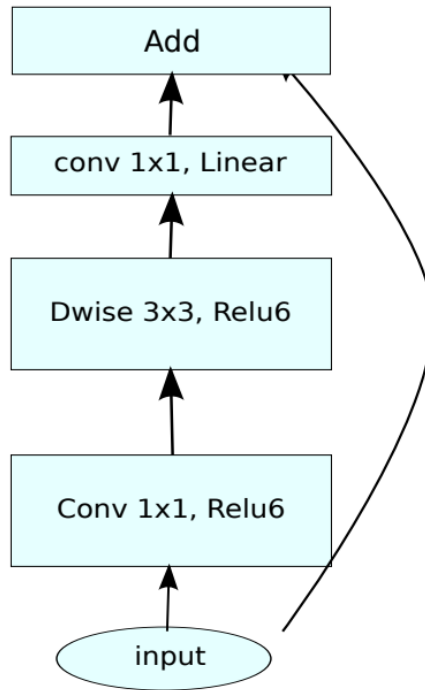


Figura 18 – Bloco residual da *MobileNetV2* [20]

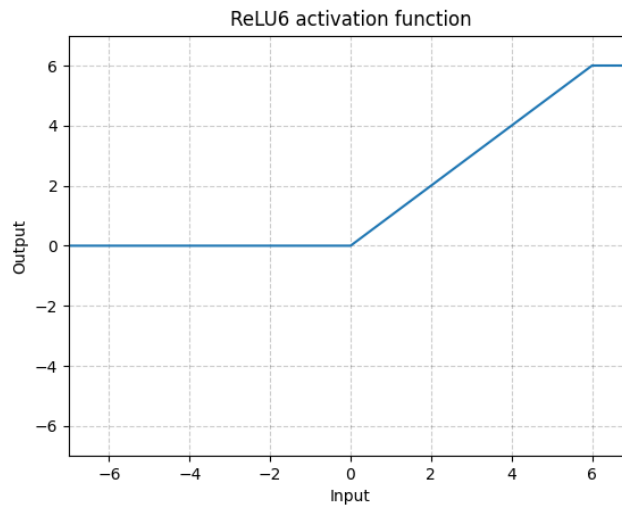


Figura 19 – Função de ativação ReLU6 [26].

Verifica-se que a melhor acurácia foi obtida para resolução 224×224 , com *batch size* 4 e sem *data augmentation*. Tal resultado reforça a hipótese anterior quanto à sutileza das características relevantes à classificação, visto que a redução da resolução levou à perda de informação. A melhor configuração foi avaliada utilizando as mesmas métricas aplicadas ao MLP na Seção 3.3.3: curva ROC (Figura 20), matriz de confusão (Figura 21), e acurácia, sensibilidade e especificidade por classe (Tabela 12).

Resolução	Batch size	<i>Data augmentation</i>	Número de épocas	Acurácia
224×224	4	Presente	7	0.3186
224×224	4	Ausente	7	0.4931
96×96	16	Presente	9	0.3959
96×96	16	Ausente	8	0.4002

Tabela 11 – Acurácia obtida para cada uma das redes treinadas e número de épocas de treinamento.

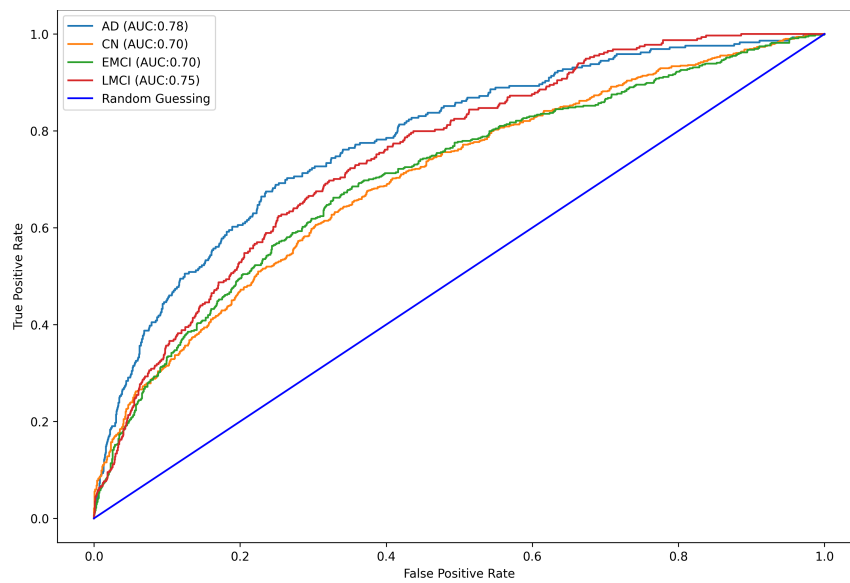


Figura 20 – Curva ROC.

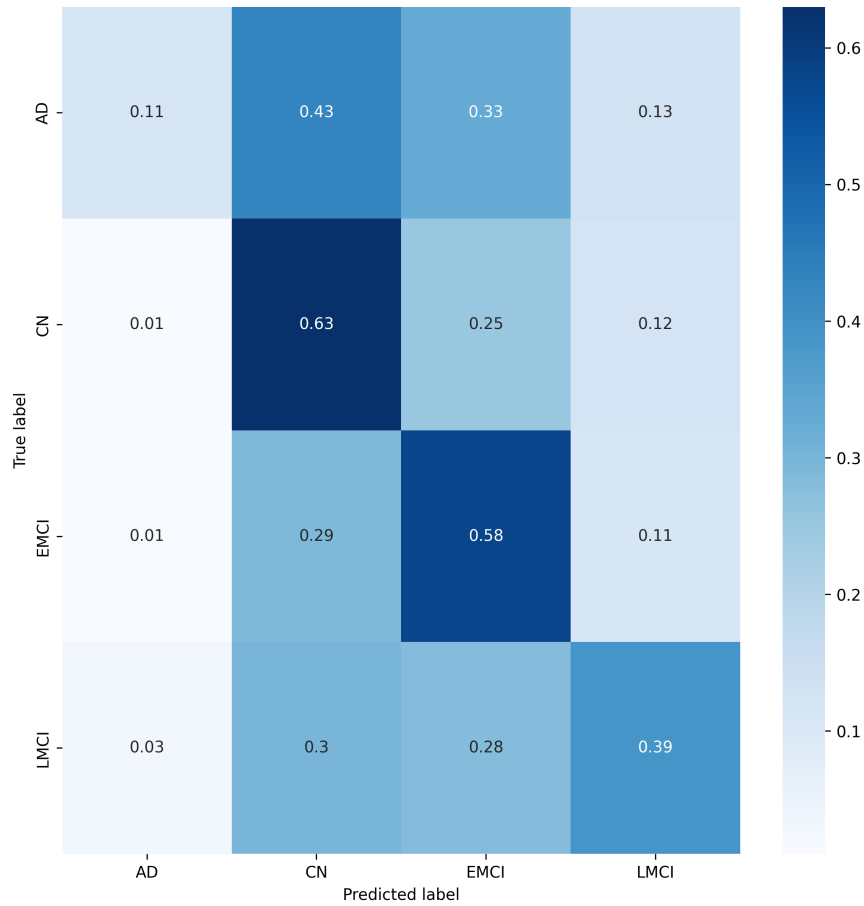


Figura 21 – Matriz de confusão.

Classe	Acurácia	Sensitividade	Especificidade
AD	0.8522	0.1073	0.9880
CN	0.6531	0.6253	0.6704
EMCI	0.6841	0.5823	0.7267
LMCI	0.7967	0.3885	0.8788

Tabela 12 – Métricas de avaliação de desempenho.

4.4 Conclusão

Em resumo, neste projeto foram aplicadas técnicas de aprendizado de máquina a fim de determinar o nível de comprometimento cognitivo de um paciente, indo desde cognitivamente normal à doença de Alzheimer, utilizando imagens de MRI cerebral. Para isso, foi utilizado o conjunto de dados do repositório ADNI, contando com 19520 imagens distribuídas entre as classes CN, EMCI, MCI, LMCI e AD.

De posse dos dados, foram aplicadas as técnicas de pré-processamento *skull stripping* e *bias field correction*. Em seguida, diante do desbalanceamento dos dados, e devido à

inviabilidade de aplicar métodos padrões de *data augmentation*, optou-se por excluir a classe MCI e utilizar pesos no treinamento dos algoritmos.

Feito isso, na parte 2 do projeto, foi realizada a extração de *features* com uma rede *MobileNetV2* pré-treinada no conjunto de dados *ImageNet*. Tais *features* foram utilizadas para treinamento de diferentes modelos KNN, SVM e MLP; a melhor performance obtida para cada algoritmo é apresentada na Tabela 13

Métodos	KNN	SVM	MLP
Acurácia	58,24%	55,19%	46,72%

Tabela 13 – Performance dos algoritmos treinados sobre as *features* extraídas do conjunto de imagens de MRI.

Por fim, na terceira parte do projeto, treinamos a rede *MobileNetV2* diretamente no conjunto de dados pré-processados, obtendo uma acurácia de 49.31%, inferior às acurácias obtidas para os algoritmos KNN e SVM. Supõe-se que o baixo desempenho se deve ao ruído presente no conjunto de dados, conforme discutido na Seção 3.4 (por ruído entende-se a presença de imagens correspondentes a planos axiais não significativos do cérebro). Ademais, devido às restrições do *hardware* disponível, utilizou-se um *batch size* pequeno, prejudicando a capacidade de aprendizado da rede.

4.5 Código fonte

Os programas desenvolvidos constam no seguinte repositório: <https://github.com/AliceDeLorenci/alzheimer-disease-detection>.

Referências

- [1] Emily C. Edmonds et al. “Early vs. Late MCI: Improved MCI Staging Using a Neuropsychological Approach”. Em: *Alzheimer’s & Dementia®: The Journal of the Alzheimer’s Association* 15 (5 2019), pp. 699–708.
- [2] Nagaraj Yamanakkanavar, Jae Young Choi e Bumshik Lee. “MRI Segmentation and Classification of Human Brain Using Deep Learning for Diagnosis of Alzheimer’s Disease: A Survey”. Em: *Sensors* 20.11 (2020). ISSN: 1424-8220. DOI: [10.3390/s20113243](https://doi.org/10.3390/s20113243). URL: <https://www.mdpi.com/1424-8220/20/11/3243>.
- [3] Christian Salvatore et al. “Magnetic resonance imaging biomarkers for the early diagnosis of Alzheimer’s disease: a machine learning approach”. Em: *Frontiers in Neuroscience* 9 (2015). ISSN: 1662-453X. DOI: [10.3389/fnins.2015.00307](https://doi.org/10.3389/fnins.2015.00307). URL: <https://www.frontiersin.org/article/10.3389/fnins.2015.00307>.
- [4] Siqi Liu et al. “Early diagnosis of Alzheimer’s disease with deep learning”. Em: *2014 IEEE 11th international symposium on biomedical imaging (ISBI)*. IEEE. 2014, pp. 1015–1018.
- [5] Heung-Il Suk et al. “Hierarchical feature representation and multimodal fusion with deep learning for AD/MCI diagnosis”. Em: *NeuroImage* 101 (2014), pp. 569–582.
- [6] Mingxia Liu et al. “Landmark-based deep multi-instance learning for brain disease diagnosis”. Em: *Medical image analysis* 43 (2018), pp. 157–168.
- [7] Bumshik Lee, Waqas Ellahi e Jae Young Choi. “Using deep CNN with data permutation scheme for classification of Alzheimer’s disease in structural magnetic resonance imaging (sMRI)”. Em: *IEICE TRANSACTIONS on Information and Systems* 102.7 (2019), pp. 1384–1395.
- [8] ADNI. *About ADNI*. URL: <https://adni.loni.usc.edu/about/> (acesso em 21/04/2022).
- [9] ADNI. *Study Design*. URL: <https://adni.loni.usc.edu/about/> (acesso em 21/04/2022).
- [10] IDA. *ADNI*. URL: <https://ida.loni.usc.edu/login.jsp?project=ADNI> (acesso em 21/04/2022).
- [11] PhDDirection. *Skull Stripping using Python*. URL: <https://www.phddirection.com/skull-stripping-using-python/>.

- [12] Uro Vovk, Franjo Pernus e Botjan Likar. “A Review of Methods for Correction of Intensity Inhomogeneity in MRI”. Em: *IEEE Transactions on Medical Imaging* 26.3 (2007), pp. 405–421. DOI: [10.1109/TMI.2006.891486](https://doi.org/10.1109/TMI.2006.891486).
- [13] Jakub Nalepa, Michal Marcinkiewicz e Michal Kawulok. “Data Augmentation for Brain-Tumor Segmentation: A Review”. Em: *Frontiers in Computational Neuroscience* 13 (2019). ISSN: 1662-5188. DOI: [10.3389/fncom.2019.00083](https://doi.org/10.3389/fncom.2019.00083). URL: <https://www.frontiersin.org/article/10.3389/fncom.2019.00083>.
- [14] Kaue T. N. Duarte et al. “Brain Extraction in Multiple T1-Weighted Magnetic Resonance Imaging slices using Digital Image Processing Techniques”. Em: *IEEE Latin America Transactions* 20 (5 2022), pp. 831–838.
- [15] Kaue T. N. Duarte et al. *Brain Extraction in Multiple T1-Weighted Magnetic Resonance Imaging slices using Digital Image Processing Techniques*. GitHub. 2022. URL: https://github.com/KaueTND/Brain_Extraction_T1wMRI.
- [16] Nicholas J. Tustison et al. “N4ITK: Improved N3 Bias Correction”. Em: *IEEE Transactions on Medical Imaging* 29 (6 2010), pp. 1310–1320.
- [17] *N4 Bias Field Correction*. Insight Toolkit. URL: https://simpleitk.readthedocs.io/en/master/link_N4BiasFieldCorrection_docs.html.
- [18] *ImageNet*. URL: <https://www.image-net.org/index.php>.
- [19] *Keras Applications*. Keras. URL: <https://keras.io/api/applications/>.
- [20] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. Em: (2019).
- [21] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. Em: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [22] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: [10.48550/ARXIV.1412.6980](https://doi.org/10.48550/ARXIV.1412.6980). URL: <https://arxiv.org/abs/1412.6980>.
- [23] *tf.keras.losses.SparseCategoricalCrossentropy*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy.
- [24] *OASIS Brains - Open Access Series of Imaging Studies*. URL: <https://www.oasis-brains.org/>.
- [25] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. Em: (2017).
- [26] *ReLU6 activation function*. URL: <https://paperswithcode.com/method/relu6>.