Identificando artroses em raixo x

1st Yago A. M. Faria
Instituto de Ciências Exatas e Informática
PUC Minas
Belo Horizonte, Brasil
yagoyagofaria@gmail.com

2st Lucas Baesse
Instituto de Ciências Exatas e Informática
PUC Minas
Belo Horizonte, Brasil
lucas.baesse@sga.pucminas.br

I. Introdução

A osteoartrite (artrose) é uma doença que se caracteriza pelo desgaste da cartilagem articular e por alterações ósseas nas articulações. O raio X é o principal exame para diagnóstico da doença que é classificada pela escala de Kellgren Lawrence, de acordo com o seu grau de severidade. O diagnóstico de artrose é confirmado para KL maior que 1.

Esse trabalho consiste no desenvolvimento, simulação, treinamento, e testagem de uma rede neural convolucioanl e de um classificador raso, com fins de identificar e diagnosticar casos de artrose de acordo com o imagem da Fig1

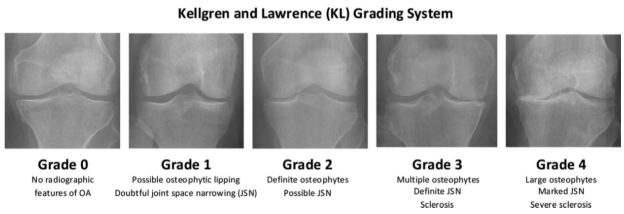

Fig. 1. Grades de artrose - Escala de Kellgren and Lawrence

O projeto será divido em duas frentes principais sendo elas o desenvolvimento de dois classificadores raso (um binário e um multi classe), baseado em características das imagens e duas redes neurais convolucionais (uma binária e uma multi classe)

A. Dados para aplicação

Os dados foram retirados de um banco de dados de uma universidade da califórnia chamada *University of California San Francisco Parnassus Campus* (UCSF). Ele proporcionou em torno de nove mil imagens de raios x de joelhos divididos em classes de 0 a 4 (Fig1).

II. Aumento de dados

A. Introdução

Para implementação, validação e aplicação dos algorítmos de processamento de imagens é necessário uma série de métodos e tratamentos afim de preparar os dados para serem corretamente consumidos pelos algorítmos, sendo eles tanto para treinamento quanto para criação dos modelos.

Para input dos dados foi-se realizado aumento de dados através de espelhamento horizontal e equalização de histogramas.

1) Equalização de histogramas: É uma técnica de transformação de intensidade que tem por objetivo balancear os níveis de cinza em uma imagem de forma automática, sem precisar de parâmetros e configurações adicionais. Dessa forma, imagens com um nível de brilho desbalanceados, ou seja, claras ou escuras demais atingem uma distribuição normalizada, o que garante um melhor contraste e visualização dos detalhes presentes na cena.

Fig. 2. Imagem equalizada

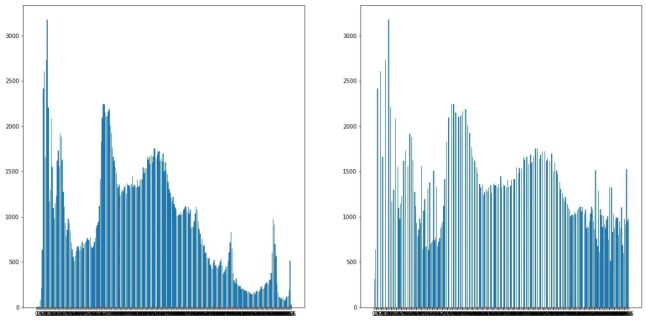

Fig. 3. Histograma da astronauta sem e com equalização respectivamente.

2) Espelhamento horizontal: A operação de espelhamento consiste em inverter a imagem. A operação consiste, no caso do horizontal, para cada píxel de uma linha, ele pega o primeiro e coloca na última posição da imagem de destino. Depois pega o segundo e coloca na penúltima posição. Esse procedimento ocorre até que todos os pixeis da linha tiverem sido colocado na sua respectiva posição da imagem de destino. Depois segue para a próxima linha até que toda a imagem tenha sido rasterizada

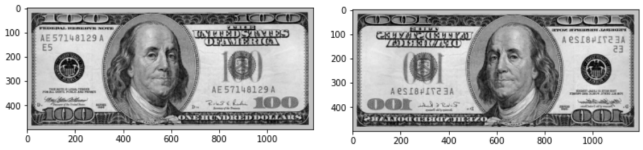

Fig. 4. Imagem espelhada horizontalmente.

III. Classificador Raso

A. *Introdução*

O aprendizado de máquina é uma aplicação de inteligência artificial, que permite que o computador opere em modo de autoaprendizagem, sem ser explicitamente programado. É um tema muito interessante e complexo, que pode impulsionar o futuro da tecnologia.

Existem inúmeras aplicações de aprendizado de máquina, das quais a classificação de imagens é uma delas. Para classificar imagens, foi-se estamos usando SVM. O Scikit-learn é uma biblioteca de aprendizado de máquina de software livre para a linguagem de programação Python e o Support vector machines (SVMs) está incluso no Scikit-learn.

1) Support vector machines (SVM) : É um algoritmo de aprendizado de máquina supervisionado que pode ser usado para desafios de classificação ou regressão. No entanto, é mais usado em problemas de classificação. Neste algoritmo SVM, foi-se plotado cada item de dados como um ponto no espaço n-dimensional (onde n é o número de recursos que você possui) com o valor de cada recurso sendo o valor de uma determinada coordenada. Em seguida, foi-se realizada a classificação, encontrando o hiperplano que diferencia muito bem as duas classes. Alguns dos principais parâmetros do SVM são:

- Gamma: Define até que ponto a influência de exemplos de treinamento único atinge valores que levam a resultados tendenciosos.
- Kernel: Os algoritmos SVM usam um conjunto de funções matemáticas que são definidas como o kernel. Os tipos de Kernels são: Linear, RBF (Função de Base Radial), Polinomial
- C: Controla o custo de erros de cálculo

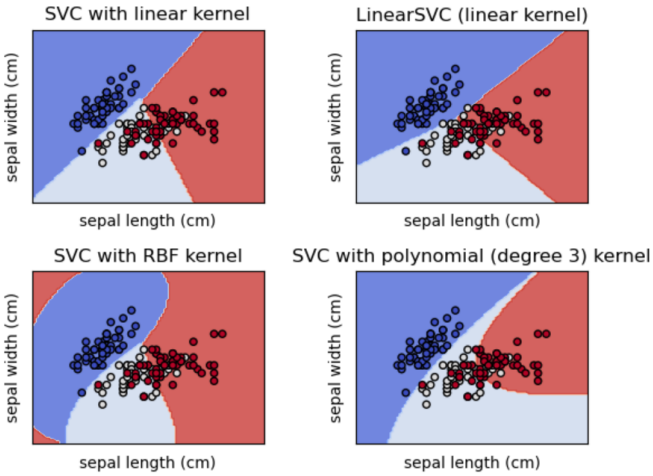

Fig. 5. Support vector machines Kernels

```
param_grid={'C':[0.1,1,10,100],'gamma':[0.0001,0.001,0.1,1],'kernel':['rbf','poly']}
```
Python

Fig. 6. Configuração dos parâmetros do SVC

2) GridSearchCV : É uma função de biblioteca que é membro do pacote modelselection do sklearn. Isso ajuda a percorrer hiperparâmetros predefinidos e ajustar seu estimador (modelo) em seu conjunto de treinamento. Portanto, no final, você pode selecionar os melhores parâmetros dos hiperparâmetros listados.

```
svc=svm.SVC(probability=True)
model=GridSearchCV(svc,param_grid)
```
Python

Fig. 7. Configuração dos parâmetros do SVC

B. *Desenvolvimento*

1) Input do Dataset : Utilizando Jupiters Notebooks no Google Collabs, o código principal foi conectado com o a base de dados de treinamento no Google Drive. 5 categorias foram criadas, sendo elas as categoricas e as binárias.

Como o SVM recebe entradas do mesmo tamanho, todas as imagens precisam ser redimensionadas para um tamanho fixo antes de inseri-las no SVM. Um quadro de dados criado (df) usando pandas e x e y são dados de entrada e saída, respectivamente.

- Binárias: Classificam de 0 a 4, os graus de Artrose
- Categoricas: Classificam se há artrose ou se não há artrose.

```python
Categories=['caso0','caso1','caso2', 'caso3', 'caso4', 'casoSA', 'casoCA']
flat_data_arr=[]
target_arr=[]
datadir='/content/drive/MyDrive/ENG COMP/8 periodo/PAI/TP2/Dataset/Train'
for i in Categories:
  print(f'Carregando : {i}')
  path=os.path.join(datadir,i)
  for img in os.listdir(path):
    try:
      img_array=imread(os.path.join(path,img))
      img_resized=resize(img_array,(150,150,3))
      flat_data_arr.append(img_resized.flatten())
      target_arr.append(Categories.index(i))
    except:
      print('Não consegui ler o arquivo')
flat_data=np.array(flat_data_arr)
target=np.array(target_arr)
df=pd.DataFrame(flat_data)
df['Target']=target
df
```

Python

Fig. 8. Desenvolvimento do tratamento e input de dados.

2) Construção e treinamento do modelo : Neste caso de projeto, o model será Support vector machines. O algoritmo para construção do modelo é o seguinte:

- Dividir o quadro de dados em dados de treinamento e em dados de teste.
- Crie um classificador Support vector machines:
- Utilizando o GridSearchCV e o paramsGrid, crie um modelo.

Os dados são divididos em duas categorias: dados de treinamento e dados de teste. Os dados de treinamento são usados para treinar o modelo, enquanto os dados de teste são usados para testar o modelo treinado. Para dividir os dados em treinamento e teste, traintestsplit() da biblioteca sklearn é usado. O modelo é treinado usando dados de treinamento na seguinte forma:

```python
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=77,stratify=y)

param_grid={'C':[0.1,1,10,100],'gamma':[0.0001,0.001,0.1,1],'kernel':['rbf','poly']}
svc=svm.SVC(probability=True)

print("The training of the model is started, please wait for while as it may take few minutes to complete")

model=GridSearchCV(svc,param_grid)
model.fit(x_train,y_train)
model.best_params_

print('The Model is trained well with the given images')
```

Python

Fig. 9. Construção e treinamento do modelo

Para fins de registro de indicadores, surgiu a necessidade de implementar um contador de tempo de execução, com a finalide de coletar o tempo consumido para o treinamento dos modelos. Utilizando a biblioteca import time o seguinte código foi desernvolvido para a finalidade mencionada.

```python
import time
from reader import feed

tic = time.perf_counter()
toc = time.perf_counter()
time = toc - tic

file = open('time1.txt', 'w')
file.write('%s' % time)
file.close()
```

Python

Fig. 10. Timer de execução do modelo

3) Teste do modelo : agora o modelo é testado usando dados de teste, desta forma model.predict(testingData). A precisão do modelo pode ser calculada usando o método AccuracyScore() de sklearn.metrics.

```python
y_pred=model.predict(x_test)
print("The predicted Data is :")
y_pred

print("The actual data is:")
np.array(y_test)

print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
```

Python

Fig. 11. Validação do modelo

4) Avaliação do modelo : Finalmente, na fase de avaliação do modelo, o modelo gerado pode ser usado para avaliar novos dados, além de um novo treinamento mdeiante à uma analise humana, após o modelo realizar uma classificação incorreta.

```python
model=pickle.load(open('img_model_Categorie.p','rb'))
url=input('Inter URL of Image: ')
img=imread(url)

img_resize=resize(img,(150,150,3))
l=[img_resize.flatten()]
probability=model.predict_proba(l)

for ind,val in enumerate(Categories):
  print(f'{val} = {probability[0][ind]*100}%')
print("The predicted image is : "+Categories[model.predict(l)[0]])
print(f'Is the image a {Categories[model.predict(l)[0]]} ?(y/n)')
```

Python

Fig. 12. Avaliação do Modelo Parte 1

```python
if(b=='n'):
  print("What is the image?")
  for i in range(len(Categories)):
    print(f"Enter {i} for {Categories[i]}")
  k=int(input())
  while(k<0 or k>=len(Categories)):
    print(f"Please enter a valid number between 0-{len(Categories)-1}")
    k=int(input())
  print("Please wait for a while for the model to learn from this image :)")
  flat_arr=flat_data_arr.copy()
  tar_arr=target_arr.copy()
  tar_arr.append(k)
  flat_arr.extend(l)
  tar_arr=np.array(tar_arr)
  flat_df=np.array(flat_arr)
  df1=pd.DataFrame(flat_df)
  df1['Target']=tar_arr
  model1=GridSearchCV(svc,param_grid)
  x1=df1.iloc[:,:-1]
  y1=df1.iloc[:,-1]
  x_train1,x_test1,y_train1,y_test1=train_test_split(x1,y1,test_size=0.20,random_state=77,stratify=y1)
  d={}
```

Python

Fig. 13. Avaliação do Modelo Parte 2

```python
for i in model.best_params_:
    d[i]=[model.best_params_[i]]
model1=GridSearchCV(svc,d)
model1.fit(x_train1,y_train1)
y_pred1=model.predict(x_test1)
print(f"The model is now {accuracy_score(y_pred1,y_test1)*100}% accurate")
pickle.dump(model1,open('img_model_Categorie.p','wb'))
```

Python

Fig. 14. Avaliação do Modelo Parte 3

C. *Resultados*

1) Input de dados: Como resultado do input e tratamento de dados, foi obtido o quadro de dados que será dividido em dados de treino e dados de teste.

	0	1	2	3	4	5	6	7	8	9	...	67491
0	0.337016	0.337016	0.337016	0.340673	0.340673	0.340673	0.346013	0.346013	0.346013	0.349020	...	0.308837
1	0.420575	0.420575	0.420575	0.426431	0.426431	0.426431	0.431598	0.431598	0.431598	0.437336	...	0.384314
2	0.552451	0.552451	0.552451	0.556810	0.556810	0.556810	0.561752	0.561752	0.561752	0.563898	...	0.432288
3	0.271794	0.271794	0.271794	0.288555	0.288555	0.288555	0.310840	0.310840	0.310840	0.327004	...	0.078431
4	0.478431	0.478431	0.478431	0.478431	0.478431	0.478431	0.477742	0.477742	0.477742	0.466928	...	0.396078
...	...	...	...	...	...	...	...	...	...	...	...	...
1474	0.477412	0.477412	0.477412	0.497882	0.497882	0.497882	0.519425	0.519425	0.519425	0.531997	...	0.326405
1475	0.341176	0.341176	0.341176	0.344078	0.344078	0.344078	0.349020	0.349020	0.349020	0.349020	...	0.333333
1476	0.301908	0.301908	0.301908	0.319477	0.319477	0.319477	0.346928	0.346928	0.346928	0.360787	...	0.003007
1477	0.411712	0.411712	0.411712	0.449699	0.449699	0.449699	0.487791	0.487791	0.487791	0.526638	...	0.364323
1478	0.602954	0.602954	0.602954	0.600768	0.600768	0.600768	0.604837	0.604837	0.604837	0.611396	...	0.452863

1479 rows × 67501 columns

Fig. 15. Quadro de dados (df)

2) Treinamento dos modelos : Após o treinamento do modelos, suas acuraciais foram geradas junto dos tempos de execução do treinamento sendo o Modelo Binário contendo uma execução de 02:40:37 e uma acurácia de 87.85714285714286 , o treinamento do Modelo Categorico apresentou um tempo de execução de 08:35:21 e uma acurácia de 41.935483870967744.

```
Splitted Successfully
The training of the model is started, please wait for while as it may take few minutes to complete
The Model is trained well with the given images
The predicted Data is :
The actual data is:
The model is 87.85714285714286% accurate
```

Fig. 16. Treinamento do Modelo Binário

```
Splitted Successfully
The training of the model is started, please wait for while as it may take few minutes to complete
The Model is trained well with the given images
The predicted Data is :
The actual data is:
The model is 41.93548387096744% accurate
```

Fig. 17. Treinamento do Modelo Categorico

3) Avaliação dos modelos : Durante a avaliação dos modelos dois principais testes foram realizados, um testando a acetividade do modelo binário que foi bem sucedida e não necessitou um retreinamento. O segundo teste foi realizado encima do modelo categorico que apresentou erro na classificação e demandou uma analise humana pra o retreinamento.

```
Enter URL of Image: /content/drive/MyDrive/ENG COMP/8 periodo/PAI/TP2/Dataset/Train/caso0/9066820R.jpg
casoSA = 29.096259672249618%
casoCA = 70.90374032775037%
The predicted image is : casoCA
Is the image a casoCA ?(y/n)
n
What is the image?
Enter 0 for casoSA
Enter 1 for casoCA
0
Please wait for a while for the model to learn from this image :)
The model is now 88.65248226950354% accurate
Thank you for your feedback
```

Fig. 18. Avaliação do Modelo Binário

```
Enter URL of Image: /content/drive/MyDrive/ENG COMP/8 periodo/PAI/TP2/Dataset/Test/caso2/9062328L.jpg
caso0 = 3.401381194906256%
caso1 = 18.096833167787494%
caso2 = 9.735216526774721%
caso3 = 30.41620631726008%
caso4 = 38.35036279408706%
The predicted image is : caso4
Is the image a caso4 ?(y/n)
n
What is the image?
Enter 0 for caso0
Enter 1 for caso1
Enter 2 for caso2
Enter 3 for caso3
Enter 4 for caso4
2
Please wait for a while for the model to learn from this image :)
```

Fig. 19. Avaliação do Modelo Categorico

4) Controle e Reulagem de Hiperparâmetros : Para melhorar a acurácia do modelo construído com SVC, foi necessário uma seuqência de testes de reulagem dos hiperparâmetros, neste caso obtivemos uma melhora não muito expresiva ao reduzir o parâmetro gamma até 0,1,pois notamos que ao reduzir o hiperparâmetro para valores menores que 0,1 a acurácia declinou novamente, mesmo fenômeno foi observado para o modelo binpario e categorico.

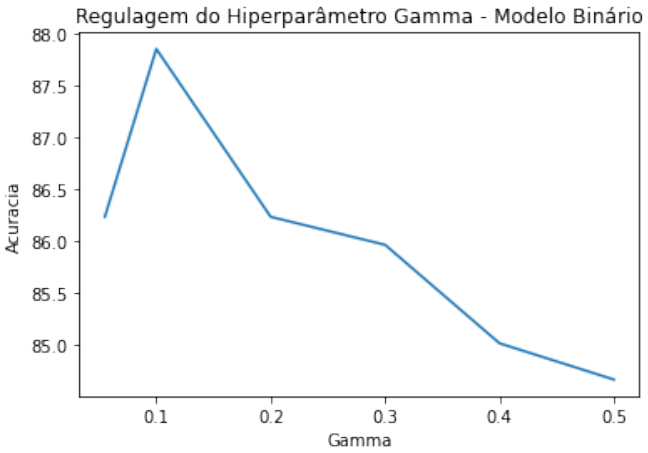

Fig. 20. Regulagem do Hiperparâmetro Gamma - Modelo Binário

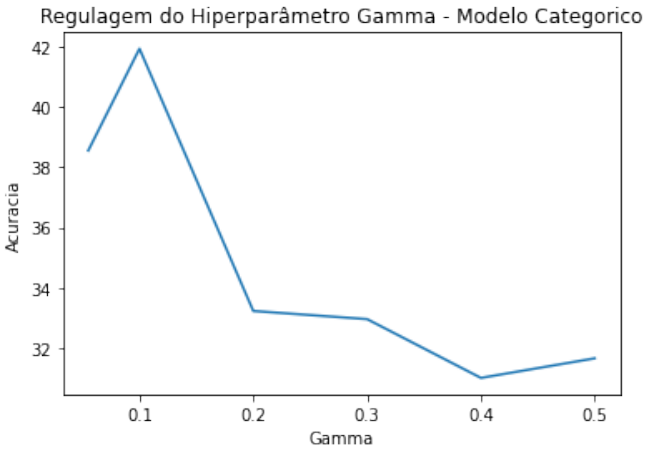

Fig. 21. Regulagem do Hiperparâmetro Gamma - Modelo Categorico

IV. CNN - Rede neural convulacional

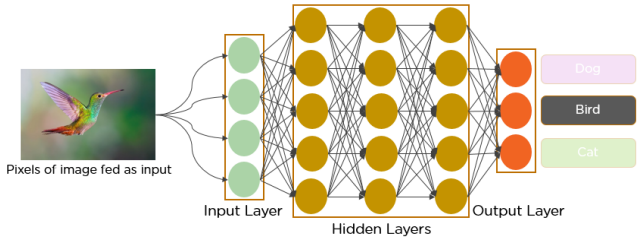

Fig. 22. Exemplo CNN

A. *Introdução*

Uma Rede Neural Convolucional ou RNC, como também é conhecida, é um algoritmo de deep learning que é capaz de captar uma imagem de entrada atribuir importâncias e diferenciar aspectos e objetos presentes nela.

Ela geralmente é composta de 3 camadas, cada camada com sua respectiva funcção:

- **Input layer (camada de enytada)** - Camada responsável por receber os dados iniciais para que sejam utilizados no processamento de dados nas camadas seguintes
- **Hidden Layers (camadas escondidas)** - Camada com funções matemáticas não lineares, em que, cada uma e modelada para retornar um valor específico ,de acordo com os dados de entrada, para que seja classificado de acordo com a Camada de saída (Output Layer)
- **Output Layer (camada de saída)** - Camada responsável por produzir o resultado final de acordo com as predições da Hidden Layers

O funcionamento final é como demonstrado na Fig24, onde o dado de entrada é uma imagem com um passarinho e após passar por todas as camadas escondidas e realizar as predições ele retorna o qué é o objeto, no casso um pássaro.

Um aspecto importante nas CNN são as funções de ativação, nesse projeto foiram utilizadas duas a sigmoid e a softmax

1) ***Sigmoid::*** A função de ativação sigmoid é comumente utilizada por redes neurais com propagação positiva (Feedforward) que precisam ter como saída apenas números positivos, em redes neurais multicamadas e em outras redes com sinais contínuos.

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Fig. 23. Equuação da sigmoid

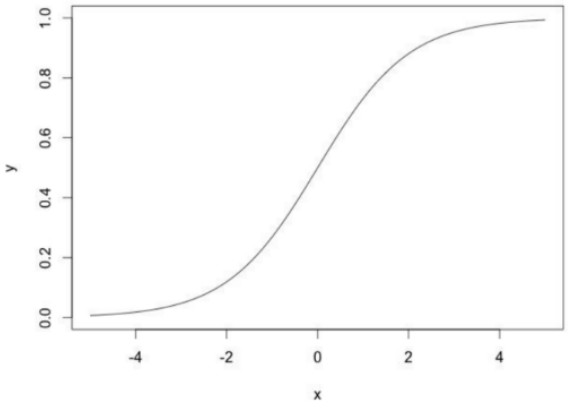

Fig. 24. Gráfico da sigmoid

2) ***Softmax::*** A função de ativação softmax é usada em redes neurais de classificação. Ela força a saída de uma rede neural a representar a probabilidade dos dados serem de uma das classes definidas. Sem ela as saídas dos neurônios são simplesmente valores numéricos onde o maior indica a classe vencedora

$$\phi_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

Fig. 25. Equação softmax

Nessa equação, i representa o índice do neurônio de saída (o) sendo calculado e j representa os índices de todos os neurônios de um nível. A variável z designa o vetor de neurônios de saída . Vale notar que a função de ativação softmax é calculada de forma diferente das demais apresentadas, uma vez que a saída de um neurônio depende dos outros neurônios de saída.

B. *Desenvolvimento*

Para que seja possivel montar nossa rede foi utilizado a linguagem Phyton 3.7.2 em uma plataforma chamada google collab, consiste em uma VM disponibilizada pela google (para mais informações acesse: http://www.overleaf.com.

Foram criadas duas redes neurais uma que chamaremos de CNN - Categórica, que categoriza os dados de entrada nas cinco classes Fig.1 e outra de CNN - binária que classifica os dados como ”com artrose” e ”sem artrose”.

As bibliotecas utilizadas para montagem foram:

- **Tensoflow** - Principal bibliotaca com os métodos para criarmos a rede
- **Keras** - Biblioteca derivada do tesnorflow que nos dá uma forma mais amigavel de utilizar os comandos do tensorflow
- **Numpy** - Para transformar os dados em vetores
- **Matplotlib** - Para plotar as matrizes e os gráficos necesários para a análise

- **Sklearn** - Para gerar a matriz de confusão

C. Resnet 50

O modelo de rede neural mais eficiente utilizado foi em ambas as redes (categóica e a binária) a rede convulacional ResNet50. Que consiste em 5 estágios cada um com uma convolução e um bloco de identidade. Cada bloco de convolução possui 3 camadas de convolução e cada bloco de identidade também possui 3 camadas de convolução. O ResNet-50 tem mais de 23 milhões de parâmetros treináveis. (Fig26). Para mais informações sobre a arquiterura da resnet https://iq.opengenus.org/resnet50-architecture/

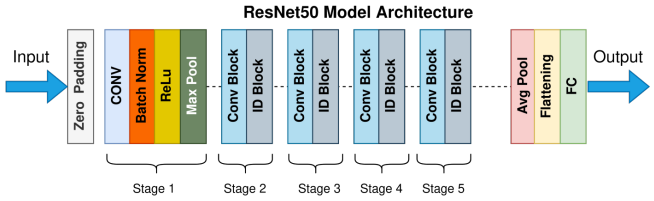

Fig. 26. Arquitetura - ResNet50

Esse modelo foi escolhido por ser um modelo eficiente e utilizado par adiversas aplicações, um exemplo seria a identificação de COVID-19 em pulmões

D. CNN - Binária

Essa rede consiste em classificar o dado de entrada de acordo com as seguintes categorias *com artrose* e *sem artrose*, em que foram agrupados os dados das clasees 0 e 1 e das classes 2,3,4 respectivamente.

1) ***Tratando os dados****:* Para o carregamento de imagens foi utulizado o método *ImageDataGenerator* para gerar imagens a partir do dataset que foi mencionadao na seção 1 para treino, validação e teste com as seguintes modificações:

- **z**oom range - Da um zoom na imagem
- **h**orizontal flip - espelha a imagem
- **s**hear - Muda os angulos das imagens
- **r**eescale - Muda o tamanho da imagem (244x244 pixels)

2) ***Código****:* Após o carregamento dos dados foram setados valores constantes para serem aplicados no treinamento:

- **E**pocas = 30
- **P**assos por época = 45
- **B**ach size = 34
- **U**m vetor de classes = [com artrose, sem artrose]

Com os padrões definidos, está na hora de aplicar o modelo:

```
res = ResNet50( input_shape=(224,224,3), include_top= False)
```

Fig. 27. Carregando dados na resnet

Depois foi adicionado uma camada Flatten para carregar os dados das outras hidden layers em um vetor e uma Dense layer com dois neuronios e uma função de ativação sigmoid

```
x = Flatten()(res.output)
x = Dense(units=2 , activation='sigmoid', name = 'predictions' )(x)

# modelo com oa RES_NET50 + as duas camadas acima.
model = Model(res.input, x)
```

Fig. 28. Carregando as duas camadas na Resnet

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4160        ['pool1_pool[0][0]']

...
Total params: 23,788,418
Trainable params: 23,735,298
Non-trainable params: 53,120
```

Fig. 29. model sumarry resumido - Obs não consta todo o model

Além disso foram adicionadas duas callback functions, uma para parar se a acurácia do modelo não melhorar depois de três épocas ele para de treinar e outra para salvar o modelo toda vez que a acurácia melhorar.

3) ***Compilando o modelo****:* Para compilação do modelo foram adicionados os seguintes parâmetros:

- **Função de otimisação Adam** - A otimização de Adam é um método de descida de gradiente estocástico baseado na estimativa adaptativa de momentos de primeira e segunda ordem. De acordo com Kingma et al., 2014, o método é ”computacionalmente eficiente, tem pouco requisito de memória, invariante ao reescalonamento diagonal de gradientes e é adequado para problemas grandes em termos de dados/parâmetros”
- **Keras categorial-crossentropy** - uma função de perda para o modelo de classificação multiclasse onde há dois ou mais rótulos de saída. O rótulo de saída é atribuído ao valor de codificação de categoria one-hot na forma de 0s e 1. O rótulo de saída, se presente no formato inteiro, é convertido em codificação categórica usando o método keras.utils to-categorical

E. Treinamento

1) ***Treinando - toda a Resnet50****:* A primeira vez foi realizada um teste com toda a rede neural Resnet, pode se ver o processo de treino na figura abaixo

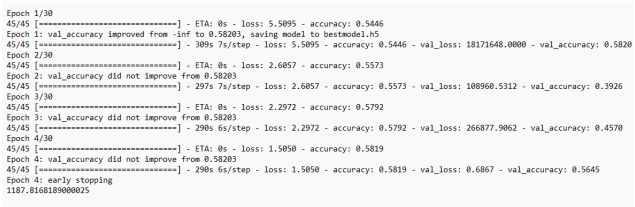

Fig. 30. Treinamento toda resnet

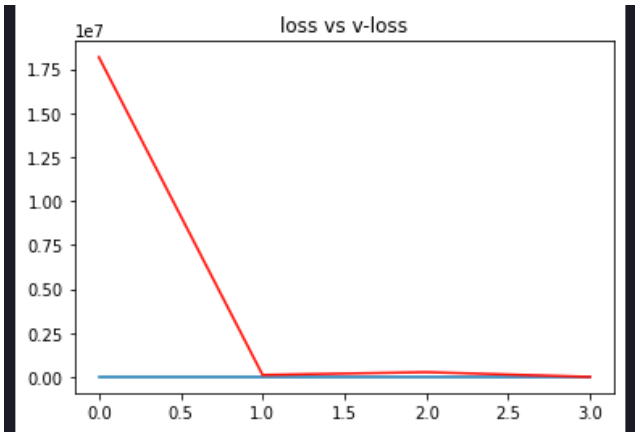

Fig. 31. Loss (azul) versus validation loss (vermelho)

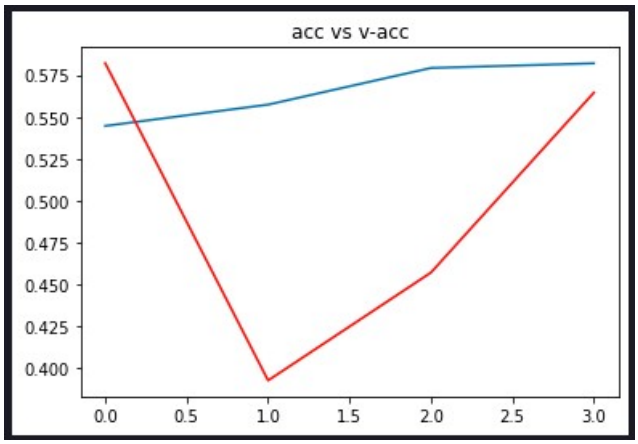

Fig. 32. Accuaracy (azul) versus validation accuaracy (vermelho)

Matriz de confusão e resumo da predição:

$$\begin{bmatrix} 318 & 27 \\ 316 & 165 \end{bmatrix}$$

–	precision	recall	f1-score	support
0	0.50	0.92	0.65	345
1	0.86	0.34	0.49	481
accuracy	–	–	0.58	826
macro avg	0.68	0.63	0.57	826
weighted avg	0.71	0.58	0.56	826

Accuracy do modelo é = 0.2610

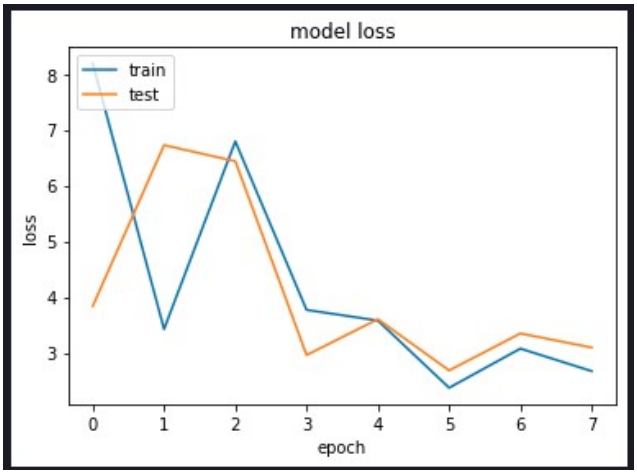

Fig. 33. Loss (azul) versus validation loss (laranja)

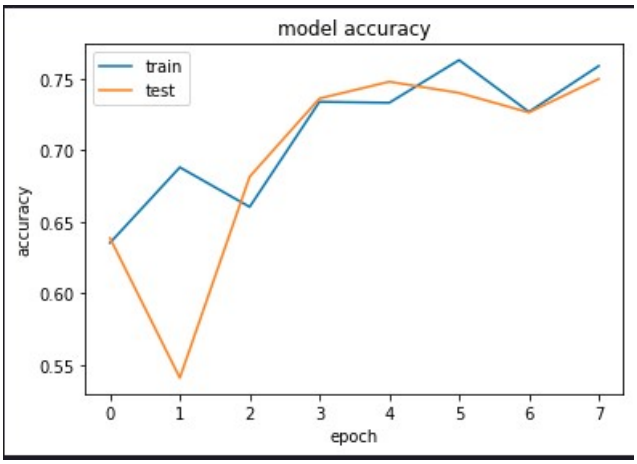

Fig. 34. Loss (azul) versus validation loss (laranja)

*2) **Treinando - Apenas uma parte da rede Resnet 50**:* Matriz de confusão e resumo da predição:

$$\begin{bmatrix} 247 & 98 \\ 137 & 344 \end{bmatrix}$$

–	precision	recall	f1-score	support
0	0.64	0.72	0.68	345
1	0.78	0.72	0.75	481
accuracy	–	–	0.72	826
macro avg	0.71	0.72	0.71	826
weighted avg	0.72	0.72	0.72	826

Accuracia do modelo é = 0.4031

F. Tetstes

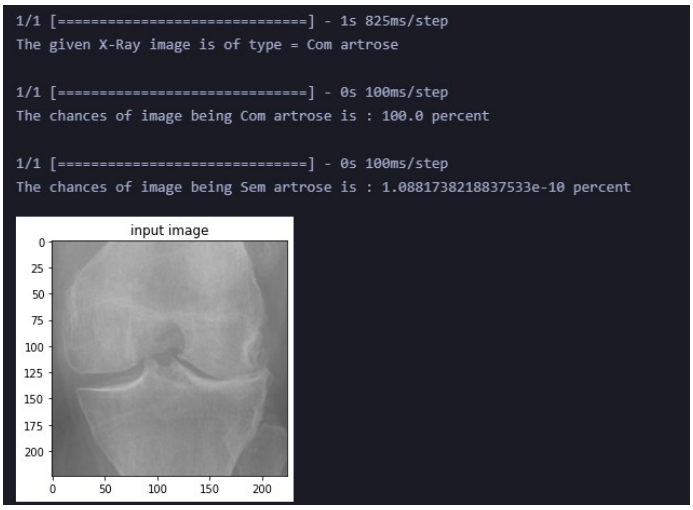

Fig. 35. Teste com artrose - acerto

Fig. 36. Teste com artrose - erro

G. Conclusão

Podemos ver que ao treinar toda a o modelelo novamente do zero os valores da accuracia e do erro se mantém no mesmo valor muito cedo, tornando o treinamento ineficiente. Já utilizando o modelo sem retreinar todo o modelo, a acuracia é maior e o comportamento dos gráficos não indicam um processamento tão descrepante do esperado(a cada época que passa a rede aprende um pouco mais).

H. CNN - Categórica

Para a implementação da rede neural categórica a lóigica foi quase mesma da CNN - binparia mas com alterações em dois aspectos, função de ativação e número de neuronios na dense layer

I. Modelo modificado

```
x = Flatten()(res.output)
x = Dense(units=5 , activation='softmax', name = 'predictions' )(x)

# Criando o modelo com as novas camadas
model = Model(res.input, x)
```

Fig. 37. Modelo modificado

1) ***Treinando - Toda a rede Resnet 50****:* Matriz de confusão e resumo da predição:

$$\begin{bmatrix} 279 & 8 & 32 & 9 & 0 \\ 124 & 3 & 23 & 3 & 0 \\ 150 & 5 & 53 & 4 & 0 \\ 50 & 2 & 50 & 4 & 0 \\ 8 & 3 & 16 & 0 & 0 \end{bmatrix}$$

–	precision	recall	f1-score	support
0	0.46	0.85	0.59	328
1	0.14	0.02	0.03	153
2	0.30	0.25	0.27	212
3	0.20	0.04	0.06	106
4	0.00	0.00	0.00	27
accuracy	–	–	0.12	826
macro avg	0.22	0.23	0.19	826
weighted avg	0.31	0.41	0.32	826

Accuracy do modelo = 0.208

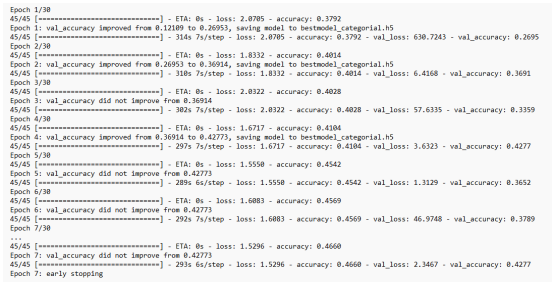

Fig. 38. Treinamento toda resnet

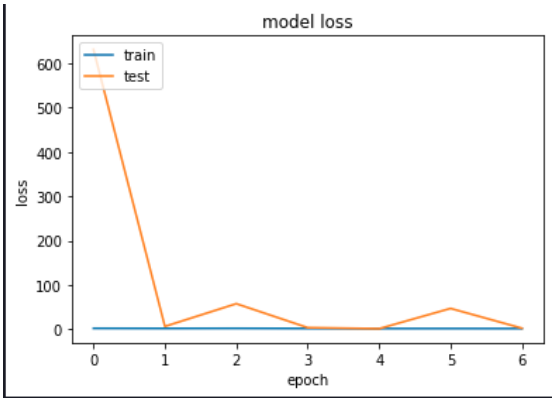

Fig. 39. Loss (azul) versus validation loss (laranja)

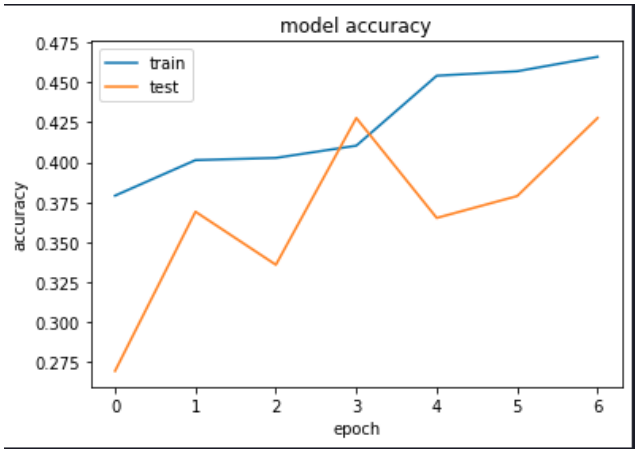

Fig. 40. Loss (azul) versus validation loss (laranja)

J. Testes

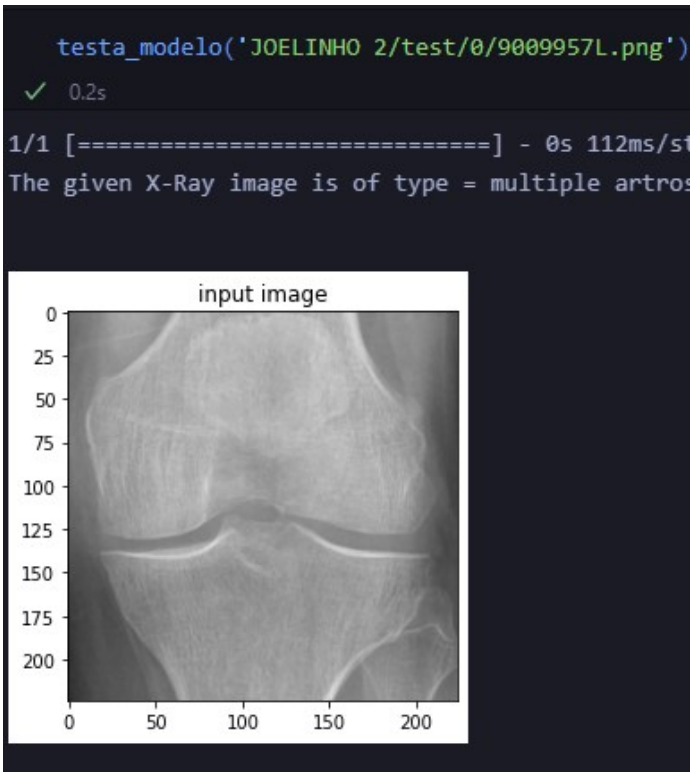

Fig. 41. Teste com artrose classe 1 - erro

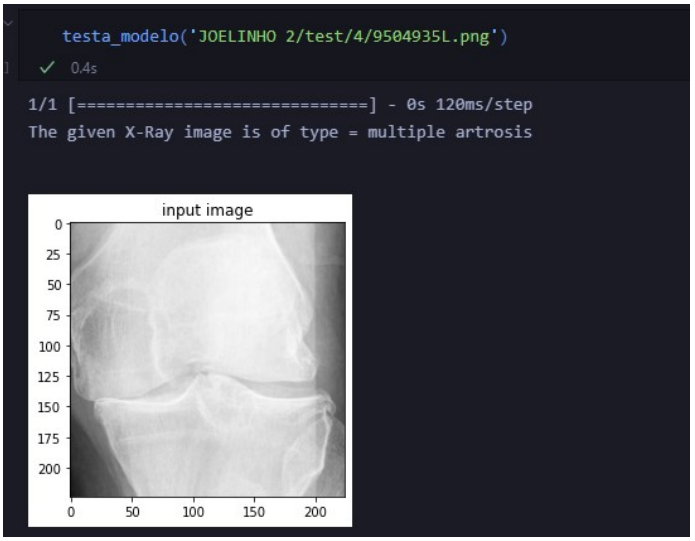

Fig. 42. Teste com artrose classe 4 - erro

*1) **Treinando - Apenas uma parte da rede Resnet 50**:* Matriz de confusão e resumo da predição:

$$\begin{bmatrix} 163 & 4 & 118 & 2 & 41 \\ 84 & 3 & 53 & 0 & 13 \\ 108 & 4 & 68 & 0 & 32 \\ 60 & 1 & 29 & 1 & 15 \\ 19 & 0 & 6 & 0 & 2 \end{bmatrix}$$

–	precision	recall	f1-score	support
0	0.57	0.67	0.62	358
1	0.21	0.10	0.13	153
2	0.39	0.24	0.30	212
3	0.31	0.63	0.42	106
4	0.68	0.56	0.61	27
accuracy	–	–	0.45	826
macro avg	0.43	0.44	0.42	826
weighted avg	0.43	0.45	0.42	826

Accuracy do modelo = 0.48671

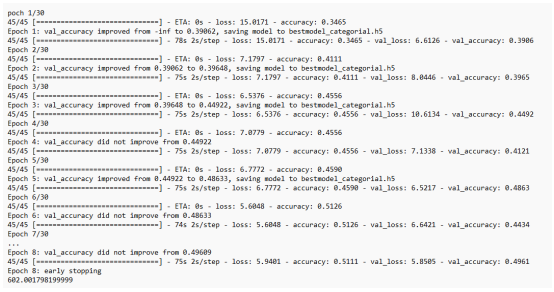

Fig. 43. Treinamento apenas uma parte da Resnet

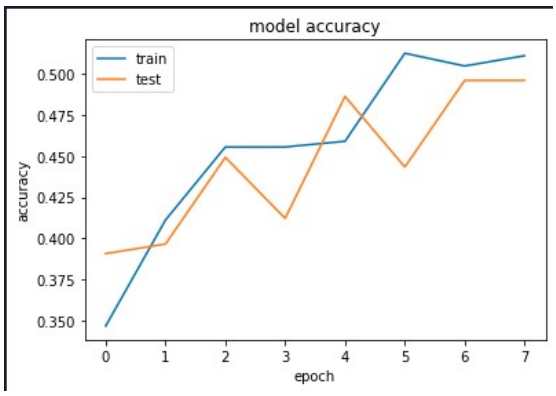

Fig. 44. Loss (azul) versus validation loss (laranja)

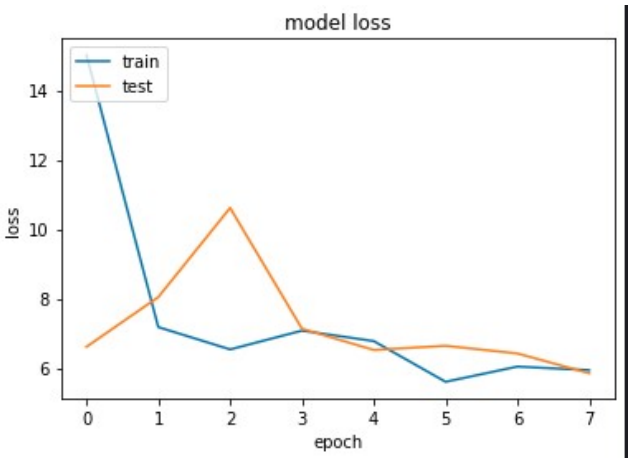

Fig. 45. Loss (azul) versus validation loss (laranja)

K. Testes

Fig. 46. Teste com artrose classe 4 - acerto

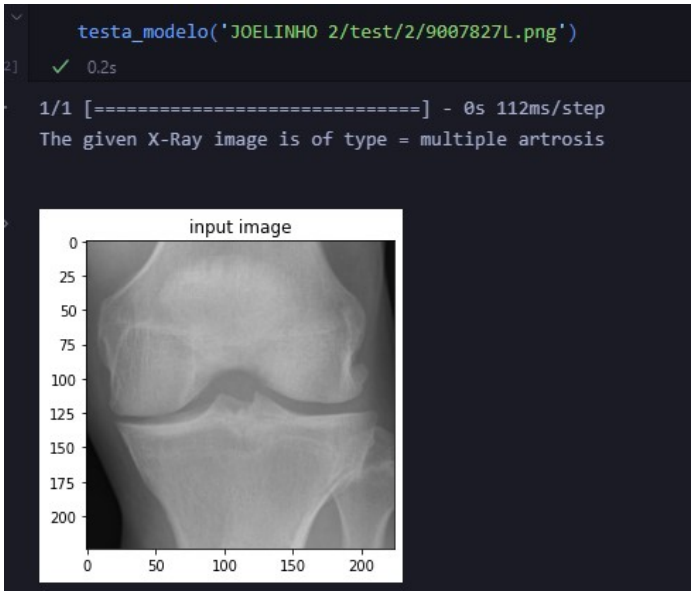

Fig. 47. Teste com artrose classe 2 - erro

Conclusão

De acorddo com os gráficos de teste e com a análise dos treinos, entramos no mesmo caso anterior que ocorreu na CNN- Binária, que é se toda a resnet for treinada os calores de acurácia e loss tem um comportamento muito incomum, com a loss ficando constante muito rápido e o acerto chegando ao um limite muito cedo.

Isso significa que é menos eficiente retreinar a rede para esse caso do que utilizar a inteligência pronta da ResNet.

V. Referências bibliográficas

- http://www.cpdee.ufmg.br/~troliveira/docs/aulas/lelti/Guia_555.pdf
- http://www.mecaweb.com.br/eletronica/content/eflip_flop
- http://www.cburch.com/logisim/docs
- https://www.mundodaeletrica.com.br/o-que-sao-amplificadores-operacionais/
- https://medium.com/analytics-vidhya/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90: :text=The
- https://keras.io/api/optimizers/adam/
- https://vitalflux.com/keras-categorical-cross-entropy-loss-function/