PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS NÚCLEO DE EDUCAÇÃO A DISTÂNCIA

Categorização e Classificação de cânceres de mama utilizando Rede Neural Convolucional (CNN)

São João de Meriti

Rodrigo de Brito de Oliveira Cabral

Classificação de cânceres de mama utilizando Rede Neural Convolucional (CNN)

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

São João de Meriti
2022

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

SIB PUC MINAS

AGRADECIMENTOS

Agradeço primeiramente a Deus pela saúde e oportunidade de concluir esse trabalho em tempos pandêmicos.

Aos meus pais, "Carlos Cesar Monteiro Cabral"e "Edinalva de Brito de Oliveira Cabral", por me acompanharem em muitos momentos importantes da minha vida e sempre incentivando o meu crescimento intelectual.

Aos amigos que me ajudaram nesse momento tão complicado que o mundo vive.

Aos professores e a PUC Minas por todo o conhecimento partilhado durante todo o curso.

Ao meu antigo mentor e minha atual mentora profissionais: "Rômulo Canella"e "Andreza Gonçalves", respectivamente.Por sempre me apoiarem enquanto a realização do curso.

Por último e não menos importante, gostaria de agradecer ao meu futuro filho, Nicolas Ravi, que está com previsão de nascimento em março de 2022. Pois, sua futura chegada me deu animo para ser cada vez melhor

LISTA DE FIGURAS

FIGURA 1 – Amostra de Histopatologia, na escala 40x 16
FIGURA 2 – Amostra de Histopatologia 17
FIGURA 3 – Amostra de Histopatologia nas amplificações (a)4x, (b)10x, (c)20x e (d)40x 17
FIGURA 4 – Screenshot da tela principal do Pycharm 18
FIGURA 5 – Código para mostrar nomes dos arquivos (Jupyter notebook) 19
FIGURA 6 – Amostra com nomes de imagens (Jupyter notebook) 19
FIGURA 7 – Função para mudar o nome das imagens (Jupyter notebook) 20
FIGURA 8 – Amostra com novos nomes, ampliação 40x 20
FIGURA 9 – Amostra com novos nomes, ampliação 100x 20
FIGURA 10 – Amostra com novos nomes, ampliação 200x 21
FIGURA 11 – Amostra com novos tamanhos das imagens 21
FIGURA 12 – Amostra da base dados Breast invasive ductal carcinomas (Jupyter notebook) 23
FIGURA 13 – Código de importação dos módulos (Google Colab) 24
FIGURA 14 – Código de criação do *"DataFrame"* (Google Colab) 25
FIGURA 15 – Amostra do *"DataFrame"* criado (Google Colab) 25
FIGURA 16 – Valores únicos de cada coluna (Google Colab) 25
FIGURA 17 – Representação gráfica de uma rede neural fictícia 32
FIGURA 18 – Representação gráfica de um neurônio fictício 33
FIGURA 19 – Representação gráfica de perceptron 34
FIGURA 20 – Representação gráfica de perceptron com entrada e saída 35
FIGURA 21 – Modelo de Perceptron não linear 35
FIGURA 22 – Diagrama de uma MLPs 36
FIGURA 23 – Diagrama de uma "Rede neural feedforward"e uma "Rede neural recorrente" 37
FIGURA 24 – Arquitetura de uma Rede neural convolucional 40
FIGURA 25 – Imagem representada em forma de matriz. 41
FIGURA 26 – Representação da camada de convolução 41

FIGURA 27 – Max pooling retângulo 2x2 42
FIGURA 28 – Average pooling retângulo 2x2 42
FIGURA 29 – Importações Biblioteca (Google Colab) 44
FIGURA 30 – Declaração das constantes (Google Colab) 44
FIGURA 31 – Segmentação Dataframe 400x (Google Colab) 44
FIGURA 32 – Segmentação Dataframe 200x (Google Colab) 45
FIGURA 33 – Diagrama de fluxo criação do modelo categoria 46
FIGURA 34 – Diagrama de fluxo criação do modelo classe 46
FIGURA 35 – Modelo Canvas 47
FIGURA 36 – Criação e carregamento de pesos (Google Colab) 48
FIGURA 37 – Código para teste do dataset (Google Colab) 48
FIGURA 38 – Criação e carregamento de pesos (Google Colab) 48
FIGURA 39 – Amostra com novos nomes, ampliação 400x 68
FIGURA 40 – Amostra com nomes de imagens (Jupyter notebook) 68
FIGURA 41 – Amostra da base dados Breast invasive ductal carcinomas (Jupyter notebook) 69
FIGURA 42 – Código para pegar nome da figura (Google Colab) 69
FIGURA 43 – Código para pegar grau de ampliação (Jupyter notebook) 69
FIGURA 44 – Código para selecionar a classe (Google Colab) 70
FIGURA 45 – Código para pegar os tipos (Jupyter notebook) 70

LISTA DE TABELAS

TABELA 1 – "BreakHis"resumo geral 16
TABELA 2 – "BreakHis"por categoria de câncer 16
TABELA 3 – Resumo geral 17
TABELA 4 – Tabela de métricas do gráfico 12 49
TABELA 5 – Tabela de métricas do gráfico 13 50
TABELA 6 – Tabela de métricas do gráfico 14 50
TABELA 7 – Tabela de métricas do gráfico 15 51
TABELA 8 – Tabela de métricas do gráfico 16 52
TABELA 9 – Tabela de métricas do gráfico 17 53
TABELA 10 – Tabela de métricas do gráfico 15 54
TABELA 11 – Tabela de métricas do gráfico 19 55
TABELA 12 – Tabela de métricas do gráfico 20 55
TABELA 13 – Tabela de métricas do gráfico 21 56
TABELA 14 – Tabela de métricas do gráfico 22 57
TABELA 15 – Tabela de métricas do gráfico 23 58
TABELA 16 – Tabela de métricas do gráfico 24 59
TABELA 17 – Tabela de métricas do gráfico 25 60
TABELA 18 – Tabela de métricas do gráfico 26 60
TABELA 19 – Tabela de métricas do gráfico 27 61
TABELA 20 – Tabela de métricas do gráfico 28 62
TABELA 21 – Tabela de métricas do gráfico 29 63
TABELA 22 – Tabela de métricas dos modelos para Categorias de câncer 29 . . 64
TABELA 23 – Tabela de métricas dos modelos para classe do câncer 29 64
TABELA 24 – Tabela de métricas das combinações 65

LISTA DE GRÁFICOS

GRÁFICO 1 – Distribuição de casos e mortes para os 10 principais cânceres mais comuns em 2020 para (A) Ambos os sexos, (B) Homens e (C) Mulheres 12
GRÁFICO 2 – Distribuição de Casos e Mortes por Área Mundial em 2020 para (A)Ambos os Sexos,(B)Homens e (C)Mulheres 13
GRÁFICO 3 – Distribuição de categorias de cancer 22
GRÁFICO 4 – Distribuição de Treino e Teste no *Dataset* 26
GRÁFICO 5 – Quantidade de dados entre Treino e Teste no *Dataset* 26
GRÁFICO 6 – Distribuição de Beigno e Maligno no Dataset 27
GRÁFICO 7 – Quantidade de dados entre Beigno e Maligno no Dataset 27
GRÁFICO 8 – Distribuição em porcentagens das classes 28
GRÁFICO 9 – Distribuição em porcentagens das classes referente aos tipos Malignos 29
GRÁFICO 10 – Distribuição em valores percentuais das classes por *"Magnification"* 30
GRÁFICO 11 – Distribuição em valores percentuais das classes base de treino e teste 31
GRÁFICO 12 – Matriz de Confusão Categoria de Cancer 400X 49
GRÁFICO 13 – Matriz de Confusão Categoria de Cancer 200X 49
GRÁFICO 14 – Matriz de Confusão Categoria de Cancer 50
GRÁFICO 15 – Matriz de Confusão Classe de Cancer 400X 51
GRÁFICO 16 – Matriz de Confusão Classe de Cancer 200X 52
GRÁFICO 17 – Matriz de Confusão Classe de Cancer 53
GRÁFICO 18 – Matriz de Confusão Categoria de Cancer 400X 54
GRÁFICO 19 – Matriz de Confusão Categoria de Cancer 200X 54
GRÁFICO 20 – Matriz de Confusão Tipo de Cancer 55
GRÁFICO 21 – Matriz de Confusão Classe de Cancer 400X 56
GRÁFICO 22 – Matriz de Confusão Classe de Cancer 200X 57
GRÁFICO 23 – Matriz de Confusão Classe de Cancer 58
GRÁFICO 24 – Matriz de Confusão Categoria de Cancer 400X 59
GRÁFICO 25 – Matriz de Confusão Categoria de Cancer 200X 59
GRÁFICO 26 – Matriz de Confusão Categoria de Cancer 60
GRÁFICO 27 – Matriz de Confusão Classe de Cancer 400X 61
GRÁFICO 28 – Matriz de Confusão Classe de Cancer 200X 62

GRÁFICO 29 – Matriz de Confusão Classe de Cancer 63

SUMÁRIO

1 INTRODUÇÃO **10**
1.1 Contextualização 10
1.2 O problema proposto 13

2 Coleta de Dados **15**
2.1 Breast Cancer Histopathological Database (BreakHis) 15
2.2 A histopathological image dataset for grading breast invasive ductal carcinomas 16

3 Processamento/Tratamento de Dados **18**
3.1 Ferramentas utilizadas 18
3.2 Tratamento das bases de dados 18
3.2.1 Tratamento da base de dados BreakHis 18
3.2.2 Tratamento da base de dados Breast invasive ductal car-cinomas . . 21

4 Análise e Exploração dos Dados **24**

5 Criação de Modelos de Machine Learning **32**
5.1 Rede Neural Natural (R.N.A.) 32
5.2 Rede Neural Artificial (R.N.A.) 33
5.2.1 Rede Neural Artificial Multicamadas ou MLPs 36
5.2.2 Funções de Ativação 37
5.2.3 Treinamento de uma Rede Neural 38
5.2.4 Algoritmo Backpropagation 39
5.2.5 Redes neurais convolucionais ou CNN 39
5.2.5.1 Image channels 40
5.2.5.2 Convolution 41
5.2.5.3 Pooling 42
5.2.5.4 Flattening 42
5.2.6 Transferência de aprendizado 43
5.3 Execução dos Modelos 43

6 Apresentação dos Resultados **47**
6.1 Modelo Preditivo - VGG19 48
6.2 Modelo Preditivo - GoogleNet 53
6.3 Modelo Preditivo - ResNet50 58
6.4 Resultado Geral 63
6.5 Combinação de Modelos 64

7 Trabalhos futuros **66**

8 Links **67**

9 Figuras / Graficos **68**

10 Código Final **71**

REFERÊNCIAS **161**

1 INTRODUÇÃO

1.1 Contextualização

A medicina é a área da ciência com o maior peso acadêmico. Através dos estudos nessa área a humanidade foi capaz de curar doenças, como, por exemplo, a tuberculose e a varíola. Porém, tão importante quanto curar uma enfermidade é saber como identifica-la, com esse objetivo, foram desenvolvidos vários métodos de detecção de enfermidades ao longo dos anos.

O diagnóstico por imagem é um conjunto de métodos que permite a obtenção de imagens do corpo, seja por inteiro ou de partes específicas. Após a obtenção da imagem, é necessário passar por uma análise clínica, com a finalidade de encontrar anomalias. No entanto, o profissional que exerce essa função se encontra sob grande pressão, dada a responsabilidade inerente de sua tarefa.

A mamografia é um tipo especial de técnica de Raios-x usado para criar imagens detalhadas da mama. É utilizado um aparelho chamado mamógrafo, onde, depois da exposição da mama aos Raios-x, o aparelho transforma a radiação em um sinal elétrico e o envia para um computador. As mamas são comprimidas com espátulas de plástico para otimizar a visualização do tecido da mama e suas anomalias.As imagens geradas ficam armazenadas em formato digital, disponíveis para impressão ou futuras consultas, tanto pela paciente quanto por radiologistas e outros profissionais da saúde .(MORSCH, 2018)

As mamografias de triagem são usadas para rastrear o câncer de mama em mulheres assintomáticas. As mamografias diagnósticas são usadas para diagnosticar doenças mamárias em mulheres com sintomas mamários, nódulos mamários palpáveis, resultados alterados no exame de mamografia que exigem maior investigação. As mamografias diagnósticas podem incluir incidências padrão e especializadas (OLIVEIRA, 2019).

A histologia é uma disciplina que estuda os tecidos dos seres vivos e suas organizações microscópicas, onde é possível entender claramente a maneira pela qual eles se relacionam entre si. Sendo a histologia patológica(ou Histopatologia) uma derivação da histologia.(O. . . , 2019)

> O exame histopatológico é, em linhas gerais, uma análise microscópica de um tecido para a detecção de possíveis alterações ou lesões, com o objetivo básico de informar ao clínico a natureza, a gravidade, a extensão, a evolução e a intensidade de algum problema do doador da amostra, além de sugerir ou confirmar a causa do problema.(O. . . , 2019)

No cenário mundial, o Câncer de Mama é uma das três categorias de maior incidência, com Câncer no Pulmão e o de próstata. Em 2020, foram encontrados 2,2 milhões de novos casos de Câncer de Mama, contribuindo com cerca de 11,7% do total de casos de câncer no mundo.Alcançando em 2020 o posto de câncer com mais incidência mundial.(SUNG et al., 2021, Tradução nossa). Como podemos ver no gráfico 1.

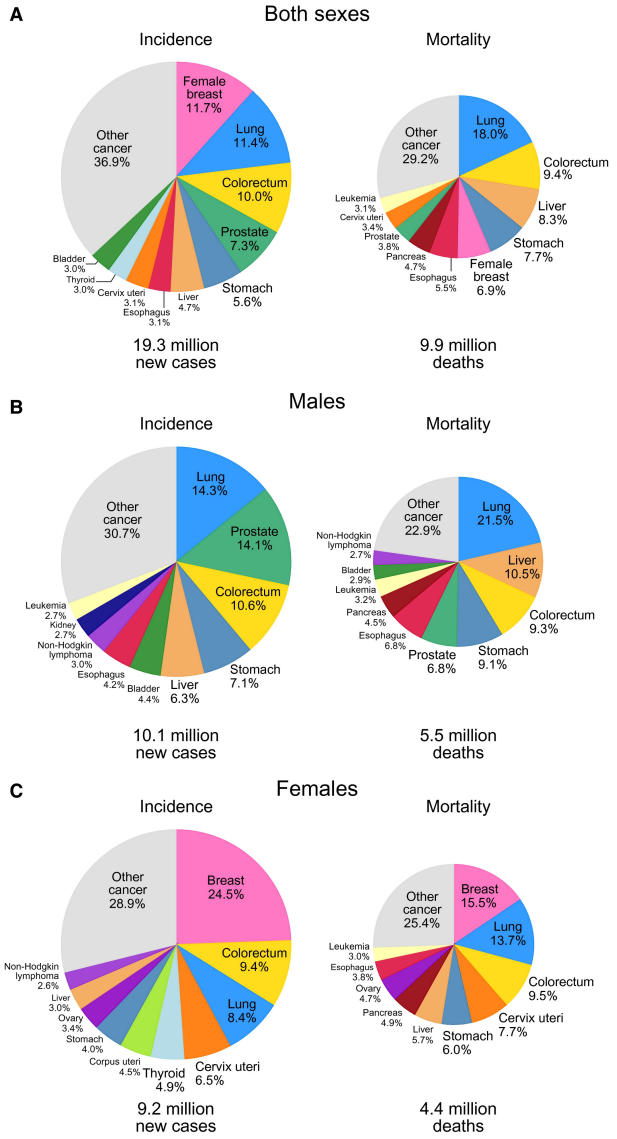
Essa categoria de câncer, é o quarto em questão de mortalidade no mundo, estimadas mais de 684,996 mil mortes em 2020 - o que representa 6,9% do total de mortes por todos os tipos da doença. O artigo também aponta que uma em cada quatro mulheres que têm um caso de câncer diagnosticado têm câncer de mama, representando 24,5% do total.(SUNG et al., 2021, Tradução nossa).Como podemos ver no gráfico 2.

No cenário Nacional, o Câncer de Mama é uma das três categorias de maior incidência INCA (2019), com Câncer de "Cólon e reto"e o de "próstata". Em 2020, foram encontrados 66.280 mil novos casos de Câncer de Mama, contribuindo com cerca de 30% do total de casos de câncer no Brasil.

Essa categoria de câncer, segundo pesquisas do INCA (2019), é o primeiro em questão de mortalidade, estimadas mais de 18 mil mortes em 2019 - o que representa 16,4% do total de mortes por todos os tipos da doença entre mulheres e ocupando o segundo lugar, quando consideramos os 2 sexos.A única região do país em que o câncer de mama não é o mais comum entre as mulheres é a norte, onde o de colo de útero ocupa a primeira posição.

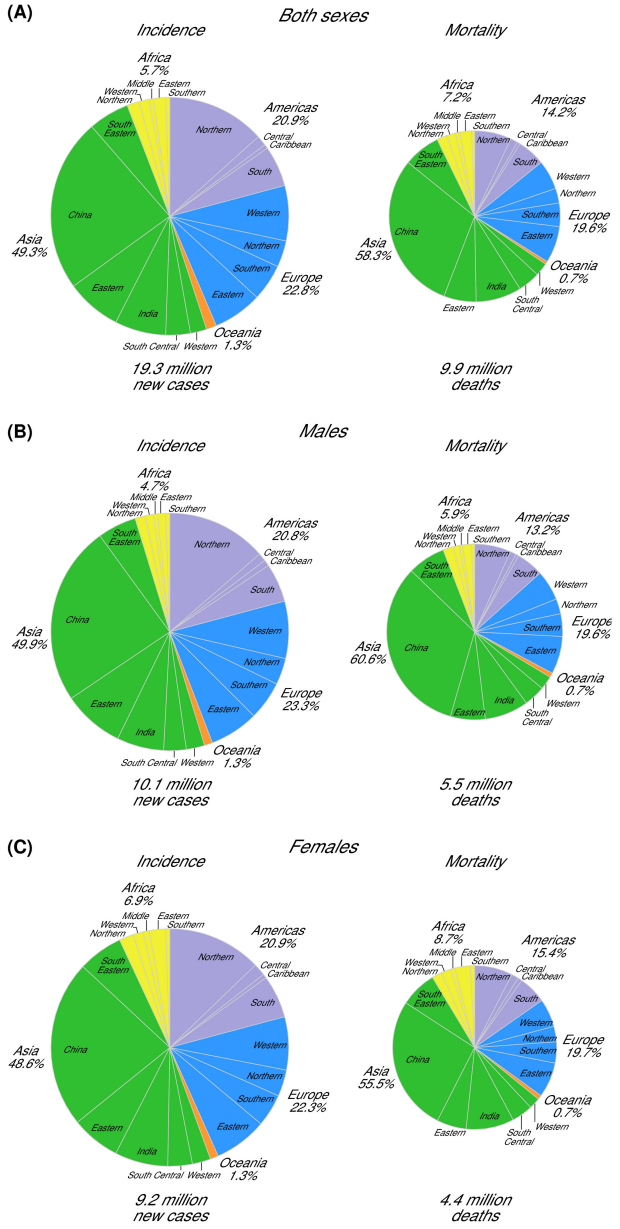
A incidência da doença aumenta em mulheres a partir dos 50 anos, onde poderemos encontrar uma incidência de 4 entre 5 mulheres.

Gráfico 1: Distribuição de casos e mortes para os 10 principais cânceres mais comuns em 2020 para (A) Ambos os sexos, (B) Homens e (C) Mulheres



Fonte: GLOBOCAN 2020

Gráfico 2: Distribuição de Casos e Mortes por Área Mundial em 2020 para (A)Ambos os Sexos,(B)Homens e (C)Mulheres



Fonte: GLOBOCAN 2020

1.2 O problema proposto

O problema proposto consiste na construção de um algoritmo para auxiliar, apenas auxiliar, os profissionais da saúde no diagnóstico de imagens, em particular a análise de imagem histopatológica do câncer de mama. Já que, nenhum algorítimo conseguirá substituir a decisão medica. O trabalho não deve ser visto como um modelo capaz de substituir uma consulta média ou tão menos a análise de um profissional qualificado.

Para facilitar o entendimento do problema e da solução a ser proposta utilizamos

a técnica do "5Ws", que consiste em responder as seguintes perguntas:

Why? Por que auxiliar médicos e profissionais na classificação do câncer a ser tratado e importante?

Segundo dados do Amarante (2021), quando o câncer é descoberto no seu início, a doença tem 95% de hipótese de cura. Segundo o Instituto Nacional de Câncer José Alencar Gomes da Silva (2021), o câncer de mama é responsável por cerca de 30% do total de casos de câncer no Brasil, representando cerca de 16,5% das mortes envolvendo câncer.

Segundo o artigo da Oncológica do Brasil Diagnóstico... (2021), um resultado falso positivo ou negativo interfere diretamente no sucesso do tratamento da doença, podendo gerar um desgaste nem um paciente saudável ou a perca da janela de tratamento em um paciente com câncer.

Who? De quem são os dados analisados?

Os dados utilizados nesse projeto foram gerados e disponibilizados para usos acadêmicos, desde que fossem referenciadas suas origens.

A primeira base de dados, *"Breast Cancer Histopathological Database"* (ou BreakHis), foi gerada para a publicação do artigo *Breast cancer histopathological image classification usingconvolutional neural networks.* (SPANHOL et al., 2016), distribuída pelos autores do artigo.

A segunda base de dados, *"A histopathological image dataset for grading breast invasiveductal carcinomas"*., foi gerada para a publicação do artigo de mesmo nome (BOLHASANI et al., 2020), distribuída pelos autores do artigo.

What? Qual o objetivo do projeto?

Já se sabe que o câncer de mama não é uma doença única, e sim um conjunto de enfermidades que têm em comum a presença da célula maligna originada na mama. Levantando essa ideia, o trabalho realizara duas classificações. Primeiramente ele irá classificar o câncer entre "Maligno"(M) ou "Benigno"(B) e caso ele seja classificado como "Maligno"(M) classificaremos esse câncer com as seguintes categorias: "carcinoma (DC)", "carcinoma lobular(CL)", "Carcinoma mucinoso (MC)"e "Carcinoma papilífero (CP)".

Where? O estudo onde as imagens foram coletadas, foi realizado no Brasil, pelo laboratório "PD Lab".(SPANHOL et al., 2016, tradução nossa)

When? Todas as imagens foram coletadas entre janeiro de 2014 e dezembro de 2014, durante a realização de um estudo clinico.(SPANHOL et al., 2016, tradução nossa)

2 COLETA DE DADOS

Neste trabalho optou-se por utilizar duas fontes de dados, ambas de estudos realizados e publicados em instituição com grande renome.

A primeira fonte de dados, descrita, é a "BreakHis". Um estudo publicado na IEEE (Institute of Electrical and Electronics Engineers) publicado no ano de 2016. Essa fonte de dados é constituída por imagens histopatológicas de cânceres de mama, subdivido entre "adenoses (A)", "Fibroadenoma (F)", "tumor Phyllodes (TP)"e "adenoma tubular (AT)"para câncer benigno e "carcinoma (DC)", "carcinoma lobular(CL)", "Carcinoma mucinoso (MC)"e "Carcinoma papilífero (CP)"para câncer maligno. Poderemos encontrar esse banco de dados em fontes diferentes, no Kaggle (https://www.kaggle.com/kritika397/breast-cancer-dataset-from-breakhis) e na pagina do artigo, no site, da Universidade Federal do Paraná (UFPR) no seguinte link (https://web.inf.ufpr.br/vri/databases/breast-cancer-histopathological-database-breakhis/)

A segunda fonte de dados, descrita, é a "A histopathological image dataset for grading breast invasive ductal carcinomas". Essa fonte contem, enumeras imagens histopatológicas, porem de cânceres benignos. Utilizarei essa fonte de dados, para complementação da "BreakHis", já que a nossa primeira fonte de dados possui um número consideravelmente superior do tipo maligno. E poderia gerar um desbalanço em nosso algorítimo.Essa fonte de dados, como posteriormente explicada, foi disponibilizada para livre uso na "Mendeley Data"

2.1 Breast Cancer Histopathological Database (BreakHis)

The Breast Cancer Histopathological Image Classification (BreakHis) é composto por 7,909 imagens microscópicas de cânceres de mama coletados de 82 pacientes diferentes fatores de ampliação (40X, 100X, 200X, e 400X). O dataset(conjunto de dados) contem 2480 canceres benignos e 5429 canceres malignos. A base de dados, foi construída em parceria entre UFPR (Universidade Federal do Paraná) e o Laboratório de PD - Anatomia Patológica e Citopatologia, Paraná, Brasil.(SPANHOL et al., 2016, tradução nossa)*

Inicialmente as imagens, foram importadas com os dados informativos em seu nome: método de produção da biopsia, classificação do tumor, subtipo do tumor, ampliação utilizada e a identificação do paciente. Como, por exemplo: "SOB-B-TA-14-4659-40-001.png"(SPANHOL et al., 2016, tradução nossa)†

Informação detalhada sobre a quantidade de "espécimes"por ampliação e categorias de cânceres estão presentes nas tabelas 1 e 2. Poderemos ver também uma amostra na figura 1.

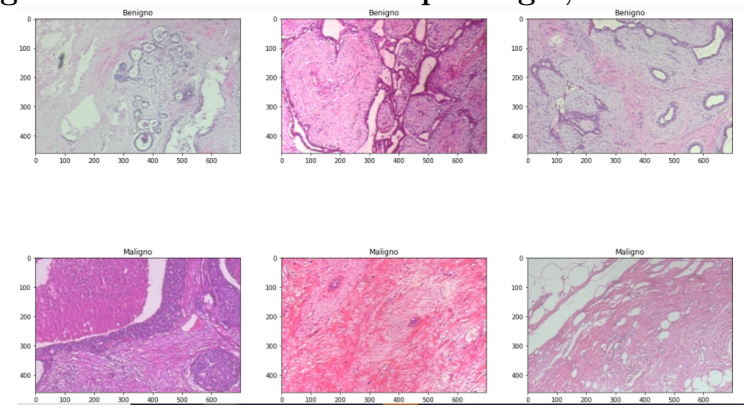
* The Breast Cancer Histopathological Image Classification (BreakHis) is composed of 9,109 microscopic images of breast tumor tissue collected from 82 patients using different magnifying factors (40X, 100X, 200X, and 400X). To date, it contains 2,480 benign and 5,429 malignant samples (700X460 pixels, 3-channel RGB, 8-bit depth in each channel, PNG format). This database has been built in collaboration with the PD Laboratory – Pathological Anatomy and Cytopathology, Parana, Brazil.

Tabela 1 – "BreakHis"resumo geral

Ampliação da imagem	Soma
40X	1995
100X	2081
200X	2013
400X	1820
Total	7909

Fonte: Spanhol et al., 2016

Figura 1 – Amostra de Histopatologia, na escala 40x



Fonte: Autor

Tabela 2 – "BreakHis"por categoria de câncer

Ampliação da imagem	Benigno	Maligno
40X	652	1370
100x	644	1437
200x	623	1390
400x	588	1232
Total	2480	5429

Fonte: Spanhol et al., 2016

2.2 A histopathological image dataset for grading breast invasive ductal carcinomas

A histopathological image dataset for grading breast invasive ductal carcinomas, é um "dataset"composto por 922 imagens de 124 pacientes com câncer de mama. Esse "dataset"foi publicado e disponibilizado em um ambiente virtual.diferentes fatores de ampliação (40X, 20X, 10X, e 4X)(BOLHASANI et al., 2020)

A marcação de espécimes neste conjunto de dados foi feita com base em uma classificação histológica de Bloom Richardson modificada. Este método de classificação avalia a quantidade de formação de túbulos, a extensão do pleomorfismo nuclear e a contagem mitótica (ou taxa mitótica).(BOLHASANI et al., 2020)

Cada imagem, ou dado, recebe uma pontuação de 1, 2 ou 3, e as pontuações são somadas para produzir uma nota. A pontuação mitótica é determinada pelo número de figuras mitóticas encontradas em 10 campos consecutivos de alta potência (HPF) na parte

mais mitoticamente ativa do tumor.(BOLHASANI et al., 2020)

Utilizaremos esse "dataset"como um enriquecimento da base de dados original, alcançando assim, uma melhor acurácia do nosso modelo. Obtendo uma menor saturação devido a pouca variedade de ampliações.

> *“ Os erros causados por dados inadequados são muito menores do que aqueles devido à falta total de dados.”* (Babbage Charles, Tradução Nossa)

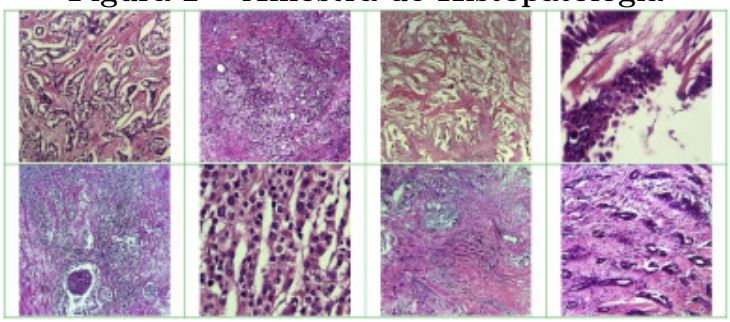
Informação detalhada sobre a quantidade de "espécimes"por ampliação e categorias de cânceres estão presentes nas tabelas 4 . Poderemos ver também duas amostras nas figuras 2 e 3.

Tabela 3 – Resumo geral

Ampliação da imagem	Soma
4X	160
10X	153
20X	155
40X	454
Total	922

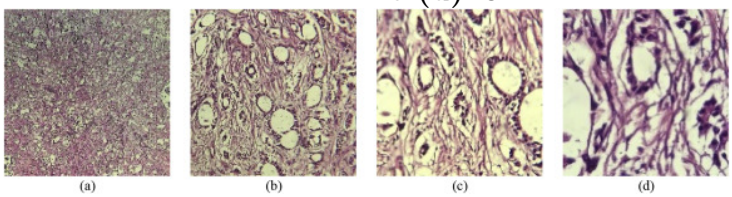
Fonte: Bolhasani et al., 2020

Figura 2 – Amostra de Histopatologia



Fonte: Bolhasani et al., 2020

Figura 3 – Amostra de Histopatologia nas amplificações (a)4x, (b)10x, (c)20x e (d)40x



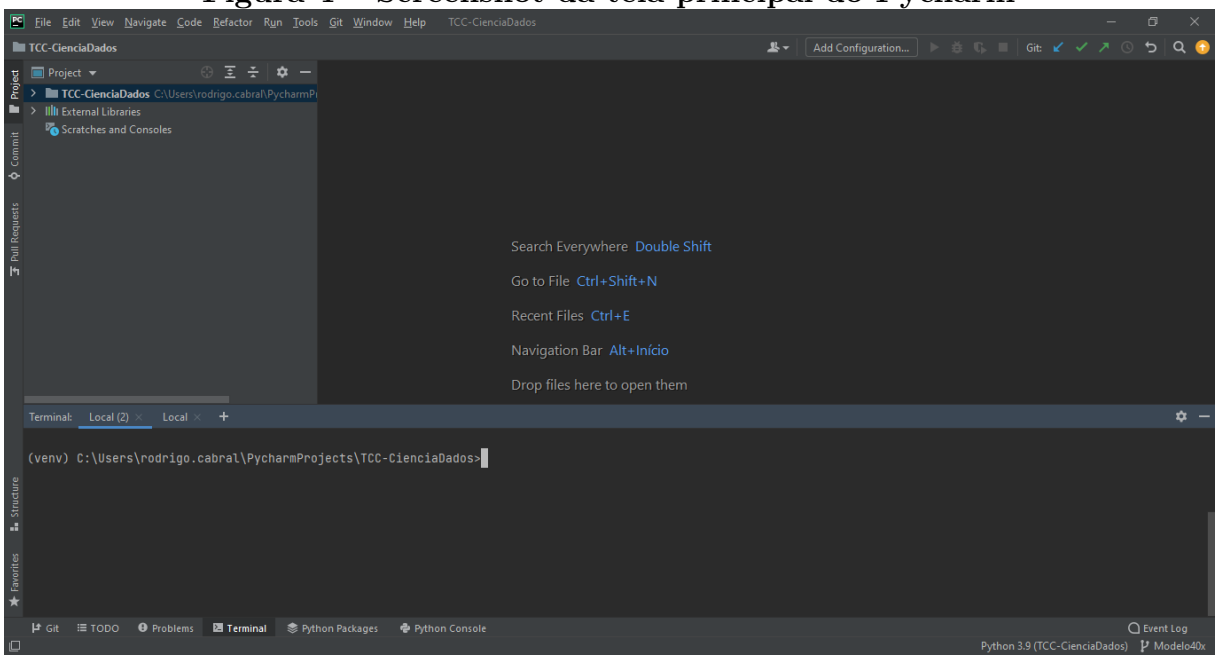
Fonte: Bolhasani et al., 2020

3 PROCESSAMENTO/TRATAMENTO DE DADOS

3.1 Ferramentas utilizadas

Inicialmente utilizamos a para desenvolvimento dos scripts em Python, o PyCharm Community Edition na versão 2021.1.3 (Figura 4) rodando o Python na versão Python 3.9.5. Utilizando também no projeto o Jupyter Notebook na versão 3.0.16. Na linguagem utilizou-se algumas "bibliotecas externas", a mais importante delas: TensorFlow ma versão 2.5.0.

Figura 4 – Screenshot da tela principal do Pycharm



Fonte: Autor

Com o decorrer do projeto, foi estudada a ideia de migrarmos o projeto para o ambiente *"Google Colab"*. O *"Colab"*(ou *"Colaboratory"*) é um sistema oferecido pela Google. No ambiente *"Colab"* são oferecidos, "CPUs","GPUs"e até "TPUs"gratuitos.

Por seu processamento, ser efetuado exclusivamente em ambiente virtual, tornando viável o treinamento paralelamente e também podendo realizar de qualquer máquina com acesso à rede.

3.2 Tratamento das bases de dados

3.2.1 Tratamento da base de dados BreakHis

Após coletados os dados e "descompactados"em máquina própria, a primeira base de dados disponibilizada estava dividida em primeiro nível por grau de amplitude e em um segundo nível, entre benigno e maligno. Primeiramente realizarei uma clusterização das imagens, separando-as por suas respectivas classes. Para isso, precisaremos analisar a composição dos dados, demonstraremos um exemplo aqui, utilizando as imagens com fator de ampliação de 40x.Será mantido um *buckup* da estrutura original, para evitar uma

perda de tempo desnecessária. Separando as imagens dessa forma, poderemos ter uma noção real do nosso *dataset*, sabermos a quantidade real de cada classe e a quantidade em relação aos outros. Para um tratamento mais fácil das imagens, será necessário saber como elas estão dispostas no repositório. Realizaremos isso usando o código da imagem 5.

Figura 5 – Código para mostrar nomes dos arquivos (Jupyter notebook)

```
import os

def mostrar_arquivos(path):
    for p, _, files in os.walk(os.path.abspath(path)):
        for file in files:
            print(os.path.join(p, file))
```

Fonte: Autor

Apos listar os arquivos, foi decido mudar suas nomenclaturas, excluindo valores que possa, se tornar desnecessário e também obsoletos.Podemos ver como é composto os nomes das imagens, como podemos ver na imagem 6. Por exemplo: SOB-B-F-14-14134E-100-004.png, é a quarta imagem, com ampliação de 100X, com a categoria de tumor "benigno", com a classificação de "Fibroadenoma", original do slide 14-14134 sendo coletado utilizando a técnica SOB.

Figura 6 – Amostra com nomes de imagens (Jupyter notebook)

```
SOB_B_A-14-22549AB-40-002.png
SOB_B_A-14-22549AB-40-003.png
SOB_B_A-14-22549AB-40-004.png
SOB_B_A-14-22549AB-40-005.png
SOB_B_A-14-22549AB-40-006.png
SOB_B_A-14-22549AB-40-007.png
SOB_B_A-14-22549AB-40-008.png
SOB_B_A-14-22549AB-40-009.png
```

Fonte: Autor

Como observado na imagem 6, o nome do arquivo leva a seguinte composição:

Procedimento de biopsia : SOB;

Categoria do tumor : M(Maligno) ou B(Benigno);

Classificação do tumor : A|F|TP|AT|DC|CL|MC|CP;*

Ano : Ano do exame;

ID do exame : Composta por números;

Grau de Ampliação : 40x|100x|200x|400x;

SEQ : Identificação do exame;

Em outro momento, foi realizada nova análise das imagens, visando eliminar esses dados, foi utilizado o algorítimo contido na imagem 7 , para a transformação dos nomes das figuras. Como podemos observar os nomes serão compostos por um número e sua categoria. Como demonstrado nas imagens 8,9,10 e 39.†

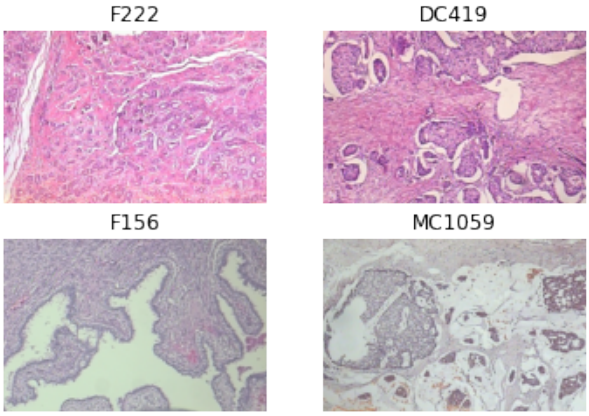
* "adenoses (A)", "Fibroadenoma (F)", "tumor Phyllodes (TP)"e "adenoma tubu-lar (AT)"para câncer benigno e "carcinoma (DC)", "carcinoma lobular(CL)", "Carcinomamucinoso (MC)"e "Carcinoma papilífero (CP)"para câncer maligno. † Todas as amostras foram tiradas aleatoriamente do *Dataset*

Figura 7 – Função para mudar o nome das imagens (Jupyter notebook)

```
import os

dicionario_cancer = {'SOB_B_A': 'A', 'SOB_B_F':'F','SOB_B_PT':'PT','SOB_B_TA':'TA','SOB_M_DC': 'DC',
                     'SOB_M_LC':'LC','SOB_M_MC':'MC','SOB_M_PC':'PC'}

def mudar_label(path):
    x = 0
    for p, _, files in os.walk(os.path.abspath(path)):
        for file in files:
            x += 1
            os.rename(p+"\\"+file,p+"\\"+dicionario_cancer.get(file.split('-')[0], 'Contato não encontrado')+f'{x}.png')
```

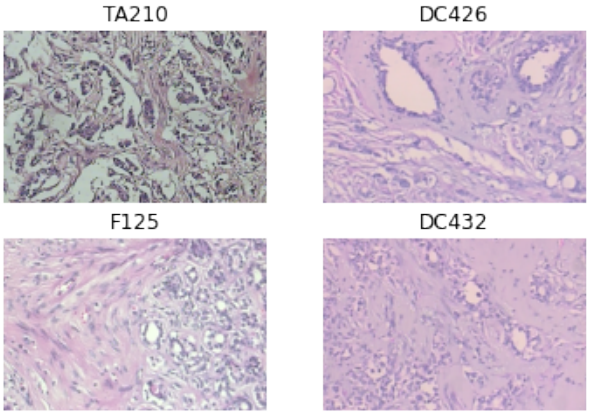
Fonte: Autor

Figura 8 – Amostra com novos nomes, ampliação 40x



Fonte: Autor

Figura 9 – Amostra com novos nomes, ampliação 100x

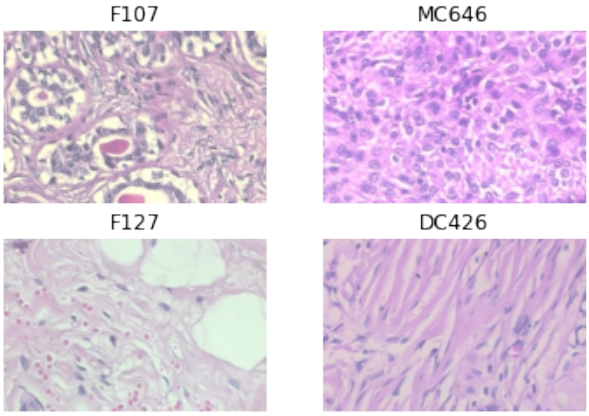


Fonte: Autor

As imagens originalmente, possuem a seguinte resolução 460 x 700 x 3, ou seja, 460 de altura, por 700 de largura e com 3 canais de cores, ou seja, colorida. Após analisar essa informação, foi decidido que seria ideal, a diminuição dessas resoluções. Além de uma melhoria no processamento, se enquadraria nas arquiteturas utilizadas.

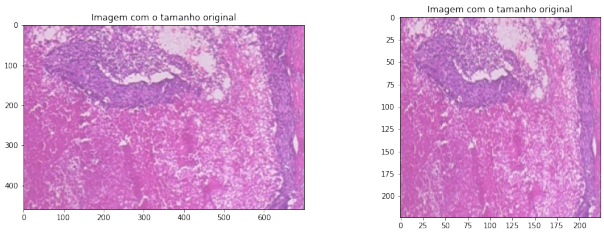
Primeiro passo será remodelar as imagens nas seguintes medidas que avaliarei conforme o andamento do nosso algoritmo, tentando balancear a taxa entre Aprendizagem x Velocidade. A primeira proporção utilizada será 224 x 224 x 3. Agora nos teremos as seguintes proporções: 224 de altura, por 224 de largura e 3 de "channels"(colorida).

Figura 10 – Amostra com novos nomes, ampliação 200x



Fonte: Autor

Figura 11 – Amostra com novos tamanhos das imagens



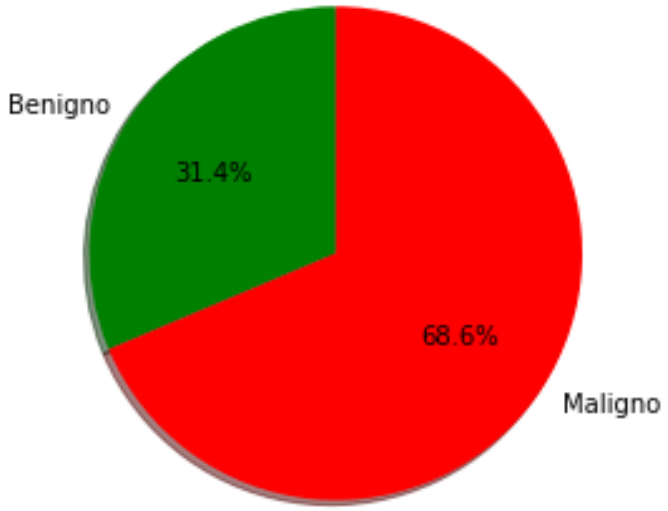
Fonte: Autor

3.2.2 Tratamento da base de dados Breast invasive ductal car-cinomas

Após coletados os dados e "descompactados"em máquina própria, a segunda base de dados disponibilizada também estava dividia, porem, esse nosso "dataset"vem em primeiro grau.dividida em 3 grupos, de acordo com sua "pontuação mitótica". A "pontuação mitótica"não será usada diretamente na em nosso algorítimo. Em um segundo nível, vemos que nossa base de dados é dividida em seus graus de amplitude.

O motivo para o uso da nova fonte de dados, é uma tentativa de realização de um balanceamento natural da base de dados. A Base de dados "BreakHis", é encontrada numa proporção aproximada de 69% para maligno enquanto apenas 31% para benignos. Como podemos ver nos gráficos 3.

Gráfico 3: Distribuição de categorias de cancer



Fonte: Autor

Utilizando o algorítimo da figura 5, conseguiremos ver a nomenclatura aplicada nessas imagens, também seguem um padrão logico:

Numero Simples : número inteiro;

Categoria do tumor : BC(Benigno)-IDC;

Numero do arquivo : número de arquivos de patologia;

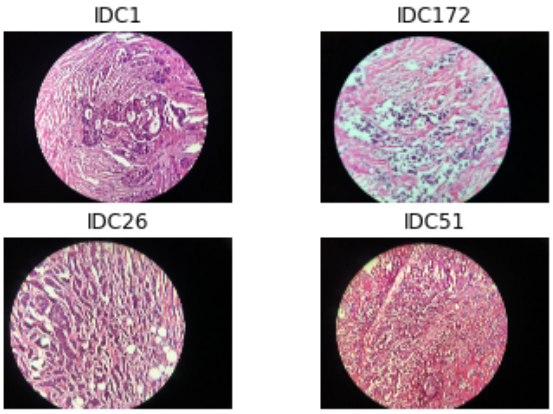
Grau de Ampliação : 40x|100x|200x|400x;

pontuação mitótica : número de figuras mitóticas encontradas em 10 campos consecutivos de alta potência (HPF)*;

Uma das principais dificuldade que encontraremos ao usar essa segunda base é a formatação de cada imagem. Como podemos ver na imagem 41 as imagens são formatadas em forma circular, diferentemente da nossa primeira base que são de formatos quadriculares. Essa diferença pode se tornar uma pode vir a se tornar uma ajuda, pois dados de fontes diferentes, sempre tenderam a vir em formatos diferentes, sendo assim, a BIAS de nossa Rede Neural, conseguirá se sair bem com formatações não padronizadas de imagens.

* A pontuação mitótica é determinada pelo número de figuras mitóticas encontradas em 10 campos consecutivos de alta potência (HPF) na parte mais mitoticamente ativa do tumor. Escores de 3 a 5, bem diferenciados (grau I); escores de 6 a 7, moderadamente diferenciados (grau II); e pontuações de 8 a 9, pouco diferenciadas (grau III)

Figura 12 – Amostra da base dados Breast invasive ductal carcinomas (Jupyter notebook)



Fonte: Autor

As imagens originalmente, possuem a seguinte resolução 3024 x 4032 x 3, ou seja, 3024 de largura, por 4032 de altura e 3 de "channels". Após analisar as imagens, percebi a existência de informações adicionais que não são necessárias para nosso treinamento. Por esse motivo, decidi realizar cortes, centrais nas imagens para melhor aproveitamento. Utilizando a parte do código presente na figura **??**.

Segundo passo será remodelar as imagens em para a proporção utilizada, pelo nosso modelo, que será 224 x 224 x 3, ou seja, 224 de altura, por 224 de largura com 3 canais de cores.

4 ANÁLISE E EXPLORAÇÃO DOS DADOS

> *"A informação é o petróleo do século XXI e a análise de dados é o motor de combustão."** - (Peter Sondergaard, Tradução Nossa)

Para a realização da análise e exploração de dados será utilizado *scripts* em Python e com apresentação no Jupyter Notebook. Os principais módulos utilizados serão: "matplotlib", "seaborn", "cv2", "numpy", "os"e "Pandas".

matplotlib e seaborn : utilizadas para a criação de gráficos;

cv2 : utilizado para leitura e manipulação das imagens;

numpy : utilizado para manipulações dos vetores de imagens;

os : biblioteca natural do python, utilizada para manipular "Paths".

Pandas : uma biblioteca de software criada para a linguagem Python para manipulação e análise de dados e dataframes.

Primeiramente faremos a importação das bibliotecas utilizadas, como podemos ver no código descrito na figura 13.

Figura 13 – Código de importação dos módulos (Google Colab)

```
import seaborn as sns
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Fonte: Autor

Em seguida, realizarei a criação de um dataframe contendo informações primordiais do nosso dataset, informações essas que serão usadas para uma análise exploratória do nosso DataSet. O nosso dataframe terá as seguintes colunas: "FikePath", "Type", "classes","Magnification"e o "Dataset".

FikePath : Caminho onde a imagem foi salva;

Type : Categoria do câncer [B|M];

classes : Classe do tumor [A|F|TP|AT|DC|CL|MC|CP];†

Magnification : grau de ampliação [40|100|200|400];

Dataset : DataSet pertencente "Teste(Te)"ou "Treino(Tr)".

---

* "Information is the oil of the 21st century, and analytics is the combustion engine." † "adenoses (A)", "Fibroadenoma (F)", "tumor Phyllodes (TP)"e "adenoma tubu-lar (AT)"para câncer benigno e "carcinoma (DC)", "carcinoma lobular(CL)", "Carcinomamucinoso (MC)"e "Carcinoma papilífero (CP)"para câncer maligno.

Serão utilizados para criação do dataframe as seguintes funções apresentadas nas figuras 14, 42, 43,44 e 45.

Figura 14 – Código de criação do *"DataFrame"* (Google Colab)

```
def montar_data_frame(path):
    nome_file, file_path = montar_nome(path)
    classes = selecionar_classe(nome_file)
    tipos = pegar_tipos(file_path)
    ampliacaos = pegar_ampliacao(file_path)
    origem = pegar_origem(file_path)
    data = {'FikePath': file_path, 'Type': tipos, 'classes':classes,'Magnification':ampliacaos, 'DataSet':origem}
    df = pd.DataFrame.from_dict(data, orient='index')
    df = df.transpose()
    return df
```

Fonte: Autor

Podemos ver um exemplo do *"DataFrame"* na figura 15.

Figura 15 – Amostra do *"DataFrame"* criado (Google Colab)

	FikePath	Type	classes	Magnification	DataSet
2821	/content/drive/MyDrive/TCC/Imagens/200x/200x_t...	M	DC	200x	Te
1676	/content/drive/MyDrive/TCC/Imagens/400x/400x_t...	B	DCI	400x	Te
825	/content/drive/MyDrive/TCC/Imagens/400x/400x_t...	M	DC	400x	Te
1373	/content/drive/MyDrive/TCC/Imagens/400x/400x_t...	B	PT	400x	Te
375	/content/drive/MyDrive/TCC/Imagens/400x/400x_T...	M	PC	400x	Tr
2663	/content/drive/MyDrive/TCC/Imagens/200x/200x_t...	M	DC	200x	Te
2801	/content/drive/MyDrive/TCC/Imagens/200x/200x_t...	M	DC	200x	Te
2840	/content/drive/MyDrive/TCC/Imagens/200x/200x_t...	M	DC	200x	Te
979	/content/drive/MyDrive/TCC/Imagens/400x/400x_t...	M	DC	400x	Te
3109	/content/drive/MyDrive/TCC/Imagens/200x/200x_t...	B	F	200x	Te

Fonte: Autor

Na figura 16, podemos confirmar que o nosso dataframe segue a composição planejada.

Figura 16 – Valores únicos de cada coluna (Google Colab)

```
[5] df_geral['Type'].unique()
    array(['M', 'B'], dtype=object)

[6] df_geral['classes'].unique()
    array(['DC', 'LC', 'MC', 'PC', 'A', 'F', 'PT', 'TA', 'DCI'], dtype=object)

[7] df_geral['Magnification'].unique()
    array(['400x', '200x'], dtype=object)

[8] df_geral['DataSet'].unique()
    array(['Tr', 'Te'], dtype=object)
```

Fonte: Autor

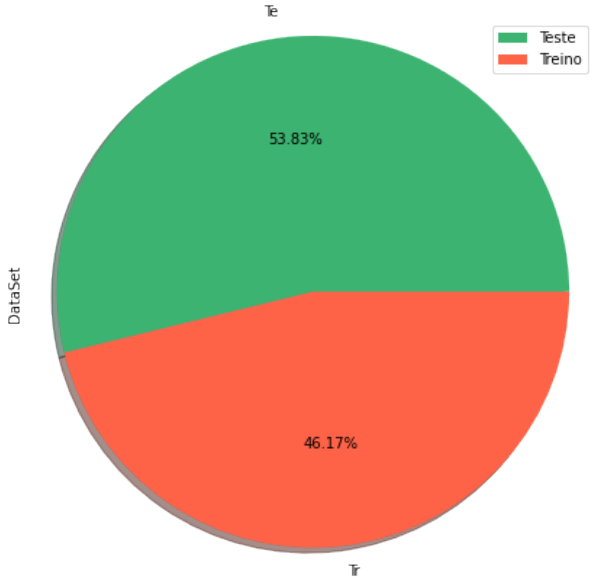
Essa abordagem se mostrou suficiente para garantirmos um estudo mais detalhado do processo de exploração de dados.O primeiro passo na análise exploratória de dados foi entender a composição do nosso *"dataframe"* como mostrado na figura 15 e principalmente na figura 16.

Observamos também, que nosso *"dataframe"* segue o padrão descrito na nossa lista, e com isso poderemos primeiro analisar a divisão do nosso *"dataset"* entre dados

que usaremos para Treinamento e dados que utilizaremos posteriormente para testarmos nosso algorítimo.

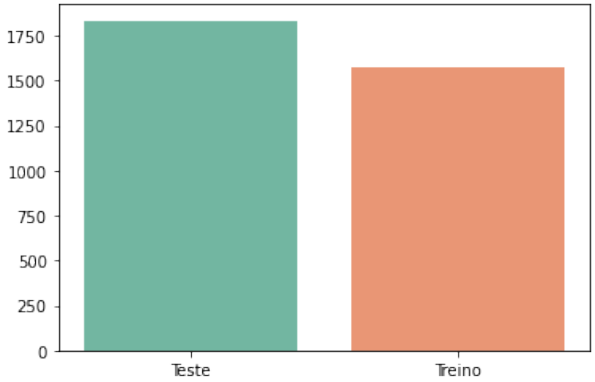
Nosso *"dataset"* segue uma divisão quase que igual entre dados de treino e dados utilizados para teste. Seguindo uma proporção de 47% de para treino e um volume um pouco maior de teste, sendo ele equivalente a 53% do nosso dataset, como demonstrado nos gráficos 4 e 5

Gráfico 4: Distribuição de Treino e Teste no *Dataset*



Fonte: Autor

Gráfico 5: Quantidade de dados entre Treino e Teste no *Dataset*

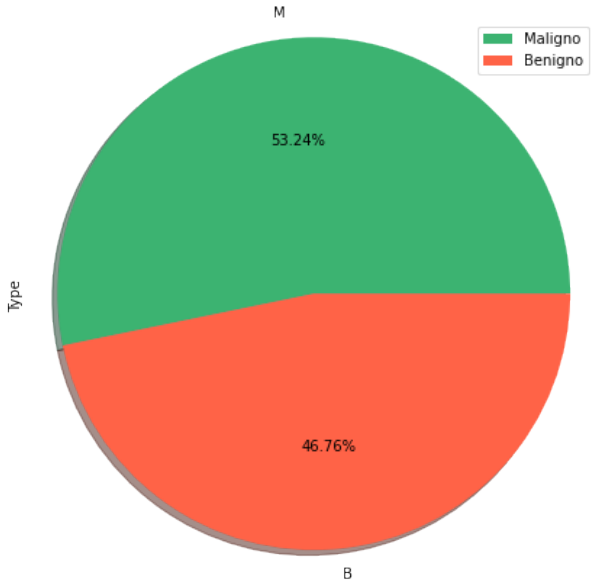


Fonte: Autor

Observamos que antes do *merge* entre as bases de dados *"BreakHis"* com a *"Breast invasive ductal carcinomas"*, era possível ver uma proporção de 31,4% para cânceres Benignos contra 68,9% para cânceres malignos. Com o *merge* das duas bases de dados, conseguimos diminuir um pouco essa proporção, agora encontramos a seguinte proporção

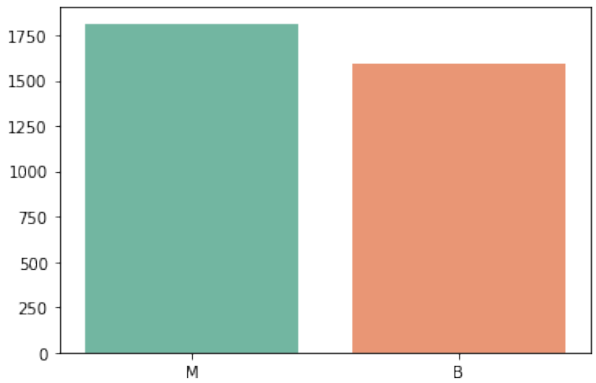
46,17% para cânceres Benignos contra 53,43% para cânceres malignos, como observado nos gráficos 6 e 7.

Gráfico 6: Distribuição de Beigno e Maligno no Dataset



Fonte: Autor

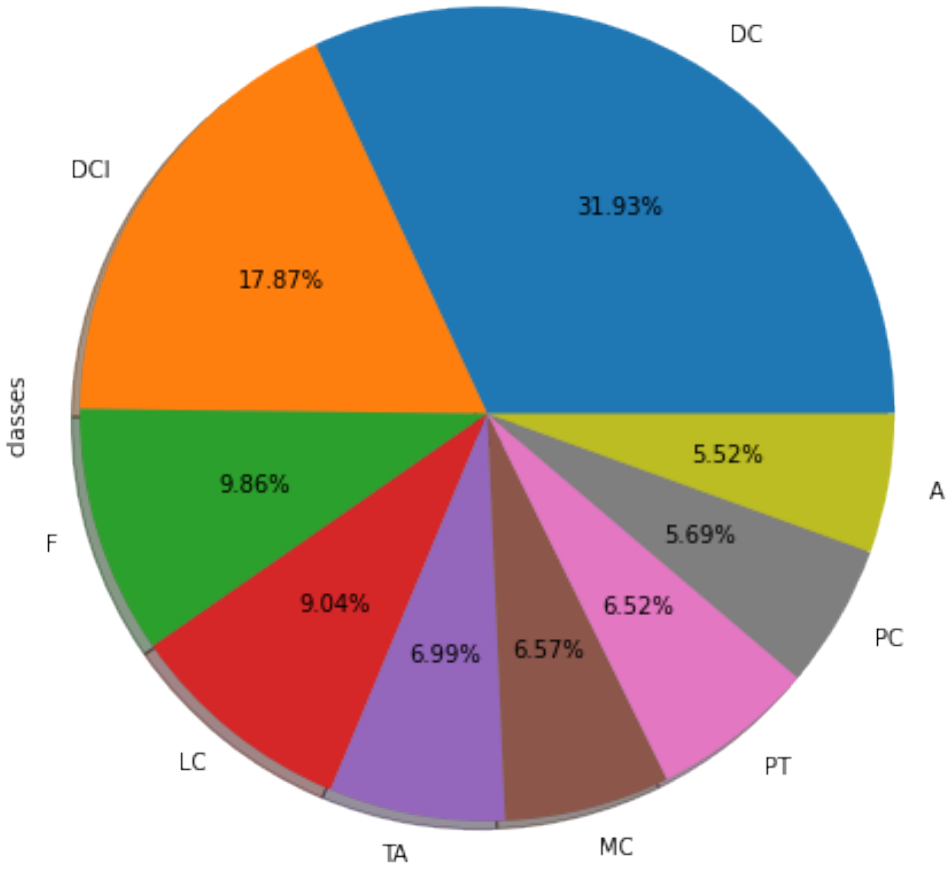
Gráfico 7: Quantidade de dados entre Beigno e Maligno no Dataset



Fonte: Autor

Ao observamos o gráfico 8, podemos perceber a distribuição em porcentagem das classes de cânceres conforme. Podemos ver também, que contemos uma quantidade superior do câncer que contem a classe "carcinoma (DC)". Estando em maior quantidade na base de dados, de ampliação "200x"e "400x".

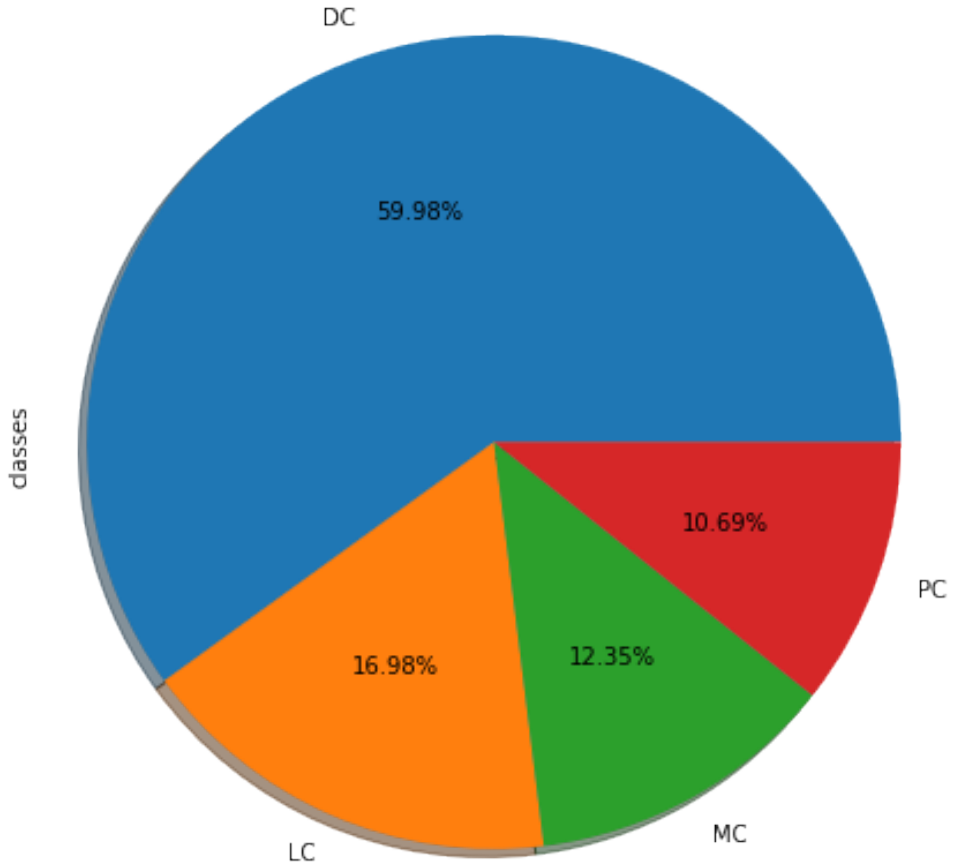
Gráfico 8: Distribuição em porcentagens das classes



Fonte: Autor

Podemos ver também essa superioridade percentual no gráfico 9, quando olhamos apenas as classes que iremos de fato prever.

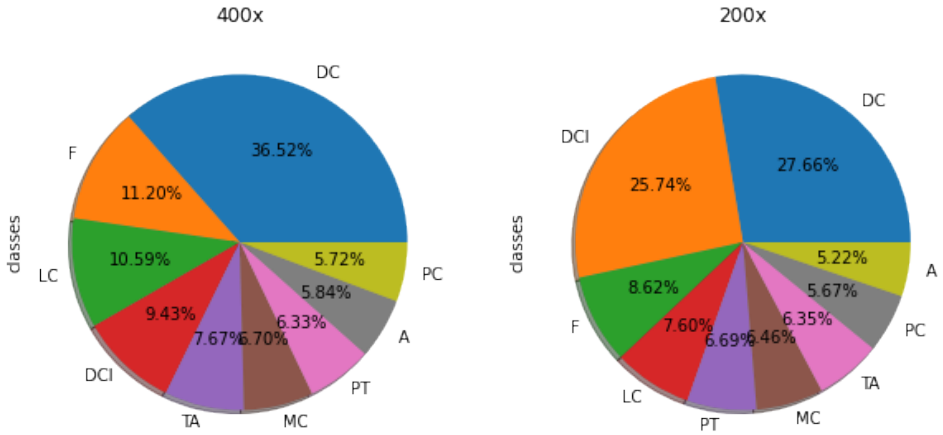
Gráfico 9: Distribuição em porcentagens das classes referente aos tipos Malignos



Fonte: Autor

Quando observamos a nossa massa de dados, percebemos um leve desbalanceamento para a classe "carcinoma (DC)"como visto no gráfico 10. O balanceamento da nossa base de dados será feita com o treinamento da nossa rede neural. O balanceamento será feito pesando no mínimo de perda de dados e o máximo de desempenho da nossa rede.

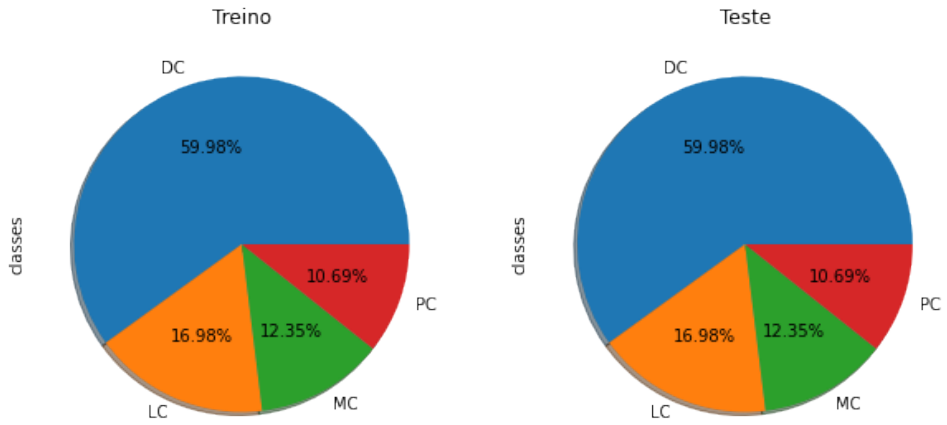
Gráfico 10: Distribuição em valores percentuais das classes por *"Magnification"*



Fonte: Autor

Podemos ver também uma proporção parecida entre os dados de treino e de teste. Como podemos ver no gráfico 11

Gráfico 11: Distribuição em valores percentuais das classes base de treino e teste



Fonte: Autor

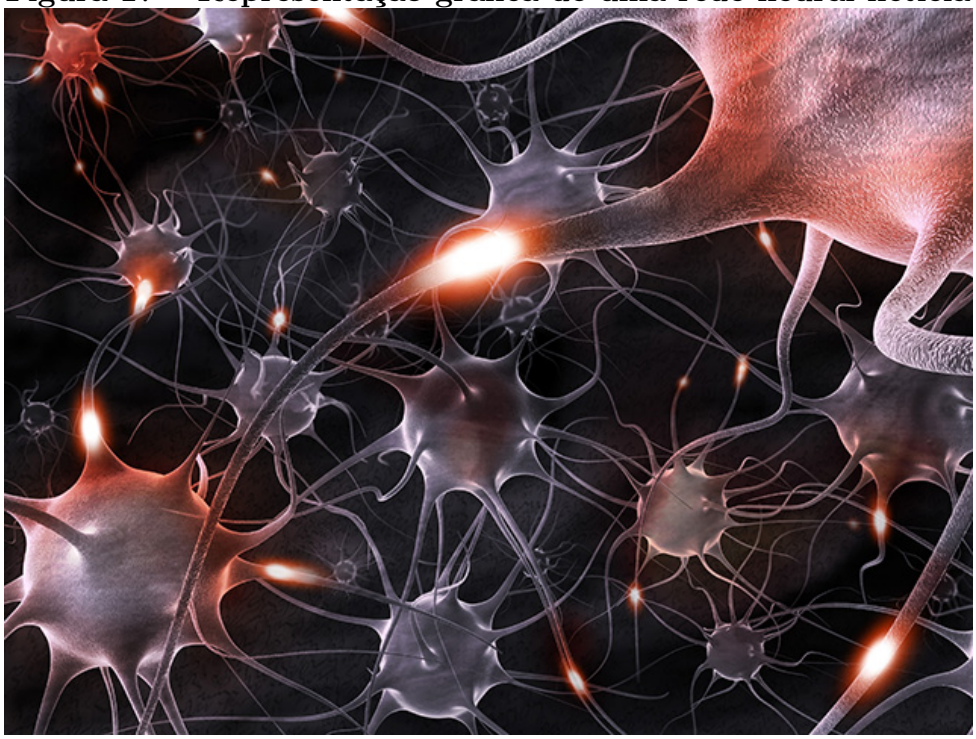
5 CRIAÇÃO DE MODELOS DE MACHINE LEARNING

Neste capítulo será desenvolvido o modelo de "Rede Neural Convolucional"o qual iremos realizar classificação de nossa base de dados. Primeiro teremos alguns conceitos teóricos sobre Aprendizagem profunda(Deep learning) e Rede neural convolucional (Convolutional neural network). As apresentações teóricas serão necessárias, para uma boa compreensão deste trabalho. Primeiro irei explicar o que seria uma rede neural natural que serviu de insparação para a tecnica que utilizaremos.

5.1 Rede Neural Natural (R.N.A.)

No nosso dia a dia, conseguimos realizar ações, como andar, comer, conversar, dirigir e etc.. A capacidade de realizarmos tais ações está ligada a presença de um rede neural. Essa composição, começou a ser estudada em 1902. Podemos ver um exemplo gráfica de uma rede neural na figura 17

Figura 17 – Representação gráfica de uma rede neural fictícia



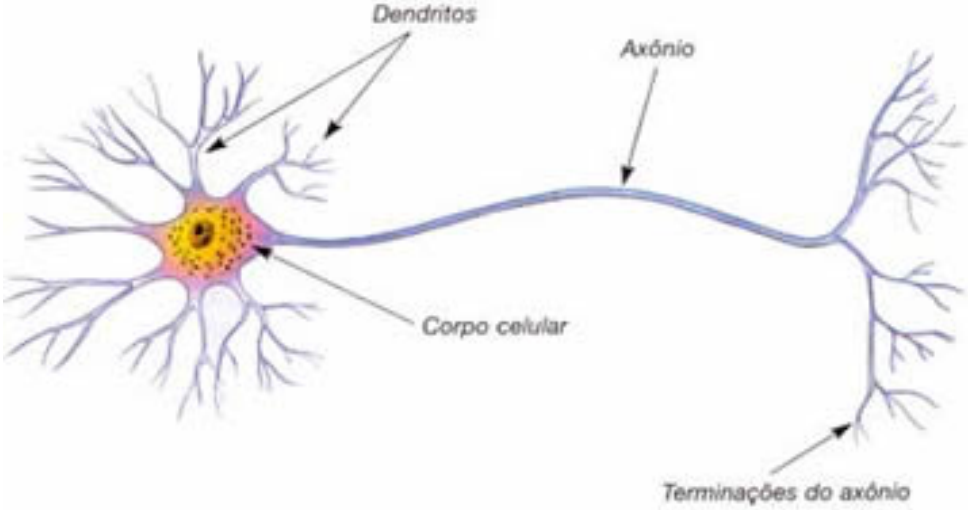
Fonte: Anônimo, 2014

Uma rede neural natural é composta por milhares de células nervosas, denominados neurônios. São células do sistema nervoso que estão relacionadas com a propagação de impulsos elétricos. Um neurônio é considerado a unidade básica desse sistema.(ANôNIMO, 2021).

Os Neurônios são compostos por 3 partes básicas: os dendritos, o axônio e o corpo celular (ANôNIMO, 2021). Como podemos ver representado na figura 18

Dendritos são as terminações dos neurônios, finos e geralmente ramificados, que

Figura 18 – Representação gráfica de um neurônio fictício



Fonte: Anônimo, 2021

conduzem os estímulos captados do ambiente ou de outras células em direção ao corpo celular.(ANôNIMO, 2021)

Axônio é um prolongamento fino, geralmente maior que os dendritos, que possuem a função de transmitir para outras células os impulsos nervosos(impulsos elétricos) provenientes do corpo celular.(ANôNIMO, 2021)

O corpo celular, é a principal parte e mais volumosa do neurônio. Nela é localizada o núcleo e a maioria das estruturas citoplasmáticas.(ANôNIMO, 2021)

No ano de 1949, Donald Hebb escreveu uma obra que apontou que caminhos neurais são fortalecidos cada vez que são usados, um conceito essencial sobre a forma que os seres humanos aprendem. Segundo Donald Hebb, Se dois neurônios disparem em simultâneo, a conexão entre eles é melhorada.(DATA SCIENCE ACADEMY, 2021)

5.2 Rede Neural Artificial (R.N.A.)

Inspirado na arquitetura do cérebro humano, as redes neurais exibem algumas características do cérebro humano, como a aprendizagem de padrões complexos e a generalização da informação aprendida. E assim como as redes neurais naturais as redes neurais artificiais, funcionam de forma parecida, ou seja, há um recebimento de estímulos, o processamento de uma função e o retorno determinado.

Em 1943, o matemático Walter Pitts e neurofisiologista Warren McCulloch (MCCULLOC; PITTS, 1943), tiveram a ideia de criar o artigo *"A Logical Calculus of the Ideas Immament in Nervous Activity"*, onde eles explicariam uma rede neural simples com a ajuda de circuitos elétricos .Essa ideia deu origem a criação do algoritmo de lógica de limiar (threshold logic).(DATA SCIENCE ACADEMY, 2021)

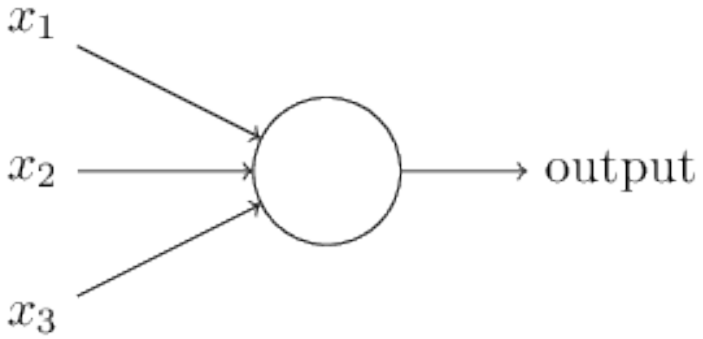
Em 1959 inciou-se os estudos sobre a aprendimento de uma R.N.A. feito por

Donald Hebb (HEBB, 1950), no artigo de nome *"The organization of behavior: A neuropsychological approach"*, nesse artigo, segundo (HEBB, 1950) a aprendizagem de rede é realizada através da inclusão de pesos em cada conexão dos neurônios.

Para Frank Rosenblatt (1958) os modelos de arquiteturas onde pesos sofriam ajuste em seus valores, poderiam ser treinados para classificação de padrões. Esse modelo de arquitetura ficou conhecido como *"perceptron"*.

O perceptron, em um resumo geral é equivalente ao neurônio. Assim como um neurônio, o perceptron possui uma estrutura um pouco mais simples, porem com 3 camadas: entrada equivalente ao Dendrito, oculta(ou intermediário) equivalente ao corpo celular e a saída que seria equivalente ao Axônio. Podemos ver um exemplo gráfico nas figuras 19.

Figura 19 – Representação gráfica de perceptron



Fonte: Data Science Academy, 2021

Um perceptron é um modelo matemático que recebem diversas entras, como no exemplo da figura 19, vemos um perceptron de 3 entradas: x1,x2,x3 que tem como resposta uma única saída binaria.(DATA SCIENCE ACADEMY, 2021)

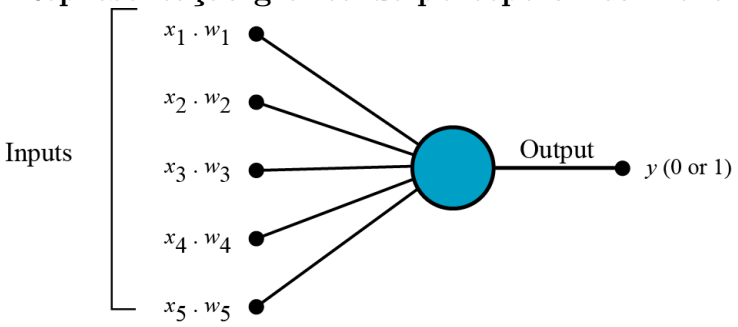
Rosenblatt (1958) propôs uma regra simples para calcular essa saída. Ele introduziu a ideia de pesos, w1,w2, .., wx . Os pesos são compostos de números reais. A saída de um neurônio, 0 ou 1, é determinada pela soma ponderada, $\sum$**jwjxj** , menor ou maior do que algum valor limiar(*threshold logi*).O threshold é um número real que é um parâmetro do neurônio.(DATA SCIENCE ACADEMY, 2021)

Podemos ver essa ideia em forma algébrica mais precisamente na seguinte função :

$$\text{output} = \begin{cases} 0 & \text{se } \sum \text{jwjxj} \leq \text{threshold} \\ 1 & \text{senão} \end{cases}$$

Como vemos, um Perceptron segue o modelo *"feed-forward"*, ou seja, ele é unilateral. As entradas são enviadas para o neuronio , processadas e resultam em uma saída específica, como podemos ver nas figuras 19 e com um grau maior de explicação na figura 20, onde podemos ver as entradas já multiplicadas pelos pesos, pesos esses que podem ser predefinidos ou definidos conforme o treinamento e podemos também ver a saída, que sempre será 0 ou 1.(DATA SCIENCE ACADEMY, 2021)

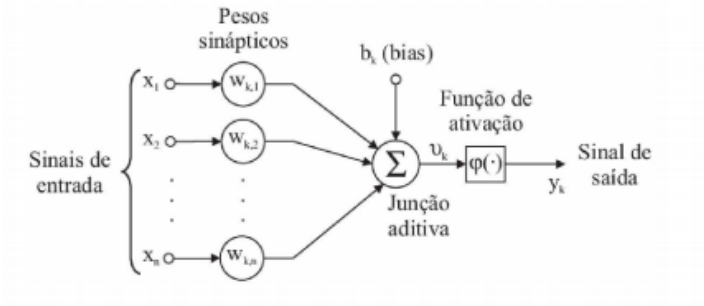
Figura 20 – Representação gráfica de perceptron com entrada e saída



Fonte: Data Science Academy, 2021

Podemos ver na figura 21, o modelo de um neurônio(Perceptron) de McCulloch e Pitts, considerado a arquitetura de uma "RNA" mais simples.

Figura 21 – Modelo de Perceptron não linear



Fonte: HAYKIN, 2007

Como visto na figura 21,um perceptron é composto por alguns elementos, já descritos no trabalho e também : Sinapse ou Peso (w), função de ativação e em alguns casos, é o *bias*.

Os Pesos(ou Sinapses) são usados para guardar o conhecimento. Esses valores são atribuídos e multiplicados pelos valores de entrada, para melhor ajuste ao desejado.

A função de ativação é responsável pela transformação não linear ao longo do sinal de entrada. Esta saída já transformada é então enviada para a camada posterior como entrada. Quando não implementamos uma função e ativação, a rede neural pode apenas obter resultados lineares.

Bias exerce a funcionalidade de um "interceptor"adicionado em uma equação linear. Ele é um parâmetro adicionada na Rede neural cuja função é ajustar a saída junto a fonte ponderada de entradas do neurônio.

A parti da criação do perceptron foram criadas as redes neurais artificiais (RNA).

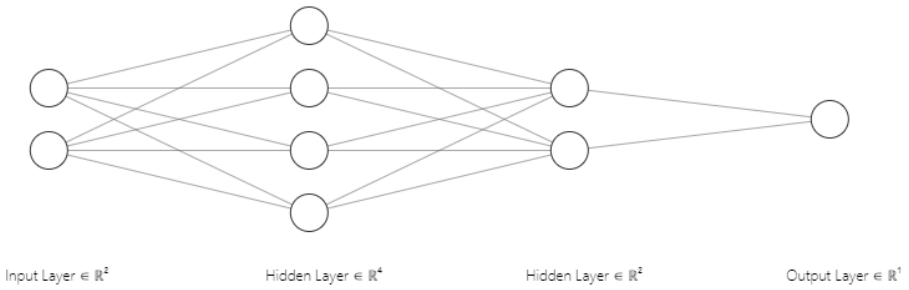
Que são um conjunto de perceptrons que seguindo a ideia das redes neurais naturais, são conectadas entre si.

5.2.1 Rede Neural Artificial Multicamadas ou MLPs

Rede Neural Artificial Multicamadas(ou Perceptron multicamadas) é uma arquitetura onde os neurônios são organizados em duas ou mais camadas.Sendo elas : "Camada de entrada"(*input layer*), onde os seus neurônios recebem o nome de "Neurônios de Entrada". A camada do oculta (hidden layers), cada neurônio dessa camada é conectado com todos os outros da próxima camada. Ela também tem a função de executar cálculos mais complexos e auxiliar a última camada. A terceira e ultima classe é a "Camada de saída"(*output layer*), que possuem os neurônios de saída.

Podemos ver na figura 24 um exemplo de Rede Neural de Múltiplas Camadas.

Figura 22 – Diagrama de uma MLPs



Fonte: Autor usando http://alexlenail.me/

Quanto a forma com que os neurônios artificiais se conectam, redes neurais podem ser classificadas em 3 categorias:

Redes neurais feedforward : Esse é o tipo mais comum de arquitetura em aplicações praticas. Nessa arquitetura a primeira camada é chamada "Entrada"e a última camada é chamada "Saida". Se houver mais de uma camada oculta, ela é chamada redes neurais profundas(ou Deep Learning). As atividades dos "neurônios"são uma função não-linear das atividades da camada anterior;

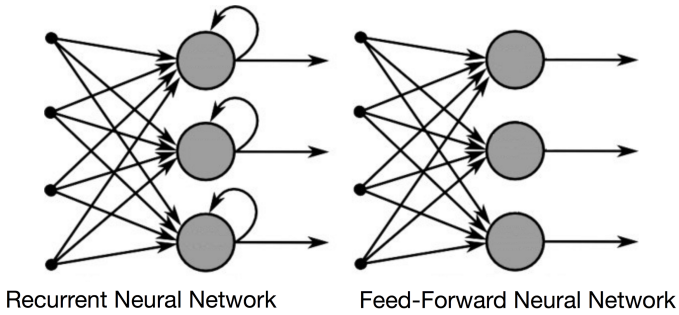
Redes neurais recorrentes : As redes recorrentes tomam como entrada, não somente os valores e os pesos, mas sim o que perceberam anteriormente no tempo.A decisão de uma rede recorrente alcançada na etapa de tempo t-1 afeta a decisão que alcançará um momento mais tarde na etapa de tempo t;

Redes neurais simétricas : Ela é semelhante à Rede Recorrente, porem as conexões entre as unidades são simétricas(Recebem o mesmo peso em ambas as direções). Se tornando uma rede mais fácil de ser analisadas.

(DATA SCIENCE ACADEMY, 2021)

Podemos ver na figura 23 uma representação grafica entre a "Rede neural feed-forward"e a "Rede neural recorrente".

Figura 23 – Diagrama de uma "Rede neural feedforward"e uma "Rede neural recorrente"



Fonte: Data Science Academy, 2021

5.2.2 Funções de Ativação

As "Funções de Ativação"tem um papel importante nas RNAs, já que elas definem saída de cada nó dado o conjunto de entradas.Quando não temos uma função de ativação, os pesos e *bias* transforma a saída em linear.

As principais funções de ativação são: Linear(ou *Linear activation*), Ativação Sigmoide, Softmax , ReLu (ou *Rectified linear unit*) e a Tangente Hiperbólica (ou TanH).

A função Linear, é a função de ativação mais básica porque ela não altera a saída do perceptron. Sendo mais utilizadas em RNAs de regressão. A função algébrica referente a essa função é a:

$$\phi(\mathrm{x}) = \mathrm{x}$$

A função de ativação Sigmoide é utilizada por redes que usem propagações positivas, que precisam de uma saída > 0. Ela é eficiente nas questões de não-linearidade.Essa característica aumenta o custo computacional.A função algébrica referente a essa função é a:

ϕ(x) = $\frac{1}{1+\varrho^{-}x}$

A função de ativação Softmax, é uma função sigmoide. Ela força a saída de uma rede neural a representar a probabilidade dos dados serem de uma das classes definidas. Ela transforma as saídas para cada classe para valores entre 0 e 1 e também divide pela soma das saídas.A função algébrica referente a essa função é a:

σ(x)j = $\frac{\varrho^{-}zj}{\sum k\varrho^{-}zk}$

A função de ativação "ReLU"é a função mais popular e amplamente usada em projetos que tenham Redes Neurais. A função de ativação "ReLU"é uma função não linear, possibilitando ter varias camadas de neurônios ativadas por ela. A função algébrica referente a essa função é a:

f(x) = max (0,x)

A função de ativação "Tangente Hiperbólica"é a função mais próxima da função indenidade, sendo uma alternativa para ativar as camadas ocultas.A função algébrica referente a essa função é a:

f(x) = $2\sigma(2x) - 1$

5.2.3 Treinamento de uma Rede Neural

A faze de aprendizagem de uma rede neural(ou treinamento), consiste em ajustar os pesos, de modo a reduzir o erro observado na saída da rede. Segundo HAYKIN (2007)

as redes neurais são estimuladas pelo ambiente, sofrem modificações nos seus parâmetros livres e responde de uma maneira nova ao ambiente de acordo com suas modificações.

No treinamento é utilizado a técnica de aprendizagem supervisionada, onde é dado um conjunto de dados já rotulados é introduzido na rede neural. É definido um sinal de erro como sendo a diferença entre a resposta verdadeira e a resposta encontrada na saída da rede.

Não existe uma regra exata para definirmos o tempo de treinamento de cada rede, geralmente ela é definida antes do início do treinamento, utilizando uma quantidade máxima de ciclos(ou épocas). Para HAYKIN (2007) a fase de treinamento deve ser finalizada quando encontramos um valor aceitável para a taxa de erro.

5.2.4 Algoritmo Backpropagation

Para corrigir os pesos sinápticos, uma das técnicas populares é o *backpropagation*. Corrigindo os valores dos pesos pela diferença entre o valor encontrado na saída da rede com o valor esperado.

Criado por Rumelhart, Hinton e Williams (1986) o algorítimo *backpropagation*, possibilitava o treinamento de redes neurais com diversas camadas através da Retropropagação do erro. Segundo o Data Science Academy (2021) a técnica do algorítimo *backpropagation* pode deixar o treinamento da rede neural até dez milhões de vezes mais rápido.

"O objetivo do backpropagation é otimizar os pesos para que a rede neural possa aprender a mapear corretamente as entradas para as saídas."(DATA SCIENCE ACADEMY, 2021)

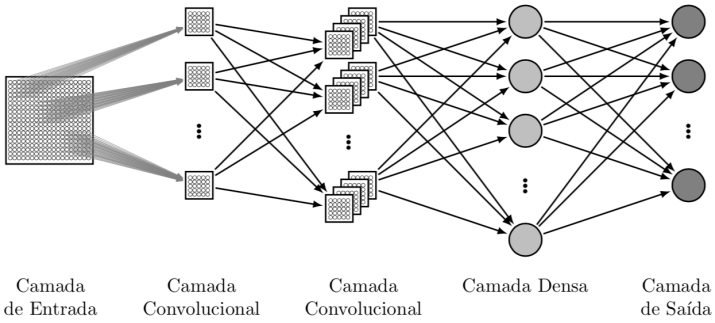
Segundo o Data Science Academy (2021) o algoritmo de backpropagation consiste em duas fases:

- Primeiramente é feito o *forward pass*, onde as entradas fazem o caminho através da rede é retornado uma sai.
- O segundo passo é feito o *backward pass*, onde será calculada a função de perda da última camada. Esse gradiente é usado para aplicar recursivamente a regra da cadeia (ou chain rule), onde iremos atualizar os pesos da Rede.

5.2.5 Redes neurais convolucionais ou CNN

A Rede Neural Convolucional (Convolutional neural network), é uma arquitetura de RNA, sendo uma variante da MLP(Rede Neural Artificial Multicamadas). As CNNs tem um uso mais voltado para fins de classificação, detecção e reconhecimento de imagens e também vídeos. A CNN tem como principal diferencial, a divisão em estágios, como visto na figura 24

Figura 24 – Arquitetura de uma Rede neural convolucional



Fonte: Sakurai, 2017

A CNN é composta de 4 camadas principais, sendo elas: A camada de entrada, camadas convolucionais, camadas densas(ou oculta) e a camada de saída. A camada convolucional, a mais importante dessa arquitetura, é responsável pela extração das principais características da imagem de entrada, essa ação é chamada "mapa de recurso"(ou feature map).

A primeira CNN foi criada por Yann LeCunn, para o reconhecimento de caracteres manuscritos, como, por exemplo, a interpretação de códigos postais. De maneira geral, a imagem é dividia em campos que alimentam a camada convolucional, que extrai então recursos da imagem.(MADHAVAN; JONES, 2017)

Segundo Samaya Madhavan (2017), a CNN é composta por 5 etapas principais, sendo elas:

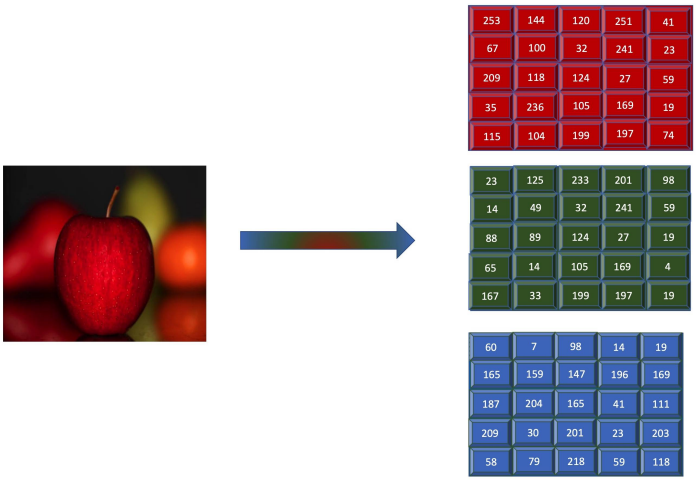
1. *Image channels*;
2. *Convolution*;
3. *Pooling*;
4. *Flattening*;
5. *Full connection.*

5.2.5.1 Image channels

Nessa primeira etapa, encontra-se o processo onde tornamos a imagem compatível com a com o algorítimo CNN. A imagem precisara ser transformada em píxeis. Cada píxel é mapeado usando números de 0 até 255, onde 0 significa branco e 255 preto.

Para imagens em preto e branco, será representada em uma matriz bidimensional. Cada célula da matriz irá conter seu valor correspondente. Para imagens coloridas, será representada em uma matriz tridimensional. Sendo cada *"channels"* da matriz uma cor, sendo elas: *red, green* e *blue* respectivamente. Poderemos ver um exemplo, na figura 26.

Figura 25 – Imagem representada em forma de matriz.



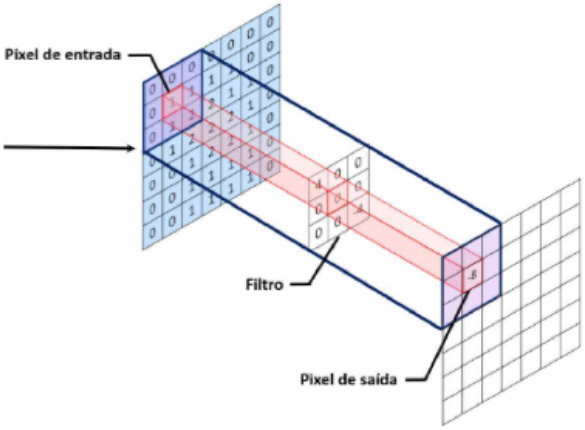
Fonte: Madhavan, 2017

5.2.5.2 Convolution

Segundo Samaya Madhavan (2017), a convolução é uma operação onde uma função modifica a forma de outra. Ainda segundo Samaya Madhavan (2017), as convoluções nas imagens são aplicadas por vários motivos, sendo eles: aumentar a nitidez, suavizar e intensificar.Na CNN, as convoluções são aplicadas para extrair os recursos proeminentes nas imagens.

A extração dos recursos principais da imagem é realizada através do uso de um filtro ou *Kernel*.Esse filtro consiste em uma matriz que será o conteúdo a ser extraído. O filtro(ou Kenel) é aplicado sobre a matriz de entrada, e a convolução resultante é uma matriz bidimensional que contém a correlação da imagem em relação ao filtro(ou Kenel). Segundo Samaya Madhavan (2017) A matriz de saída é conhecida como *"feature map"*.

Figura 26 – Representação da camada de convolução



Fonte: USHIZIMA, 2017

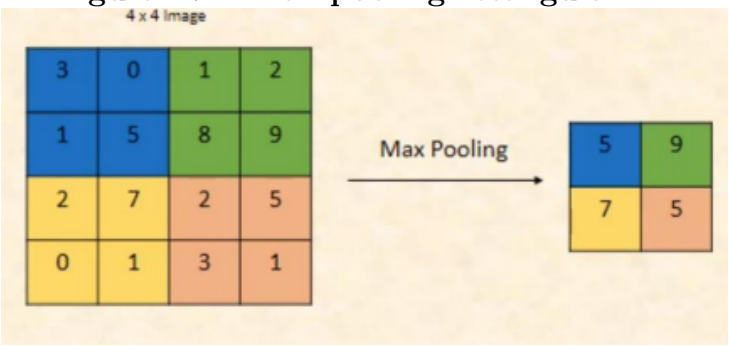
5.2.5.3 Pooling

Após as camadas convolucionais, com o objetivo de reduzir a dimensão espacial do mapa gerado, é aplicada a camada de *Pooling*. A camada possibilita que as camadas mais profundas possam ter uma quantidade maior de filtros sem existir um custo computacional muito elevado. Segundo Madhavan (2017) esse procedimento ajuda a compactar ainda mais a dimensão do *feature map*.

O *"Pooling"* pode ser obtido pegando o máximo, o mínimo ou a média dentro de um grupo de células. Sendo Conhecidos como: *Max Pooling, Min Pooling* e *Average Pooling*. Sendo os *Max Pooling* e *Average Pooling* mais utilizados.

No *Max Pooling*, o valor maximo da matriz é retornado para cada retangulo percorrido. Segundo essa técnica é a mais utilizada.Podemos ver um exemplo na figura 27.

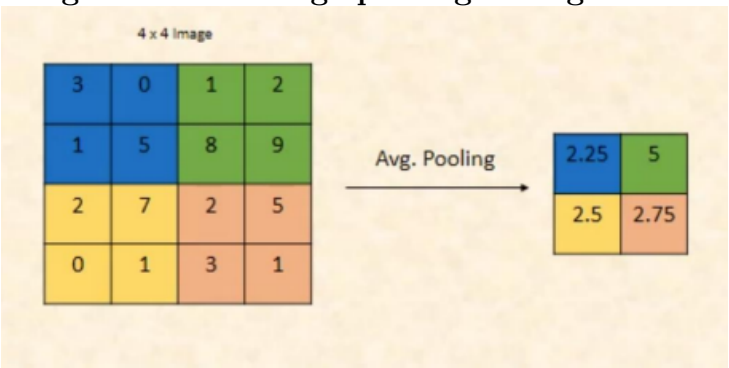
Figura 27 – Max pooling retângulo 2x2



Fonte: RIZWAN, 2018

No *Average Pooling*, a média dos valores no interior do retângulo é retornado para cada retângulo percorrido. Segundo GOODFELLOW Yoshua BENGIO (2016) essa técnica é a mais utilizada.Podemos ver um exemplo na figura 28.

Figura 28 – Average pooling retângulo 2x2



Fonte: RIZWAN, 2018

5.2.5.4 Flattening

A camada *Flattening* tem como objetivo transformar os resultados da CNN em dados compatíveis a um RNA(Rede Neural Artificial). Um RNA tem como entrada valores

em forma de vetor.

5.2.6 Transferência de aprendizado

A transferência de aprendizado consiste em reaproveitar apenas as camadas que deseja ("parte de baixo" ou *"floor"*) de uma rede já treinada, podendo aproveitar os pesos alinhados e com isso treinar apenas as camadas acrescentadas("parte de cima"ou *"top"*).

O processo de "transferência de aprendizado" é útil, pois economiza uma quantidade enorme de processamento, já que, o *"floor"* só processará cada padrão uma única vez. No nosso projeto utilizaremos esse processo em 3 redes diferentes, sendo elas ,"ResNet50", "VGG19"e "InceptionV3". Elas serão treinadas utilizando os pesos do treinamento "ImageNet".

5.3 Execução dos Modelos

Nessa seção apresentarei o "passo a passo"da execução dos modelos. Começaremos apresentando como foi realizado a criação do modelo *"ResNet50"*, logo em seguida será apresentado a criação do modelo *"VGG19"* e por fim a utilização da arquitetura *"InceptionV3"*.

Hoje sabemos que o quanto mais profundo uma modelo pode ir, melhor seu desempenho. A arquitetura *Resnet* foi avaliada como um dos trabalhos mais inovadores em visão computacional, conseguindo ganhar a competição na tarefa de classificação *"ILSVRC-2015"*.

Criado pelo Departamento de Engenharia, Universidade de Oxford(Department of Engineering Science, University of Oxford.) a "VGG19"é uma arquitetura de rede neural convolucional com 19 camadas de profundidade. 16 camadas convolucionais e 3 camadas totalmente conectadas.

A proposta desse modelo consiste em identificar e classificar as categorias de câncer e após sua classificação como "Maligno", utilizaremos a técnica de transferia de aprendizagem para a clusterização por classes.

Inicialmente, importaremos as Bibliotecas necessárias para a criação do modelo. A principais bibliotecas serão *"TensorFlow"* para criação das estruturas da "RedeNeural"e *"Pandas"* para criação e manipulação dos *"DataFremes"*. Poderemos ver o código referente a essas importações na figura 29.

Logo apos as importações das Bibliotecas será feito também a declaração de algumas constantes, como: caminho principal dos dados, tamanho das imagens, o "Batch Size"e o número de "Epocas"padrão no treinamento. Usamos por padrão "25 épocas"e formos aumentando conforme o necessário. Podemos ver esse trecho de código na figura 30.

Figura 29 – Importações Biblioteca (Google Colab)

```
[1] import re
    from tensorflow.keras.layers import *
    from tensorflow.keras.optimizers import *
    import seaborn as sns
    from tensorflow.keras.models import *
    from tensorflow.keras.preprocessing.image import *
    from tensorflow.keras.callbacks import *
    from tensorflow.keras import layers, models, metrics
    from tensorflow.keras.applications.efficientnet import *
    from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
    from sklearn.model_selection import train_test_split
    from tensorflow.keras.applications.inception_v3 import InceptionV3
    from tensorflow.keras.applications.vgg19 import VGG19
    from tensorflow.keras.applications.resnet50 import ResNet50
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split, StratifiedKFold
    import cv2
    from glob import glob
    from random import shuffle
    import tensorflow as tf
    import numpy as np
    import pandas as pd
    from pathlib import Path
    import os.path
```

Fonte: Autor

Figura 30 – Declaração das constantes (Google Colab)

```
[37]
    AUTOTUNE = tf.data.experimental.AUTOTUNE
    GCS_PATH = r"/content/drive/MyDrive/TCC"
    BATCH_SIZE = 16
    IMAGE_SIZE = [224, 224]
    EPOCHS = 25
```

Fonte: Autor

Apos importarmos as bibliotecas necessárias, e declararmos as constantes, será iniciado a fase onde será preparado os dados para o treinamento, validação e teste. Como mostrado no capítulo 4 na figura 15 será criado um "Dataframe"com as principais informações do nosso *"DataSet"*, sendo elas *"FikePath","Type","class","Magnification"* e *"Dataset"* como descritos na lista 4. Serão utilizados para criação do dataframe as seguintes funções apresentadas nas figuras 14, 42, 43,44 e 45.

Com o "dataframe"conseguiremos segmentar os dados de acordo com sua *"Magnification"*. Como já explicado começarei com o modelo utilizando apenas *"Magnification"= 400x* e depois será feita a separação entre *"Dataset de treino"* e *"Dataset de teste"*. Poderemos observar o código referente a essas segmentações na figura 31.

Figura 31 – Segmentação Dataframe 400x (Google Colab)

```
[ ] df_treino_400x = df_geral_400x[df_geral_400x['DataSet'] == 'Tr'][df_geral_400x['Magnification'] == '400x']
    df_teste_400x = df_geral_400x[df_geral_400x['DataSet'] == 'Te'][df_geral_400x['Magnification'] == '400x']
    df_treino_400x = df_treino_400x.drop(columns=['Magnification','DataSet'])
    df_teste_400x = df_teste_400x.drop(columns=['Magnification','DataSet'])
```

Fonte: Autor

Foi realizado o mesmo procedimento para buscarmos as imagens com *"Magnification"= 200x* e também o separaremos pelo tipo de dataset, sendo eles: *"Dataset de treino"* e *"Dataset de teste"*.Poderemos observar o código referente a essas segmentações na figura 32.

Para realizarmos o treinamento, precisaremos definir a taxa de aprendizagem, que no nosso caso será 10e-4 e a separação das imagens em *batchs*, que nesse caso terão o tamanho de 16.

Todos os nossos modelos de Rede serão uma junção da *"parte de baixo"("floor")* das arquiteturas. Pegaremos todas as camadas convolucionais e também o resultado dos pesos adquiridos no treinamento com a base de dados "ImageNet". Integraremos com a *"parte de cima"("top")* criada para criar um pós-processamentos vindo das camadas convolucionais.

O método de optimização utilizado no projeto será o *"ADAM"*.Segundo Ribeiro e Junior (2020) *"ADAM"* é um método de otimização que calcula taxas de aprendizado

Figura 32 – Segmentação Dataframe 200x (Google Colab)

```
[17] df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/200x')
     df_treino = df_geral[df_geral['DataSet'] == 'Tr'][df_geral['Magnification'] == '200x']
     df_teste = df_geral[df_geral['DataSet'] == 'Te'][df_geral['Magnification'] == '200x']
```

Fonte: Autor

adaptativas para cada parâmetro. *"ADAM"* mantêm uma média exponencial decrescente dos gradientes anteriores.

Como demonstrado no capítulo 3 na subseção 3.2 a imagem será redimensionada para "224 x 224 x 3". Sendo esse o padrão da camada de entrada(*input layer*) de todos os nossos modelos.

A sessão de treinamento é divida em 2 processos, sendo o primeiro a separação entre cânceres malignos e benignos a segunda a identificação de suas classes.

Na primeira sessão de treinamento como falado antes, prepararemos a nossa rede neural para identificar um câncer maligno. Seguiremos o seguinte passo-a-passo para esse treinamento:

1. Realização do treinamento para imagens com amplitude de "400x"por 25 épocas;
2. Realização de uma nova rodada de treinamento agora por 15 épocas caso a anterior não alcance uma taxa minima de 93% de acurácia;;
3. Uma última rodada de treinamento com apenas 10 épocas caso a anterior não alcance uma taxa minima de 93% de acurácia;
4. validação do treinamento com o *dataset* de teste.
5. Criação de um *dataset* contendo apenas canceres malignos.
6. Importação dos pesos adquiridos nos processos anteriores;
7. Realização uma rodada de treinamento para imagens com amplitude de "200x"por 10 épocas;
8. validação do treinamento com o *dataset* de teste.
9. Criação de um *dataset* contendo apenas canceres malignos.

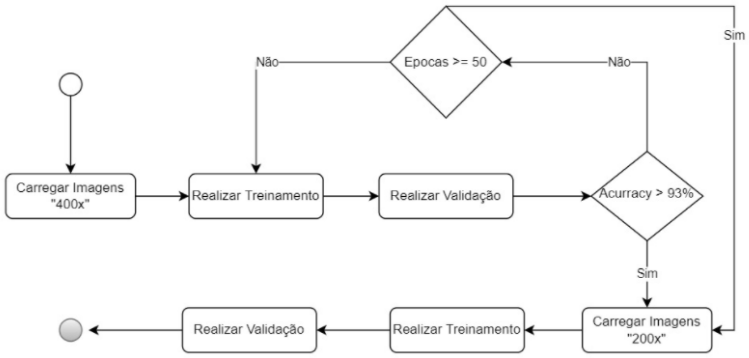
Apos o primeiro processo de treinamento, visto no conjunto de passos 5.3, será feita a segunda etapa onde classificaremos os canceres.Já tendo realizado a primeira etapa de treinamento e validação realizaremos as novas etapas de treino:

1. Realização do treinamento para imagens com amplitude de "400x"por 50 épocas;
2. Realização de uma nova rodada de treinamento agora por 25 épocas, aso a anterior não alcance uma taxa minima de 93% de acurácia;

3. Uma última rodada de treinamento com apenas 25 épocas caso a anterior não alcance uma taxa minima de 93% de acurácia;
4. validação do treinamento com o *dataset* de teste.
5. Importação dos pesos adquiridos nos processos anteriores;
6. Realização uma rodada de treinamento para imagens com amplitude de "200x"por 25 épocas;
7. validação do treinamento com o *dataset* de teste.
8. Criação de um *dataset* contendo os canceres e suas classes.

Podemos ver ilustrativamente, na figura 33, o processo descrito no passo a passo da criação do modelo de predição para categorias.

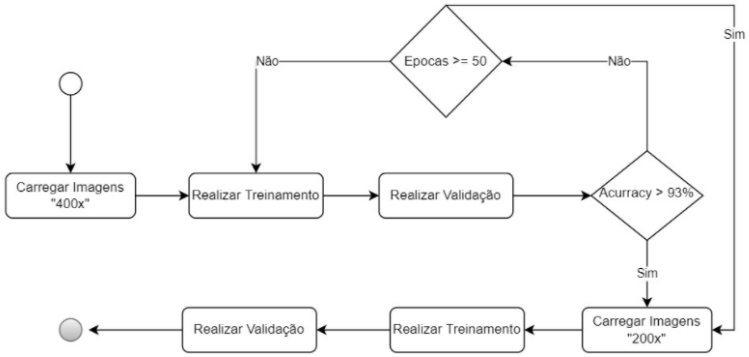
Figura 33 – Diagrama de fluxo criação do modelo categoria



Fonte: Autor

Podemos ver ilustrativamente, na figura 35, o processo descrito no passo a passo da criação do modelo de predição para classes.

Figura 34 – Diagrama de fluxo criação do modelo classe

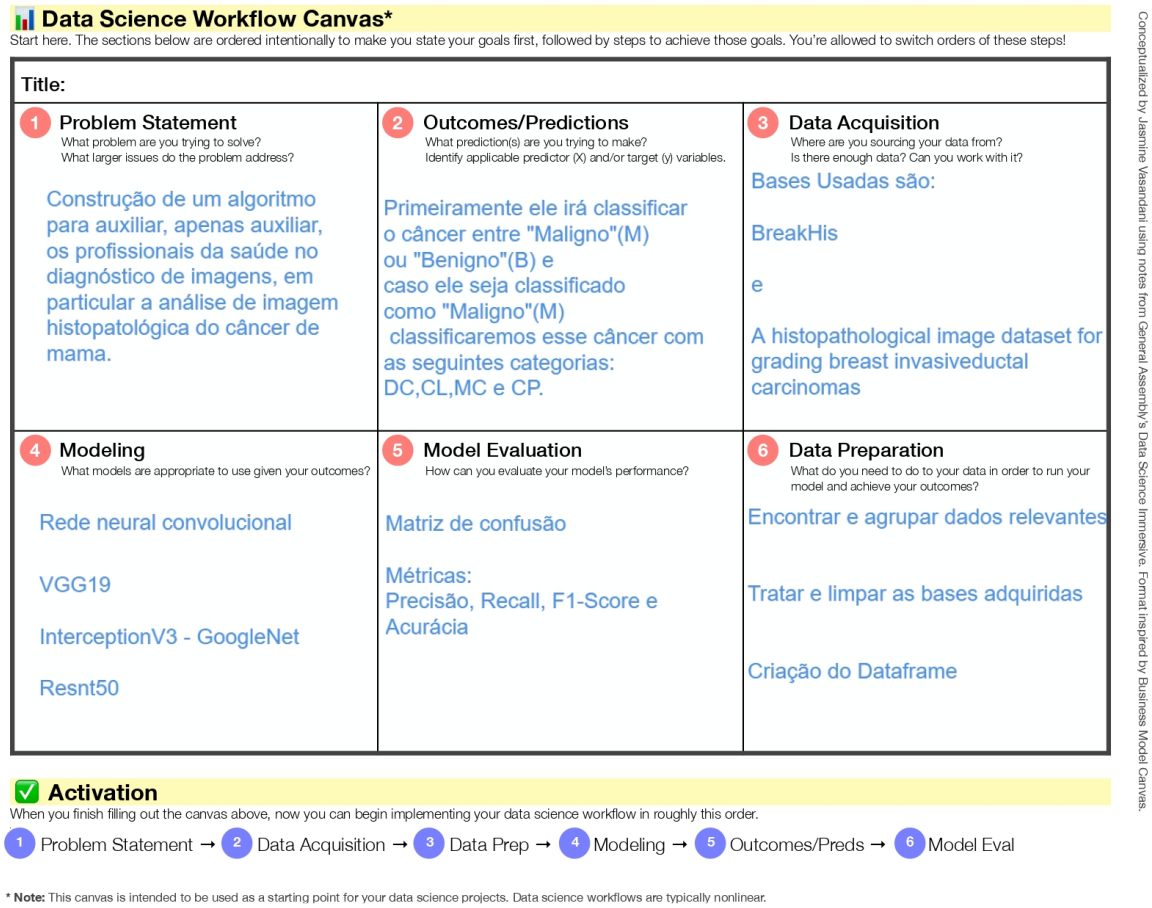


Fonte: Autor

6 APRESENTAÇÃO DOS RESULTADOS

Os resultados do projeto foram obtidos a partir do estabelecimento das metas mapeadas no modelo Canvas de Vasandani conforme apresentado na figura 35

Figura 35 – Modelo Canvas



📊 **Data Science Workflow Canvas***

Start here. The sections below are ordered intentionally to make you state your goals first, followed by steps to achieve those goals. You're allowed to switch orders of these steps!

Title:

1 Problem Statement What problem are you trying to solve? What larger issues do the problem address?	2 Outcomes/Predictions What prediction(s) are you trying to make? Identify applicable predictor (X) and/or target (y) variables.	3 Data Acquisition Where are you sourcing your data from? Is there enough data? Can you work with it?
Construção de um algoritmo para auxiliar, apenas auxiliar, os profissionais da saúde no diagnóstico de imagens, em particular a análise de imagem histopatológica do câncer de mama.	Primeiramente ele irá classificar o câncer entre "Maligno"(M) ou "Benigno"(B) e caso ele seja classificado como "Maligno"(M) classificaremos esse câncer com as seguintes categorias: DC,CL,MC e CP.	Bases Usadas são: BreakHis e A histopathological image dataset for grading breast invasiveductal carcinomas
4 Modeling What models are appropriate to use given your outcomes?	**5 Model Evaluation** How can you evaluate your model's performance?	**6 Data Preparation** What do you need to do to your data in order to run your model and achieve your outcomes?
Rede neural convolucional VGG19 InterceptionV3 - GoogleNet Resnt50	Matriz de confusão Métricas: Precisão, Recall, F1-Score e Acurácia	Encontrar e agrupar dados relevantes Tratar e limpar as bases adquiridas Criação do Dataframe

✅ **Activation**

When you finish filling out the canvas above, now you can begin implementing your data science workflow in roughly this order.

1 Problem Statement → 2 Data Acquisition → 3 Data Prep → 4 Modeling → 5 Outcomes/Preds → 6 Model Eval

* **Note:** This canvas is intended to be used as a starting point for your data science projects. Data science workflows are typically nonlinear.

Conceptualized by Jasmine Vasandani using notes from General Assembly's Data Science Immersive. Format inspired by Business Model Canvas.

Fonte: Autor

A seguir apresentaremos os resultados adquiridos pelos modelos das nossas *"CNN"*. Apresentaremos os resultados em uma "Matriz de confusão"e uma tabela mostrando sua eficiência.

Antes de seguirmos com a apresentação dos resultados, precisarmos realizar as importações das bibliotecas necessárias, como visto na figura 29

Os resultados serão apresentados seguindo a seguinte ordem:

1. Resultados para "Tipo de Câncer"com o modelo VGG19
2. Resultados para "Classe de Câncer"com o modelo VGG19
3. Resultados para "Tipo de Câncer"com o modelo GoogleNet
4. Resultados para "Classe de Câncer"com o modelo GoogleNet

5. Resultados para "Tipo de Câncer"com o modelo ResNet

6. Resultados para "Classe de Câncer"com o modelo ResNet

Primeiro passo para apresentação do resultado será a criação da rede e o carregamento dos pesos adquiridos nos treinamentos. Na imagem a seguir, mostraremos um exemplo desse código.

Figura 36 – Criação e carregamento de pesos (Google Colab)

```
[14] VGG19 = build_model("VGG19")
     VGG19.load_weights('/content/drive/MyDrive/TCC/VGG19_MB_TESTE.h5')
```

Fonte: Autor

Realizaremos os testes, para as amplitudes 400x, 200x e para o *dataset* completo.

Figura 37 – Código para teste do dataset (Google Colab)

```
[16] pred = VGG19.predict(test_images_VGG19_MB)
     pred=np.argmax(pred,axis=1)
```

Fonte: Autor

Apos testados, criaremos o *dataset* com o resultado.

Figura 38 – Criação e carregamento de pesos (Google Colab)

```
[22] pred_df_400=df_teste_400.copy()
     labels={}
     for l,v in test_images_VGG19_MB.class_indices.items():
         labels.update({v:l})
     pred_df_400['pred']=pred
     pred_df_400['pred']=pred_df_400['pred'].apply(lambda x: labels[x])
```

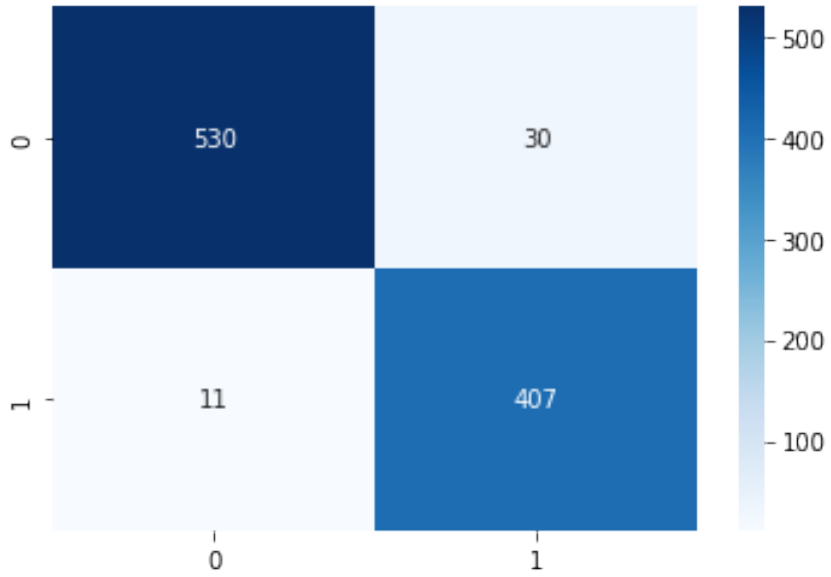
Fonte: Autor

Os exemplos presentes nas figuras 36,37 e 38 são referentes ao modelo *"VGG19"*, porem o padrão é replicado para os modelos *"GoogleNet"* e *"ResNet"*.

6.1 Modelo Preditivo - VGG19

A matriz de confusão abaixo (gráfico 12) mostra o resultado das predições efetuadas nos testes, na amostra onde temos apenas imagens com 400x.

Gráfico 12: Matriz de Confusão Categoria de Cancer 400X



Fonte: Autor

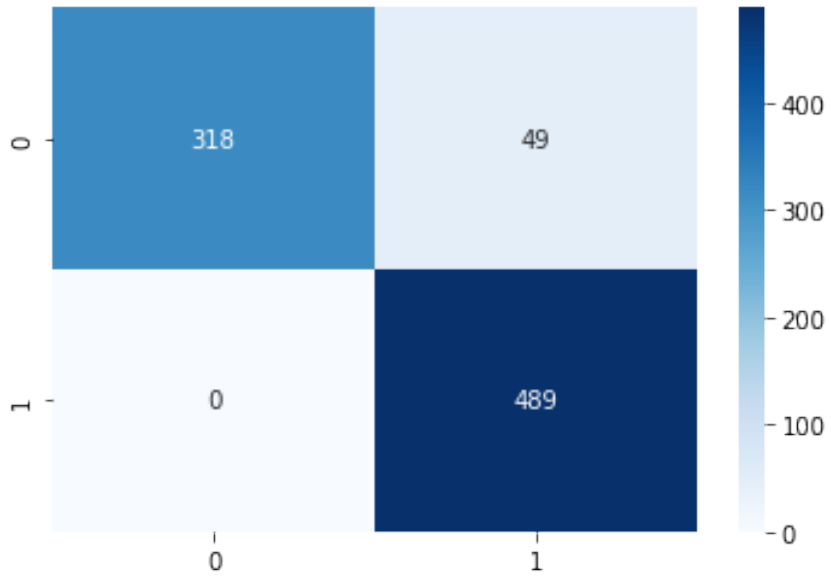
Tabela 4 – Tabela de métricas do gráfico 12

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	98%	95%	96%	560
Maligno	93%	97%	95%	418

Fonte: Autor

A matriz de confusão abaixo (gráfico 13) mostra o resultado das predições efetuadas nos testes, na amostra onde temos apenas imagens com 200x.

Gráfico 13: Matriz de Confusão Categoria de Cancer 200X



Fonte: Autor

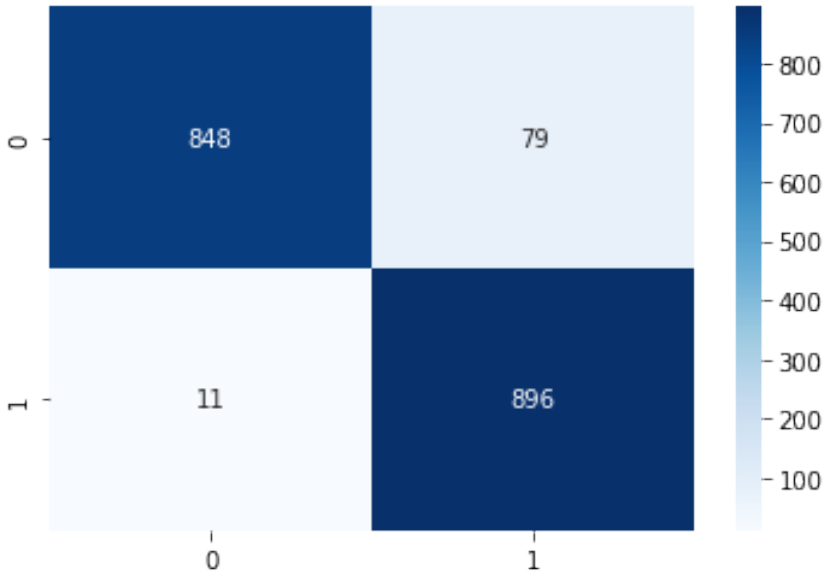
Tabela 5 – Tabela de métricas do gráfico 13

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	100%	87%	93%	367
Maligno	91%	100%	95%	489

Fonte: Autor

A matriz de confusão abaixo (gráfico 14) mostra o resultado das predições efetuadas nos testes com todas as imagens.

Gráfico 14: Matriz de Confusão Categoria de Cancer



Fonte: Autor

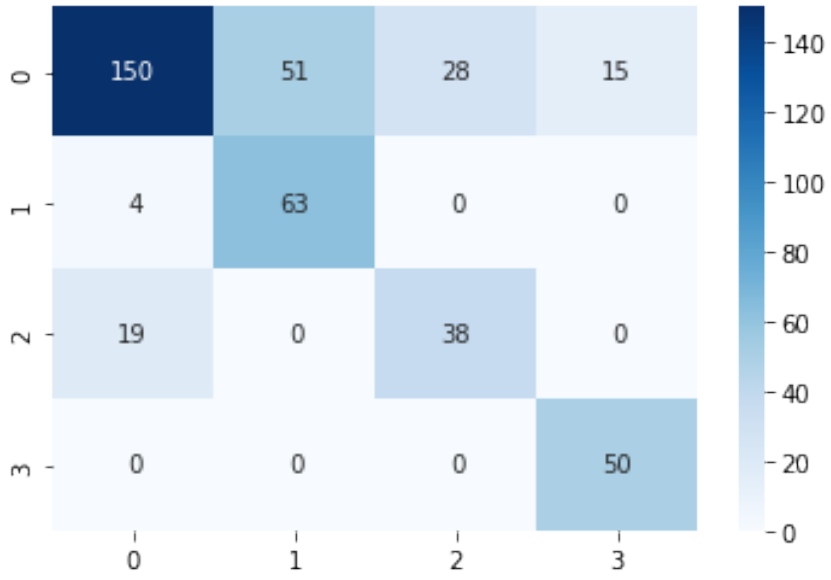
Tabela 6 – Tabela de métricas do gráfico 14

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	99%	91%	95%	1301
Maligno	92%	99%	95%	1406

Fonte: Autor

A matriz de confusão abaixo (gráfico 15) mostra o resultado das predições efetuadas nos testes, para classes de câncer na amostra onde temos apenas imagens com 400x.

Gráfico 15: Matriz de Confusão Classe de Cancer 400X



Fonte: Autor

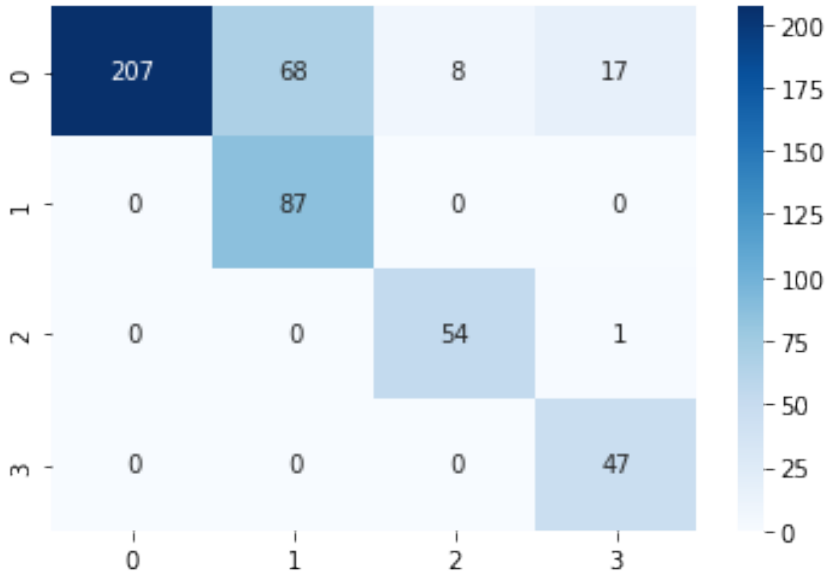
Tabela 7 – Tabela de métricas do gráfico 15

Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	87%	61%	72%	244
LC	55%	94%	70%	67
MC	58%	67%	62%	57
PC	77%	100%	87%	50

Fonte: Autor

A matriz de confusão abaixo (gráfico 16) mostra o resultado das predições efetuadas nos testes, para classes de câncer na amostra onde temos apenas imagens com 200x.

Gráfico 16: Matriz de Confusão Classe de Cancer 200X



Fonte: Autor

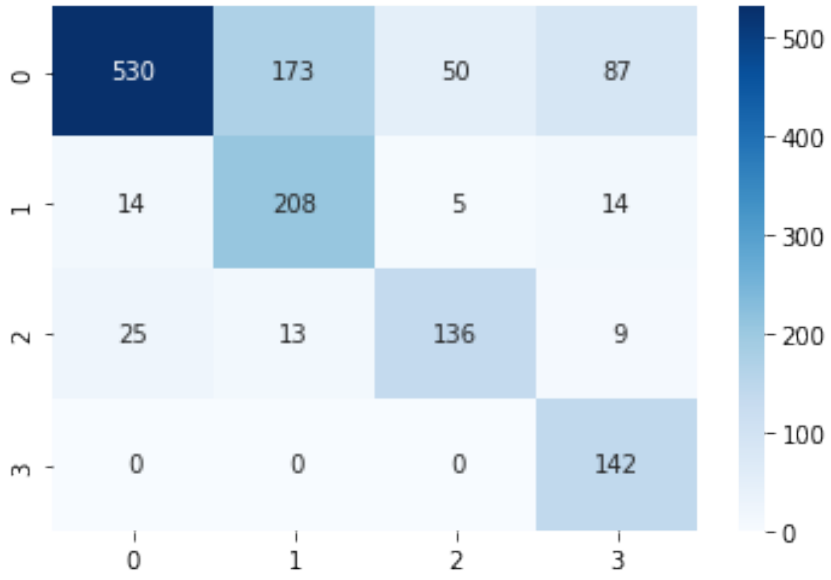
Tabela 8 – Tabela de métricas do gráfico 16

Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	100%	69%	82%	300
LC	56%	100%	72%	87
MC	87%	98%	92%	55
PC	72%	100%	84%	47

Fonte: Autor

A matriz de confusão abaixo (gráfico 17) mostra o resultado das predições efetuadas nos testes com todas as imagens.

Gráfico 17: Matriz de Confusão Classe de Cancer



Fonte: Autor

Tabela 9 – Tabela de métricas do gráfico 17

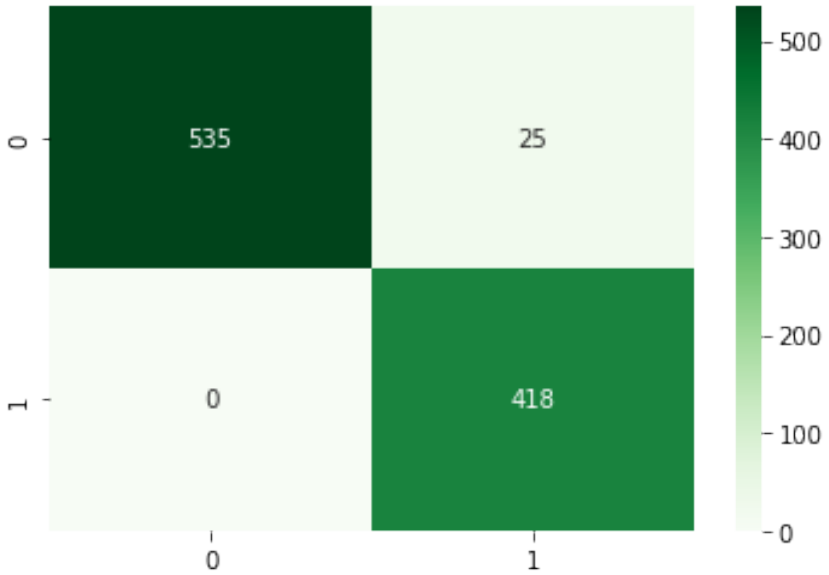
Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	93%	63%	75%	840
LC	53%	86%	66%	241
MC	71%	74%	73%	183
PC	56%	100%	72%	142

Fonte: Autor

6.2 Modelo Preditivo - GoogleNet

A matriz de confusão abaixo (gráfico 18) mostra o resultado das predições efetuadas nos testes, na amostra onde temos apenas imagens com 400x.

Gráfico 18: Matriz de Confusão Categoria de Cancer 400X



Fonte: Autor

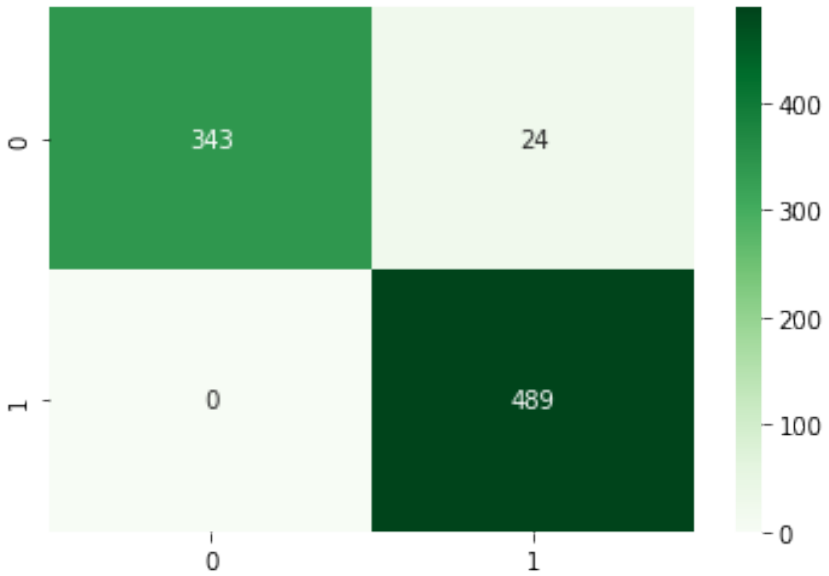
Tabela 10 – Tabela de métricas do gráfico 15

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	100%	96%	98%	560
Maligno	94%	100%	97%	418

Fonte: Autor

A matriz de confusão abaixo (gráfico 19) mostra o resultado das predições efetuadas nos testes, na amostra onde temos apenas imagens com 200x.

Gráfico 19: Matriz de Confusão Categoria de Cancer 200X



Fonte: Autor

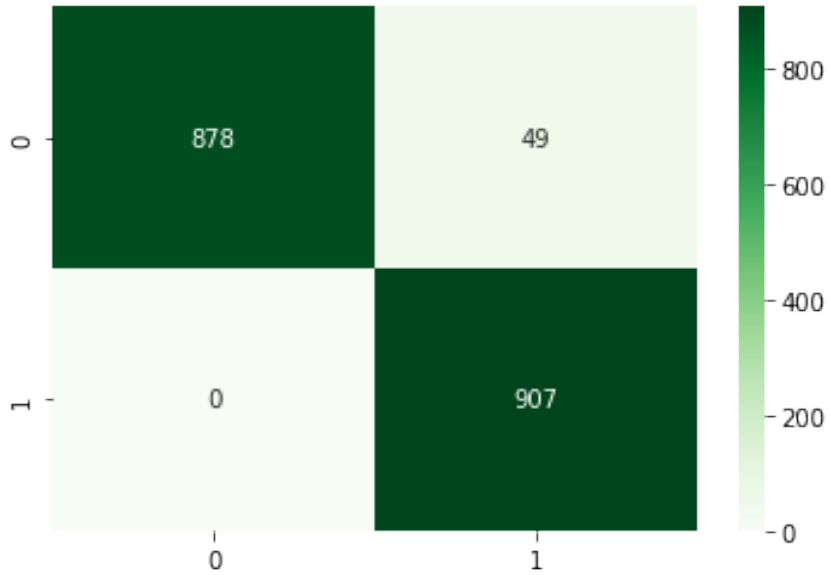
Tabela 11 – Tabela de métricas do gráfico 19

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	100%	93%	97%	367
Maligno	95%	100%	98%	489

Fonte: Autor

A matriz de confusão abaixo (gráfico 20) mostra o resultado das predições efetuadas nos testes com todas as imagens.

Gráfico 20: Matriz de Confusão Tipo de Cancer



Fonte: Autor

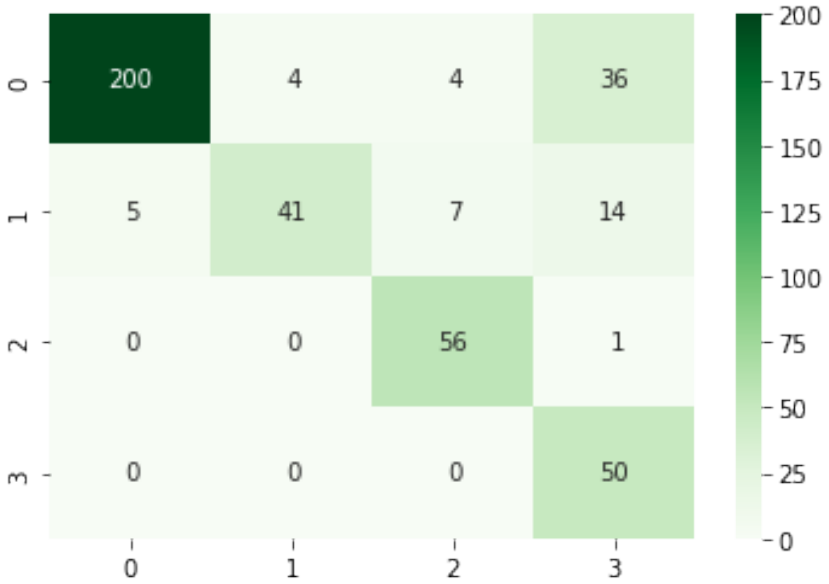
Tabela 12 – Tabela de métricas do gráfico 20

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	100%	95%	97%	927
Maligno	96%	100%	98%	907

Fonte: Autor

A matriz de confusão abaixo (gráfico 21) mostra o resultado das predições efetuadas nos testes, para classes de câncer na amostra onde temos apenas imagens com 400x.

Gráfico 21: Matriz de Confusão Classe de Cancer 400X



Fonte: Autor

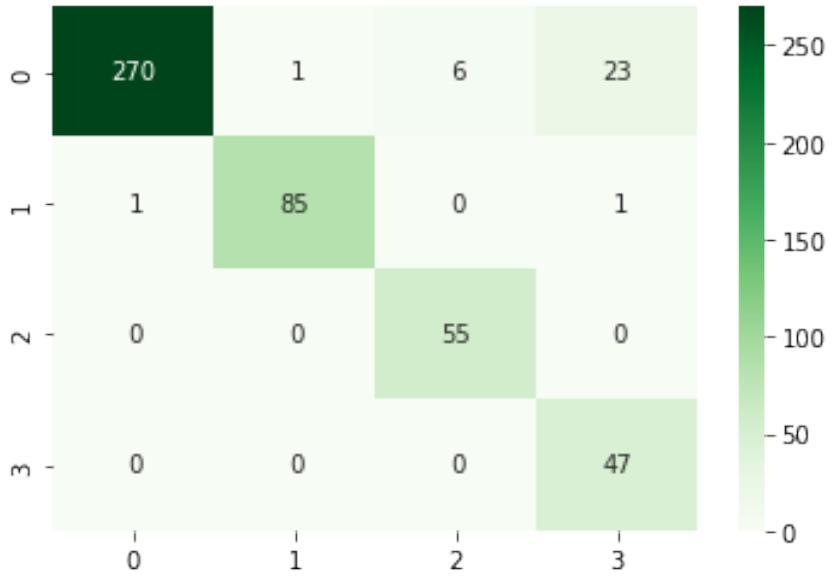
Tabela 13 – Tabela de métricas do gráfico 21

Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	98%	82%	89%	244
LC	91%	61%	73%	67
MC	84%	98%	90%	57
PC	50%	100%	66%	50

Fonte: Autor

A matriz de confusão abaixo (gráfico 22) mostra o resultado das predições efetuadas nos testes, para classes de câncer na amostra onde temos apenas imagens com 200x.

Gráfico 22: Matriz de Confusão Classe de Cancer 200X



Fonte: Autor

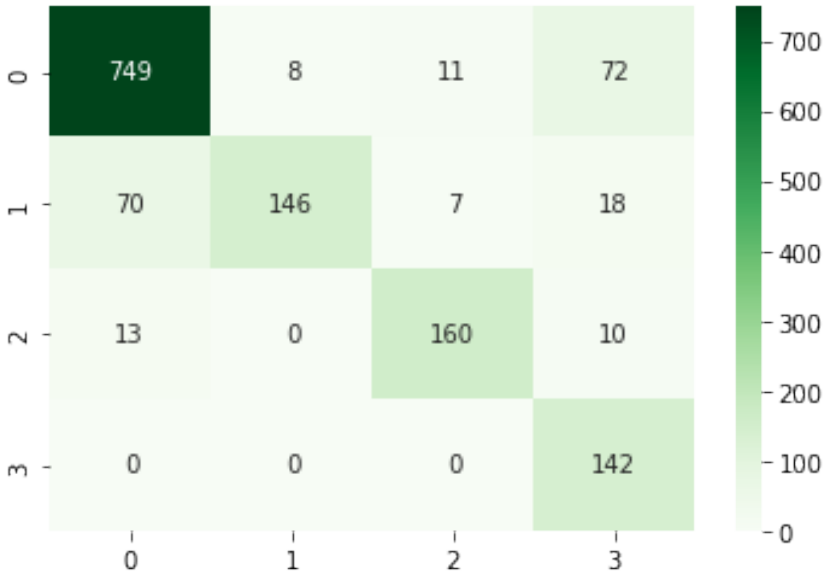
Tabela 14 – Tabela de métricas do gráfico 22

Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	100%	90%	95%	300
LC	99%	98%	98%	87
MC	90%	100%	95%	55
PC	66%	100%	80%	47

Fonte: Autor

A matriz de confusão abaixo (gráfico 23) mostra o resultado das predições efetuadas nos testes com todas as imagens.

Gráfico 23: Matriz de Confusão Classe de Cancer



Fonte: Autor

Tabela 15 – Tabela de métricas do gráfico 23

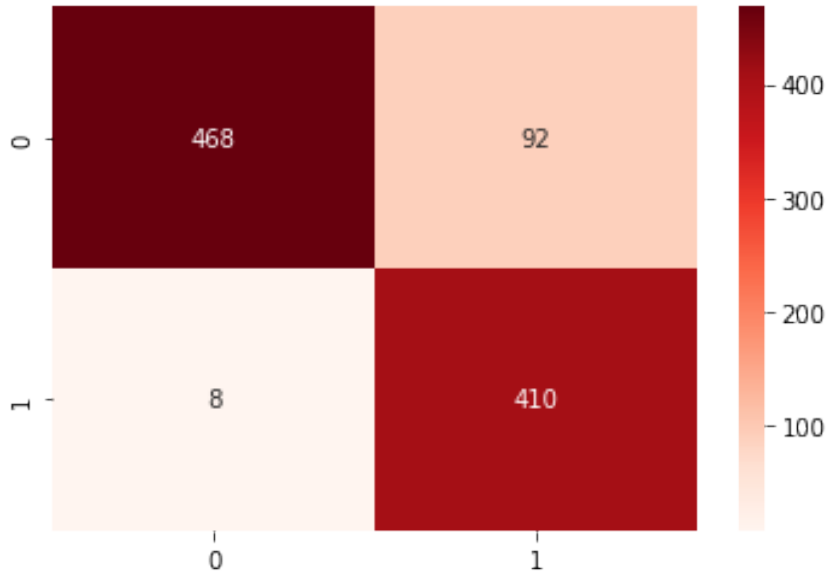
Classe do Câncer	Precisão	Recall	f1-score	Support
DC	90%	89%	90%	840
LC	95%	61%	74%	241
MC	90%	87%	89%	183
PC	59%	100%	74%	142

Fonte: Autor

6.3 Modelo Preditivo - ResNet50

A matriz de confusão abaixo (gráfico 24) mostra o resultado das predições efetuadas nos testes, na amostra onde temos apenas imagens com 400x.

Gráfico 24: Matriz de Confusão Categoria de Cancer 400X



Fonte: Autor

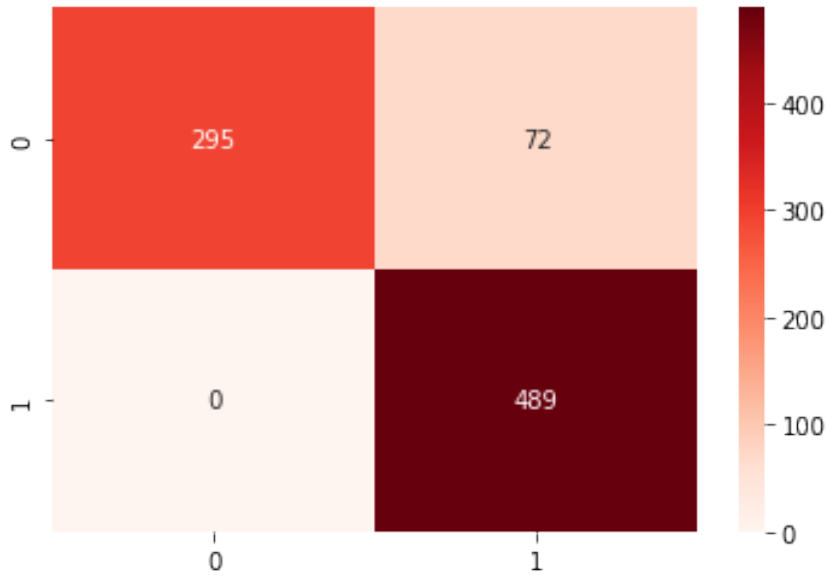
Tabela 16 – Tabela de métricas do gráfico 24

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	98%	84%	90%	560
Maligno	82%	98%	89%	418

Fonte: Autor

A matriz de confusão abaixo (gráfico 25) mostra o resultado das predições efetuadas nos testes, na amostra onde temos apenas imagens com 200x.

Gráfico 25: Matriz de Confusão Categoria de Cancer 200X



Fonte: Autor

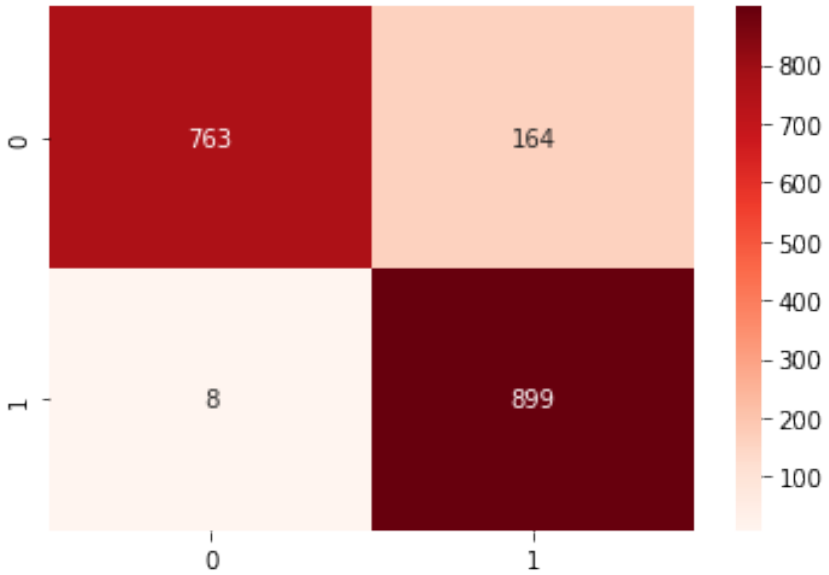
Tabela 17 – Tabela de métricas do gráfico 25

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	100%	80%	89%	367
Maligno	87%	100%	93%	489

Fonte: Autor

A matriz de confusão abaixo (gráfico 26) mostra o resultado das predições efetuadas nos testes com todas as imagens.

Gráfico 26: Matriz de Confusão Categoria de Cancer



Fonte: Autor

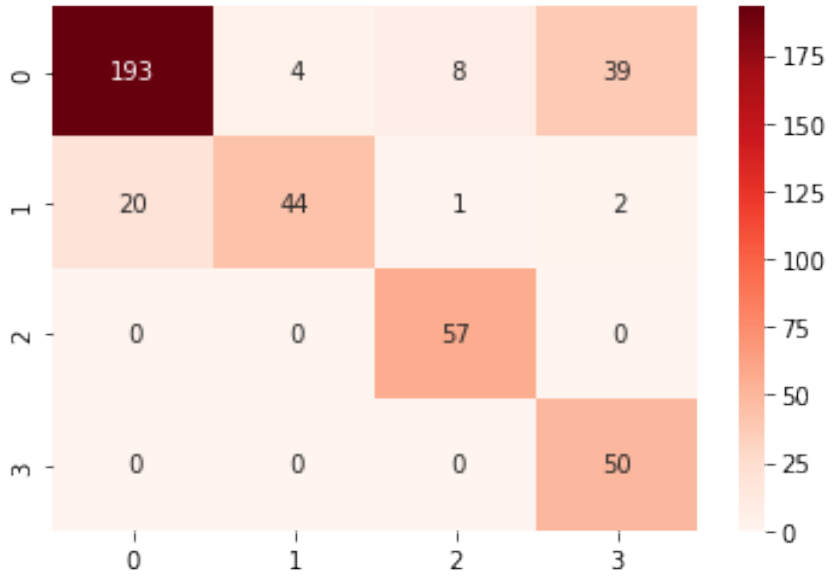
Tabela 18 – Tabela de métricas do gráfico 26

Tipo de Câncer	Precisão	Recall	F1-Score	Support
Benigno	99%	82%	90%	927
Maligno	85%	99%	91%	907

Fonte: Autor

A matriz de confusão abaixo (gráfico 27) mostra o resultado das predições efetuadas nos testes, para classes de câncer na amostra onde temos apenas imagens com 400x.

Gráfico 27: Matriz de Confusão Classe de Cancer 400X



Fonte: Autor

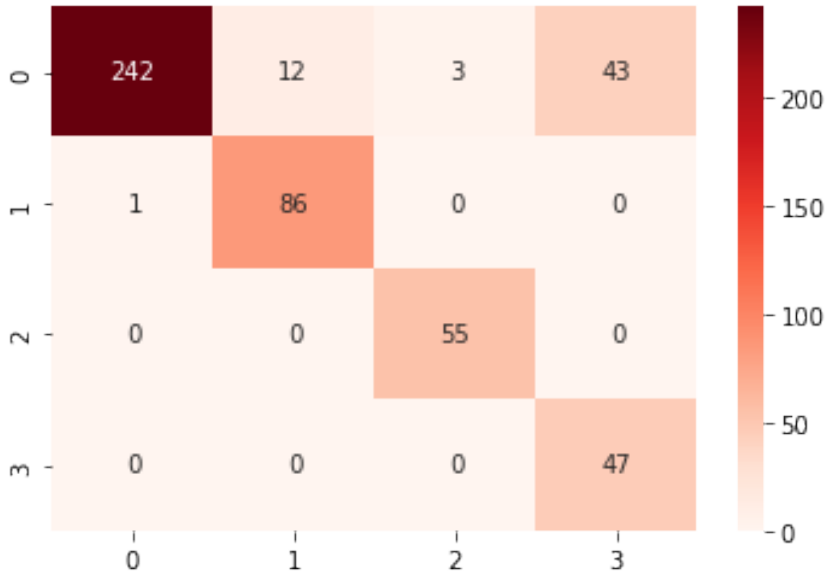
Tabela 19 – Tabela de métricas do gráfico 27

Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	91%	79%	84%	224
LC	92%	66%	77%	67
MC	86%	100%	93%	57
PC	55%	100%	71%	50

Fonte: Autor

A matriz de confusão abaixo (gráfico 28) mostra o resultado das predições efetuadas nos testes, para classes de câncer na amostra onde temos apenas imagens com 200x.

Gráfico 28: Matriz de Confusão Classe de Cancer 200X



Fonte: Autor

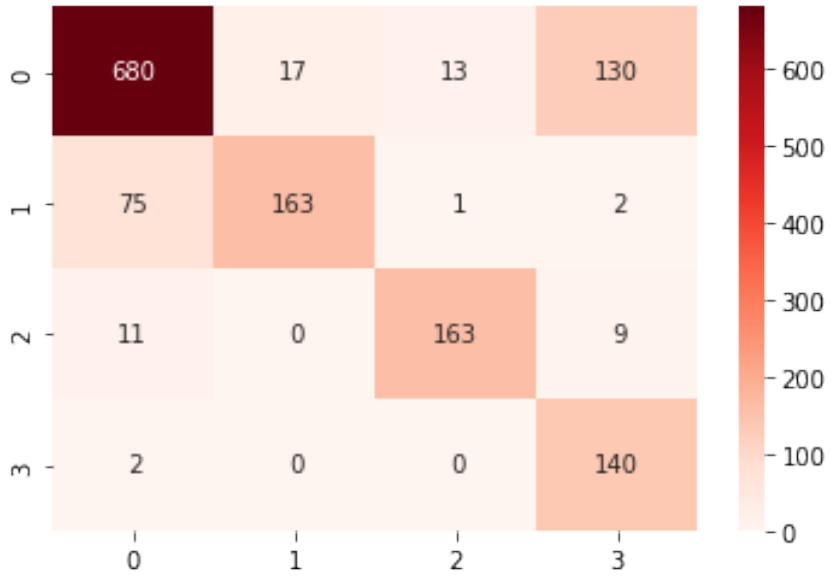
Tabela 20 – Tabela de métricas do gráfico 28

Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	100%	81%	89%	300
LC	88%	99%	93%	87
MC	95%	100%	97%	55
PC	52%	100%	69%	47

Fonte: Autor

A matriz de confusão abaixo (gráfico 29) mostra o resultado das predições efetuadas nos testes com todas as imagens.

Gráfico 29: Matriz de Confusão Classe de Cancer



Fonte: Autor

Tabela 21 – Tabela de métricas do gráfico 29

Classe do Câncer	Precisão	Recall	F1-Score	Support
DC	89%	81%	85%	840
LC	91%	68%	77%	241
MC	92%	89%	91%	183
PC	50%	99%	66%	142

Fonte: Autor

6.4 Resultado Geral

Nessa seção apresentaremos os resultados médios de cada previsão apresentadas nas seções anteriores. Serão presentados em duas tabelas. A primeira Representado a classificação por categoria de câncer a segunda referente a classificação por classes de câncer.

Os valores apresentados nas tabelas são médias ponderadas (ou *"weighted avg"*) das métricas obtidas em cada modelo ao prever sua saída (ou *"target"*). As métricas apresentadas serão: "Precisão", "Recall", "F1-Score"e "Acurracy".

Na tabela abaixo(tabela 22) é apresentado os resultados referente a predição da categoria do câncer.Nesta tabela, teremos uma visão de cada modelo prevendo as seguintes saídas: "M"e "B".*

* "Maligno (M)", "Benigno(B)".

Tabela 22 – Tabela de métricas dos modelos para Categorias de câncer 29

Modelo de CNN	Precisão	Recall	F1-Score	Acurácia
VGG19	95%	95%	95%	95%
GoogleNet	97%	97%	97%	97%
ResNet50	92%	91%	91%	91%

Fonte: Autor

Na tabela abaixo(tabela 23) é apresentado os resultados referente a predição da classe do câncer. Nesta tabela, teremos uma visão de cada modelo prevendo as seguintes saídas: "DC","LC","MC"e "PC"*

Tabela 23 – Tabela de métricas dos modelos para classe do câncer 29

Modelo de CNN	Precisão	Recall	F1-Score	Acurácia
VGG19	68%	81%	71%	72%
GoogleNet	92%	89%	89%	89%
ResNet50	89%	85%	86%	85%

Fonte: Autor

6.5 Combinação de Modelos

Após as execuções dos modelos, e tendo visto separadamente cada avaliação percebemos que os melhores modelos até então são, o GoogleNet(InterceptionV3) para categoria e GoogleNet(InterceptionV3) para classe.

Agora realizaremos a combinação dos modelos, para definirmos a conjunção final do Projeto ou confirmamos o que já tínhamos encontrado. Em um primeiro momento descartaremos os Modelos "ResNet Categoria"e "VGG19 Classes", por não atingirem resultados mínimos esperados.

Primeira conjunto de combinações a serem realizadas, serão as combinações entre "VGG19 Categoria"com o modelo "GoogleNet Classe". Essa combinação será chamada: "Combinação 1". A segunda combinação será entre "VGG19 Categoria"e o modelo "ResNet Classe". Essa combinação receberá o nome de "Combinação 2".

O segundo conjunto de combinações, terá como base o modelo "GoogleNet Categoria". A "Combinação 3"será a junção dos modelos "GoogleNet Categoria"com o modelo "ResNet Classe"e por último a "Combinação 4"composta por "GoogleNet Categoria"e "GoogleNet Classe".

Podemos observar na tabela 24 as métricas obtidas de "cada combinação".

A parti dos resultados apresentados no decorrer do capítulo, é possível perceber que a melhor combinação de modelos é a "Combinação 4", como já tínhamos analisado anteriormente. Mesmo com o desbalanceamento de classes, foi encontrado Resultados suficientemente bons para aplicarmos em nosso modelo final.

Conseguirmos concluir o modelo com uma métrica média de F1-Score em 90%,

* "carcinoma (DC)", "carcinoma lobular(CL)", "Carcinomamucinoso (MC)"e "Carcinoma papilífero (CP)"

Tabela 24 – Tabela de métricas das combinações

Modelo de CNN	Precisão	Recall	F1-Score	Acurácia
Combinação 1	88%	90%	88%	90%
Combinação 2	86%	88%	87%	92%
Combinação 3	89%	90%	89%	90%
Combinação 4	91%	92%	90%	92%

Fonte: Autor

métrica obtida através de uma média simples entre o modelo de categoria e de classe, um resultado bom considerando nossa base de dados e modelos, e uma acurácia média de 92%, considerando também uma média simples entre o modelo de categoria e de classe.

Conforme os dados anteriores, conseguimos encontrar uma combinação, considerada até onde podemos ver, "ótima"para nosso modelo. Uma combinação onde poderemos encontrar uma taxa de sucesso de 90%.

Com o modelo descrito nesse trabalho, poderemos ajudar a diminuir a taxa de falsos negativos, o que trazem o risco de um diagnóstico tardio da doença o que aumenta o risco de óbito, devido ao câncer, sendo esse o quarto em questão de mortalidade.

7 TRABALHOS FUTUROS

8 LINKS

A seguir, estão os links do vídeo de apresentação e do repositório contendo os dados utilizados no projeto.

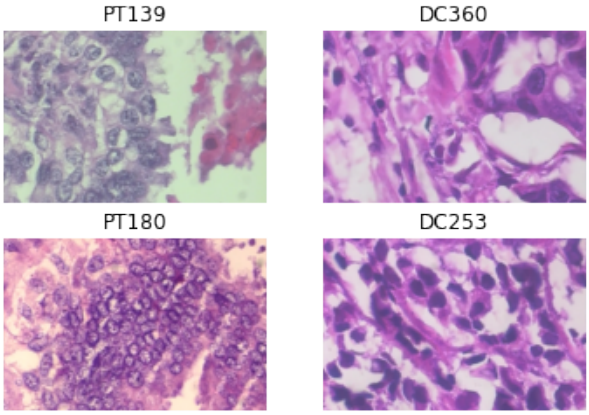
Link para base de dados(imagens de teste): <https://drive.google.com/drive/folders/1VQpWhlzK__2EUfd4CMeUWQ8-wAHaWGeT?usp=sharing>

Link para repositório: <https://github.com/RodrigoOBC/TCC-CienciaDados>

Apresentação:

9 FIGURAS / GRAFICOS

Figura 39 – Amostra com novos nomes, ampliação 400x



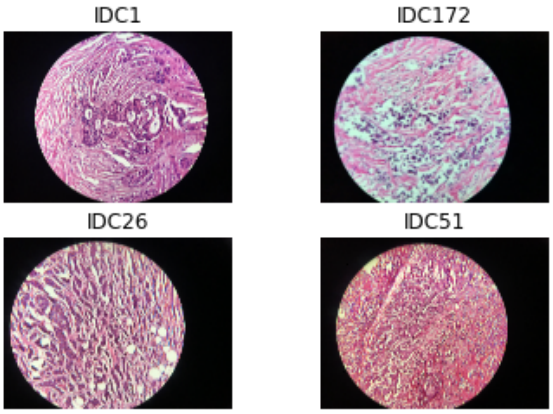
Fonte: Autor

Figura 40 – Amostra com nomes de imagens (Jupyter notebook)

```
01_BC_G1_9057_10x_1.JPG
02_BC_G1_9103_10x_1.JPG
03_BC_G1_10959_10x_1.JPG
04_BC_G1_10783_10x_1.JPG
05_BC_G1_9610_10x_1.JPG
06_BC_G1_7967_10x_1.JPG
01_BC_G1_9057_20x_1.JPG
01_BC_G1_9057_40x_1.JPG
01_BC_G1_9057_4x_1.jpg
```

Fonte: Autor

Figura 41 – Amostra da base dados Breast invasive ductal carcinomas (Jupyter notebook)



Fonte: Autor

Figura 42 – Código para pegar nome da figura (Google Colab)

```
def montar_nome(path):
    nome_file = []
    file_path = []
    x = 0
    for p, _, files in os.walk(os.path.abspath(path)):
        for file_name in files:
            if 'ResNet'  in file_name or "VGG19" in file_name or "Interception" in file_n
              continue
            else:
              nome_file.append(file_name)
              file_path.append(p+'//'+file_name)
    return nome_file, file_path
```

Fonte: Autor

Figura 43 – Código para pegar grau de ampliação (Jupyter notebook)

```
def pegar_ampliacao(file_path):
    ampliacao = []
    for path in file_path:
        if '40x' in path:
            ampliacao.append('40x')
        elif '100x' in path:
            ampliacao.append('100x')
        elif '200x' in path:
            ampliacao.append('200x')
        elif '400x' in path:
            ampliacao.append('400x')
    return ampliacao
```

Fonte: Autor

Figura 44 – Código para selecionar a classe (Google Colab)

```
def selecionar_classe(nomes):
    classe = []
    for nome in nomes:
        nome = ''.join([i for i in nome if not i.isdigit()])
        nome = nome.split('.')[0]
        if "TA" == nome:
            classe.append('TA')
        elif "DCI" == nome:
            classe.append('DCI')
        elif "F" == nome:
            classe.append('F')
        elif "PT" == nome:
            classe.append('PT')
        elif "DC" == nome:
            classe.append('DC')
        elif "LC" == nome:
            classe.append('LC')
        elif "MC" == nome:
            classe.append('MC')
        elif "PC" == nome:
            classe.append('PC')
        elif 'A' == nome:
            classe.append('A')
    return classe
```

Fonte: Autor

Figura 45 – Código para pegar os tipos (Jupyter notebook)

```
def pegar_tipos(file_path):
    tipo = []
    for path in file_path:
        path = path.split('/')[7]
        if 'B' in path:
            tipo.append('B')
        if 'M' in path:
            tipo.append('M')
    return tipo
```

Fonte: Autor

10 CÓDIGO FINAL

#Fluxo Independente

Para realizar o download da base de dados é só entrar no seguinte link:

https://drive.google.com/drive/folders/1VQpWhlzK__2EUfd4CMeUWQ8-wAHaWGeT?usp=sharing

Inicialmente, importaremos as Bibliotecas necessárias para a criação do modelo.A principais bibliotecas serão"TensorFlow"para criação das estruturas da "RedeNeural"e"Pandas"para criação e manipulação dos"DataFremes"

0.1 Importa Bibliotecas

[1]:
```
import re
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
import seaborn as sns
from tensorflow.keras.models import *
from tensorflow.keras.preprocessing.image import *
from tensorflow.keras.callbacks import *
from tensorflow.keras import layers, models, metrics
from tensorflow.keras.applications.efficientnet import *
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet50 import ResNet50
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import precision_score, recall_score,f1_score
import cv2
from glob import glob
from random import shuffle
import tensorflow as tf
import numpy as np
import pandas as pd
from pathlib import Path
import os.path
```

##Declaração de constantes

Declaração de algumas constantes, como: caminho principal dos dados, tamanho das imagens, o "BatchSize" e o número de "Epocas" padrão no treinamento. Usamos por padrão "25 épocas" e formos aumentando conforme o necessário

[2]:
```
AUTOTUNE = tf.data.experimental.AUTOTUNE
GCS_PATH = r'/content/drive/MyDrive/TCC'
BATCH_SIZE = 32
IMAGE_SIZE = [224, 224]
```

```
EPOCHS = 25
```

0.2 Criação dos DataFremes

[3]:
```
def montar_nome(path):
    nome_file = []
    file_path = []
    x = 0
    for p, _, files in os.walk(os.path.abspath(path)):
        for file_name in files:
            nome_file.append(file_name)
            file_path.append(p+'//'+file_name)
    return nome_file, file_path

def selecionar_classe(nomes):
    classe = []
    for nome in nomes:
        nome = ''.join([i for i in nome if not i.isdigit()])
        nome = nome.split('.')[0]
        if "TA" == nome:
            classe.append('TA')
        elif "DCI" == nome:
            classe.append('DCI')
        elif "F" == nome:
            classe.append('F')
        elif "PT" == nome:
            classe.append('PT')
        elif "DC" == nome:
            classe.append('DC')
        elif "LC" == nome:
            classe.append('LC')
        elif "MC" == nome:
            classe.append('MC')
        elif "PC" == nome:
            classe.append('PC')
        elif 'A' == nome:
            classe.append('A')
    return classe

def pegar_tipos(file_path):
    tipo = []
    for path in file_path:
        path = path.split('/')[8]
        if 'B' in path:
            tipo.append('B')
        if 'M' in path:
            tipo.append('M')
```

```
    return tipo

def pegar_ampliacao(file_path):
    ampliacao = []
    for path in file_path:
        if '40x' in path:
            ampliacao.append('40x')
        elif '100x' in path:
            ampliacao.append('100x')
        elif '200x' in path:
            ampliacao.append('200x')
        elif '400x' in path:
            ampliacao.append('400x')
    return ampliacao

def pegar_origem(file_path):
    origem = []
    for path in file_path:
        if 'treino' in path:
            origem.append('Te')
        elif 'Teste' in path:
            origem.append('Tr')
    return origem

def montar_data_frame(path):
    nome_file, file_path = montar_nome(path)
    classes = selecionar_classe(nome_file)
    tipos = pegar_tipos(file_path)
    ampliacaos = pegar_ampliacao(file_path)
    origem = pegar_origem(file_path)
    data = {'FikePath': file_path, 'Type': tipos, 'classes':
 ↪classes,'Magnification':ampliacaos, 'DataSet':origem}
    df = pd.DataFrame.from_dict(data, orient='index')
    df = df.transpose()
    return df
```

[4]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_treino = df_geral[df_geral['DataSet'] == 'Tr']
df_teste = df_geral[df_geral['DataSet'] == 'Te']
```

[5]:
```
df_treino.head()
```

[5]:
```
                                          FikePath Type  ... Magnification
DataSet
0  /content/drive/MyDrive/TCC/Imagens/400x/400x_T...    M  ...          400x
Tr
1  /content/drive/MyDrive/TCC/Imagens/400x/400x_T...    M  ...          400x
```

```
Tr
2  /content/drive/MyDrive/TCC/Imagens/400x/400x_T...    M  ...          400x
Tr
3  /content/drive/MyDrive/TCC/Imagens/400x/400x_T...    M  ...          400x
Tr
4  /content/drive/MyDrive/TCC/Imagens/400x/400x_T...    M  ...          400x
Tr

[5 rows x 5 columns]
```

[6]:
```
df_teste.head()
```

[6]:
```
                                          FikePath  ... DataSet
786  /content/drive/MyDrive/TCC/Imagens/400x/400x_t...  ...      Te
787  /content/drive/MyDrive/TCC/Imagens/400x/400x_t...  ...      Te
788  /content/drive/MyDrive/TCC/Imagens/400x/400x_t...  ...      Te
789  /content/drive/MyDrive/TCC/Imagens/400x/400x_t...  ...      Te
790  /content/drive/MyDrive/TCC/Imagens/400x/400x_t...  ...      Te

[5 rows x 5 columns]
```

0.3 Segmentação do Dataframe

Com o *"dataframe"* conseguiremos segmentar os dados de acordo com sua *"Magnification"*.

Separando para 200x e para 400x

[7]:
```
df_treino_400 = df_treino[df_treino['Magnification'] == '400x']
df_treino_400 = df_treino_400.drop(columns=['Magnification'])

df_teste_400 = df_teste[df_teste['Magnification'] == '400x']
df_teste_400 = df_teste_400.drop(columns=['Magnification'])



df_treino_200 = df_treino[df_treino['Magnification'] == '200x']
df_treino_200 = df_treino_200.drop(columns=['Magnification'])

df_teste_200 = df_teste[df_teste['Magnification'] == '200x']
df_teste_200 = df_teste_200.drop(columns=['Magnification'])
```

[8]:
```
df_treino_400_classe = df_treino[df_treino['Magnification'] ==␣
 ↪'400x'][df_treino['Type'] == 'M']
df_treino_400_classe = df_treino_400_classe.drop(columns=['Magnification'])

df_teste_400_classe = df_teste[df_teste['Magnification'] ==␣
 ↪'400x'][df_teste['Type'] == 'M']
df_teste_400_classe = df_teste_400_classe.drop(columns=['Magnification'])
```

```
df_treino_200_classe = df_treino[df_treino['Magnification'] ==␣
↪'200x'][df_treino['Type'] == 'M']
df_treino_200_classe = df_treino_200_classe.drop(columns=['Magnification'])

df_teste_200_classe  = df_teste[df_teste['Magnification'] ==␣
↪'200x'][df_teste['Type'] == 'M']
df_teste_200_classe = df_teste_200_classe.drop(columns=['Magnification'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if sys.path[0] == '':
```

1 Criação dos datas Generatos

Todos os 3 ImageDataGenerator terão em comum as adiçoes de rotação em 10%, a de adição de zoom em 20%. No caso de validação será sempre 20% do dado de treinamento

1.0.1 ImageDataGenerator para VGG19

[9]:
```
train_generator_VGG19 = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg19.preprocess_input,
    rotation_range=0.1,
    zoom_range=0.2,
    validation_split=0.2
)
test_generator_VGG19= tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg19.preprocess_input
)
```

1.0.2 ImageDataGenerator para GoogleNet

[10]:
```
train_generator_googleNet = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
    rotation_range=0.1,
```

```
    zoom_range=0.2,
    validation_split=0.2
)

test_generator_googleNet = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
)
```

1.0.3 ImageDataGenerator para ResNet50

[11]:
```
train_generator_ResNet = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
    rotation_range=0.1,
    zoom_range=0.2,
    validation_split=0.2
)

test_generator_ResNet  = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
)
```

2 Execução dos Modelos

Nessa seção apresentarei o "passo a passo" da execução dos modelos. Começare-mos apresentando como foi realizado a criação do modelo *VGG19,* logo em seguidaserá apresentado a criação do modelo *InceptionV3* e por fim a utilização da arquitetura *RESNET50.*

2.1 Modelo Tipo de Cancer

Na primeira sessão de treinamento como falado antes, prepararemos a nossa redeneural para identificar um câncer maligno. Seguiremos o seguinte passo-a-passo para esse treinamento.

[ ]:
```
def build_model(Tipo_modelo):
  if Tipo_modelo == "VGG19":
    base_VGG19 = tf.keras.applications.VGG19(weights = 'imagenet', include_top =␣
 ↪False)
    x = base_VGG19.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
 ↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(2, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_VGG19.input, outputs = preds)
  elif Tipo_modelo == "GoogleNet":
```

```
    base_Interception = InceptionV3(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(2, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)
  elif Tipo_modelo == "Resnet":
    base_RESNET = tf.keras.applications.ResNet50(weights = 'imagenet',␣
↪include_top = False)
    x = base_RESNET.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(2, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_RESNET.input, outputs = preds)

  return model
```

2.1.1 Modelo VGG19 Para grau de ampliação 400

[ ]:
```
train_images_VGG19_MB = train_generator_VGG19.flow_from_dataframe(
    dataframe=df_treino_400,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images_VGG19_MB = train_generator_VGG19.flow_from_dataframe(
    dataframe=df_treino_400,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
```

```
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images_VGG19_MB = test_generator_VGG19.flow_from_dataframe(
    dataframe=df_teste_400,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 629 validated image filenames belonging to 2 classes.
Found 157 validated image filenames belonging to 2 classes.
Found 978 validated image filenames belonging to 2 classes.
```

[ ]:
```
VGG19 = build_model("VGG19")
VGG19.load_weights('/content/drive/MyDrive/TCC/weights/VGG19/VGG_MB_com_pesos.
 ↪h5')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 0s 0us/step
80150528/80134624 [==============================] - 0s 0us/step
```

[ ]:
```
opt = Adam(learning_rate=0.0001)
VGG19.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

[ ]:
```
pred = VGG19.predict(test_images_VGG19_MB)
pred=np.argmax(pred,axis=1)
```

[ ]:
```
pred_df_400=df_teste_400.copy()
labels={}
for l,v in test_images_VGG19_MB.class_indices.items():
    labels.update({v:l})
pred_df_400['pred']=pred
pred_df_400['pred']=pred_df_400['pred'].apply(lambda x: labels[x])
```

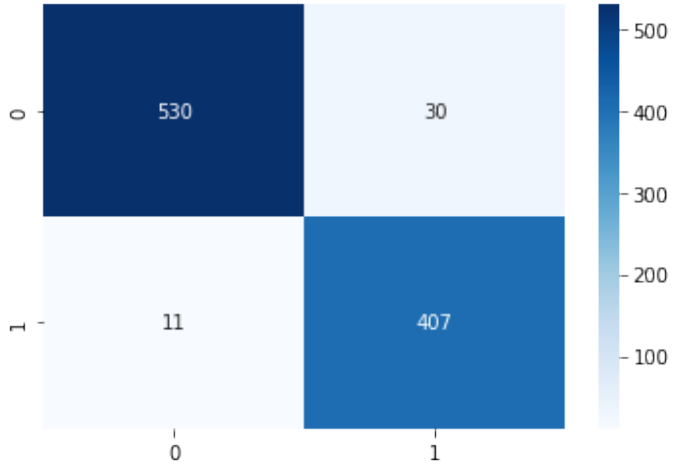
[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df_400['Type'],pred_df_400['pred'])}")
```

```
Metrics Score:                precision    recall  f1-score   support

           B       0.98      0.95      0.96       560
           M       0.93      0.97      0.95       418

    accuracy                           0.96       978
   macro avg       0.96      0.96      0.96       978
weighted avg       0.96      0.96      0.96       978
```

[ ]:
```
print(f"Accuracy Score:␣
↪{accuracy_score(pred_df_400['Type'],pred_df_400['pred'])}")
```

```
Accuracy Score: 0.9580777096114519
```

[ ]:
```
sns.heatmap(confusion_matrix(pred_df_400['Type'],pred_df_400['pred']),␣
↪annot=True, fmt='2d',  cmap="Blues");
```



2.1.2 Modelo VGG19 Para grau de ampliação 200

[ ]:
```
train_images_VGG19_MB = train_generator_VGG19.flow_from_dataframe(
    dataframe=df_treino_200,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
```

```
    class_mode='categorical',
    batch_size=16,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images_VGG19_MB = train_generator_VGG19.flow_from_dataframe(
    dataframe=df_treino_200,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images_VGG19_MB = test_generator_VGG19.flow_from_dataframe(
    dataframe=df_teste_200,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 630 validated image filenames belonging to 2 classes.
Found 157 validated image filenames belonging to 2 classes.
Found 856 validated image filenames belonging to 2 classes.
```

[ ]:
```
VGG19 = build_model("VGG19")
VGG19.load_weights('/content/drive/MyDrive/TCC/weights/VGG19/VGG_MB_com_pesos.
 ↪h5')
opt = Adam(learning_rate=0.0001)
VGG19.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

[ ]:
```
pred = VGG19.predict(test_images_VGG19_MB)
pred=np.argmax(pred,axis=1)
```

[ ]:
```
pred_df_200=df_teste_200.copy()
labels={}
```

```
for l,v in test_images_VGG19_MB.class_indices.items():
    labels.update({v:l})
pred_df_200['pred']=pred
pred_df_200['pred']=pred_df_200['pred'].apply(lambda x: labels[x])
```

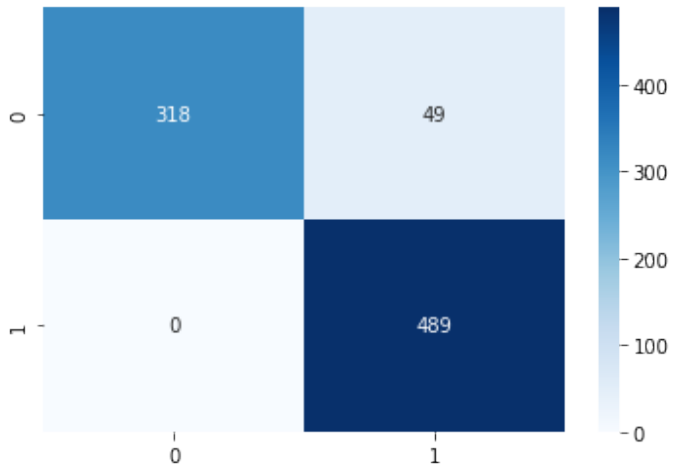
[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df_200['Type'],pred_df_200['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       1.00      0.87      0.93       367
           M       0.91      1.00      0.95       489

    accuracy                           0.94       856
   macro avg       0.95      0.93      0.94       856
weighted avg       0.95      0.94      0.94       856
```

[ ]:
```
print(f"Accuracy Score:␣
↪{accuracy_score(pred_df_200['Type'],pred_df_200['pred'])}")
```

```
Accuracy Score: 0.9427570093457944
```

[ ]:
```
sns.heatmap(confusion_matrix(pred_df_200['Type'],pred_df_200['pred']),␣
↪annot=True, fmt='2d',  cmap="Blues");
```



2.1.3 Modelo VGG19 Validação final

[ ]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_treino = df_geral[df_geral['DataSet'] == 'Tr']
df_teste = df_geral[df_geral['DataSet'] == 'Te']
```

[ ]:
```
print(f'quantidade de dados na base de Teste {df_teste.shape[0]}')
print(f'quantidade de dados na base de Treino {df_treino.shape[0]}')
```

```
quantidade de dados na base de Teste 1834
quantidade de dados na base de Treino 1573
```

[ ]:
```
test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg19.preprocess_input
)
```

[ ]:
```
test_images = test_generator.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 1834 validated image filenames belonging to 2 classes.
```

[ ]:
```
opt = Adam(learning_rate=0.0001)
VGG19_rede_Geral = build_model('VGG19')
checkpoint_path = r'/content/drive/MyDrive/TCC/weights/VGG19/VGG_MB_com_pesos.h5'
VGG19_rede_Geral.load_weights(checkpoint_path)
VGG19_rede_Geral.compile(loss='categorical_crossentropy', optimizer=opt, metrics␣
↪= ['accuracy'])
```

[ ]:
```
pred = VGG19_rede_Geral.predict(test_images)
pred=np.argmax(pred,axis=1)
```

[ ]:
```
pred_df=df_teste.copy()
labels={}
for l,v in test_images.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score: {classification_report(pred_df['Type'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support
```

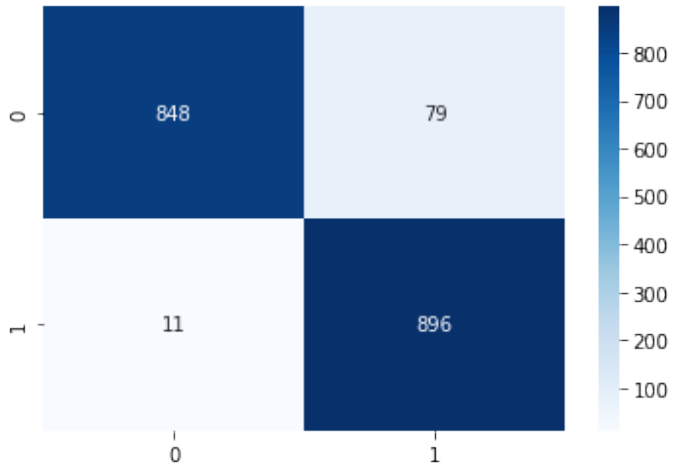
```
           B       0.99      0.91      0.95       927
           M       0.92      0.99      0.95       907

    accuracy                           0.95      1834
   macro avg       0.95      0.95      0.95      1834
weighted avg       0.95      0.95      0.95      1834
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
```

```
Accuracy Score: 0.95092693565976
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['Type'],pred_df['pred']), annot=True,␣
↪fmt='2d',  cmap="Blues");
```

```
Accuracy Score: 0.95092693565976
```



[ ]:
```

```

2.1.4 Modelo GoogleNet Para grau de ampliação 400

[ ]:
```
train_images_googlent_MB = train_generator_googleNet.flow_from_dataframe(
    dataframe=df_treino_400,
    x_col='FikePath',
    y_col='Type',
```

```
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images_googlent_MB = train_generator_googleNet.flow_from_dataframe(
    dataframe=df_treino_400,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images_googlent_MB = test_generator_googleNet.flow_from_dataframe(
    dataframe=df_teste_400,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
Found 629 validated image filenames belonging to 2 classes.
Found 157 validated image filenames belonging to 2 classes.
Found 978 validated image filenames belonging to 2 classes.
```

[ ]:
```
Interception = build_model("GoogleNet")
Interception.load_weights(filepath=r'/content/drive/MyDrive/TCC/weights/
 ↪Interception/Interception_MB_com_pesos.h5')
Interception.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
87924736/87910968 [==============================] - 1s 0us/step
Model: "model_3"
__________________________________________________________________________________
```

```
------------------
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 input_4 (InputLayer)           [(None, None, None,  0           []
                                 3)]

 conv2d (Conv2D)                (None, None, None,   864
['input_4[0][0]']
                                32)

 batch_normalization (BatchNorm  (None, None, None,   96
['conv2d[0][0]']
 alization)                     32)

 activation (Activation)        (None, None, None,   0
['batch_normalization[0][0]']
                                32)

 conv2d_1 (Conv2D)              (None, None, None,   9216
['activation[0][0]']
                                32)

 batch_normalization_1 (BatchNo  (None, None, None,   96
['conv2d_1[0][0]']
 rmalization)                   32)

 activation_1 (Activation)      (None, None, None,   0
['batch_normalization_1[0][0]']
                                32)

 conv2d_2 (Conv2D)              (None, None, None,   18432
['activation_1[0][0]']
                                64)

 batch_normalization_2 (BatchNo  (None, None, None,   192
['conv2d_2[0][0]']
 rmalization)                   64)

 activation_2 (Activation)      (None, None, None,   0
['batch_normalization_2[0][0]']
                                64)

 max_pooling2d (MaxPooling2D)   (None, None, None,   0
['activation_2[0][0]']
                                64)

 conv2d_3 (Conv2D)              (None, None, None,   5120
```

```
['max_pooling2d[0][0]']
                                 80)

 batch_normalization_3 (BatchNo  (None, None, None,   240
['conv2d_3[0][0]']
 rmalization)                    80)

 activation_3 (Activation)       (None, None, None,   0
['batch_normalization_3[0][0]']
                                 80)

 conv2d_4 (Conv2D)               (None, None, None,   138240
['activation_3[0][0]']
                                 192)

 batch_normalization_4 (BatchNo  (None, None, None,   576
['conv2d_4[0][0]']
 rmalization)                    192)

 activation_4 (Activation)       (None, None, None,   0
['batch_normalization_4[0][0]']
                                 192)

 max_pooling2d_1 (MaxPooling2D)  (None, None, None,   0
['activation_4[0][0]']
                                 192)

 conv2d_8 (Conv2D)               (None, None, None,   12288
['max_pooling2d_1[0][0]']
                                 64)

 batch_normalization_8 (BatchNo  (None, None, None,   192
['conv2d_8[0][0]']
 rmalization)                    64)

 activation_8 (Activation)       (None, None, None,   0
['batch_normalization_8[0][0]']
                                 64)

 conv2d_6 (Conv2D)               (None, None, None,   9216
['max_pooling2d_1[0][0]']
                                 48)

 conv2d_9 (Conv2D)               (None, None, None,   55296
['activation_8[0][0]']
                                 96)

 batch_normalization_6 (BatchNo  (None, None, None,   144
```

```
['conv2d_6[0][0]']
 rmalization)                   48)

 batch_normalization_9 (BatchNo  (None, None, None,   288
['conv2d_9[0][0]']
 rmalization)                   96)

 activation_6 (Activation)      (None, None, None,   0
['batch_normalization_6[0][0]']
                                48)

 activation_9 (Activation)      (None, None, None,   0
['batch_normalization_9[0][0]']
                                96)

 average_pooling2d (AveragePool  (None, None, None,   0
['max_pooling2d_1[0][0]']
 ing2D)                         192)

 conv2d_5 (Conv2D)              (None, None, None,   12288
['max_pooling2d_1[0][0]']
                                64)

 conv2d_7 (Conv2D)              (None, None, None,   76800
['activation_6[0][0]']
                                64)

 conv2d_10 (Conv2D)             (None, None, None,   82944
['activation_9[0][0]']
                                96)

 conv2d_11 (Conv2D)             (None, None, None,   6144
['average_pooling2d[0][0]']
                                32)

 batch_normalization_5 (BatchNo  (None, None, None,   192
['conv2d_5[0][0]']
 rmalization)                   64)

 batch_normalization_7 (BatchNo  (None, None, None,   192
['conv2d_7[0][0]']
 rmalization)                   64)

 batch_normalization_10 (BatchN  (None, None, None,   288
['conv2d_10[0][0]']
 ormalization)                  96)

 batch_normalization_11 (BatchN  (None, None, None,   96
```

```
['conv2d_11[0][0]']
 ormalization)                  32)

 activation_5 (Activation)      (None, None, None,   0
['batch_normalization_5[0][0]']
                                64)

 activation_7 (Activation)      (None, None, None,   0
['batch_normalization_7[0][0]']
                                64)

 activation_10 (Activation)     (None, None, None,   0
['batch_normalization_10[0][0]']
                                96)

 activation_11 (Activation)     (None, None, None,   0
['batch_normalization_11[0][0]']
                                32)

 mixed0 (Concatenate)           (None, None, None,   0
['activation_5[0][0]',
                                256)
'activation_7[0][0]',
'activation_10[0][0]',
'activation_11[0][0]']

 conv2d_15 (Conv2D)             (None, None, None,   16384
['mixed0[0][0]']
                                64)

 batch_normalization_15 (BatchN  (None, None, None,   192
['conv2d_15[0][0]']
 ormalization)                  64)

 activation_15 (Activation)     (None, None, None,   0
['batch_normalization_15[0][0]']
                                64)

 conv2d_13 (Conv2D)             (None, None, None,   12288
['mixed0[0][0]']
                                48)

 conv2d_16 (Conv2D)             (None, None, None,   55296
['activation_15[0][0]']
                                96)

 batch_normalization_13 (BatchN  (None, None, None,   144
['conv2d_13[0][0]']
```

```
 ormalization)                  48)

 batch_normalization_16 (BatchN  (None, None, None,   288
['conv2d_16[0][0]']
 ormalization)                  96)

 activation_13 (Activation)     (None, None, None,   0
['batch_normalization_13[0][0]']
                                48)

 activation_16 (Activation)     (None, None, None,   0
['batch_normalization_16[0][0]']
                                96)

 average_pooling2d_1 (AveragePo  (None, None, None,   0
['mixed0[0][0]']
 oling2D)                       256)

 conv2d_12 (Conv2D)             (None, None, None,   16384
['mixed0[0][0]']
                                64)

 conv2d_14 (Conv2D)             (None, None, None,   76800
['activation_13[0][0]']
                                64)

 conv2d_17 (Conv2D)             (None, None, None,   82944
['activation_16[0][0]']
                                96)

 conv2d_18 (Conv2D)             (None, None, None,   16384
['average_pooling2d_1[0][0]']
                                64)

 batch_normalization_12 (BatchN  (None, None, None,   192
['conv2d_12[0][0]']
 ormalization)                  64)

 batch_normalization_14 (BatchN  (None, None, None,   192
['conv2d_14[0][0]']
 ormalization)                  64)

 batch_normalization_17 (BatchN  (None, None, None,   288
['conv2d_17[0][0]']
 ormalization)                  96)

 batch_normalization_18 (BatchN  (None, None, None,   192
['conv2d_18[0][0]']
```

```
 ormalization)                  64)

 activation_12 (Activation)     (None, None, None,   0
['batch_normalization_12[0][0]']
                                64)

 activation_14 (Activation)     (None, None, None,   0
['batch_normalization_14[0][0]']
                                64)

 activation_17 (Activation)     (None, None, None,   0
['batch_normalization_17[0][0]']
                                96)

 activation_18 (Activation)     (None, None, None,   0
['batch_normalization_18[0][0]']
                                64)

 mixed1 (Concatenate)           (None, None, None,   0
['activation_12[0][0]',
                                288)
'activation_14[0][0]',
'activation_17[0][0]',
'activation_18[0][0]']

 conv2d_22 (Conv2D)             (None, None, None,   18432
['mixed1[0][0]']
                                64)

 batch_normalization_22 (BatchN  (None, None, None,   192
['conv2d_22[0][0]']
 ormalization)                  64)

 activation_22 (Activation)     (None, None, None,   0
['batch_normalization_22[0][0]']
                                64)

 conv2d_20 (Conv2D)             (None, None, None,   13824
['mixed1[0][0]']
                                48)

 conv2d_23 (Conv2D)             (None, None, None,   55296
['activation_22[0][0]']
                                96)

 batch_normalization_20 (BatchN  (None, None, None,   144
['conv2d_20[0][0]']
 ormalization)                  48)
```

```
 batch_normalization_23 (BatchN  (None, None, None,   288
['conv2d_23[0][0]']
 ormalization)                  96)

 activation_20 (Activation)     (None, None, None,   0
['batch_normalization_20[0][0]']
                                48)

 activation_23 (Activation)     (None, None, None,   0
['batch_normalization_23[0][0]']
                                96)

 average_pooling2d_2 (AveragePo  (None, None, None,   0
['mixed1[0][0]']
 oling2D)                       288)

 conv2d_19 (Conv2D)             (None, None, None,   18432
['mixed1[0][0]']
                                64)

 conv2d_21 (Conv2D)             (None, None, None,   76800
['activation_20[0][0]']
                                64)

 conv2d_24 (Conv2D)             (None, None, None,   82944
['activation_23[0][0]']
                                96)

 conv2d_25 (Conv2D)             (None, None, None,   18432
['average_pooling2d_2[0][0]']
                                64)

 batch_normalization_19 (BatchN  (None, None, None,   192
['conv2d_19[0][0]']
 ormalization)                  64)

 batch_normalization_21 (BatchN  (None, None, None,   192
['conv2d_21[0][0]']
 ormalization)                  64)

 batch_normalization_24 (BatchN  (None, None, None,   288
['conv2d_24[0][0]']
 ormalization)                  96)

 batch_normalization_25 (BatchN  (None, None, None,   192
['conv2d_25[0][0]']
 ormalization)                  64)
```

```
 activation_19 (Activation)     (None, None, None,   0
['batch_normalization_19[0][0]']
                                64)

 activation_21 (Activation)     (None, None, None,   0
['batch_normalization_21[0][0]']
                                64)

 activation_24 (Activation)     (None, None, None,   0
['batch_normalization_24[0][0]']
                                96)

 activation_25 (Activation)     (None, None, None,   0
['batch_normalization_25[0][0]']
                                64)

 mixed2 (Concatenate)           (None, None, None,   0
['activation_19[0][0]',
                                288)
'activation_21[0][0]',
'activation_24[0][0]',
'activation_25[0][0]']

 conv2d_27 (Conv2D)             (None, None, None,   18432
['mixed2[0][0]']
                                64)

 batch_normalization_27 (BatchN  (None, None, None,   192
['conv2d_27[0][0]']
 ormalization)                  64)

 activation_27 (Activation)     (None, None, None,   0
['batch_normalization_27[0][0]']
                                64)

 conv2d_28 (Conv2D)             (None, None, None,   55296
['activation_27[0][0]']
                                96)

 batch_normalization_28 (BatchN  (None, None, None,   288
['conv2d_28[0][0]']
 ormalization)                  96)

 activation_28 (Activation)     (None, None, None,   0
['batch_normalization_28[0][0]']
                                96)
```

```
 conv2d_26 (Conv2D)             (None, None, None,   995328
['mixed2[0][0]']
                                384)

 conv2d_29 (Conv2D)             (None, None, None,   82944
['activation_28[0][0]']
                                96)

 batch_normalization_26 (BatchN  (None, None, None,   1152
['conv2d_26[0][0]']
 ormalization)                  384)

 batch_normalization_29 (BatchN  (None, None, None,   288
['conv2d_29[0][0]']
 ormalization)                  96)

 activation_26 (Activation)     (None, None, None,   0
['batch_normalization_26[0][0]']
                                384)

 activation_29 (Activation)     (None, None, None,   0
['batch_normalization_29[0][0]']
                                96)

 max_pooling2d_2 (MaxPooling2D)  (None, None, None,   0
['mixed2[0][0]']
                                288)

 mixed3 (Concatenate)           (None, None, None,   0
['activation_26[0][0]',
                                768)
'activation_29[0][0]',
'max_pooling2d_2[0][0]']

 conv2d_34 (Conv2D)             (None, None, None,   98304
['mixed3[0][0]']
                                128)

 batch_normalization_34 (BatchN  (None, None, None,   384
['conv2d_34[0][0]']
 ormalization)                  128)

 activation_34 (Activation)     (None, None, None,   0
['batch_normalization_34[0][0]']
                                128)

 conv2d_35 (Conv2D)             (None, None, None,   114688
['activation_34[0][0]']
```

```
                                 128)

 batch_normalization_35 (BatchN  (None, None, None,   384
['conv2d_35[0][0]']
 ormalization)                   128)

 activation_35 (Activation)      (None, None, None,   0
['batch_normalization_35[0][0]']
                                 128)

 conv2d_31 (Conv2D)              (None, None, None,   98304
['mixed3[0][0]']
                                 128)

 conv2d_36 (Conv2D)              (None, None, None,   114688
['activation_35[0][0]']
                                 128)

 batch_normalization_31 (BatchN  (None, None, None,   384
['conv2d_31[0][0]']
 ormalization)                   128)

 batch_normalization_36 (BatchN  (None, None, None,   384
['conv2d_36[0][0]']
 ormalization)                   128)

 activation_31 (Activation)      (None, None, None,   0
['batch_normalization_31[0][0]']
                                 128)

 activation_36 (Activation)      (None, None, None,   0
['batch_normalization_36[0][0]']
                                 128)

 conv2d_32 (Conv2D)              (None, None, None,   114688
['activation_31[0][0]']
                                 128)

 conv2d_37 (Conv2D)              (None, None, None,   114688
['activation_36[0][0]']
                                 128)

 batch_normalization_32 (BatchN  (None, None, None,   384
['conv2d_32[0][0]']
 ormalization)                   128)

 batch_normalization_37 (BatchN  (None, None, None,   384
['conv2d_37[0][0]']
```

```
 ormalization)                  128)

 activation_32 (Activation)     (None, None, None,   0
['batch_normalization_32[0][0]']
                                128)

 activation_37 (Activation)     (None, None, None,   0
['batch_normalization_37[0][0]']
                                128)

 average_pooling2d_3 (AveragePo  (None, None, None,   0
['mixed3[0][0]']
 oling2D)                       768)

 conv2d_30 (Conv2D)             (None, None, None,   147456
['mixed3[0][0]']
                                192)

 conv2d_33 (Conv2D)             (None, None, None,   172032
['activation_32[0][0]']
                                192)

 conv2d_38 (Conv2D)             (None, None, None,   172032
['activation_37[0][0]']
                                192)

 conv2d_39 (Conv2D)             (None, None, None,   147456
['average_pooling2d_3[0][0]']
                                192)

 batch_normalization_30 (BatchN  (None, None, None,   576
['conv2d_30[0][0]']
 ormalization)                  192)

 batch_normalization_33 (BatchN  (None, None, None,   576
['conv2d_33[0][0]']
 ormalization)                  192)

 batch_normalization_38 (BatchN  (None, None, None,   576
['conv2d_38[0][0]']
 ormalization)                  192)

 batch_normalization_39 (BatchN  (None, None, None,   576
['conv2d_39[0][0]']
 ormalization)                  192)

 activation_30 (Activation)     (None, None, None,   0
['batch_normalization_30[0][0]']
```

```
                                192)

 activation_33 (Activation)     (None, None, None,   0
['batch_normalization_33[0][0]']
                                192)

 activation_38 (Activation)     (None, None, None,   0
['batch_normalization_38[0][0]']
                                192)

 activation_39 (Activation)     (None, None, None,   0
['batch_normalization_39[0][0]']
                                192)

 mixed4 (Concatenate)           (None, None, None,   0
['activation_30[0][0]',
                                768)
'activation_33[0][0]',
'activation_38[0][0]',
'activation_39[0][0]']

 conv2d_44 (Conv2D)             (None, None, None,   122880
['mixed4[0][0]']
                                160)

 batch_normalization_44 (BatchN  (None, None, None,   480
['conv2d_44[0][0]']
 ormalization)                  160)

 activation_44 (Activation)     (None, None, None,   0
['batch_normalization_44[0][0]']
                                160)

 conv2d_45 (Conv2D)             (None, None, None,   179200
['activation_44[0][0]']
                                160)

 batch_normalization_45 (BatchN  (None, None, None,   480
['conv2d_45[0][0]']
 ormalization)                  160)

 activation_45 (Activation)     (None, None, None,   0
['batch_normalization_45[0][0]']
                                160)

 conv2d_41 (Conv2D)             (None, None, None,   122880
['mixed4[0][0]']
                                160)
```

```
 conv2d_46 (Conv2D)             (None, None, None,   179200
['activation_45[0][0]']
                                160)

 batch_normalization_41 (BatchN  (None, None, None,   480
['conv2d_41[0][0]']
 ormalization)                  160)

 batch_normalization_46 (BatchN  (None, None, None,   480
['conv2d_46[0][0]']
 ormalization)                  160)

 activation_41 (Activation)     (None, None, None,   0
['batch_normalization_41[0][0]']
                                160)

 activation_46 (Activation)     (None, None, None,   0
['batch_normalization_46[0][0]']
                                160)

 conv2d_42 (Conv2D)             (None, None, None,   179200
['activation_41[0][0]']
                                160)

 conv2d_47 (Conv2D)             (None, None, None,   179200
['activation_46[0][0]']
                                160)

 batch_normalization_42 (BatchN  (None, None, None,   480
['conv2d_42[0][0]']
 ormalization)                  160)

 batch_normalization_47 (BatchN  (None, None, None,   480
['conv2d_47[0][0]']
 ormalization)                  160)

 activation_42 (Activation)     (None, None, None,   0
['batch_normalization_42[0][0]']
                                160)

 activation_47 (Activation)     (None, None, None,   0
['batch_normalization_47[0][0]']
                                160)

 average_pooling2d_4 (AveragePo  (None, None, None,   0
['mixed4[0][0]']
 oling2D)                       768)
```

```
 conv2d_40 (Conv2D)             (None, None, None,   147456
['mixed4[0][0]']
                                192)

 conv2d_43 (Conv2D)             (None, None, None,   215040
['activation_42[0][0]']
                                192)

 conv2d_48 (Conv2D)             (None, None, None,   215040
['activation_47[0][0]']
                                192)

 conv2d_49 (Conv2D)             (None, None, None,   147456
['average_pooling2d_4[0][0]']
                                192)

 batch_normalization_40 (BatchN  (None, None, None,   576
['conv2d_40[0][0]']
 ormalization)                  192)

 batch_normalization_43 (BatchN  (None, None, None,   576
['conv2d_43[0][0]']
 ormalization)                  192)

 batch_normalization_48 (BatchN  (None, None, None,   576
['conv2d_48[0][0]']
 ormalization)                  192)

 batch_normalization_49 (BatchN  (None, None, None,   576
['conv2d_49[0][0]']
 ormalization)                  192)

 activation_40 (Activation)     (None, None, None,   0
['batch_normalization_40[0][0]']
                                192)

 activation_43 (Activation)     (None, None, None,   0
['batch_normalization_43[0][0]']
                                192)

 activation_48 (Activation)     (None, None, None,   0
['batch_normalization_48[0][0]']
                                192)

 activation_49 (Activation)     (None, None, None,   0
['batch_normalization_49[0][0]']
                                192)
```

```
 mixed5 (Concatenate)           (None, None, None,   0
['activation_40[0][0]',
                                768)
'activation_43[0][0]',
'activation_48[0][0]',
'activation_49[0][0]']

 conv2d_54 (Conv2D)             (None, None, None,   122880
['mixed5[0][0]']
                                160)

 batch_normalization_54 (BatchN  (None, None, None,   480
['conv2d_54[0][0]']
 ormalization)                  160)

 activation_54 (Activation)     (None, None, None,   0
['batch_normalization_54[0][0]']
                                160)

 conv2d_55 (Conv2D)             (None, None, None,   179200
['activation_54[0][0]']
                                160)

 batch_normalization_55 (BatchN  (None, None, None,   480
['conv2d_55[0][0]']
 ormalization)                  160)

 activation_55 (Activation)     (None, None, None,   0
['batch_normalization_55[0][0]']
                                160)

 conv2d_51 (Conv2D)             (None, None, None,   122880
['mixed5[0][0]']
                                160)

 conv2d_56 (Conv2D)             (None, None, None,   179200
['activation_55[0][0]']
                                160)

 batch_normalization_51 (BatchN  (None, None, None,   480
['conv2d_51[0][0]']
 ormalization)                  160)

 batch_normalization_56 (BatchN  (None, None, None,   480
['conv2d_56[0][0]']
 ormalization)                  160)
```

```
 activation_51 (Activation)     (None, None, None,   0
['batch_normalization_51[0][0]']
                                160)

 activation_56 (Activation)     (None, None, None,   0
['batch_normalization_56[0][0]']
                                160)

 conv2d_52 (Conv2D)             (None, None, None,   179200
['activation_51[0][0]']
                                160)

 conv2d_57 (Conv2D)             (None, None, None,   179200
['activation_56[0][0]']
                                160)

 batch_normalization_52 (BatchN  (None, None, None,   480
['conv2d_52[0][0]']
 ormalization)                  160)

 batch_normalization_57 (BatchN  (None, None, None,   480
['conv2d_57[0][0]']
 ormalization)                  160)

 activation_52 (Activation)     (None, None, None,   0
['batch_normalization_52[0][0]']
                                160)

 activation_57 (Activation)     (None, None, None,   0
['batch_normalization_57[0][0]']
                                160)

 average_pooling2d_5 (AveragePo  (None, None, None,   0
['mixed5[0][0]']
 oling2D)                       768)

 conv2d_50 (Conv2D)             (None, None, None,   147456
['mixed5[0][0]']
                                192)

 conv2d_53 (Conv2D)             (None, None, None,   215040
['activation_52[0][0]']
                                192)

 conv2d_58 (Conv2D)             (None, None, None,   215040
['activation_57[0][0]']
                                192)
```

```
 conv2d_59 (Conv2D)             (None, None, None,   147456
['average_pooling2d_5[0][0]']
                                192)

 batch_normalization_50 (BatchN  (None, None, None,   576
['conv2d_50[0][0]']
 ormalization)                  192)

 batch_normalization_53 (BatchN  (None, None, None,   576
['conv2d_53[0][0]']
 ormalization)                  192)

 batch_normalization_58 (BatchN  (None, None, None,   576
['conv2d_58[0][0]']
 ormalization)                  192)

 batch_normalization_59 (BatchN  (None, None, None,   576
['conv2d_59[0][0]']
 ormalization)                  192)

 activation_50 (Activation)     (None, None, None,   0
['batch_normalization_50[0][0]']
                                192)

 activation_53 (Activation)     (None, None, None,   0
['batch_normalization_53[0][0]']
                                192)

 activation_58 (Activation)     (None, None, None,   0
['batch_normalization_58[0][0]']
                                192)

 activation_59 (Activation)     (None, None, None,   0
['batch_normalization_59[0][0]']
                                192)

 mixed6 (Concatenate)           (None, None, None,   0
['activation_50[0][0]',
                                768)
'activation_53[0][0]',
'activation_58[0][0]',
'activation_59[0][0]']

 conv2d_64 (Conv2D)             (None, None, None,   147456
['mixed6[0][0]']
                                192)

 batch_normalization_64 (BatchN  (None, None, None,   576
```

```
['conv2d_64[0][0]']
 ormalization)                  192)

 activation_64 (Activation)     (None, None, None,   0
['batch_normalization_64[0][0]']
                                192)

 conv2d_65 (Conv2D)             (None, None, None,   258048
['activation_64[0][0]']
                                192)

 batch_normalization_65 (BatchN  (None, None, None,   576
['conv2d_65[0][0]']
 ormalization)                  192)

 activation_65 (Activation)     (None, None, None,   0
['batch_normalization_65[0][0]']
                                192)

 conv2d_61 (Conv2D)             (None, None, None,   147456
['mixed6[0][0]']
                                192)

 conv2d_66 (Conv2D)             (None, None, None,   258048
['activation_65[0][0]']
                                192)

 batch_normalization_61 (BatchN  (None, None, None,   576
['conv2d_61[0][0]']
 ormalization)                  192)

 batch_normalization_66 (BatchN  (None, None, None,   576
['conv2d_66[0][0]']
 ormalization)                  192)

 activation_61 (Activation)     (None, None, None,   0
['batch_normalization_61[0][0]']
                                192)

 activation_66 (Activation)     (None, None, None,   0
['batch_normalization_66[0][0]']
                                192)

 conv2d_62 (Conv2D)             (None, None, None,   258048
['activation_61[0][0]']
                                192)

 conv2d_67 (Conv2D)             (None, None, None,   258048
```

```
['activation_66[0][0]']
                                192)

 batch_normalization_62 (BatchN  (None, None, None,   576
['conv2d_62[0][0]']
 ormalization)                  192)

 batch_normalization_67 (BatchN  (None, None, None,   576
['conv2d_67[0][0]']
 ormalization)                  192)

 activation_62 (Activation)     (None, None, None,   0
['batch_normalization_62[0][0]']
                                192)

 activation_67 (Activation)     (None, None, None,   0
['batch_normalization_67[0][0]']
                                192)

 average_pooling2d_6 (AveragePo  (None, None, None,   0
['mixed6[0][0]']
 oling2D)                       768)

 conv2d_60 (Conv2D)             (None, None, None,   147456
['mixed6[0][0]']
                                192)

 conv2d_63 (Conv2D)             (None, None, None,   258048
['activation_62[0][0]']
                                192)

 conv2d_68 (Conv2D)             (None, None, None,   258048
['activation_67[0][0]']
                                192)

 conv2d_69 (Conv2D)             (None, None, None,   147456
['average_pooling2d_6[0][0]']
                                192)

 batch_normalization_60 (BatchN  (None, None, None,   576
['conv2d_60[0][0]']
 ormalization)                  192)

 batch_normalization_63 (BatchN  (None, None, None,   576
['conv2d_63[0][0]']
 ormalization)                  192)

 batch_normalization_68 (BatchN  (None, None, None,   576
```

```
['conv2d_68[0][0]']
 ormalization)                  192)

 batch_normalization_69 (BatchN  (None, None, None,   576
['conv2d_69[0][0]']
 ormalization)                  192)

 activation_60 (Activation)     (None, None, None,   0
['batch_normalization_60[0][0]']
                                192)

 activation_63 (Activation)     (None, None, None,   0
['batch_normalization_63[0][0]']
                                192)

 activation_68 (Activation)     (None, None, None,   0
['batch_normalization_68[0][0]']
                                192)

 activation_69 (Activation)     (None, None, None,   0
['batch_normalization_69[0][0]']
                                192)

 mixed7 (Concatenate)           (None, None, None,   0
['activation_60[0][0]',
                                768)
'activation_63[0][0]',
'activation_68[0][0]',
'activation_69[0][0]']

 conv2d_72 (Conv2D)             (None, None, None,   147456
['mixed7[0][0]']
                                192)

 batch_normalization_72 (BatchN  (None, None, None,   576
['conv2d_72[0][0]']
 ormalization)                  192)

 activation_72 (Activation)     (None, None, None,   0
['batch_normalization_72[0][0]']
                                192)

 conv2d_73 (Conv2D)             (None, None, None,   258048
['activation_72[0][0]']
                                192)

 batch_normalization_73 (BatchN  (None, None, None,   576
['conv2d_73[0][0]']
```

```
 ormalization)                  192)

 activation_73 (Activation)     (None, None, None,   0
['batch_normalization_73[0][0]']
                                192)

 conv2d_70 (Conv2D)             (None, None, None,   147456
['mixed7[0][0]']
                                192)

 conv2d_74 (Conv2D)             (None, None, None,   258048
['activation_73[0][0]']
                                192)

 batch_normalization_70 (BatchN  (None, None, None,   576
['conv2d_70[0][0]']
 ormalization)                  192)

 batch_normalization_74 (BatchN  (None, None, None,   576
['conv2d_74[0][0]']
 ormalization)                  192)

 activation_70 (Activation)     (None, None, None,   0
['batch_normalization_70[0][0]']
                                192)

 activation_74 (Activation)     (None, None, None,   0
['batch_normalization_74[0][0]']
                                192)

 conv2d_71 (Conv2D)             (None, None, None,   552960
['activation_70[0][0]']
                                320)

 conv2d_75 (Conv2D)             (None, None, None,   331776
['activation_74[0][0]']
                                192)

 batch_normalization_71 (BatchN  (None, None, None,   960
['conv2d_71[0][0]']
 ormalization)                  320)

 batch_normalization_75 (BatchN  (None, None, None,   576
['conv2d_75[0][0]']
 ormalization)                  192)

 activation_71 (Activation)     (None, None, None,   0
['batch_normalization_71[0][0]']
```

```
                                 320)

 activation_75 (Activation)     (None, None, None,   0
['batch_normalization_75[0][0]']
                                 192)

 max_pooling2d_3 (MaxPooling2D)  (None, None, None,   0
['mixed7[0][0]']
                                 768)

 mixed8 (Concatenate)           (None, None, None,   0
['activation_71[0][0]',
                                 1280)
'activation_75[0][0]',
'max_pooling2d_3[0][0]']

 conv2d_80 (Conv2D)             (None, None, None,   573440
['mixed8[0][0]']
                                 448)

 batch_normalization_80 (BatchN  (None, None, None,   1344
['conv2d_80[0][0]']
 ormalization)                   448)

 activation_80 (Activation)     (None, None, None,   0
['batch_normalization_80[0][0]']
                                 448)

 conv2d_77 (Conv2D)             (None, None, None,   491520
['mixed8[0][0]']
                                 384)

 conv2d_81 (Conv2D)             (None, None, None,   1548288
['activation_80[0][0]']
                                 384)

 batch_normalization_77 (BatchN  (None, None, None,   1152
['conv2d_77[0][0]']
 ormalization)                   384)

 batch_normalization_81 (BatchN  (None, None, None,   1152
['conv2d_81[0][0]']
 ormalization)                   384)

 activation_77 (Activation)     (None, None, None,   0
['batch_normalization_77[0][0]']
                                 384)
```

```
 activation_81 (Activation)     (None, None, None,   0
['batch_normalization_81[0][0]']
                                384)

 conv2d_78 (Conv2D)             (None, None, None,   442368
['activation_77[0][0]']
                                384)

 conv2d_79 (Conv2D)             (None, None, None,   442368
['activation_77[0][0]']
                                384)

 conv2d_82 (Conv2D)             (None, None, None,   442368
['activation_81[0][0]']
                                384)

 conv2d_83 (Conv2D)             (None, None, None,   442368
['activation_81[0][0]']
                                384)

 average_pooling2d_7 (AveragePo  (None, None, None,   0
['mixed8[0][0]']
 oling2D)                       1280)

 conv2d_76 (Conv2D)             (None, None, None,   409600
['mixed8[0][0]']
                                320)

 batch_normalization_78 (BatchN  (None, None, None,   1152
['conv2d_78[0][0]']
 ormalization)                  384)

 batch_normalization_79 (BatchN  (None, None, None,   1152
['conv2d_79[0][0]']
 ormalization)                  384)

 batch_normalization_82 (BatchN  (None, None, None,   1152
['conv2d_82[0][0]']
 ormalization)                  384)

 batch_normalization_83 (BatchN  (None, None, None,   1152
['conv2d_83[0][0]']
 ormalization)                  384)

 conv2d_84 (Conv2D)             (None, None, None,   245760
['average_pooling2d_7[0][0]']
                                192)
```

```
 batch_normalization_76 (BatchN  (None, None, None,   960
['conv2d_76[0][0]']
 ormalization)                  320)

 activation_78 (Activation)     (None, None, None,   0
['batch_normalization_78[0][0]']
                                384)

 activation_79 (Activation)     (None, None, None,   0
['batch_normalization_79[0][0]']
                                384)

 activation_82 (Activation)     (None, None, None,   0
['batch_normalization_82[0][0]']
                                384)

 activation_83 (Activation)     (None, None, None,   0
['batch_normalization_83[0][0]']
                                384)

 batch_normalization_84 (BatchN  (None, None, None,   576
['conv2d_84[0][0]']
 ormalization)                  192)

 activation_76 (Activation)     (None, None, None,   0
['batch_normalization_76[0][0]']
                                320)

 mixed9_0 (Concatenate)         (None, None, None,   0
['activation_78[0][0]',
                                768)
'activation_79[0][0]']

 concatenate (Concatenate)      (None, None, None,   0
['activation_82[0][0]',
                                768)
'activation_83[0][0]']

 activation_84 (Activation)     (None, None, None,   0
['batch_normalization_84[0][0]']
                                192)

 mixed9 (Concatenate)           (None, None, None,   0
['activation_76[0][0]',
                                2048)
'mixed9_0[0][0]',
'concatenate[0][0]',
'activation_84[0][0]']
```

```
 conv2d_89 (Conv2D)             (None, None, None,   917504
['mixed9[0][0]']
                                448)

 batch_normalization_89 (BatchN  (None, None, None,   1344
['conv2d_89[0][0]']
 ormalization)                  448)

 activation_89 (Activation)     (None, None, None,   0
['batch_normalization_89[0][0]']
                                448)

 conv2d_86 (Conv2D)             (None, None, None,   786432
['mixed9[0][0]']
                                384)

 conv2d_90 (Conv2D)             (None, None, None,   1548288
['activation_89[0][0]']
                                384)

 batch_normalization_86 (BatchN  (None, None, None,   1152
['conv2d_86[0][0]']
 ormalization)                  384)

 batch_normalization_90 (BatchN  (None, None, None,   1152
['conv2d_90[0][0]']
 ormalization)                  384)

 activation_86 (Activation)     (None, None, None,   0
['batch_normalization_86[0][0]']
                                384)

 activation_90 (Activation)     (None, None, None,   0
['batch_normalization_90[0][0]']
                                384)

 conv2d_87 (Conv2D)             (None, None, None,   442368
['activation_86[0][0]']
                                384)

 conv2d_88 (Conv2D)             (None, None, None,   442368
['activation_86[0][0]']
                                384)

 conv2d_91 (Conv2D)             (None, None, None,   442368
['activation_90[0][0]']
                                384)
```

```
 conv2d_92 (Conv2D)             (None, None, None,   442368
['activation_90[0][0]']
                                384)

 average_pooling2d_8 (AveragePo  (None, None, None,   0
['mixed9[0][0]']
 oling2D)                       2048)

 conv2d_85 (Conv2D)             (None, None, None,   655360
['mixed9[0][0]']
                                320)

 batch_normalization_87 (BatchN  (None, None, None,   1152
['conv2d_87[0][0]']
 ormalization)                  384)

 batch_normalization_88 (BatchN  (None, None, None,   1152
['conv2d_88[0][0]']
 ormalization)                  384)

 batch_normalization_91 (BatchN  (None, None, None,   1152
['conv2d_91[0][0]']
 ormalization)                  384)

 batch_normalization_92 (BatchN  (None, None, None,   1152
['conv2d_92[0][0]']
 ormalization)                  384)

 conv2d_93 (Conv2D)             (None, None, None,   393216
['average_pooling2d_8[0][0]']
                                192)

 batch_normalization_85 (BatchN  (None, None, None,   960
['conv2d_85[0][0]']
 ormalization)                  320)

 activation_87 (Activation)     (None, None, None,   0
['batch_normalization_87[0][0]']
                                384)

 activation_88 (Activation)     (None, None, None,   0
['batch_normalization_88[0][0]']
                                384)

 activation_91 (Activation)     (None, None, None,   0
['batch_normalization_91[0][0]']
                                384)
```

```
 activation_92 (Activation)     (None, None, None,   0
['batch_normalization_92[0][0]']
                                384)

 batch_normalization_93 (BatchN  (None, None, None,   576
['conv2d_93[0][0]']
 ormalization)                  192)

 activation_85 (Activation)     (None, None, None,   0
['batch_normalization_85[0][0]']
                                320)

 mixed9_1 (Concatenate)         (None, None, None,   0
['activation_87[0][0]',
                                768)
'activation_88[0][0]']

 concatenate_1 (Concatenate)    (None, None, None,   0
['activation_91[0][0]',
                                768)
'activation_92[0][0]']

 activation_93 (Activation)     (None, None, None,   0
['batch_normalization_93[0][0]']
                                192)

 mixed10 (Concatenate)          (None, None, None,   0
['activation_85[0][0]',
                                2048)
'mixed9_1[0][0]',
'concatenate_1[0][0]',
'activation_93[0][0]']

 global_average_pooling2d_3 (Gl  (None, 2048)        0
['mixed10[0][0]']
 obalAveragePooling2D)

 dense_12 (Dense)               (None, 1024)         2098176
['global_average_pooling2d_3[0][0
                                                                ]']

 dense_13 (Dense)               (None, 1024)         1049600
['dense_12[0][0]']

 dropout_3 (Dropout)            (None, 1024)         0
['dense_13[0][0]']
```

```
 dense_14 (Dense)                (None, 512)          524800
['dropout_3[0][0]']

 dense_15 (Dense)                (None, 2)            1026
['dense_14[0][0]']

==================================================================================
==================
Total params: 25,476,386
Trainable params: 25,441,954
Non-trainable params: 34,432
----------------------------------------------------------------------------------
------------------
```

[ ]:
```
opt = Adam(learning_rate=0.0001)
Interception.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
↪['accuracy'])
```

[ ]:
```
pred = Interception.predict(test_images_googlent_MB)
pred=np.argmax(pred,axis=1)
```

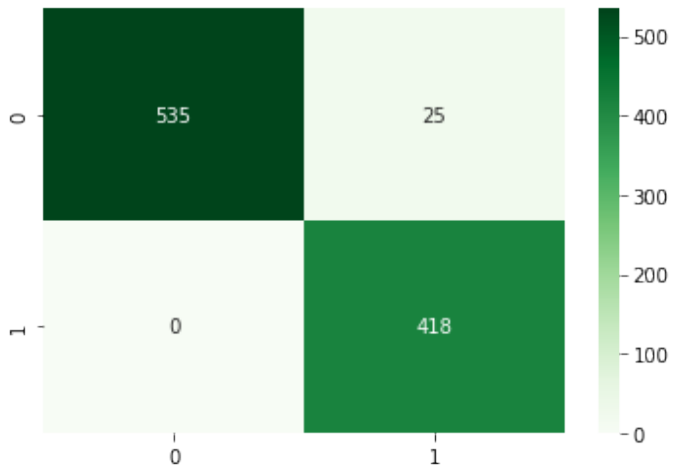
[ ]:
```
pred_df=df_teste_400.copy()
labels={}
for l,v in test_images_googlent_MB.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score: {classification_report(pred_df['Type'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       1.00      0.96      0.98       560
           M       0.94      1.00      0.97       418

    accuracy                           0.97       978
   macro avg       0.97      0.98      0.97       978
weighted avg       0.98      0.97      0.97       978
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['Type'],pred_df['pred']), annot=True,␣
↪fmt='2d',   cmap="Greens");
```

```
Accuracy Score: 0.9744376278118609
```



2.1.5 Modelo GoogleNet Para grau de ampliação 200

[ ]:
```
train_images_googlent_MB = train_generator_googleNet.flow_from_dataframe(
    dataframe=df_treino_200,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images_googlent_MB = train_generator_googleNet.flow_from_dataframe(
    dataframe=df_treino_200,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
```

```
    subset='validation'
)

test_images_googlent_MB = test_generator_googleNet.flow_from_dataframe(
    dataframe=df_teste_200,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
Found 630 validated image filenames belonging to 2 classes.
Found 157 validated image filenames belonging to 2 classes.
Found 856 validated image filenames belonging to 2 classes.
```

[ ]:
```
pred = Interception.predict(test_images_googlent_MB)
pred=np.argmax(pred,axis=1)
```

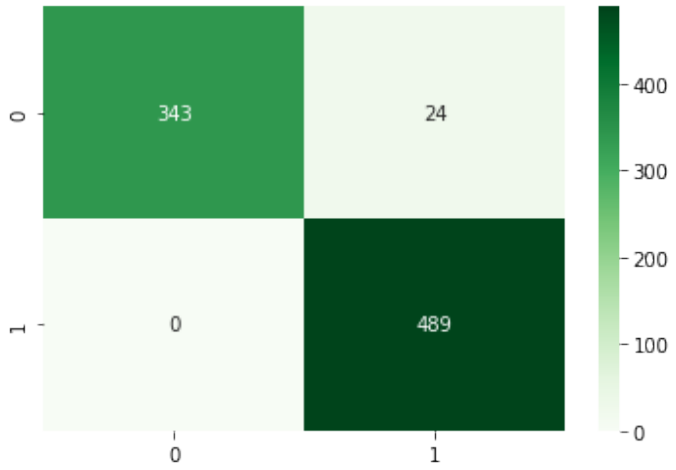
[ ]:
```
pred_df=df_teste_200.copy()
labels={}
for l,v in test_images_googlent_MB.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score: {classification_report(pred_df['Type'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       1.00      0.93      0.97       367
           M       0.95      1.00      0.98       489

    accuracy                           0.97       856
   macro avg       0.98      0.97      0.97       856
weighted avg       0.97      0.97      0.97       856
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['Type'],pred_df['pred']), annot=True,␣
 ↪fmt='2d',   cmap="Greens");
```

```
Accuracy Score: 0.9719626168224299
```



[ ]:

```

```

2.1.6 Modelo GoogleNet validação geral

[ ]:

```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_treino = df_geral[df_geral['DataSet'] == 'Tr']
df_teste = df_geral[df_geral['DataSet'] == 'Te']
```

[ ]:

```
train_images = train_generator_googleNet.flow_from_dataframe(
    dataframe=df_treino,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images = train_generator_googleNet.flow_from_dataframe(
    dataframe=df_treino,
    x_col='FikePath',
    y_col='Type',
```

```
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images = test_generator_googleNet.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 1259 validated image filenames belonging to 2 classes.
Found 314 validated image filenames belonging to 2 classes.
Found 1834 validated image filenames belonging to 2 classes.
```

[ ]:
```
pred = Interception.predict(test_images)
pred=np.argmax(pred,axis=1)
```

[ ]:
```
pred_df=df_teste.copy()
labels={}
for l,v in test_images.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

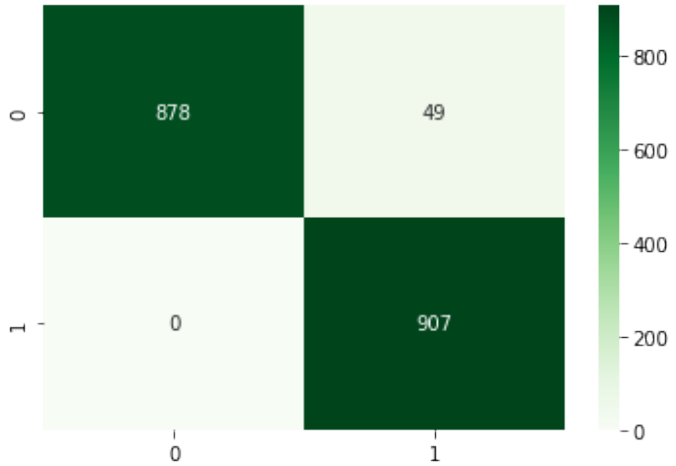
[ ]:
```
print(f"Metrics Score: {classification_report(pred_df['Type'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       1.00      0.95      0.97       927
           M       0.95      1.00      0.97       907

    accuracy                           0.97      1834
   macro avg       0.97      0.97      0.97      1834
weighted avg       0.97      0.97      0.97      1834
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
```

```
sns.heatmap(confusion_matrix(pred_df['Type'],pred_df['pred']), annot=True,␣
↪fmt='2d',  cmap="Greens");
```

```
Accuracy Score: 0.9732824427480916
```



[ ]:

```
```

2.1.7 Modelo ResNet50 Para grau de ampliação 400

[ ]:

```
train_images_ResNet_MB = train_generator_ResNet.flow_from_dataframe(
    dataframe=df_treino_400,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images_ResNet_MB = train_generator_ResNet.flow_from_dataframe(
    dataframe=df_treino_400,
    x_col='FikePath',
    y_col='Type',
```

```
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images_ResNet_MB = test_generator_ResNet.flow_from_dataframe(
    dataframe=df_teste_400,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
Found 629 validated image filenames belonging to 2 classes.
Found 157 validated image filenames belonging to 2 classes.
Found 978 validated image filenames belonging to 2 classes.
```

[ ]:
```
ResNet = build_model("Resnet")
ResNet.load_weights('/content/drive/MyDrive/TCC/weights/ResNet50/
 ↪ResNet_MB_com_pesos.h5')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
```

[ ]:
```
opt = Adam(learning_rate=0.0001)
ResNet.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

[ ]:
```
pred = ResNet.predict(test_images_ResNet_MB)
pred=np.argmax(pred,axis=1)
```

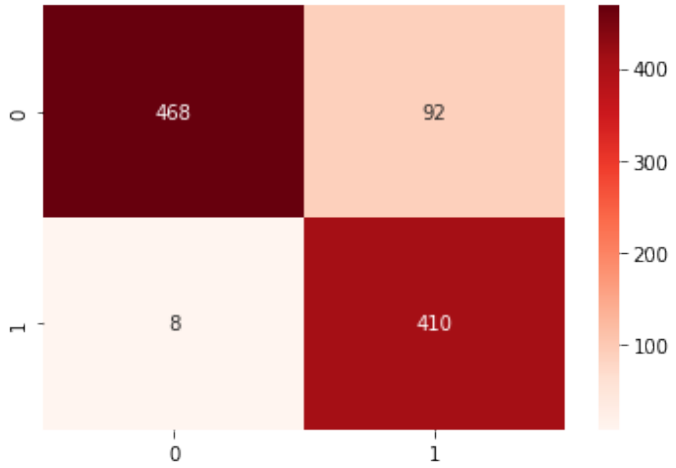
[ ]:
```
pred_df=df_teste_400.copy()
labels={}
for l,v in test_images_ResNet_MB.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score: {classification_report(pred_df['Type'],pred_df['pred'])}")
```

```
Metrics Score:                precision    recall  f1-score   support

           B       0.98      0.84      0.90       560
           M       0.82      0.98      0.89       418

    accuracy                           0.90       978
   macro avg       0.90      0.91      0.90       978
weighted avg       0.91      0.90      0.90       978
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['Type'],pred_df['pred']), annot=True,␣
↪fmt='2d',  cmap="Reds");
```

```
Accuracy Score: 0.8977505112474438
```



2.1.8 Modelo ResNet50 Para grau de ampliação 200

[ ]:
```
test_images_ResNet_MB = test_generator_ResNet.flow_from_dataframe(
    dataframe=df_teste_200,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
```

```
    shuffle=False
)
```

```
Found 856 validated image filenames belonging to 2 classes.
```

[ ]:
```
opt = Adam(learning_rate=0.0001)
ResNet.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
↪['accuracy'])
```

[ ]:
```
pred = ResNet.predict(test_images_ResNet_MB)
pred=np.argmax(pred,axis=1)
```

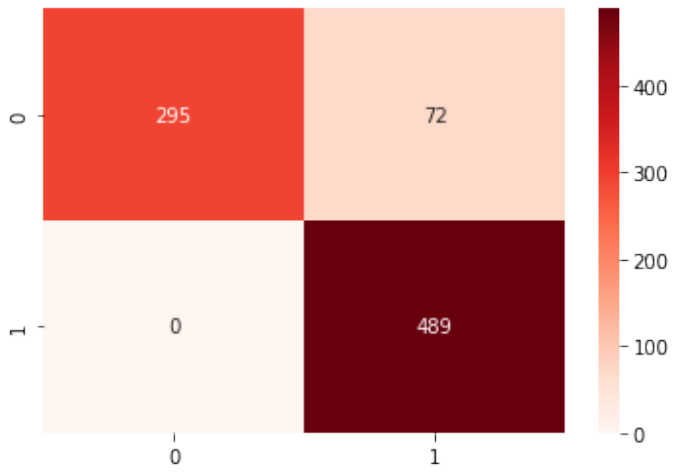
[ ]:
```
pred_df=df_teste_200.copy()
labels={}
for l,v in test_images_ResNet_MB.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score: {classification_report(pred_df['Type'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       1.00      0.80      0.89       367
           M       0.87      1.00      0.93       489

    accuracy                           0.92       856
   macro avg       0.94      0.90      0.91       856
weighted avg       0.93      0.92      0.91       856
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['Type'],pred_df['pred']), annot=True,␣
↪fmt='2d',  cmap="Reds");
```

```
Accuracy Score: 0.9158878504672897
```



2.1.9 Modelo ResNet50 validação geral

[ ]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_treino = df_geral[df_geral['DataSet'] == 'Tr']
df_teste = df_geral[df_geral['DataSet'] == 'Te']
```

[ ]:
```
test_images_ResNet_MB = test_generator_ResNet.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
Found 1834 validated image filenames belonging to 2 classes.
```

[ ]:
```
opt = Adam(learning_rate=0.0001)
ResNet.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

[ ]:
```
pred = ResNet.predict(test_images_ResNet_MB)
pred=np.argmax(pred,axis=1)
```

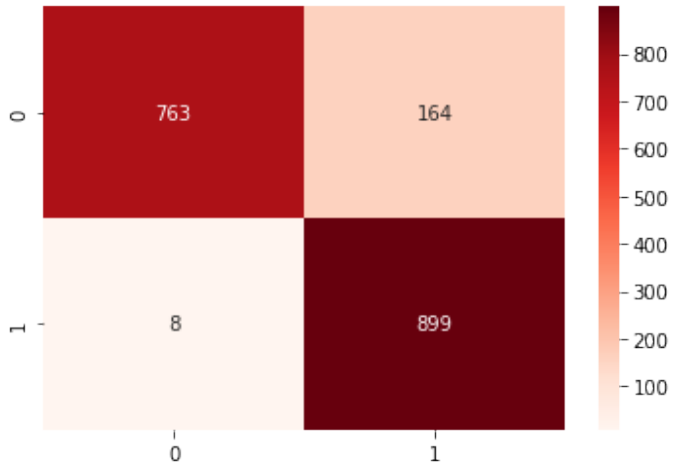
[ ]:
```
pred_df=df_teste.copy()
labels={}
for l,v in test_images_ResNet_MB.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score: {classification_report(pred_df['Type'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       0.99      0.82      0.90       927
           M       0.85      0.99      0.91       907

    accuracy                           0.91      1834
   macro avg       0.92      0.91      0.91      1834
weighted avg       0.92      0.91      0.91      1834
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['Type'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['Type'],pred_df['pred']), annot=True,␣
 ↪fmt='2d', cmap="Reds");
```

```
Accuracy Score: 0.906215921483097
```



2.2 Modelo Classe do Cancer

Apos o primeiro processo de treinamento. Será feita a segunda etapa onde classificaremos os canceres. Já tendo realizado a primeira etapade treinamento e validação realizaremos as novas etapas de treino.

[ ]:
```
def build_model_classe(Tipo_modelo):
  if Tipo_modelo == "VGG19":
    base_VGG19 = tf.keras.applications.VGG19(weights = 'imagenet', include_top =␣
↪False)
    x = base_VGG19.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_VGG19.input, outputs = preds)
  elif Tipo_modelo == "GoogleNet":
    base_Interception = InceptionV3(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)
  elif Tipo_modelo == "Resnet":
    base_Interception = ResNet50(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)

  return model
```

2.2.1 Modelo VGG19 Para grau de ampliação 400

[ ]:
```
test_images_VGG19_Classes_400 = test_generator_VGG19.flow_from_dataframe(
    dataframe=df_teste_400_classe,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 418 validated image filenames belonging to 4 classes.
```

[ ]:
```
VGG19_classes = build_model_classe("VGG19")
VGG19_classes.load_weights('/content/drive/MyDrive/TCC/weights/VGG19/VGG19_class.
 ↪h5')
opt = Adam(learning_rate=0.0001)
VGG19_classes.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
```

[ ]:
```
pred = VGG19_classes.predict(test_images_VGG19_Classes_400)
pred=np.argmax(pred,axis=1)
```

[ ]:
```
pred_df=df_teste_400_classe.copy()
labels={}
for l,v in test_images_VGG19_Classes_400.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

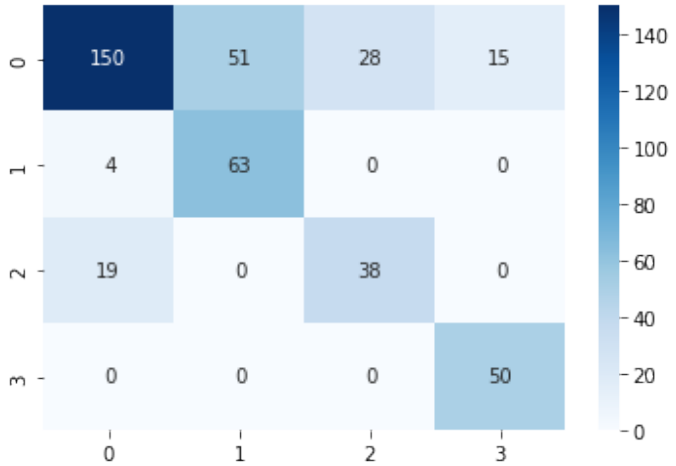
[ ]:
```
from sklearn.metrics import classification_report
```

[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           DC       0.87      0.61      0.72       244
           LC       0.55      0.94      0.70        67
           MC       0.58      0.67      0.62        57
           PC       0.77      1.00      0.87        50
```

```
    accuracy                           0.72       418
   macro avg       0.69      0.81      0.73       418
weighted avg       0.77      0.72      0.72       418
```

[ ]:
```
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
↪fmt='2d', cmap="Blues");
```



2.2.2 Modelo VGG19 Para grau de ampliação 200

[ ]:
```
test_images_VGG19_Classes_200 = test_generator_VGG19.flow_from_dataframe(
    dataframe=df_teste_200_classe,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 489 validated image filenames belonging to 4 classes.
```

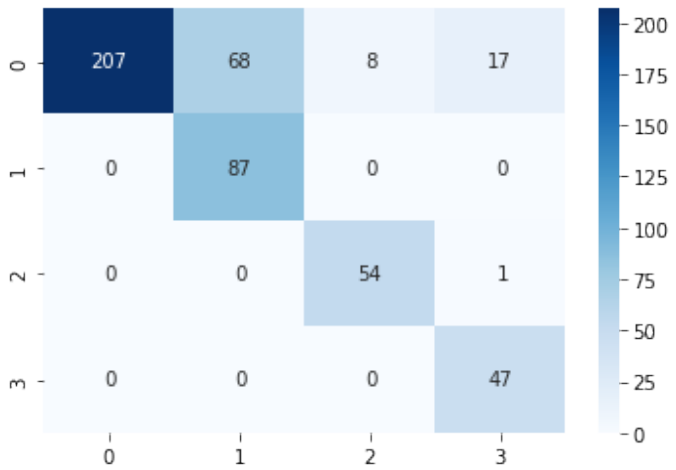
[ ]:
```
pred = VGG19_classes.predict(test_images_VGG19_Classes_200)
pred=np.argmax(pred,axis=1)
```

[ ]:
```
pred_df=df_teste_200_classe.copy()
labels={}
for l,v in test_images_VGG19_Classes_200.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

          DC       1.00      0.69      0.82       300
          LC       0.56      1.00      0.72        87
          MC       0.87      0.98      0.92        55
          PC       0.72      1.00      0.84        47

    accuracy                           0.81       489
   macro avg       0.79      0.92      0.82       489
weighted avg       0.88      0.81      0.81       489
```

[ ]:
```
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
↪fmt='2d', cmap="Blues");
```



2.2.3 Modelo VGG19 Validação geral

[ ]:
```
VGG19_classes = build_model_classe("VGG19")
VGG19_classes.load_weights('/content/drive/MyDrive/TCC/weights/VGG19/VGG19_class.
↪h5')
opt = Adam(learning_rate=0.0001)
VGG19_classes.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
↪['accuracy'])
```

[ ]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_treino = df_geral[df_geral['DataSet'] == 'Tr']
df_teste = df_geral[df_geral['DataSet'] == 'Te'][df_geral['Type'] == 'M']
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[ ]:
```
test_images_VGG19_Classes = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg19.preprocess_input
)
```

[ ]:
```
test_images_VGG19 = test_images_VGG19_Classes.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 907 validated image filenames belonging to 4 classes.
```

[ ]:
```
pred = VGG19_classes.predict(test_images_VGG19)
pred=np.argmax(pred,axis=1)
```

[ ]:
```
pred_df=df_teste.copy()
labels={}
for l,v in test_images_VGG19.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

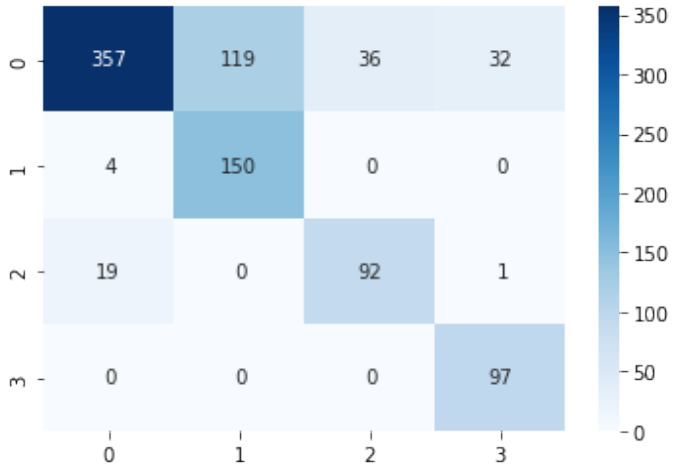
[ ]:
```
print(f"Accuracy Score:␣
↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Accuracy Score:               precision    recall  f1-score   support
```

```
           DC       0.94      0.66      0.77       544
           LC       0.56      0.97      0.71       154
           MC       0.72      0.82      0.77       112
           PC       0.75      1.00      0.85        97

     accuracy                           0.77       907
    macro avg       0.74      0.86      0.78       907
weighted avg        0.83      0.77      0.77       907
```

[ ]:
```
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
↪fmt='2d', cmap="Blues")
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faebfcf8110>
```



2.2.4 Modelo GoogleNet Para grau de ampliação 400

[ ]:
```
test_images_googleNet_Classes_400 = test_generator_googleNet.flow_from_dataframe(
    dataframe=df_teste_400_classe,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
```

```
    shuffle=False
)
```

```
Found 418 validated image filenames belonging to 4 classes.
```

[ ]:
```
interception_classes = build_model_classe("GoogleNet")
interception_classes.load_weights('/content/drive/MyDrive/TCC/weights/
 ↪Interception/Interception_class.h5')
opt = Adam(learning_rate=0.0001)
interception_classes.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics = ['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
87924736/87910968 [==============================] - 1s 0us/step
```

[ ]:
```
pred = interception_classes.predict(test_images_googleNet_Classes_400)
pred=np.argmax(pred,axis=1)
```

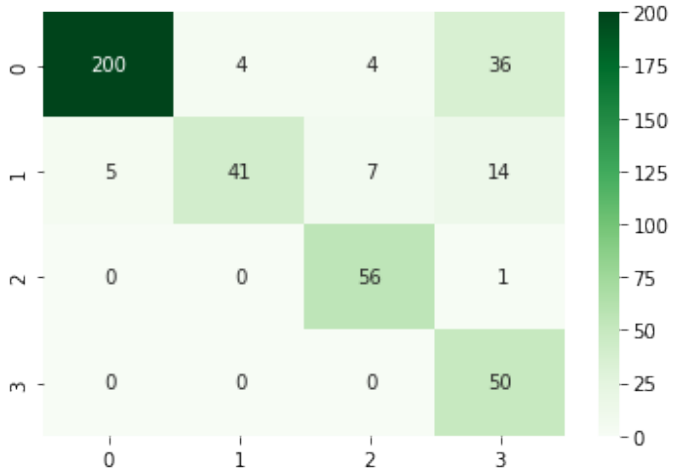
[ ]:
```
pred_df=df_teste_400_classe.copy()
labels={}
for l,v in test_images_googleNet_Classes_400.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           DC       0.98      0.82      0.89       244
           LC       0.91      0.61      0.73        67
           MC       0.84      0.98      0.90        57
           PC       0.50      1.00      0.66        50

     accuracy                           0.83       418
    macro avg       0.80      0.85      0.80       418
weighted avg        0.89      0.83      0.84       418
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['classes'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
 ↪fmt='2d', cmap='Greens');
```

```
Accuracy Score: 0.8301435406698564
```



[ ]:

```

```

2.2.5 Modelo GoogleNet Para grau de ampliação 200

[ ]:

```
test_images_googleNet_Classes_200 = test_generator_googleNet.flow_from_dataframe(
    dataframe=df_teste_200_classe,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
      shuffle=False
)
```

```
Found 489 validated image filenames belonging to 4 classes.
```

[ ]:

```
pred = interception_classes.predict(test_images_googleNet_Classes_200)
pred=np.argmax(pred,axis=1)
```

[ ]:

```
pred_df=df_teste_200_classe.copy()
labels={}
for l,v in test_images_googleNet_Classes_200.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
```

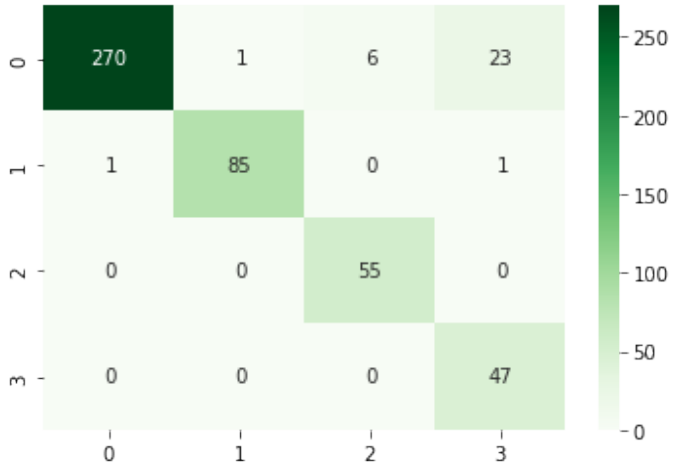
```
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:                precision    recall  f1-score   support

           DC       1.00      0.90      0.95       300
           LC       0.99      0.98      0.98        87
           MC       0.90      1.00      0.95        55
           PC       0.66      1.00      0.80        47

     accuracy                           0.93       489
    macro avg       0.89      0.97      0.92       489
weighted avg        0.95      0.93      0.94       489
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['classes'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
↪fmt='2d', cmap='Greens');
```

```
Accuracy Score: 0.934560327198364
```



[ ]:
```

```

2.2.6 Modelo GoogleNet Validação geral

[ ]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_treino = df_geral[df_geral['DataSet'] == 'Tr']
df_teste = df_geral[df_geral['DataSet'] == 'Te'][df_geral['Type'] == 'M']
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[ ]:
```
test_images_googleNet = test_generator_googleNet.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 907 validated image filenames belonging to 4 classes.
```

[ ]:
```
pred = interception_classes.predict(test_images_googleNet)
pred=np.argmax(pred,axis=1)
```

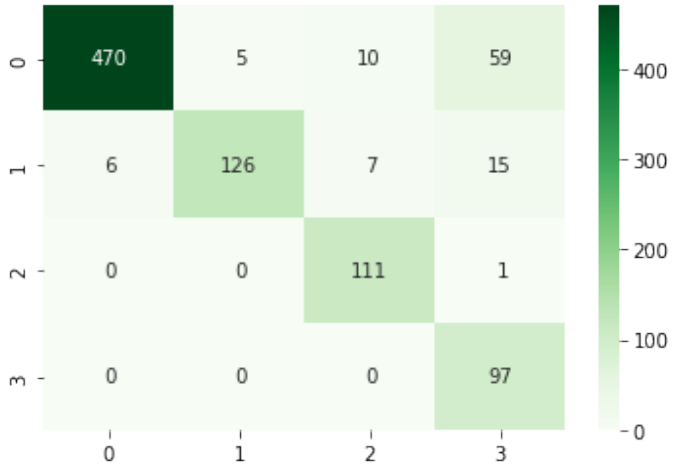
[ ]:
```
pred_df=df_teste.copy()
labels={}
for l,v in test_images_googleNet.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

          DC       0.99      0.86      0.92       544
          LC       0.96      0.82      0.88       154
          MC       0.87      0.99      0.92       112
          PC       0.56      1.00      0.72        97

    accuracy                           0.89       907
   macro avg       0.85      0.92      0.86       907
weighted avg       0.92      0.89      0.89       907
```

```
[ ]: print(f"Accuracy Score: {accuracy_score(pred_df['classes'],pred_df['pred'])}")
     sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
      ↪fmt='2d', cmap='Greens');
```

```
Accuracy Score: 0.886438809261301
```




```
[ ]:
```

2.2.7 Modelo ResNet Para grau de ampliação 400

```
[ ]: test_images_Resnet_Classes_400 = test_generator_ResNet.flow_from_dataframe(
         dataframe=df_teste_400_classe,
         x_col='FikePath',
         y_col='classes',
         target_size=(224, 224),
         color_mode='rgb',
         class_mode='categorical',
         batch_size=16,
         shuffle=False
     )
```

```
Found 418 validated image filenames belonging to 4 classes.
```

```
[ ]: ResNet_classes = build_model_classe("Resnet")
     ResNet_classes.load_weights('/content/drive/MyDrive/TCC/weights/ResNet50/
      ↪ResNet_class.h5')
```

```
opt = Adam(learning_rate=0.0001)
ResNet_classes.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
↪['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
```

[ ]:
```
pred = ResNet_classes.predict(test_images_Resnet_Classes_400)
pred=np.argmax(pred,axis=1)
```

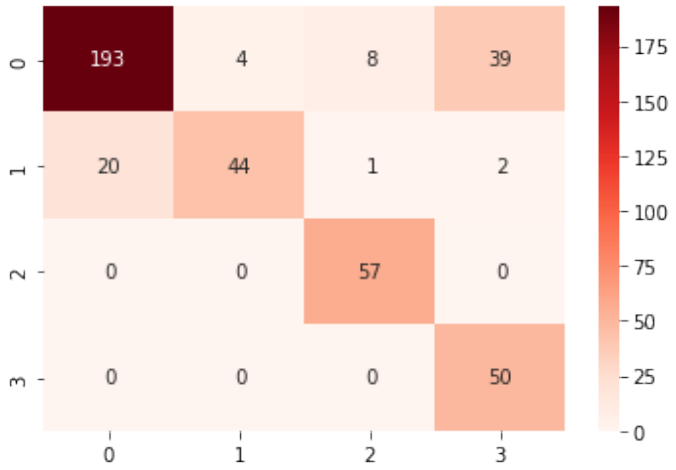
[ ]:
```
pred_df=df_teste_400_classe.copy()
labels={}
for l,v in test_images_Resnet_Classes_400.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           DC       0.91      0.79      0.84       244
           LC       0.92      0.66      0.77        67
           MC       0.86      1.00      0.93        57
           PC       0.55      1.00      0.71        50

     accuracy                           0.82       418
    macro avg       0.81      0.86      0.81       418
weighted avg        0.86      0.82      0.83       418
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['classes'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
↪fmt='2d', cmap='Reds');
```

```
Accuracy Score: 0.8229665071770335
```



[ ]:
```
```

2.2.8 Modelo ResNet Para grau de ampliação 200

[ ]:
```
test_images_Resnet_Classes_200 = test_generator_ResNet.flow_from_dataframe(
    dataframe=df_teste_200_classe,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 489 validated image filenames belonging to 4 classes.
```

[ ]:
```
ResNet_classes = build_model_classe("Resnet")
ResNet_classes.load_weights('/content/drive/MyDrive/TCC/weights/ResNet50/
 ↪ResNet_class.h5')
opt = Adam(learning_rate=0.0001)
ResNet_classes.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

[ ]:
```
pred = ResNet_classes.predict(test_images_Resnet_Classes_200)
pred=np.argmax(pred,axis=1)
```

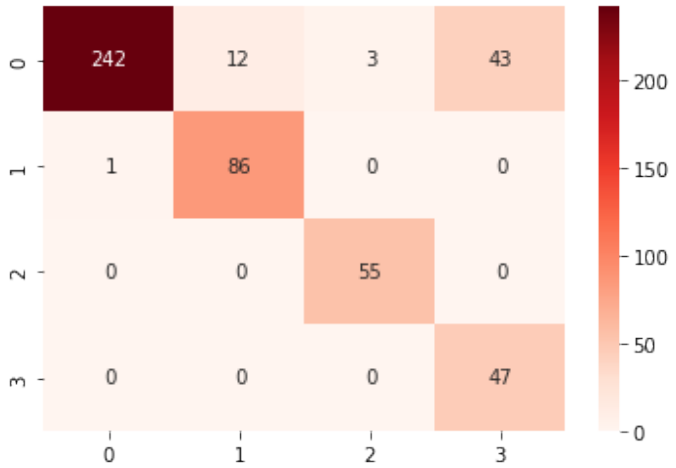
[ ]:
```
pred_df=df_teste_200_classe.copy()
labels={}
for l,v in test_images_Resnet_Classes_200.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           DC       1.00      0.81      0.89       300
           LC       0.88      0.99      0.93        87
           MC       0.95      1.00      0.97        55
           PC       0.52      1.00      0.69        47

     accuracy                           0.88       489
    macro avg       0.84      0.95      0.87       489
 weighted avg       0.92      0.88      0.89       489
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['classes'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
↪fmt='2d', cmap='Reds');
```

```
Accuracy Score: 0.8793456032719836
```



[ ]:

2.2.9 Modelo ResNet Validação geral

[ ]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_treino = df_geral[df_geral['DataSet'] == 'Tr']
df_teste = df_geral[df_geral['DataSet'] == 'Te'][df_geral['Type'] == 'M']
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[ ]:
```
test_images_ResNet = test_generator_ResNet.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='classes',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 907 validated image filenames belonging to 4 classes.
```

[ ]:
```
ResNet_classes = build_model_classe("Resnet")
ResNet_classes.load_weights('/content/drive/MyDrive/TCC/weights/ResNet50/
 ↪ResNet_class.h5')
opt = Adam(learning_rate=0.0001)
ResNet_classes.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

[ ]:
```
pred = ResNet_classes.predict(test_images_ResNet)
pred=np.argmax(pred,axis=1)
```

[ ]:

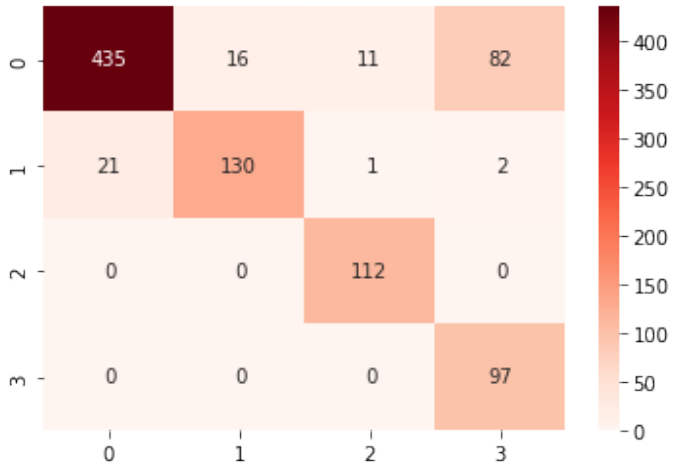
[ ]:
```
pred_df=df_teste.copy()
labels={}
for l,v in test_images_ResNet.class_indices.items():
    labels.update({v:l})
pred_df['pred']=pred
pred_df['pred']=pred_df['pred'].apply(lambda x: labels[x])
```

[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df['classes'],pred_df['pred'])}")
```

```
Metrics Score:                precision    recall  f1-score   support

           DC       0.95      0.80      0.87       544
           LC       0.89      0.84      0.87       154
           MC       0.90      1.00      0.95       112
           PC       0.54      1.00      0.70        97

     accuracy                           0.85       907
    macro avg       0.82      0.91      0.85       907
weighted avg        0.89      0.85      0.86       907
```

[ ]:
```
print(f"Accuracy Score: {accuracy_score(pred_df['classes'],pred_df['pred'])}")
sns.heatmap(confusion_matrix(pred_df['classes'],pred_df['pred']), annot=True,␣
↪fmt='2d', cmap='Reds');
```

```
Accuracy Score: 0.8533627342888643
```



[ ]:
```

```

#Combinação dos modelos

Após as execuções dos modelos, e tendo visto separadamente cada avaliação percebemos que os melhores modelos até então são, o GoogleNet(InterceptionV3) para categoria e GoogleNet(InterceptionV3) para classe.

Agora realizaremos a combinação dos modelos, para definirmos a conjunção final do Projeto.

2.2.10 Definição do dataframe e dos modelos

[ ]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_teste_categoria = df_geral[df_geral['DataSet'] == 'Te']
```

[ ]:
```
def build_model(Tipo_modelo):
  if Tipo_modelo == "VGG19":
    base_VGG19 = tf.keras.applications.VGG19(weights = 'imagenet', include_top =␣
↪False)
    x = base_VGG19.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(2, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_VGG19.input, outputs = preds)
  elif Tipo_modelo == "GoogleNet":
    base_Interception = InceptionV3(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(2, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)
  elif Tipo_modelo == "Resnet":
    base_RESNET = tf.keras.applications.ResNet50(weights = 'imagenet',␣
↪include_top = False)
    x = base_RESNET.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(2, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_RESNET.input, outputs = preds)

  return model
```

[ ]:
```
def build_model_classe(Tipo_modelo):
  if Tipo_modelo == "VGG19":
```

```
    base_VGG19 = tf.keras.applications.VGG19(weights = 'imagenet', include_top =␣
↪False)
    x = base_VGG19.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_VGG19.input, outputs = preds)
  elif Tipo_modelo == "GoogleNet":
    base_Interception = InceptionV3(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)
  elif Tipo_modelo == "Resnet":
    base_Interception = ResNet50(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)

  return model
```

2.2.11 Configuração da VGG19 Categoria

[ ]:
```
test_generator_VGG19_MB = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg19.preprocess_input
)
```

[ ]:
```
test_images_VGG19_MB = test_generator_VGG19_MB.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='Type',
```

```
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 1834 validated image filenames belonging to 2 classes.
```

[ ]:
```
opt = Adam(learning_rate=0.0001)
VGG19_rede_MB = build_model('VGG19')
checkpoint_path = r'/content/drive/MyDrive/TCC/weights/VGG19/VGG_MB_com_pesos.h5'
VGG19_rede_MB.load_weights(checkpoint_path)
VGG19_rede_MB.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
↪['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 0s 0us/step
80150528/80134624 [==============================] - 0s 0us/step
```

2.3 Combinação 1

2.3.1 Modelo VGG19 Categoria com Modelo GoogleNet Classes

2.3.2 Execução modelo VGG19 Categoria

[ ]:
```
pred_VGG19 = VGG19_rede_MB.predict(test_images_VGG19_MB)
pred_VGG19_gmax=np.argmax(pred_VGG19,axis=1)
```

[ ]:
```
labels={}
pred_df_VGG19_MB=df_teste.copy()
for l,v in test_images_VGG19_MB.class_indices.items():
    labels.update({v:l})
pred_df_VGG19_MB['pred']=pred_VGG19_gmax
pred_df_VGG19_MB['pred']=pred_df_VGG19_MB['pred'].apply(lambda x: labels[x])
pred_df_VGG19_MB =  pred_df_VGG19_MB[pred_df_VGG19_MB["pred"] == "M"]
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df_VGG19_MB['Type'],pred_df_VGG19_MB['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       0.00      0.00      0.00        79
           M       0.92      1.00      0.96       896

    accuracy                           0.92       975
   macro avg       0.46      0.50      0.48       975
weighted avg       0.84      0.92      0.88       975
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
pred_df_VGG19_MB["classes"][(pred_df_VGG19_MB['classes'] != 'DC') &␣
↪(pred_df_VGG19_MB['classes'] != 'LC') & (pred_df_VGG19_MB['classes'] != 'MC')␣
↪& (pred_df_VGG19_MB['classes'] != 'PC')] = "Erro"
```

2.3.3 Configuração da GoogleNet Classe

[ ]:
```
test_generator_googleNet_classe = tf.keras.preprocessing.image.
↪ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
)
```

[ ]:
```
test_images_googleNet_classe = test_generator_googleNet_classe.
↪flow_from_dataframe(
    dataframe=pred_df_VGG19_MB,
    x_col='FikePath',
    y_col='classes',
    classes=['DC','LC','MC','PC'],
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 896 validated image filenames belonging to 4 classes.
```

[ ]:
```
interception_classes = build_model_classe("GoogleNet")
interception_classes.load_weights('/content/drive/MyDrive/TCC/weights/
↪Interception/Interception_class.h5')
opt = Adam(learning_rate=0.0001)
```

```
interception_classes.compile(loss='categorical_crossentropy', optimizer=opt,␣
↪metrics = ['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
87924736/87910968 [==============================] - 1s 0us/step
```

[ ]:
```
pred_interception = interception_classes.predict(test_images_googleNet_classe)
pred_interception_gmax=np.argmax(pred_interception,axis=1)
```

[ ]:
```
pred_df_VGG_GOOGLE =  pred_df_VGG19_MB[pred_df_VGG19_MB['classes'] != "Erro"]
labels={}
for l,v in test_images_googleNet_classe.class_indices.items():
    labels.update({v:l})
pred_df_VGG_GOOGLE['pred_CLASS']=pred_interception_gmax
pred_df_VGG_GOOGLE['pred_CLASS']=pred_df_VGG_GOOGLE['pred_CLASS'].apply(lambda x:
↪ labels[x])
```

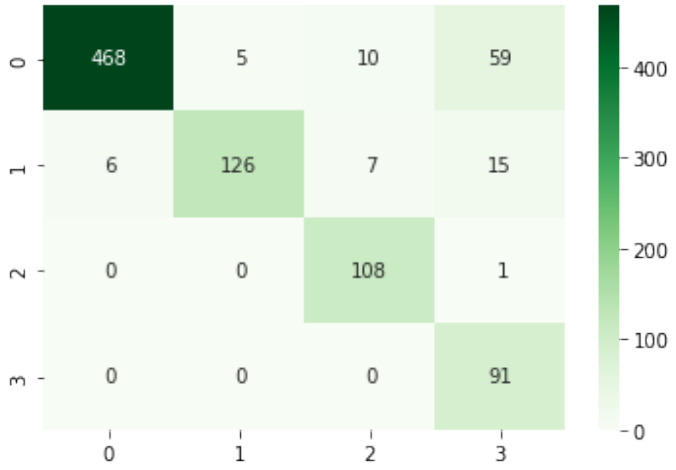
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df_VGG_GOOGLE['classes'],pred_df_VGG_GOOGLE['pred_CLASS'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

          DC       0.99      0.86      0.92       542
          LC       0.96      0.82      0.88       154
          MC       0.86      0.99      0.92       109
          PC       0.55      1.00      0.71        91

    accuracy                           0.89       896
   macro avg       0.84      0.92      0.86       896
weighted avg       0.92      0.89      0.89       896
```

```
[ ]: print(f"Accuracy Score:␣
     ↪{accuracy_score(pred_df_VGG_GOOGLE['classes'],pred_df_VGG_GOOGLE['pred_CLASS'])}")
     sns.
     ↪heatmap(confusion_matrix(pred_df_VGG_GOOGLE['classes'],pred_df_VGG_GOOGLE['pred_CLASS']),␣
     ↪annot=True, fmt='2d', cmap='Greens');
```

```
Accuracy Score: 0.8850446428571429
```




```
[ ]:
```

2.4 Combinação 2

2.4.1 Modelo VGG19 Categoria com Modelo ResNet Classes

2.4.2 Execução modelo VGG19 Categoria

```
[ ]: pred_VGG19 = VGG19_rede_MB.predict(test_images_VGG19_MB)
     pred_VGG19_gmax=np.argmax(pred_VGG19,axis=1)
```

```
[ ]: labels={}
     pred_df_VGG19_MB=df_teste.copy()
     for l,v in test_images_VGG19_MB.class_indices.items():
         labels.update({v:l})
     pred_df_VGG19_MB['pred']=pred_VGG19_gmax
     pred_df_VGG19_MB['pred']=pred_df_VGG19_MB['pred'].apply(lambda x: labels[x])
```

```
pred_df_VGG19_MB =  pred_df_VGG19_MB[pred_df_VGG19_MB["pred"] == "M"]
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df_VGG19_MB['Type'],pred_df_VGG19_MB['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       0.00      0.00      0.00        79
           M       0.92      1.00      0.96       896

    accuracy                           0.92       975
   macro avg       0.46      0.50      0.48       975
weighted avg       0.84      0.92      0.88       975


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
pred_df_VGG19_MB["classes"][(pred_df_VGG19_MB['classes'] != 'DC') &␣
↪(pred_df_VGG19_MB['classes'] != 'LC') & (pred_df_VGG19_MB['classes'] != 'MC')␣
↪& (pred_df_VGG19_MB['classes'] != 'PC')] = "Erro"
```

2.4.3 Configuração da ResNet Classe

[ ]:
```
test_generator_ResNet_classe = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
)
```

[ ]:
```
test_images_ResNet_classe = test_generator_ResNet_classe.flow_from_dataframe(
    dataframe=pred_df_VGG19_MB,
    x_col='FikePath',
    y_col='classes',
    classes=['DC','LC','MC','PC'],
    target_size=(224, 224),
    color_mode='rgb',
```

```
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 896 validated image filenames belonging to 4 classes.
```

[ ]:
```
resNet_classes = build_model_classe("Resnet")
resNet_classes.load_weights('/content/drive/MyDrive/TCC/weights/ResNet50/
 ↪ResNet_class.h5')
opt = Adam(learning_rate=0.0001)
resNet_classes.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
 ↪['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 0s 0us/step
94781440/94765736 [==============================] - 0s 0us/step
```

[ ]:
```
pred_ResNet = resNet_classes.predict(test_images_ResNet_classe)
pred_ResNet_gmax=np.argmax(pred_ResNet,axis=1)
```

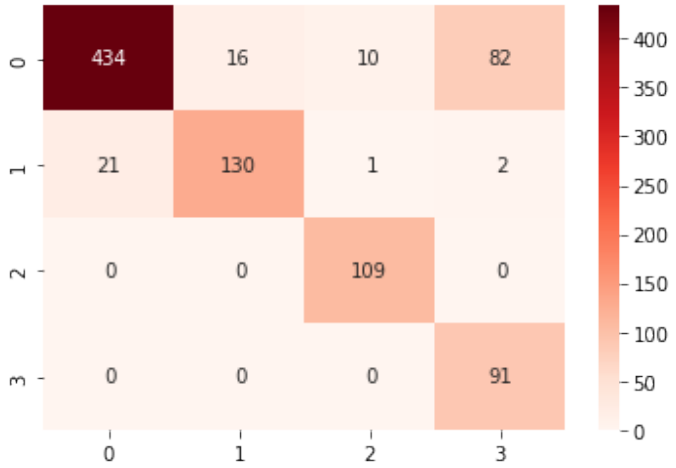
[ ]:
```
pred_df_VGG_ResNet =  pred_df_VGG19_MB[pred_df_VGG19_MB['classes'] != "Erro"]
labels={}
for l,v in test_images_ResNet_classe.class_indices.items():
    labels.update({v:l})
pred_df_VGG_ResNet['pred_CLASS']=pred_ResNet_gmax
pred_df_VGG_ResNet['pred_CLASS']=pred_df_VGG_ResNet['pred_CLASS'].apply(lambda x:
 ↪ labels[x])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df_VGG_ResNet['classes'],pred_df_VGG_ResNet['pred_CLASS'])}")
```

```
Metrics Score:                precision    recall  f1-score   support

           DC       0.95      0.80      0.87       542
           LC       0.89      0.84      0.87       154
           MC       0.91      1.00      0.95       109
           PC       0.52      1.00      0.68        91

     accuracy                           0.85       896
    macro avg       0.82      0.91      0.84       896
weighted avg        0.89      0.85      0.86       896
```

[ ]:
```
sns.
↪heatmap(confusion_matrix(pred_df_VGG_ResNet['classes'],pred_df_VGG_ResNet['pred_CLASS']),␣
↪annot=True, fmt='2d', cmap='Reds');
```



2.5 Combinação 3

2.5.1 Modelo GoogleNet Categoria com Modelo ResNet Classes

2.5.2 Configuração da GoogleNet Categoria

[ ]:
```
test_generator_googleNet_MB = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
)
```

[ ]:
```
test_images_googleNet_MB  = test_generator_googleNet_MB.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 1834 validated image filenames belonging to 2 classes.
```

[ ]:
```
Interception_rede_MB = build_model("GoogleNet")
Interception_rede_MB.load_weights(filepath=r'/content/drive/MyDrive/TCC/weights/
 ↪Interception/Interception_MB_com_pesos.h5')
opt = Adam(learning_rate=0.0001)
Interception_rede_MB.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics = ['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 0s 0us/step
87924736/87910968 [==============================] - 0s 0us/step
```

[ ]:
```
pred_Interception_MB = Interception_rede_MB.predict(test_images_googleNet_MB)
pred_Interception_MB_gmax=np.argmax(pred_Interception_MB,axis=1)
```

[ ]:
```
labels={}
pred_df_Interceptio_MB=df_teste.copy()
for l,v in test_images_googleNet_MB.class_indices.items():
    labels.update({v:l})
pred_df_Interceptio_MB['pred']=pred_Interception_MB_gmax
pred_df_Interceptio_MB['pred']=pred_df_Interceptio_MB['pred'].apply(lambda x:␣
 ↪labels[x])
pred_df_Interceptio_MB =  pred_df_Interceptio_MB[pred_df_Interceptio_MB["pred"]␣
 ↪== "M"]
```

[ ]:
```
pred_df_Interceptio_MB["classes"][(pred_df_Interceptio_MB['classes'] != 'DC') &␣
 ↪(pred_df_Interceptio_MB['classes'] != 'LC') &␣
 ↪(pred_df_Interceptio_MB['classes'] != 'MC') &␣
 ↪(pred_df_Interceptio_MB['classes'] != 'PC')] = "Erro"
```

[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df_Interceptio_MB['Type'],pred_df_Interceptio_MB['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       0.00      0.00      0.00        49
```

```
           M       0.95      1.00      0.97       907

    accuracy                           0.95       956
   macro avg       0.47      0.50      0.49       956
weighted avg       0.90      0.95      0.92       956

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

2.5.3 Configuração da ResNet Classe

[ ]:
```
test_generator_ResNet_classe = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
)
```

[ ]:
```
test_images_ResNet_classe = test_generator_ResNet_classe.flow_from_dataframe(
    dataframe=pred_df_Interceptio_MB,
    x_col='FikePath',
    y_col='classes',
    classes=['DC','LC','MC','PC'],
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 907 validated image filenames belonging to 4 classes.
```

[ ]:
```
resNet_classes = build_model_classe("Resnet")
resNet_classes.load_weights('/content/drive/MyDrive/TCC/weights/ResNet50/
 ↪ResNet_class.h5')
opt = Adam(learning_rate=0.0001)
```

```
resNet_classes.compile(loss='categorical_crossentropy', optimizer=opt, metrics =␣
↪['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
```

[ ]:
```
pred_ResNet = resNet_classes.predict(test_images_ResNet_classe)
pred_ResNet_gmax=np.argmax(pred_ResNet,axis=1)
```

[ ]:
```
pred_df_VGG_ResNet =  pred_df_Interceptio_MB[pred_df_Interceptio_MB['classes'] !
↪= "Erro"]
labels={}
for l,v in test_images_ResNet_classe.class_indices.items():
    labels.update({v:l})
pred_df_VGG_ResNet['pred_CLASS']=pred_ResNet_gmax
pred_df_VGG_ResNet['pred_CLASS']=pred_df_VGG_ResNet['pred_CLASS'].apply(lambda x:
↪ labels[x])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df_VGG_ResNet['classes'],pred_df_VGG_ResNet['pred_CLASS'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

          DC       0.95      0.80      0.87       544
          LC       0.89      0.84      0.87       154
          MC       0.90      1.00      0.95       112
          PC       0.54      1.00      0.70        97

    accuracy                           0.85       907
   macro avg       0.82      0.91      0.85       907
```

```
weighted avg       0.89      0.85      0.86       907
```

[ ]:
```
sns.
↪heatmap(confusion_matrix(pred_df_VGG_ResNet['classes'],pred_df_VGG_ResNet['pred_CLASS']),␣
↪annot=True, fmt='2d', cmap='Reds');
```

[ ]:
```

```

2.6 Combinação 4

2.6.1 Modelo GoogleNet Categoria com Modelo GoogleNet Classes

2.6.2 Configuração da GoogleNet Categoria

[ ]:
```
test_generator_googleNet_MB = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
)
```

[ ]:
```
test_images_googleNet_MB  = test_generator_googleNet_MB.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
```

```
    batch_size=16,
    shuffle=False
)
```

```
Found 1834 validated image filenames belonging to 2 classes.
```

[ ]:
```
Interception_rede_MB = build_model("GoogleNet")
Interception_rede_MB.load_weights(filepath=r'/content/drive/MyDrive/TCC/weights/
 ↪Interception/Interception_MB_com_pesos.h5')
opt = Adam(learning_rate=0.0001)
Interception_rede_MB.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics = ['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 0s 0us/step
87924736/87910968 [==============================] - 0s 0us/step
```

[ ]:
```
pred_Interception_MB = Interception_rede_MB.predict(test_images_googleNet_MB)
pred_Interception_MB_gmax=np.argmax(pred_Interception_MB,axis=1)
```

[ ]:
```
labels={}
pred_df_Interceptio_MB=df_teste.copy()
for l,v in test_images_googleNet_MB.class_indices.items():
    labels.update({v:l})
pred_df_Interceptio_MB['pred']=pred_Interception_MB_gmax
pred_df_Interceptio_MB['pred']=pred_df_Interceptio_MB['pred'].apply(lambda x:␣
 ↪labels[x])
pred_df_Interceptio_MB =  pred_df_Interceptio_MB[pred_df_Interceptio_MB["pred"]␣
 ↪== "M"]
```

[ ]:
```
pred_df_Interceptio_MB["classes"][(pred_df_Interceptio_MB['classes'] != 'DC') &␣
 ↪(pred_df_Interceptio_MB['classes'] != 'LC') &␣
 ↪(pred_df_Interceptio_MB['classes'] != 'MC') &␣
 ↪(pred_df_Interceptio_MB['classes'] != 'PC')] = "Erro"
```

[ ]:
```
print(f"Metrics Score:␣
 ↪{classification_report(pred_df_Interceptio_MB['Type'],pred_df_Interceptio_MB['pred'])}")
```

```
Metrics Score:               precision    recall  f1-score   support

           B       0.00      0.00      0.00        49
           M       0.95      1.00      0.97       907

    accuracy                           0.95       956
   macro avg       0.47      0.50      0.49       956
weighted avg       0.90      0.95      0.92       956
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

2.6.3 Configuração da GoogleNet Classe

[ ]:
```
test_generator_googleNet_classe = tf.keras.preprocessing.image.
↪ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
)
```

[ ]:
```
test_images_googleNet_classe = test_generator_googleNet_classe.
↪flow_from_dataframe(
    dataframe=pred_df_Interceptio_MB,
    x_col='FikePath',
    y_col='classes',
    classes=['DC','LC','MC','PC'],
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 907 validated image filenames belonging to 4 classes.
```

[ ]:
```
interception_classes = build_model_classe("GoogleNet")
interception_classes.load_weights('/content/drive/MyDrive/TCC/weights/
↪Interception/Interception_class.h5')
opt = Adam(learning_rate=0.0001)
interception_classes.compile(loss='categorical_crossentropy', optimizer=opt,␣
↪metrics = ['accuracy'])
```

[ ]:
```
pred_interception = interception_classes.predict(test_images_googleNet_classe)
pred_interception_gmax=np.argmax(pred_interception,axis=1)
```

[ ]:
```
pred_df_Interceptio_GOOGLE = ␣
↪pred_df_Interceptio_MB[pred_df_Interceptio_MB['classes'] != "Erro"]
labels={}
for l,v in test_images_googleNet_classe.class_indices.items():
    labels.update({v:l})
pred_df_Interceptio_GOOGLE['pred_CLASS']=pred_interception_gmax
pred_df_Interceptio_GOOGLE['pred_CLASS']=pred_df_Interceptio_GOOGLE['pred_CLASS'].
↪apply(lambda x: labels[x])
```

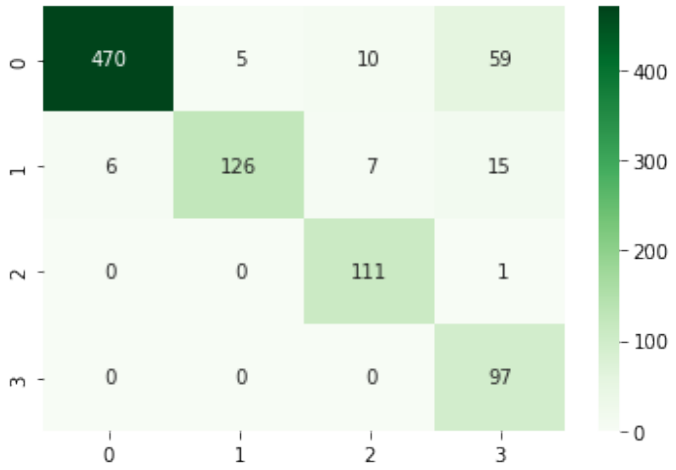
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

[ ]:
```
print(f"Metrics Score:␣
↪{classification_report(pred_df_Interceptio_GOOGLE['classes'],pred_df_Interceptio_GOOGLE['pred
```

```
Metrics Score:               precision    recall  f1-score   support

          DC       0.99      0.86      0.92       544
          LC       0.96      0.82      0.88       154
          MC       0.87      0.99      0.92       112
          PC       0.56      1.00      0.72        97

    accuracy                           0.89       907
   macro avg       0.85      0.92      0.86       907
weighted avg       0.92      0.89      0.89       907
```

[ ]:
```
print(f"Accuracy Score:␣
↪{accuracy_score(pred_df_Interceptio_GOOGLE['classes'],pred_df_Interceptio_GOOGLE['pred_CLASS'
sns.
↪heatmap(confusion_matrix(pred_df_Interceptio_GOOGLE['classes'],pred_df_Interceptio_GOOGLE['pr
↪annot=True, fmt='2d', cmap='Greens');
```

```
Accuracy Score: 0.886438809261301
```



[ ]:

```
```

3 Modelo Final

A parti dos resultados apresentados no decorrer do capítulo, é possível perceber que a melhor combinação de modelos é a "Combinação 4", como já tínhamos analisado anteriormente. Mesmo com o desbalanceamento de classes, foi encontrado Resultados suficientemente bons para aplicarmos em nosso modelo final.

Nessa seção apresentaremos o modelo final aplicado.

[21]:
```
df_geral =  montar_data_frame(r'/content/drive/MyDrive/TCC/Imagens')
df_teste_categoria = df_geral[df_geral['DataSet'] == 'Te']
```

[22]:
```
def build_model(Tipo_modelo):
  if Tipo_modelo == "Categoria":
    base_Interception = InceptionV3(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(2, activation='softmax')(x)
```

```
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)
  elif Tipo_modelo == "Classe":
    base_Interception = InceptionV3(weights = 'imagenet', include_top = False)
    x = base_Interception.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation='relu',␣
↪kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(1024, activation='relu')(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    preds = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(inputs = base_Interception.input, outputs =␣
↪preds)

  return model
```

[23]:
```
test_generator_googleNet_MB = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
)
```

[24]:
```
test_images_googleNet_MB  = test_generator_googleNet_MB.flow_from_dataframe(
    dataframe=df_teste,
    x_col='FikePath',
    y_col='Type',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 1834 validated image filenames belonging to 2 classes.
```

[25]:
```
Interception_rede_MB = build_model("Categoria")
Interception_rede_MB.load_weights(filepath=r'/content/drive/MyDrive/TCC/weights/
↪Interception/Interception_MB_com_pesos.h5')
opt = Adam(learning_rate=0.0001)
Interception_rede_MB.compile(loss='categorical_crossentropy', optimizer=opt,␣
↪metrics = ['accuracy'])
```

[26]:
```
pred_Interception_MB = Interception_rede_MB.predict(test_images_googleNet_MB)
pred_Interception_MB_gmax=np.argmax(pred_Interception_MB,axis=1)
```

[27]:
```
labels={}
pred_df_Interceptio_MB=df_teste.copy()
for l,v in test_images_googleNet_MB.class_indices.items():
```

```
    labels.update({v:l})
pred_df_Interceptio_MB['pred']=pred_Interception_MB_gmax
pred_df_Interceptio_MB['pred']=pred_df_Interceptio_MB['pred'].apply(lambda x:␣
↪labels[x])
pred_df_Interceptio_MB =  pred_df_Interceptio_MB[pred_df_Interceptio_MB["pred"]␣
↪== "M"]
```

[28]:
```
pred_df_Interceptio_MB["classes"][(pred_df_Interceptio_MB['classes'] != 'DC') &␣
↪(pred_df_Interceptio_MB['classes'] != 'LC') &␣
↪(pred_df_Interceptio_MB['classes'] != 'MC') &␣
↪(pred_df_Interceptio_MB['classes'] != 'PC')] = "Erro"
```

[29]:
```
test_generator_googleNet_classe = tf.keras.preprocessing.image.
↪ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
)
```

[30]:
```
test_images_googleNet_classe = test_generator_googleNet_classe.
↪flow_from_dataframe(
    dataframe=pred_df_Interceptio_MB,
    x_col='FikePath',
    y_col='classes',
    classes=['DC','LC','MC','PC'],
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=16,
    shuffle=False
)
```

```
Found 907 validated image filenames belonging to 4 classes.
```

[31]:
```
interception_classes = build_model("Classe")
interception_classes.load_weights('/content/drive/MyDrive/TCC/weights/
↪Interception/Interception_class.h5')
opt = Adam(learning_rate=0.0001)
interception_classes.compile(loss='categorical_crossentropy', optimizer=opt,␣
↪metrics = ['accuracy'])
```

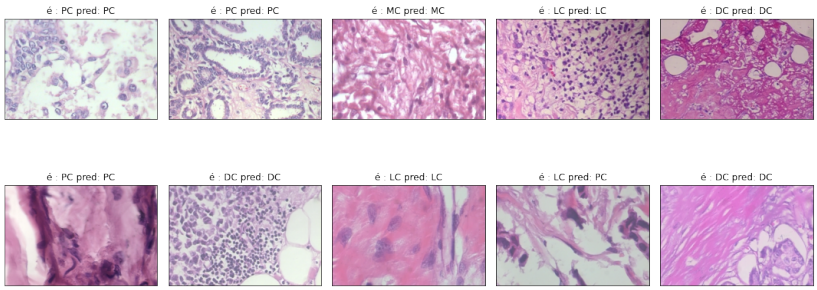
[32]:
```
pred_interception = interception_classes.predict(test_images_googleNet_classe)
pred_interception_gmax=np.argmax(pred_interception,axis=1)
```

[33]:
```
pred_df_Interceptio_GOOGLE = ␣
↪pred_df_Interceptio_MB[pred_df_Interceptio_MB['classes'] != "Erro"]
labels={}
for l,v in test_images_googleNet_classe.class_indices.items():
    labels.update({v:l})
```

```
pred_df_Interceptio_GOOGLE['pred_CLASS']=pred_interception_gmax
pred_df_Interceptio_GOOGLE['pred_CLASS']=pred_df_Interceptio_GOOGLE['pred_CLASS'].
 ↪apply(lambda x: labels[x])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

[36]:
```
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(15, 7),
                         subplot_kw={'xticks': [], 'yticks': []})
for i, ax in enumerate(axes.flat):
    ponteiro = np.random.randint(1,pred_df_Interceptio_GOOGLE.shape[0]-1)
    ax.imshow(plt.imread(list(pred_df_Interceptio_GOOGLE.FikePath)[ponteiro]))
    ax.set_title(f'é : {list(pred_df_Interceptio_GOOGLE.classes)[ponteiro]} pred:
 ↪ {list(pred_df_Interceptio_GOOGLE.pred_CLASS)[ponteiro]}')
plt.tight_layout()
plt.show()
```



[ ]:
```

```

REFERÊNCIAS

AMARANTE, S. Câncer de mama: a importância do diagnóstico precoce. In: IFF/FIOCRUZ. Rio de Janeiro, 2021. Disponível em: <http://www.iff.fiocruz.br/index.php/8-noticias/274-cancerdemama>. Acesso em: 12 set. 2021. 14

ANôNIMO. Como funcionam os neurônios. p. 1, 2014. Disponível em: <https://metodosupera.com.br/como-funcionam-os-neuronios/>. Publicado em: 17/09/2014. 32

ANôNIMO. "células nervosas"em só biologia. p. 1, 2021. Consultado em 19/08/2021 às 19:30. Disponível na Internet em <https://www.sobiologia.com.br/conteudos/FisiologiaAnimal/nervoso2.php>. 32, 33

BOLHASANI, H. et al. “a histopathological image dataset for grading breast invasive ductal carcinomas”. v. 19, n. 100341, p. 1, 2020. 14, 16, 17

DATA SCIENCE ACADEMY. *Deep Learning Book*. 1. ed. [S.l.]: Data Science Academy, 2021. Disponível em: <https://www.deeplearningbook.com.br/>.Acesso em: 19/08/2021 às 16:10. 33, 34, 35, 36, 37, 39

DIAGNóSTICO: o que interfere no falso positivo ou negativo. 2021. Disponível em: <https://www.oncologicadobrasil.com.br/blog/diagnostico-falso-positivo-negativo/>. acessado em: 11 de set.de 2021. 14

GOODFELLOW YOSHUA BENGIO, A. C. I. Deep learning. In: . [S.l.]: MIT., 2016. 42

HAYKIN, S. *Redes neurais: princípios e prática.* 1. ed. [S.l.]: Bookman Editora, 2007. 35, 38, 39

HEBB, D. *The organization of behavior: A neuropsychological approach.* 1. ed. [S.l.]: John Wiley and Sons, Inc., 1950. 34

INCA. *Atlas on-line de mortalidade.* 1. ed. [S.l.]: INSTITUTO NACIONAL DE CÂNCER JOSÉ ALENCAR GOMES DA SILVA, 2019. Disponível em: <https://www.inca.gov.br/aplicativos/atlas-de-mortalidade-por-cancer/>.Acesso em: 19/08/2021 às 16:10. 11

MADHAVAN, S. Introduction to convolutional neural networks. p. 1, 2017. Disponível em: <https://developer.ibm.com/articles/introduction-to-convolutional-neural-networks/>. Publicado em: 13/07/2021. 40, 41, 42

MADHAVAN, S.; JONES, M. T. Deep learning architectures. p. 1, 2017. Disponível em: <https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/>. Publicado em: 08/09/2017 e atualizado em: 25/01/2021. 40

MCCULLOC, W.; PITTS, W. The bulletin of mathematical biophysics. v. 5, n. 4, p. 115–133, 1943. 33

MORSCH, J. Como é feita a mamografia digital: para que serve, preparo e vantagens. In: . Rio Grande do Sul: [s.n.], 2018. Disponível em: <https://telemedicinamorsch.com.br/blog/como-e-feita-a-mamografia-digital>. Acesso em: 12 set. 2021. 10

O que é o diagnóstico histopatológico e qual sua importância? Anticorpos Laboratório, 2019. Disponível em: <https://www.anticorpos.com.br/artigos/o-que-e-o-diagnostico-histopatologico-e-qual-sua-importancia/>. acessado em: 13 de dez.de 2021. 10

OLIVEIRA, A. VisÃo computacional aplicada À anÁlise de mamografias. v. 1, n. 1, p. 31 – 31, 2019. TCC (Bacharelado em Ciência da Computação) - Centro Universitário Carioca, Rio de Janeiro - Rio de Janeiro. 10

RIBEIRO, A.; JUNIOR, F. Um estudo comparativo entre cinco métodos de otimização aplicados em uma rnc voltada ao diagnóstico do glaucoma. *Revista de Sistemas e Computação*, CS, v. 10, n. 1, p. 122–130, 2020. Disponível em: <http://www.revistas.unifacs.br/index.php/rsc>. 44

RIZWAN, M. Convolutional neural networks – in a nut shell. In: ENGMRK. [S.l.], 2018. Disponível em: <https://engmrk.com/convolutional-neural-network-3/> Acesso em: 20/9/2021. 42

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, v. 6, n. 65, p. 386–408, 1958. 34

RUMELHART, D.; HINTON, G.; WILLIAMS, R. J. Learning internal representations by error propagation. In: . [S.l.: s.n.], 1986. 39

SAKURAI, R. Implementando a estrutura de uma rede neural convolucional utilizando o mapreduce do spark. In: . [S.l.: s.n.], 2017. Disponível em: <http://www.sakurai.dev.br/cnn-mapreduce/>. Publicado em: 20/12/2017. 40

SILVA, I. N. de Câncer José Alencar Gomes da. Detecção precoce. In: INCA. Rio de Janeiro, 2021. Disponível em: <https://www.inca.gov.br/controle-do-cancer-de-mama/acoes-de-controle/deteccao-precoce>. Acesso em: 12 set. 2021. 14

SPANHOL, F. A. et al. Breast cancer histopathological image classification using convolutional neural networks. p. 1, 2016. 14, 15, 16

SUNG, H. et al. Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: A Cancer Journal for Clinicians*, ACS JOURNALS, v. 71, n. 3, p. 209–249, 2021. 10, 11

USHIZIMA. Redes neurais convolucionais com tensorflow: Teoria e prática. p. 382–406, 2017. 41