**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**

**Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data**

**Cesar Augusto Siqueira Santos**

**ALGORITMO DE CLASSIFICAÇÃO BINÁRIA USANDO CNNS: MELANOMA VERSUS NEVUS.**

Belo Horizonte

2022

**Cesar Augusto Siqueira Santos**

**ALGORITMO DE CLASSIFICAÇÃO BINÁRIA USANDO CNNS: MELANOMA VERSUS NEVUS.**

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.

Belo Horizonte

2022 **SUMÁRIO**

[**1. Introdução**](#_heading=h.gjdgxs) **5**

[1.1. Contextualização](#_heading=h.p6mam5rtf8p) 5

[1.2. O problema proposto](#_heading=h.s2y2hfthdyz2) 6

[1.3. Objetivos](#_heading=h.p3tr8prre52z) 7

[**2. Coleta de Dados**](#_heading=h.eui6sh21veu1) **7**

[2.1 HAM10000](#_heading=h.l2u3jtjznz8k) 8

[2.2 MED-NODE](#_heading=h.bmkvpswke6i5) 10

[**3. Processamento/Tratamento de Dados**](#_heading=h.pvxihicta43o) **11**

[**4. Análise e Exploração dos Dados**](#_heading=h.tyjcwt) **13**

[**5. Criação de Modelos de Machine Learning**](#_heading=h.3dy6vkm) **18**

[5.1 VGG-16](#_heading=h.noean36a5lvh) 18

[5.1.1 Aumentação de dados](#_heading=h.mrc5nhflsx1) 21

[5.1.2 Separação de bases de treinamento e validação](#_heading=h.kz9vxjakjuv9) 21

[5.1.3 Carregamento do modelo e compilação](#_heading=h.odrttbxz9rrl) 22

[5.1.4 Treinamento](#_heading=h.u4hw0hiea2mq) 23

[5.2 Inception:](#_heading=h.ym1tishfrq8x) 23

[5.2.1 Aumentação de dados](#_heading=h.p9fzsc937eki) 25

[5.2.2 Separação de bases de treinamento e validação](#_heading=h.ecsxkbqn399) 25

[5.2.3 Carregamento do modelo e compilação](#_heading=h.mpf2ihs1905m) 26

[5.2.4 Treinamento](#_heading=h.g38evh9vxeez) 27

[5.3 ResNet50](#_heading=h.t1dttlpievv2) 27

[5.3.1 Aumentação de dados](#_heading=h.kudc7oog94xr) 30

[5.3.2 Separação de bases de treinamento e validação](#_heading=h.z4m59uruc0p8) 30

[5.3.3 Carregamento do modelo e compilação](#_heading=h.qsy43tqrbwlf) 31

[5.3.4 Treinamento](#_heading=h.e0qs617fae8r) 32

[**6. Interpretação dos Resultados**](#_heading=h.epy5ocap7rih) **32**

[6.1 ResNet50](#_heading=h.n4c79n1mtm4p) 33

[6.2 Inception](#_heading=h.ivm4jhlkxp33) 35

[6.3 VGG16](#_heading=h.xwvkdvk40iyh) 36

[6.4 Conclusão](#_heading=h.4lxr0w33wvuu) 37

[**7. Apresentação dos Resultados**](#_heading=h.4d34og8) **38**

[**8. Links**](#_heading=h.ee5etyhv4rco) **39**

[8.1. Link Vídeo](#_heading=h.yy7uvfcwk9kx) 39

[8.2. Link Repositório](#_heading=h.7iqvrdm4tc9s) 40

[8.3. Link Dataset Final](#_heading=h.o53yx6nfmd1g) 40

[**9.Referências**](#_heading=h.3rdcrjn) **40**

[**Apêndice**](#_heading=h.26in1rg) **42**

# 1. Introdução

Este é um projeto com ênfase acadêmica, desenvolvido como trabalho de conclusão para o curso de especialização em ciência de dados e *big data*, trata sobre a coleta, tratamento e processamento de dados, que culminou na criação de três modelos de machine learning (CNNs pré-treinadas) para classificação de melanoma ou pintas normais (nevus), através do diagnóstico de imagens.

## 1.1. Contextualização

Câncer é o principal problema de saúde pública no mundo, em paralelo, as doenças de pele são um dos principais tópicos debatidos nos meios acadêmicos. Numa estimativa mundial conduzida no ano de 2018, aponta que ocorreram no mundo 18 milhões de casos novos de câncer e cerca de 9,6 milhões de óbitos. A incidência em homens (9,5 milhões) representa 53% dos casos novos, sendo um pouco maior do em mulheres, com 8,6 milhões (47%) de novos casos. (Instituto Nacional do Câncer, 2021).   
 Discorrendo agora sobre os cânceres de pele, um dos mais comuns no mundo. Este projeto tem ênfase na identificação de melanoma, o câncer de pele com maiores índices de mortalidade, ainda que represente apenas 1% dos cânceres de pele identificados, é de longe o maior causador de mortes.

Tomando por exemplo os Estados Unidos, para o ano de 2022 é esperado o diagnóstico de 99.780 novos casos de melanoma, distribuídos conforme: 57,3% para homens (57.180) e 42,7% em mulheres (42.600). Sobre mortalidade, é esperado que 7.650 pessoas venham a óbito em virtude do melanoma.

É fato que o melanoma atinge em sua maioria pessoas de pele branca, sendo 20 vezes mais comum do que em pessoas de pele preta. Sumarizando, o risco de desenvolver melanoma ao longo da vida para pessoas brancas é de cerca de 2,6%, para pessoas de pele preta, por outro lado, apenas 0,1%.

O risco de melanoma aumenta conforme a idade avançada, a idade média de diagnóstico, nos Estados Unidos, é de 65 anos, entretanto o melanoma não é tão incomum abaixo dos 30, é na verdade um dos cânceres mais comuns entre jovens adultos, especialmente mulheres. (Sociedade Americana de Câncer, 2022)

A detecção precoce, consiste em diagnosticar o câncer ainda em um estágio inicial, possibilitando assim melhores resultados no tratamento, pode ser buscado com a investigação de sinais e sintomas como o aparecimento de pintas escuras com bordas irregulares acompanhada de coceira e descamação, ou até mesmo alteração de pintas já existentes, como aumento de tamanho, mudança de cor e forma, passando também a apresentar bordas irregulares.

Uma regra adotada internacionalmente é a do “ABCDE” , utilizado na identificação precoce de melanoma. Não é necessariamente 100% assertiva, mas pode indicar uma certa atenção e justificar uma consulta médica com um dermatologista especializado:

* **Assimetria**: uma metade da mancha é diferente da outra;
* **Bordas irregulares**: contorno mal definido;
* **Cor variável**: presença de várias cores em uma mesma lesão (preta, castanha, branca, avermelhada ou azul);
* **Diâmetro**: maior que 6 milímetros;
* **Evolução**: mudanças observadas em suas características (tamanho, forma ou cor).

Discorrendo sobre o tratamento, o mais indicado para maioria dos casos é remoção da mancha de pelo através de cirurgia, porém radioterapia e/ou quimioterapia, podem ser utilizadas dependendo do estágio da doença. Quando acontece metástase (câncer já se espalhou para outros órgãos), o melanoma tem sido tratado hoje com novos medicamentos, que vem apresentando boas taxas de sucesso no tratamento, que hoje consistente em atrasar o avanço da doença o máximo possível, oferecendo assim, chance de sobrevida mais longa a pacientes que antes tinham um prognóstico mais desanimador (Instituto Nacional do Câncer, 2021).

## 1.2. O problema proposto

Os diversos cânceres de pele estão entre os cânceres mais comuns no mundo, embora o melanoma represente apenas 1% da variedade de cânceres de pele diagnosticados pelo globo, é de longe o mais mortífero. Se identificado em estágio inicial, pode realmente significar a sobrevivência de milhares de pessoas espalhadas pelo globo anualmente.

Este projeto visa classificar as imagens de câncer entre *melanocytic nevi* ou nevus(macha de pele comum e não cancerígena ) e melanoma(mancha de pele cancerígena) de maneira prática, auxiliando em diagnósticos precoces da doença.

Dessa forma, foram coletadas imagens de bases públicas disponibilizadas por iniciativas médicas ou acadêmicas. Neste projeto foram usados dados e imagens provenientes do HAM10000, extraídos do Kaggle, dados estes que foram reunidos entre o ano de 2016 e 2018. Além disso, os dados foram agrupados com uma base menor de dados provenientes de um Instituto dos Países Baixos, a base conhecida como MED-NODE e foi coletada até meados de 2015. Unidas as bases, totalizam cerca de 8 mil imagens distribuídas em 6805 caracterizadas como nevus e outras 1183 já identificadas como melanoma.

A análise matemática deste projeto consiste em converter as imagens em matrizes normalizadas de números inteiros representando assim as nuances das imagens, e através de modelos de *deep* *learning* (CNNs) pré-treinados, classificar as imagens separados as em lesões comuns e lesões cancerígenas. Foi necessário normalizar, separar em diretórios e redimensionar as imagens antes de efetivamente processar os dados.

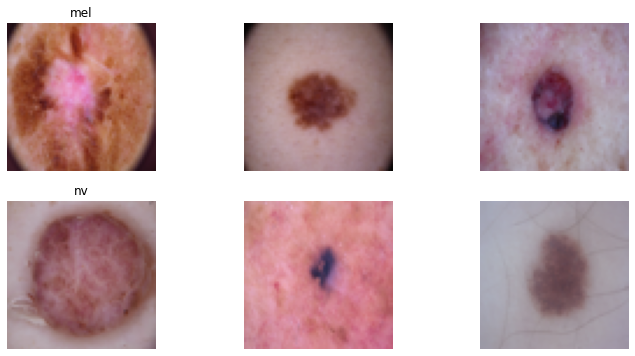
## 1.3. Objetivos

Este projeto possui apenas ênfase acadêmica, e visa colocar em prática diversos temas aprendidos durante a especialização *stricto sensu*, passando por coleta de dados, tratamentos e transformações, classificação e enriquecimento de dados, modelos de *machine learning* e *deep learning.* Todas estas etapas foram construídas utilizando Python como linguagem de programação, documentadas e executadas através de Jupyter Notebook. Logo, o objetivo não foi obter um aplicativo em estado de *deploy* em produção ou software finalizado para identificar melanoma ou mesmo substituir exames clínicos performados por médicos, nem mesmo altos índices de acurácia ou mesmo explorar em demasia atributos e funções de diferentes modelos de *machine learning*. O foco foi realmente a exploração e aprimoramento de conhecimentos através de uma atividade prática atrelada a realidade.

Assim sendo, o objetivo deste projeto foi testar pelo menos três modelos de classificação de imagem e definir também o modelo mais assertivo de *machine learning* para identificar melanoma, dada a base de imagens coletada, através do diagnóstico de imagens de manchas de pele.

# 2. **Coleta** de Dados

O tema deste projeto tem sido trabalhado por vários projetos ou textos académicos ao longo dos últimos anos, porém existem poucas bases, mas que são muito populares, praticamente todas providas pela mesma fonte a ISIC que anualmente (desde 2016) tem lançado um desafio de *machine learning*, disponível no Kaggle, e providenciado datasets com milhares de imagens com este tema. Entretanto, como exigência do projeto fez-se necessário buscar uma base de dados distinta e após pesquisa, um dataset pequeno foi encontrado, produzido pela MED-NODE, mas que forneceu imagens de nevus normais e melanoma. A fim de unir as bases, fez-se necesário reduzir o escopo do dataset provido pela HAM10000 (imagens colhidas pela ISIC em 2018), em apenas nevus ou melanomas. Para efeito de esclarecimento, seguem abaixo três imagens identificadas como melanoma e outras três identificadas como nevus:



Fonte: elaborado pelo autor em 2022.

Partindo agora para o detalhamento das bases de imagens e como foram devidamente unidas:

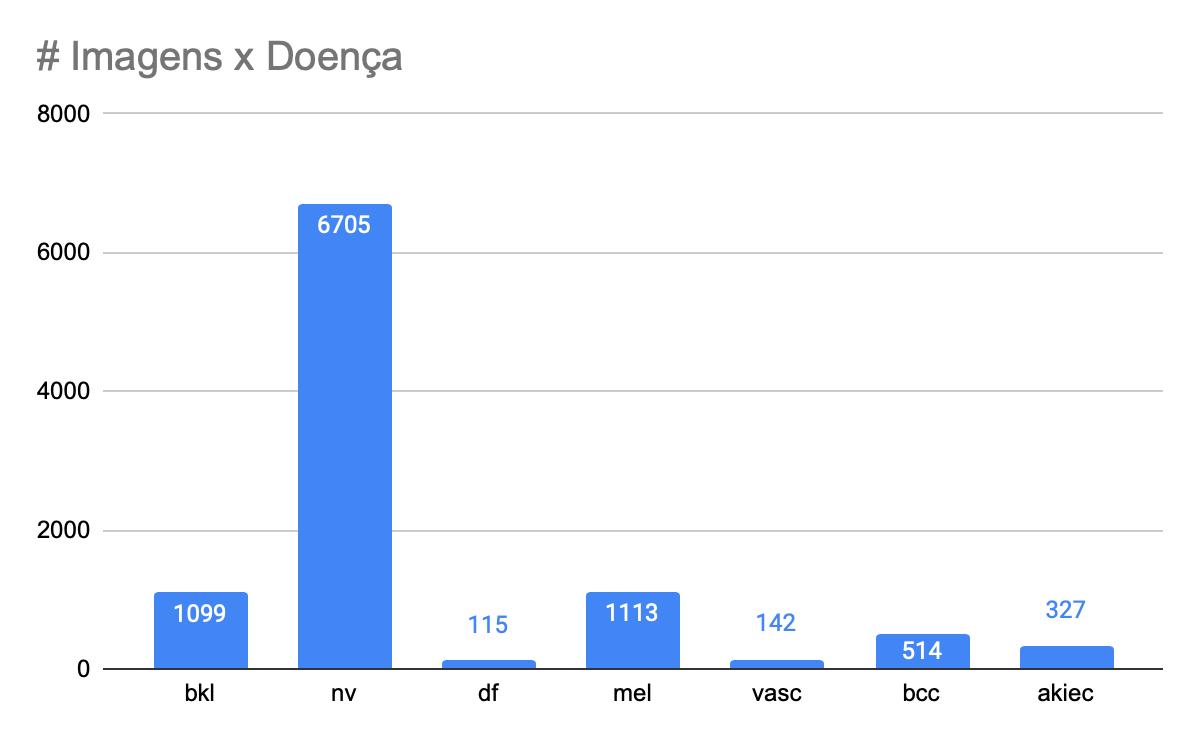
## 2.1 HAM10000

*Human Against Machine with 10000 training images*, ouhumanos contra máquinas usando 10000 imagens para treinamento, em tradução literal. Esta base de imagens possui este nome pois foi usada em desafios de *machine learning*, os dados foram coletados de diferentes populações, o dataset final é composto por 10015 imagens dermatoscópicas, que podem ser usadas para treinar diferentes tipos de modelos ou com propósitos distintos. Esta base de imagens contém um acervo representativo de imagens e dados da mais importantes lesões de pigmentação de pele, dentre elas:

| **Abreviação** | **Nome da Doença** | **Número de registros** | **Mantido?** |
| --- | --- | --- | --- |
| akiec | Doença de Bowen | 327 | Não |
| bcc | Carcinoma basocelular | 514 | Não |
| bkl | Verruga seborreica | 1099 | Não |
| df | Dermatofibroma | 115 | Não |
| mel | Melanoma | 1113 | Sim |
| vasc | Lesões Vasculares | 142 | Não |
| nv | Nevo Melanocítico ou nevus | 6705 | Sim |

Fonte: elaborado pelo autor em 2022.

Um gráfico para melhor interpretação da distribuição dos dados:



Fonte: elaborado pelo autor em 2022.

Mais de 50% das lesões presentes na base de dados foram confirmadas através de histopatologia, estudo de uma doença a partir de um conjunto de células, também conhecido como biópsia. Os outros resultados foram inferidos através de exames, consenso de especialistas ou confirmação através de microscopia confocal sem extração. Dentro da base de dados, existem algumas lesões com mais de uma imagem.

Junto a base de dados também foi providenciado um arquivo CSV contendo as seguintes informações:

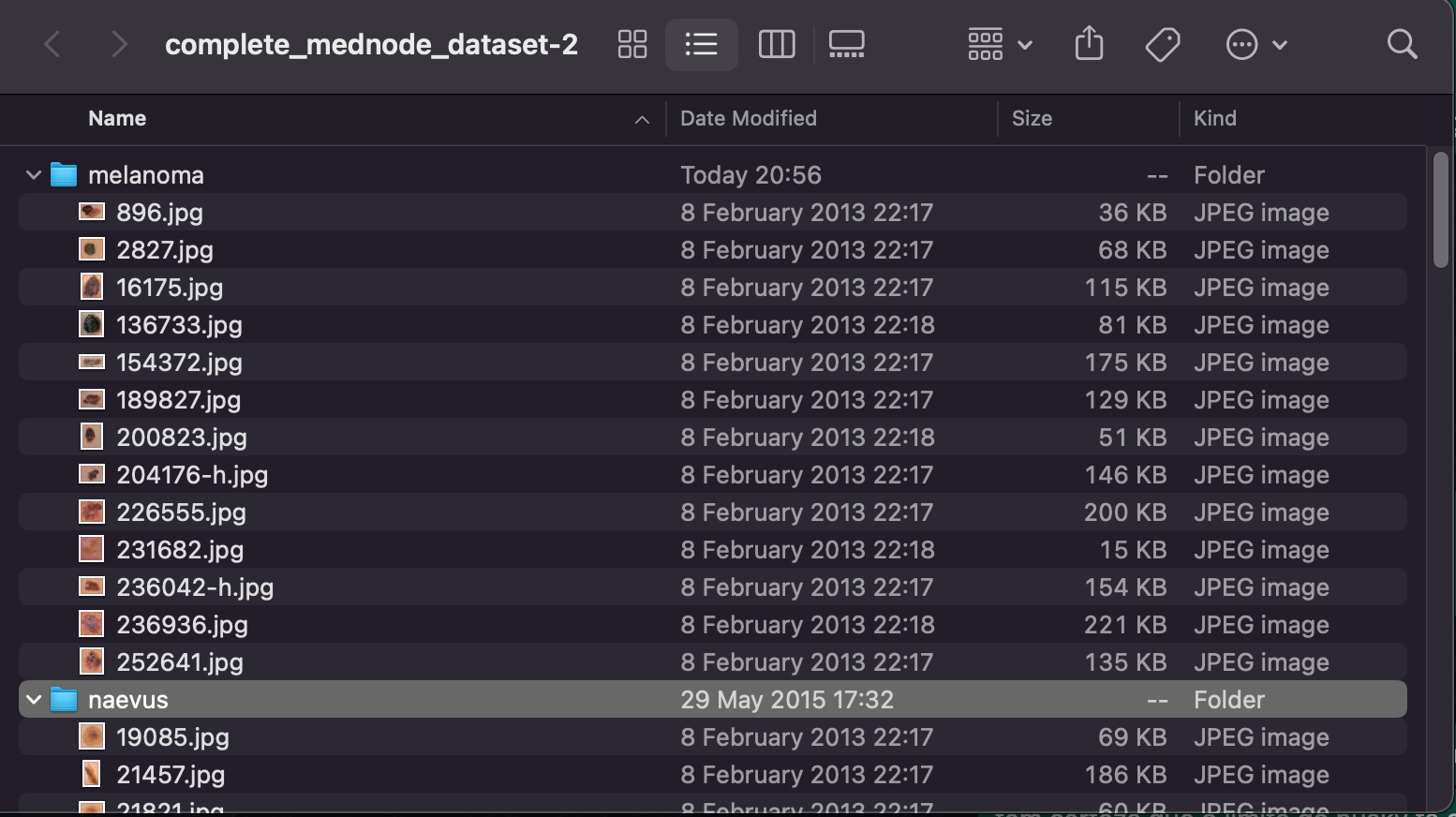
| Nome do campo | Descrição | Tipo |
| --- | --- | --- |
| lession\_id | Identificador da lesão | texto |
| image\_id | Identificador da imagem | texto |
| dx | Abreviatura da lesão constatada | texto |
| dx\_type | Método de confirmação da lesão | texto |
| age | Idade do paciente | inteiro |
| sex | Sexo do paciente | texto |
| localization | Parte do corpo que contém a lesão | texto |
| dataset | Dataset de origem | texto |

Fonte: elaborado pelo autor em 2022.

Entretanto, dado os requisitos deste projeto, foi necessário abrir mão de várias imagens e também registros de metadados, em virtude da natureza do próximo conjunto de dados a ser explorado. Esta base está disponível para download pelo site de Harvard: <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T>

## 2.2 MED-NODE

Conjunto de imagens reduzido, composto por 70 imagens de melanoma e 100 nevus, estas imagens foram coletadas e arquivadas pelo Departamento de Dermatologia da *University Medical Center Groningen (UMCG)*, localizada nos Países Baixos. Este dataset foi coletado para desenvolvimento e testes de um sistema para identificação de câncer de pele através de imagens macroscópicas.

Este dataset não fornece nenhum arquivo CSV contendo metadados ou algo do gênero, ele apenas consiste num arquivo zip contendo as imagens em formato JPEG, em diretórios separados, conforme imagem abaixo:  


Fonte: elaborado pelo autor em 2022.

A base de imagens encontra-se disponível para download pelo site:  
[complete\_mednode\_dataset.zip](http://www.cs.rug.nl/~imaging/databases/melanoma_naevi/complete_mednode_dataset.zip)

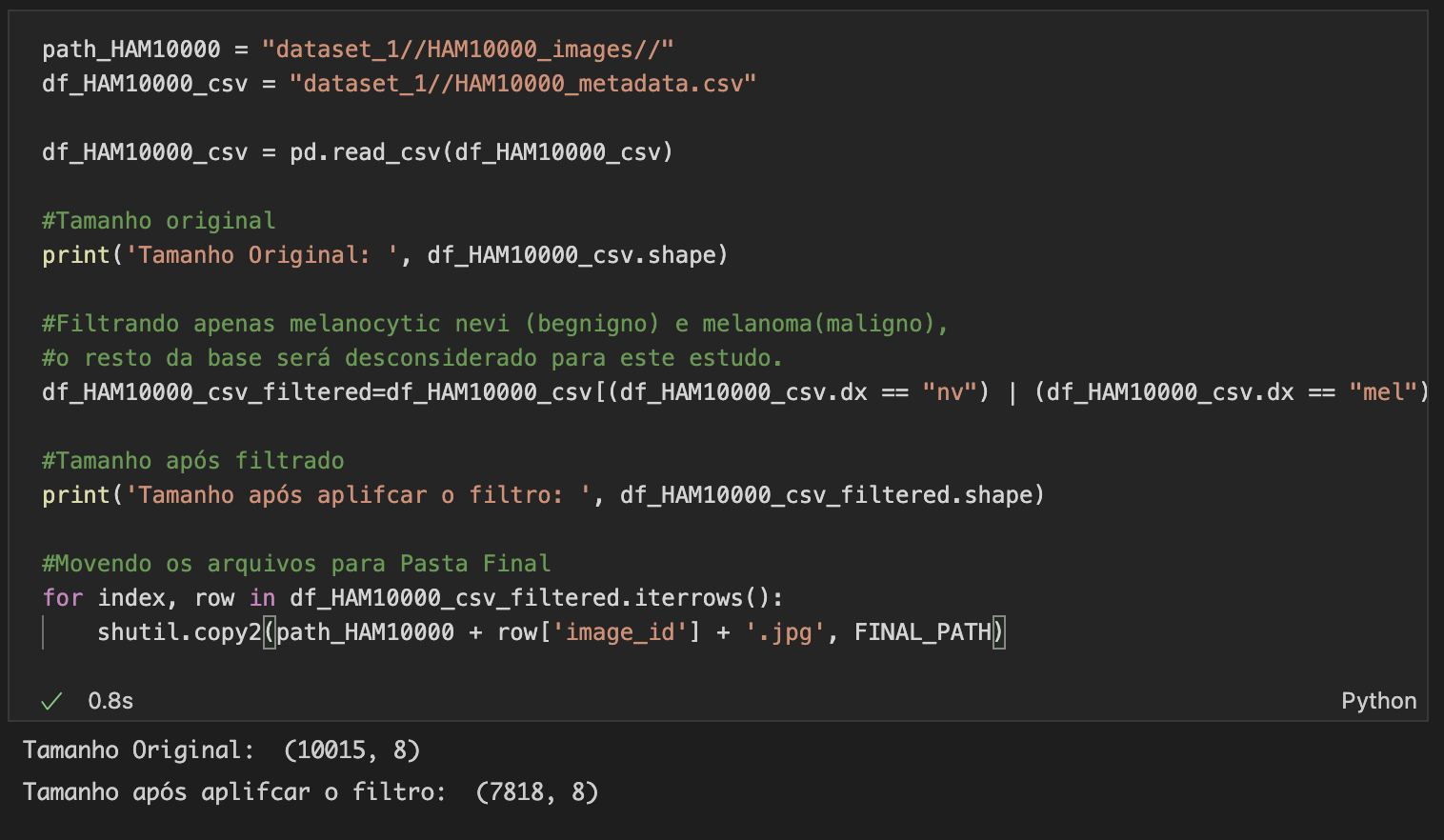
# 3. Processamento/Tratamento de Dados

Ambos os datasets possuíam suas peculiaridades, em um ( HAM10000 ) temos mais doenças de pele, inclui ainda um arquivo de metadados e dados que poderia enriquecer o modelo, melhorando acurácia, considerando que informações como região do corpo que apresenta a lesão e também idade do paciente são informações bastante relevantes até mesmo para avaliação de um profissional de saúde. A base da MED-NODE por outro lado possuía poucos registros e apenas duas classificações separadas através de diretórios.

Dessa forma foi necessário fazer uma limpeza nos dados vindos da HAM10000, eliminando portanto, cerca de 2200 imagens referentes a doença de Bowen, carcinoma basocelular, verruga seborréica, dermatofibroma e lesões vasculares, além do arquivo de metadados, que foi usado apenas no momento de tratamento para classificar as imagens e carregá-las para um diretório tido como final. Na sequência o arquivo também foi descartado e não foi usado durante os treinamentos ou avaliações dos modelos de machine learning.

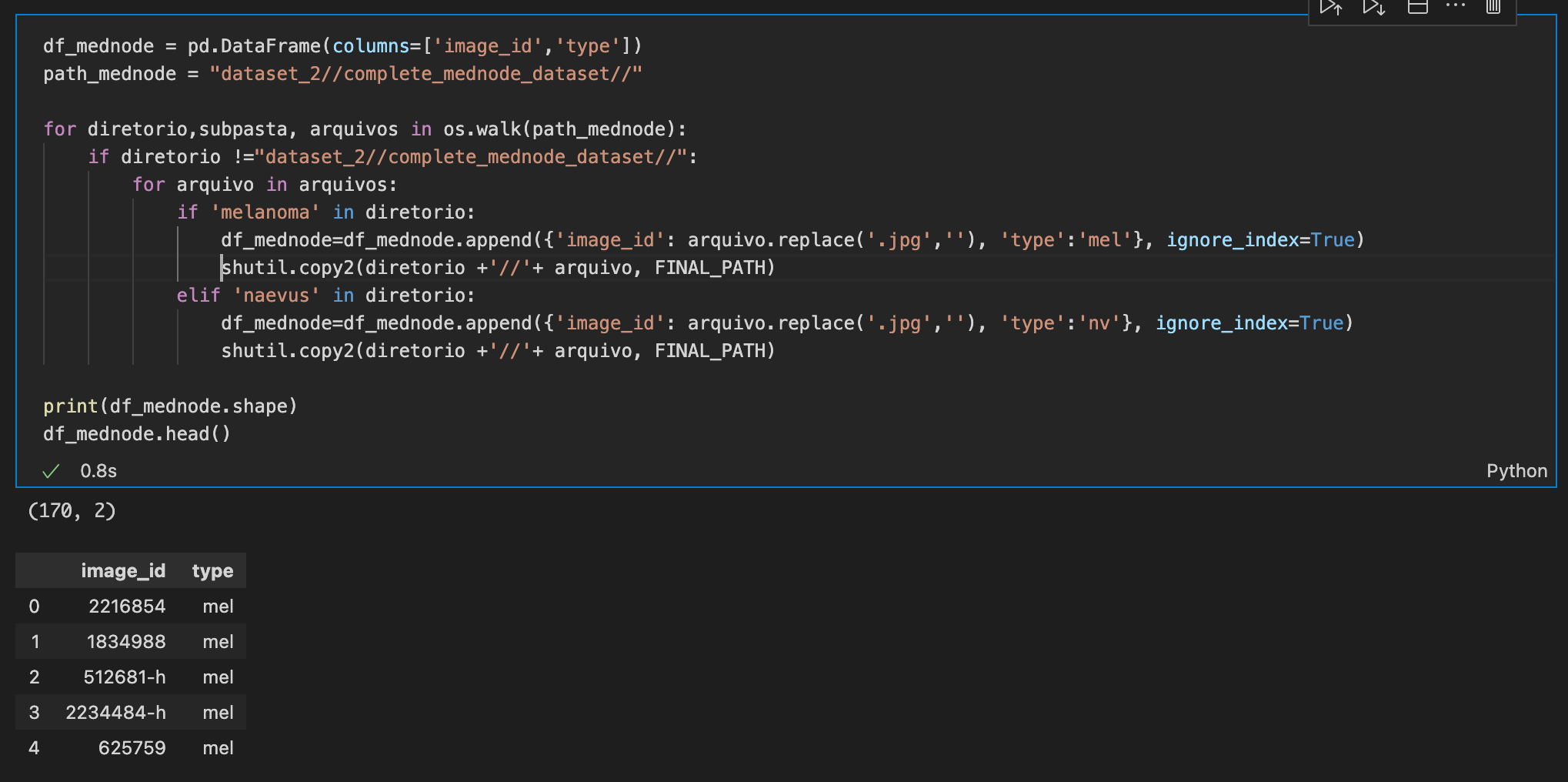
O tratamento em si de ambas as bases, se deu através de bibliotecas python de manipulação de arquivos e também pandas para ler e compor um arquivo destino por parte de ambas as bases de dados.

Para tratar a base da HAM10000, foi necessário ler o arquivo CSV a fim de identificar a classificação das lesões e também excluir aquelas que não fariam sentido neste projeto, para isso, foram excluídas do dataframe (criado pela leitura do arquivo de metadados providenciado), filtrando apenas as lesões referentes a nevus ou melanoma.



Fonte: elaborado pelo autor em 2022.

A imagem acima retrata a leitura do CSV, o filtro do dataframe e também o processo de cópia de todos os arquivos referentes a melanoma e nevus para o FINAL\_PATH. Também percebe-se a redução do dataframe que antes continha 10015 imagens e após o filtro foi reduzida para 7818 imagens.

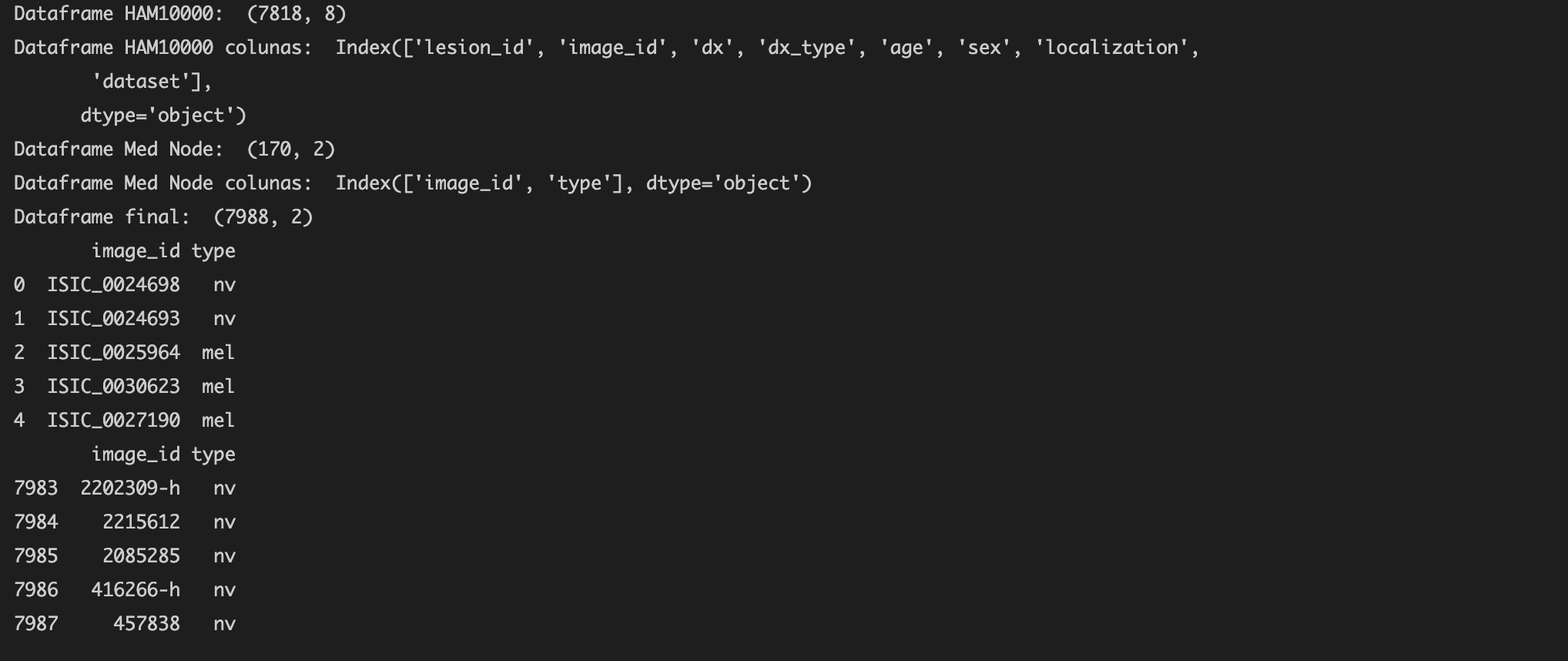
Por parte da base MED-NODE, nenhum tratamento foi necessário no sentido de limpeza ou tratamento dos dados, foi preciso apenas fazer a leitura das imagens nos diretórios correspondentes e juntar todas as imagens, após classificá-las. Conforme descrito pelo código destacado na imagem abaixo:  


Fonte: elaborado pelo autor em 2022.

O trecho de código acima, faz a leitura dos diretórios e imagens provenientes da base MED-NODE, foi necessário criar um dataframe pandas contendo o ID da imagem, nesse caso preenchido pelo nome do arquivo e também o tipo da lesão , extraído pelo nome da pasta anterior, se a imagem está na pasta melanoma receberá o label de "mel", se estiver na pasta nevus, receberá o label de "nv". O dataframe criado será unido posteriormente ao dataframe tratado do HAM10000. Além de construir o dataframe, o código destacado acima também copia os arquivos, utilizando da biblioteca python shuttil, para um diretório chamado de FINAL\_PATH, onde todas as imagens do projeto ficarão, sejam melanoma ou nevus.

Após o tratamento individual de ambas as bases e também o mover das imagens para o FINAL\_PATH, o código abaixo retrata o tratamento e união dos dataframes, tornando-os em uma único flat file (CSV). Dessa forma na hora de criar os modelos, basta ter acesso ao arquivo de metadados (CSV) com a classificação de imagens e as imagens em si.



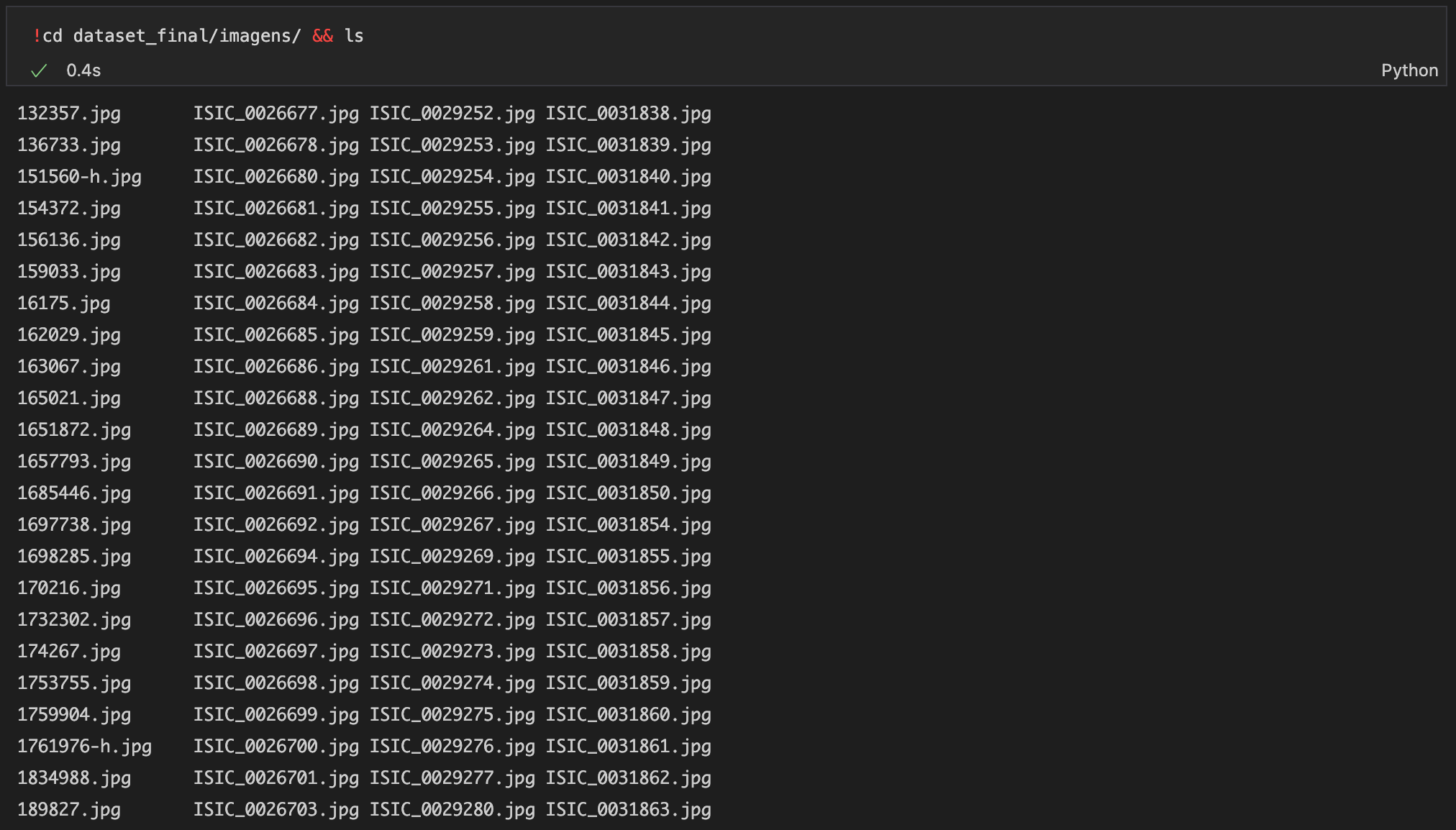


Fonte: elaborado pelo autor em 2022.

Sobre os resultados e outputs do código acima, podemos ver que por parte do HAM10000 foram ingeridas 7818 imagens, enquanto do MED-NODE apenas 170 imagens (todas que foram fornecidas), o dataframe final ficou com 7988 imagens e duas colunas: identificador único da imagem e tipo de lesão.

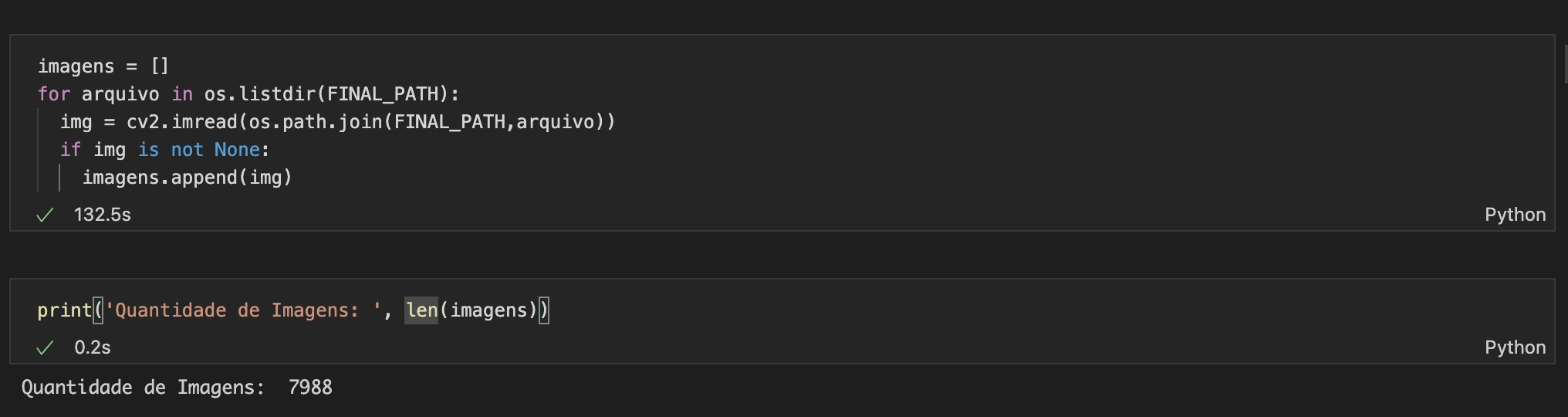
Embora o requisito do projeto fosse uma base de apenas 1000 registros, para este projeto fez sentido manter mais registros, a fim de obter uma melhor acurácia do modelo, considerando que a variável a ser usada serão apenas as imagens em si.

# 4. Análise e Exploração dos Dados

Para este projeto, não foi necessário realizar grandes análises ou mesmo exploração dos dados, dada a natureza dos modelos a serem usados. Ainda assim, foi preciso tratar alguns outros pontos, num primeiro momento a pasta definida como FINAL\_PATH foi acessada e os arquivos foram listados para confirmar que todos existiam no local correto. Conforme evidência abaixo:  


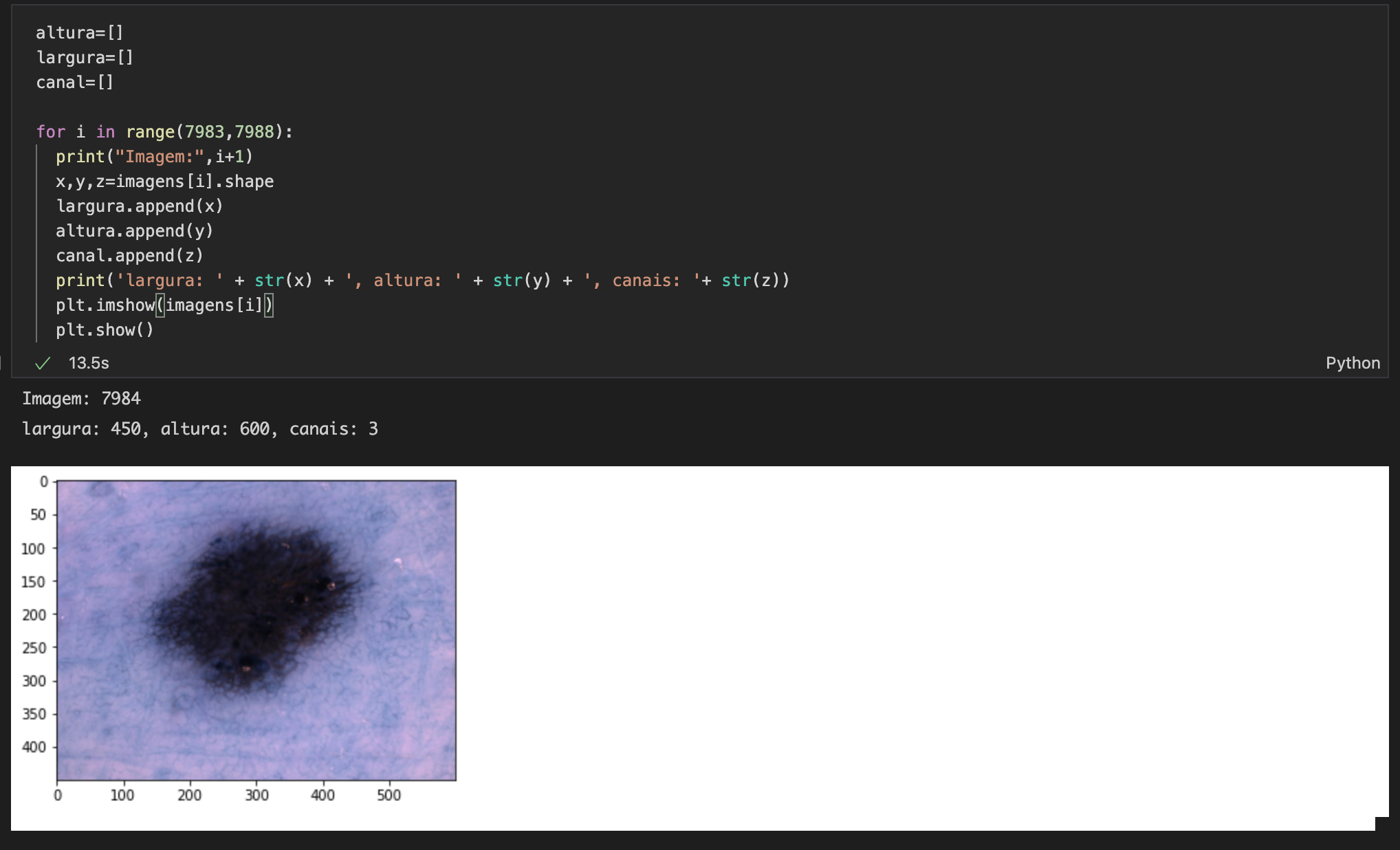
Fonte: elaborado pelo autor em 2022.

Em seguida listei a quantidade de imagens dentro da pasta, confirmando a existência de todas as 7988 imagens unidas pelo processo de tratamento dos dados.



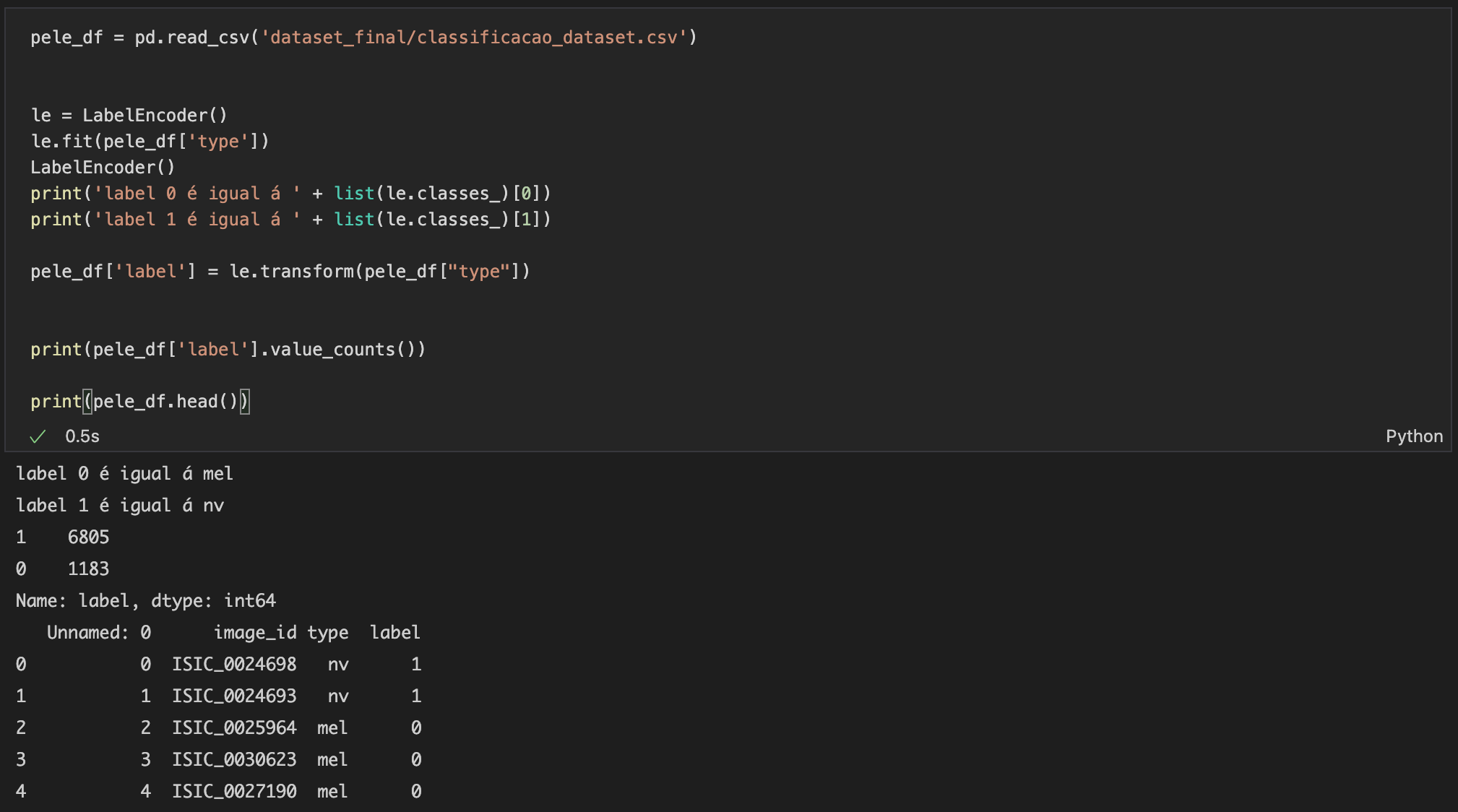
Fonte: elaborado pelo autor em 2022.

A fim de mostrar algumas das imagens e também confirmar informações como altura, largura e número de canais, que representa o número de componentes usados para representar cada pixel, 3 canais por exemplo representam RGB (Red, Green, Blue), obtidos através da biblioteca Python OpenCV.



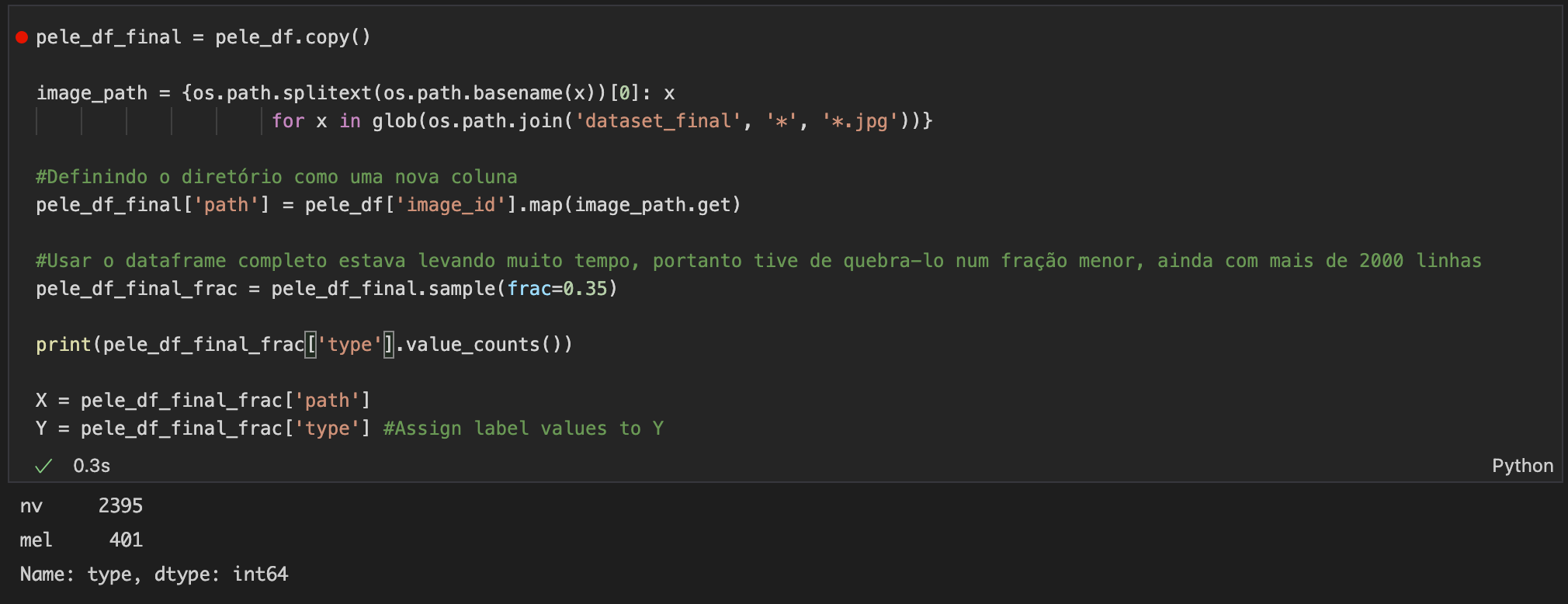
Fonte: elaborado pelo autor em 2022.

Por fim, antes de efetivamente fazer a criação de modelos, foi necessário ainda separar e classificar os arquivos com label antes de disponibilizá-los aos modelos. Para isso fiz a leitura do arquivo CSV gerado durante a etapa de Processamento/Tratamento de dados, além disso retornei também o as quantidades, mostrando que 6805 imagens são de nevus e 1183 imagens são de melanoma. O dataframe em si é composto por identificador de imagens, tipo e também label de maneira binária.



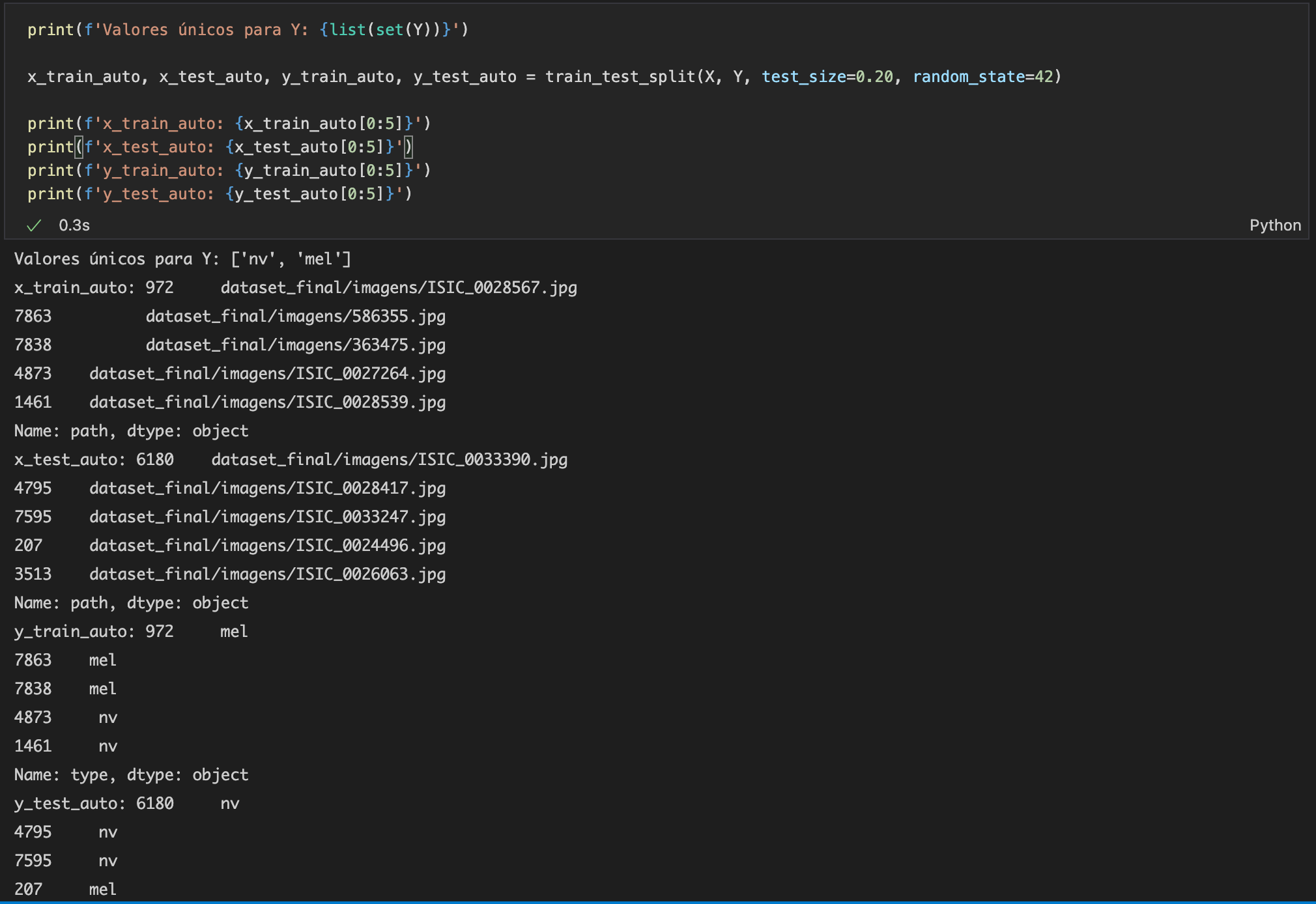
Fonte: elaborado pelo autor em 2022.

Seguindo com a análise e exploração dos dados, a base totalizava 7988 imagens, um eventual processamento de uma base desse tamanho seria muito custoso e tomaria tempo demasiado, portanto foi necessário reduzir a base de imagens para aproximadamente 3000 imagens, utilizando 35% da base tratada através de um fracionamento. Que resultou em 2367 imagens classificadas como nevuse outras 429 imagens classificadas como melanoma. Por fim, atribui ao eixo X a coluna de path contendo o arquivo da imagem que será analisada, ao eixo Y foi atribuído o label de classificação entre nv (nevus) e mel (melanoma).



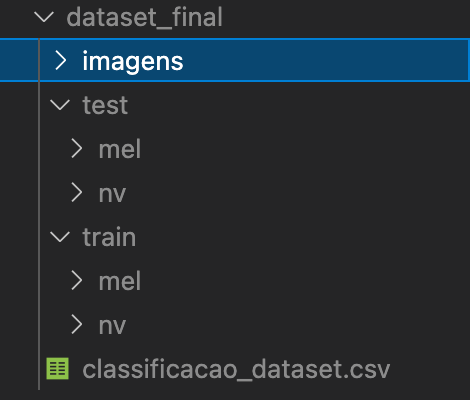
Fonte: elaborado pelo autor em 2022.

Na sequência base de treino e validação foi separada, usando a função de *train\_test\_split* fornecida pela biblioteca *sklearn.model\_selection*. A base foi divida usando o paradigma de 80/20, onde 80% da base será usada para treinamento e 20% da base validará o modelo. Finalmente, um parâmetro de *random\_state* foi usado, a fim de permitir a replicação da mesma base de imagens entre todos modelos testados, evitando assim algum viés individual na validação do modelo.



Fonte: elaborado pelo autor em 2022.

Por fim, para continuar com a construção dos modelos. Foram copiadas, usando novamente a biblioteca shuttil, as imagens da base de validação (test) e também treino (train), cada uma destas pastas contém ainda outras duas pastas uma para melanoma (mel) e outra para nevus (nv), conforme a seguinte estrutura:



Fonte: elaborado pelo autor em 2022.

Dentro desta mesma pasta dataset\_final, a subpasta de imagens, em evidência na imagem acima, contém todas as 7988 imagens provenientes do tratamento das bases HAM10000 e MED-NODE, além do arquivo CSV também gerado durante o tratamento das bases de dados. Este arquivo CSV não foi usado durante o treinamento dos modelos.

# 5. Criação de Modelos de Machine Learning

Antes de seguir com as explicações e também códigos utilizados para modelar e treinar os modelos, vale ressaltar alguns conhecimentos bastante explorados nos textos seguintes. Dentre eles:

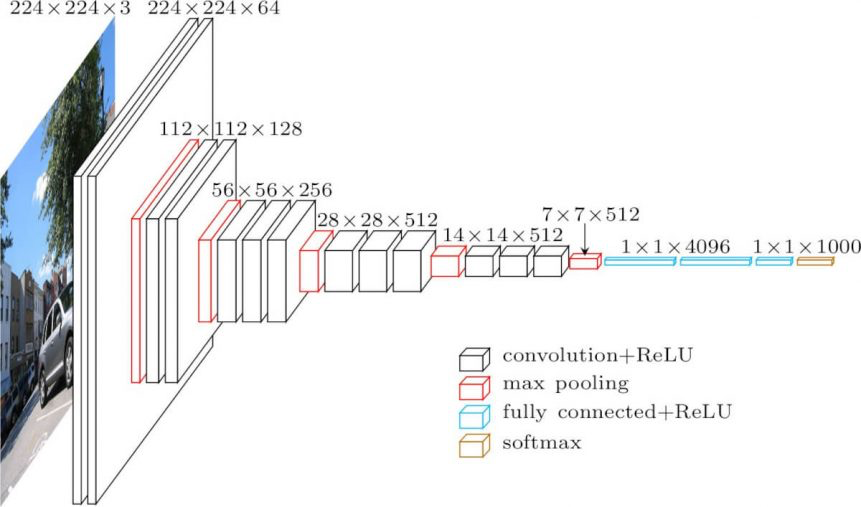
**CNN** ou **Convolutional Neural Network**, em tradução livre Redes Neurais Convolucionais, são um tipo de rede neural que tem obtido grande destaque e sucesso no que tange à visão computacional e processamento de imagem. Classificação de imagens e segmentação, além de detecção de objetos, processamento de vídeos, processamento de linguagem natural e reconhecimento de discurso, são apenas algumas das áreas de aplicação das CNNs. A capacidade de CNNs serem utilizadas de maneira espacial ou temporal buscando correlações nos dados é uma de suas principais propriedades. A CNN é composta por numerosas etapas de aprendizagem, que consistem em uma mistura de camadas de convolução, unidades de processamento não linear e camadas de subamostragem. CNN é uma rede de multicamadas hierarquias visionárias, onde cada camada conduz diversas transformações usando um banco de kernels convolucionais. O procedimento de convolução auxilia na extração de características valiosas de data points que estão conectados espacialmente. Exemplos de CNNs: *LeNet*, *AlexNet*, *ResNet*, *InceptionNet*, VGG, *PolyNet*, *Attention Based CNNs*, entre outros.

**Convolução** é um operador linear que, a partir de duas funções dadas, resulta numa terceira que mede a soma do produto dessas funções ao longo da região subentendida pela superposição delas em função do deslocamento existente entre elas.

**Modelos Pré-treinados**, eliminam a necessidade de grandes volumes de dados de treinamento, recursos de computação massivos e também amplo conhecimento de Inteligência Artificial (IA), através de modelos pré-treinados é possível acelerar os fluxos de trabalho e desenvolvimento.

Agora que esses conceitos já foram brevemente detalhados, é possível entender um pouco mais dos seguintes modelos de machine learning:

## 5.1 VGG-16

O modelo pré treinado *Very Deep Convolutional Networks for Large-Scale Image Recognition* (em tradução literal: Redes Convolucional Muito Profundas para Reconhecimento de Imagem em Larga Escala), mais conhecido como VGG-16 é um modelos pré-treinados mais populares para classificação de imagem. Introduzido na conferência ILSVRC de 2014, ainda é um dos modelos mais assertivos e modernos no tempos de hoje, cerca de 8 anos após seu anúncio. Desenvolvido pelo Grupo de Visuais Gráficos da Universidade de Oxford. O VGG-16 bateu o até então padrão *AlexNet* e rapidamente foi adotado por pesquisadores e também por indústrias para tarefas de classificação de imagem. Segue a arquitetura por trás do VGG-16:  
Fonte: Coletado por Purva91 em 2020.

Abaixo um modelo mais intuitivo do layout usado pelo modelo VGG-16:



Fonte: Coletado por Purva91 em 2020.

O modelo é composto pelas seguintes camadas:

* + Camadas de Convolução: 13
  + Camadas de Conjugação (também chamadas de Pool): 5
  + Camadas de Compactação: 3

Explorando em detalhes:

1. Entrada: Imagem com dimensões: altura de 224, largura de 224 e 3 camadas.
2. Camada de Convolução Conv1:
   1. Conv1-1: 64 filtros;
   2. Conv1-2: 64 filtros e também *Max Pooling;*
   3. Imagem com as seguintes dimensões: 224 x 224.
3. Camada de Convolução Conv2: Filtros aumentados para 128.
   1. Imagem ingerida com dimensões reduzidas: 112 x 112;
   2. Conv2-1: 128 filtros;
   3. Conv2-2: 128 filtros;
4. Camada de Convolução Conv3: Novamente, filtros dobrados para 256, mais uma camada de convolução
   1. Imagem ingerida com dimensões reduzidas: 56 x 56;
   2. Conv3-1: 256 filtros;
   3. Conv3-2: 256 filtros;
   4. Conv3-3: 256 filtros e também *Max Pooling.*
5. Camada de Convolução Conv4: Semelhante ao Conv3, mas agora com 512 filtros:
   1. Imagem ingerida com dimensões novamente reduzidas: 28 x 28.
   2. Conv4-1: 512 filtros;
   3. Conv4-2: 512 filtros;
   4. Conv4-3: 512 filtros e *Max Pooling.*
6. Camada de Convolução Conv5: Semelhante a Conv4.
   1. Imagem ingerida com dimensões novamente reduzidas: 14 x 14.
   2. Conv5-1: 512 filtros;
   3. Conv5-2: 512 filtros;
   4. Conv5-3: 512 filtros;
   5. Na saída as dimensões da imagem são de 7 x 7. Nesse momento a saída desta camada gera um vetor.
7. Totalmente Conectada/Compactação FC1: 4096 nós, gerando um vetor de features de tamanho 1 x 4096
8. Totalmente Conectada/Compactação FC2:4096 nós, gerando novamente um vetor de features de tamanho 1 x 4096.
9. Totalmente Conectada/Compactação FC3: 4096 nós, gerando 1000 canais para 1000 classes. Então é passado para função de ativação do *Softmax*
10. Camada de Saída

Como pode ser visto, o modelo é sequencial por natureza e faz uso de muitos filtros. Nesse estágio, pequenos filtros de 3 \* 3 são usados para reduzir o número de parâmetros, todas as camadas ocultas usam da função de ativação ReLU. Mesmo assim, o número de parâmetros é da ordem de 138 bilhões, o que torna o processamento todo, muito mais devagar em modelos consideravelmente maiores, ou seja, dado a natureza dos filtros o tempo de processamento escala exponencialmente em modelos maiores. Durante os testes deste projeto, este modelo levou cerca de 28 minutos em treinamento.

Existem ainda variações ao modelo VGG16, que no geral são apenas aprimoramentos em relação ao mesmo, como por exemplo o VGG19, composto por 19 camadas. Abaixo segue o detalhamento, junto aos códigos aplicando o modelo VGG16:

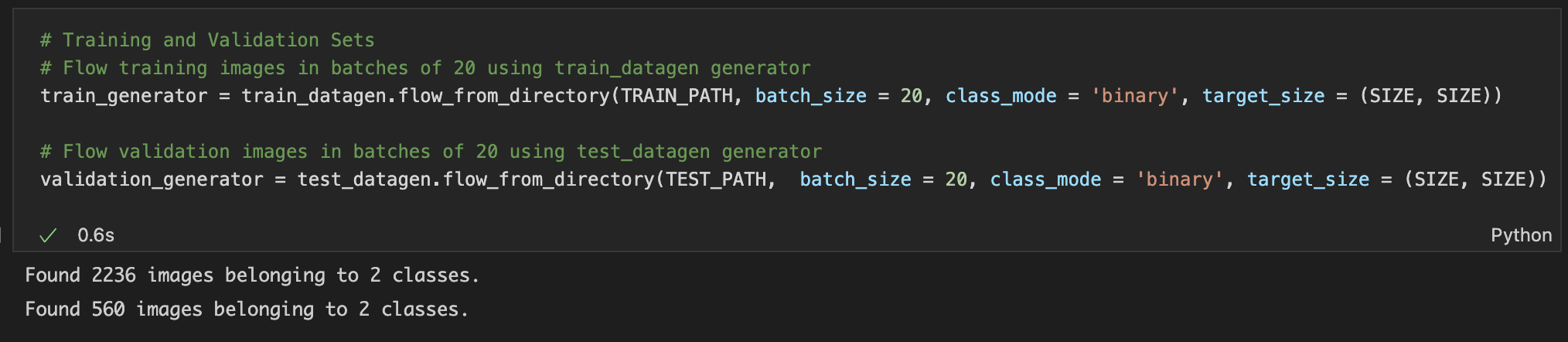
### 5.1.1 Aumentação de dados



Fonte: elaborado pelo autor em 2022.

Para aumentar a base de dados foi utilizado o *ImageDataGenerator*, construído pela biblioteca TensorFlow e Keras. É utilizado para gerar lotes de imagens em formato de dados de imagem *tensor* (próprio para modelos de machine learning baseados em TensorFlow), contendo dados aumentados em tempo real.

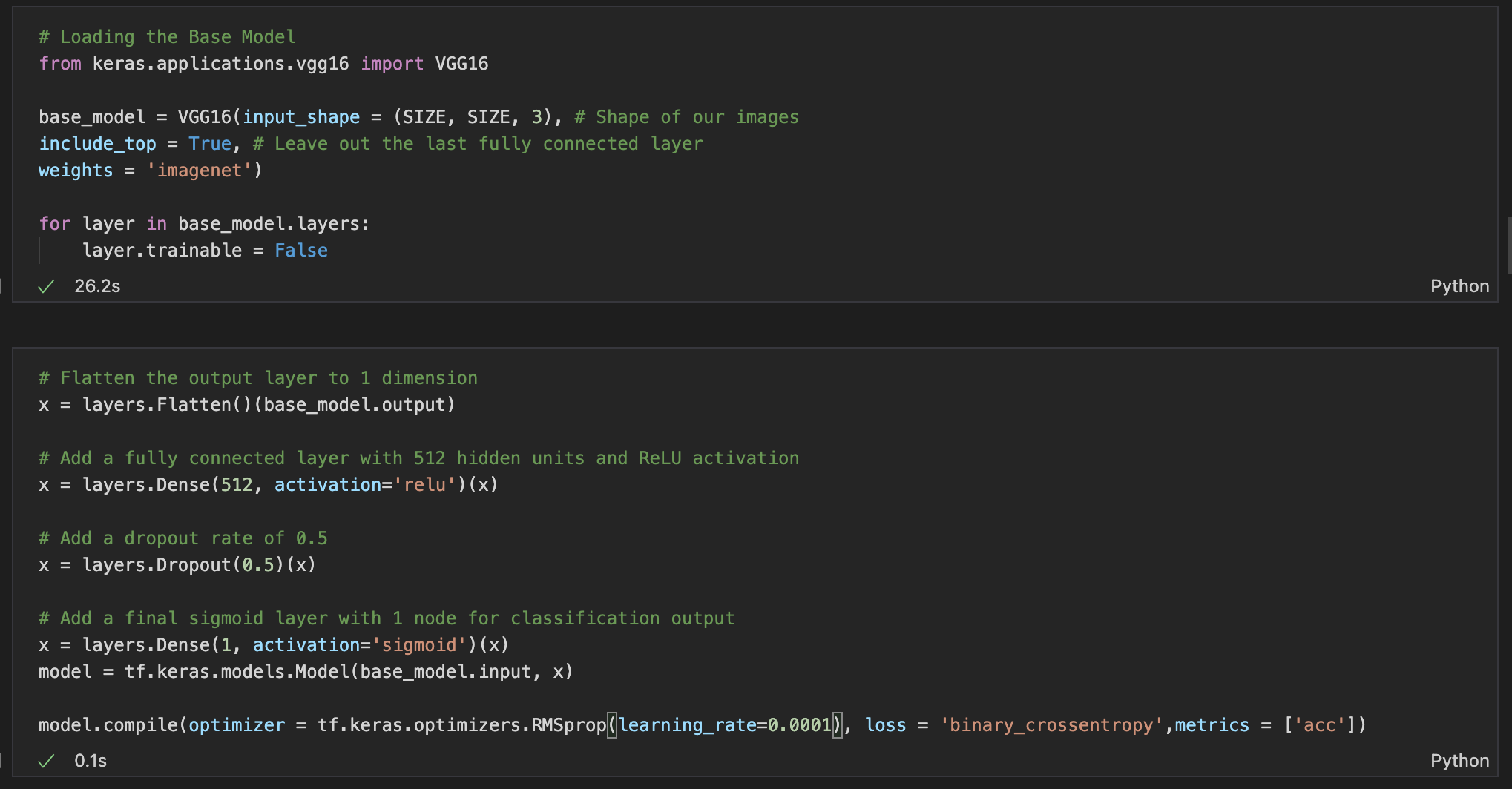
### 5.1.2 Separação de bases de treinamento e validação



Fonte: elaborado pelo autor em 2022.

Etapa simples onde é feita a separação entre as bases de treino e validação, através dos diretórios definidos na etapa de Análise e Exploração dos Dados. Foi utilizado a função de *flow\_from\_directory*, classificando as imagens de maneira binária. A base de treino contém 2236 imagens, classificadas entre melanoma ou nevus*.* A base de validação, por outro lado, contém 560 imagens, também classificadas em duas classes: melanoma ou nevus*.*

### 5.1.3 Carregamento do modelo e compilação



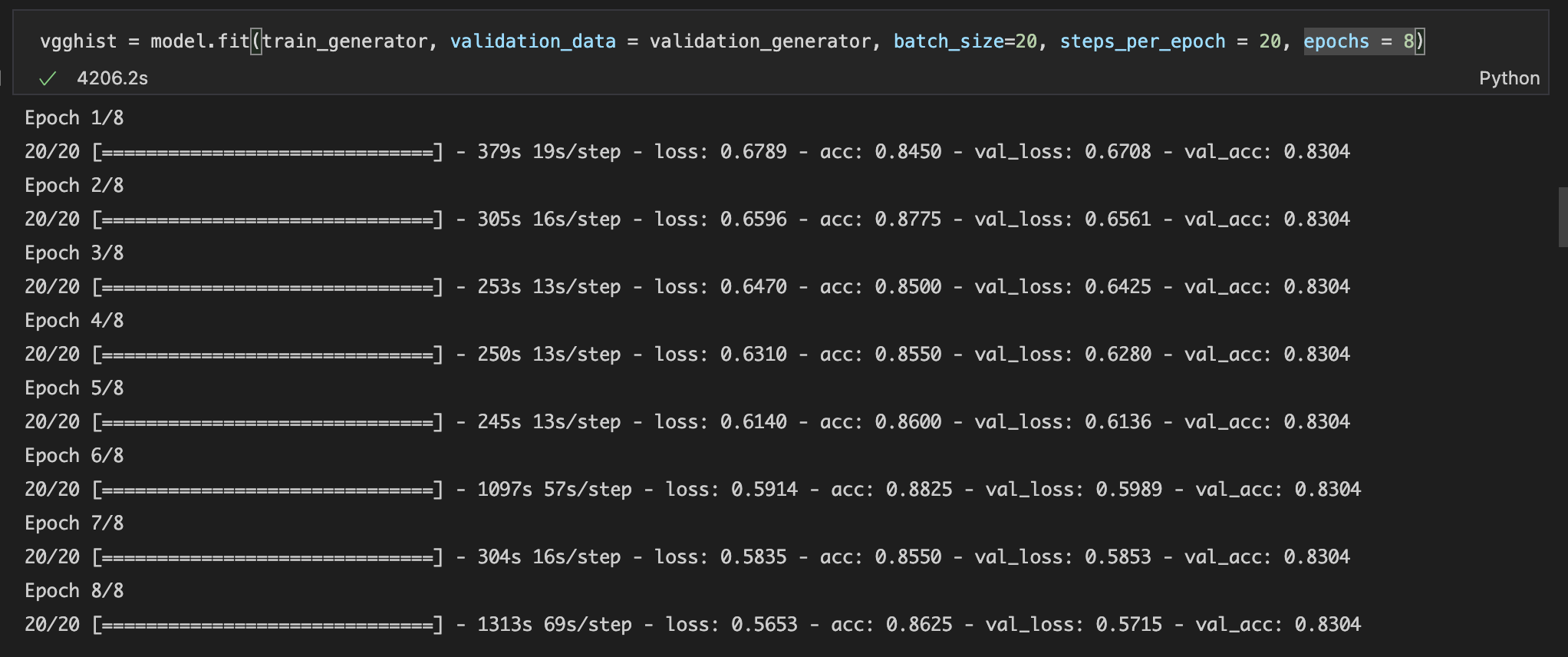
Fonte: elaborado pelo autor em 2022.

Nesta etapa, é feito o carregamento do modelo VGG16. Conforme descrito acima, o VGG16 aceita como entrada na primeira camada uma imagem de 224 pixels de altura por 224 pixels de largura e 3 classes. O *RMSprop* foi o otimizador utilizado, com índice de aprendizado de 10ˆ-4. O otimizador RMSprop implementa o algoritmo de *RMSprop*, que tem como base os seguintes conceitos matemáticos:

* Manter a média móvel (descontada) dos gradientes quadráticos.
* Divide o gradiente pela raiz da média.

*RMSprop*, faz uso de momentum plano, ao invés do momentum *Nesterov*. A versão centralizada deste otimizador mantém ainda as médias móveis dos gradientes, e usa essa média para estimar a variação.

### 5.1.4 Treinamento



Fonte: elaborado pelo autor em 2022.

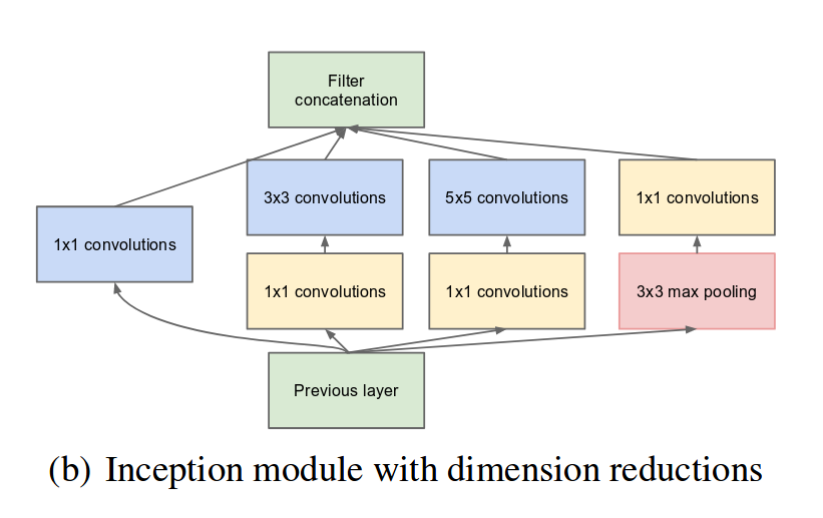
O treinamento deste modelo faz uso da função *fit*, inteirando por lotes de 20. Após alguns testes locais, ficou claro que 20 passos para cada *epoch* e 8 *epochs*,foram mais do que suficientes para atingir uma acurácia de 83,04%.

## 5.2 Inception:

*Inception* é um modelo pré-treinado para classificação de imagens, modelos pré-treinados são alternativas menos custosas para treinamento em máquinas locais de pequena capacidade computacional. O modelo pré-treinado *Inception* é segurado por ninguém menos que o Google, era inicialmente conhecido como *GoogLeNet*, e agora mais recentemente como Inception.

O documento original propôs o modelo *Inceptionv1*, contendo 7 milhões de parâmetros, muito menor que outros modelos predecessores como VGG e *AlexNet*. Adicionando a isso um índice menor de falha, possível ver abaixo no detalhamento do modelo. Além disso, a maior inovação presente no documento original que propôs esse modelo, foi na verdade outro detalhamento do módulo Inception.

Em outras palavras, o módulo Inception apenas executa convoluções com diferentes filtros aplicados na entrada dos dados, performando portanto *Max Pooling*, e concatenando o resultado com o próximo módulo Inception. A introdução de 1 para 1 nas operações de convolução reduziu o número de parâmetros drasticamente.

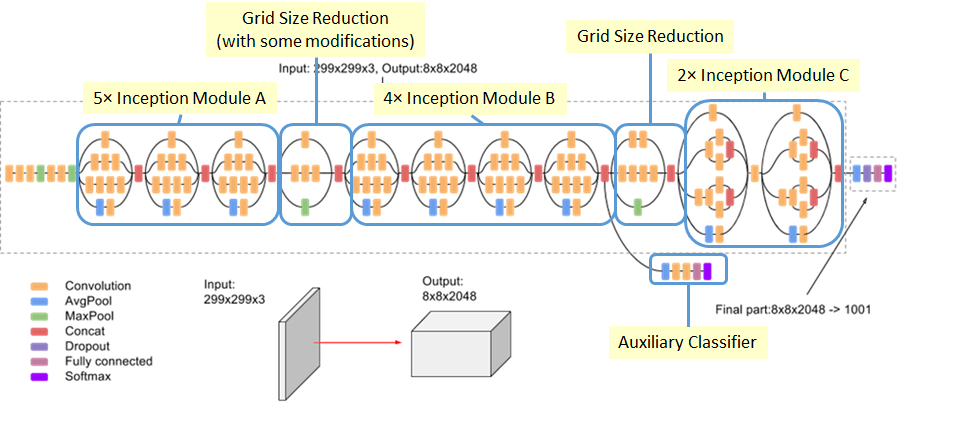


Fonte: Coletado por Purva91 em 2020.

Em tradução literal acima: (b) Módulo Inception com reduções de dimensões.

O modelo pré-treinado *Inceptionv1* é composto de 22 camadas, uma redução também massiva de parâmetros tornou esse modelo dificilmente batível por outros. Já o modelo *Inceptionv2*, teve ainda um grande aprimoramento em relação ao *Inceptionv1*, o que basicamente aumentou a acurácia e tornou também com o passar do tempo o modelo menos complexo. No mesmo documento que o modelo *Inceptionv2* foi proposto, o autor introduziu ainda o modelo *Inceptionv3* com mais algumas melhorias em relação à versão anterior também recém introduzida. Dentre as maiores melhorias, essas são as de maior destaque:

* Introdução da Normalização por Lote;
* Mais fatorização;
* Otimizador *RMSprop.*

Fonte: Coletado por Purva91 em 2020.

Não é possível explicar em detalhes todas as camadas e detalhes do modelo *Inception* dentro deste pequeno texto acadêmico. Entretanto é possível ver que o número de camadas no modelo *Inceptionv2* é de 42 camadas, versus as 22 camadas do *Inceptionv1*, o modelo *Inceptionv3* reduziu ainda o índice de erro para apenas 4.2%.

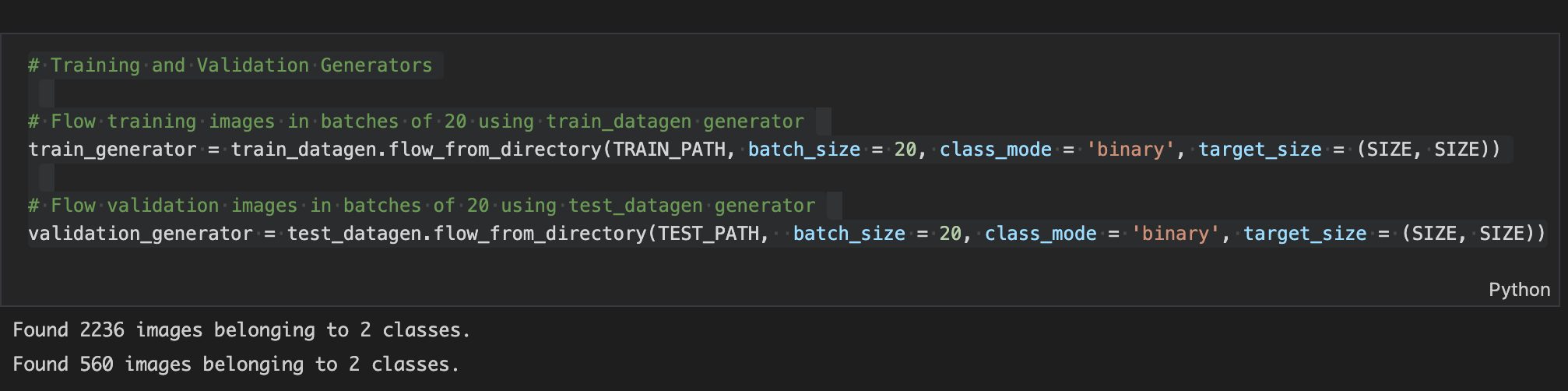
Entrando agora no desenvolvimento do modelo em si, aplicado à base de imagens coletada e contexto de classificação de imagens de lesões de pele. Segue o detalhamento dos códigos:

### 5.2.1 Aumentação de dados

Fonte: elaborado pelo autor em 2022.

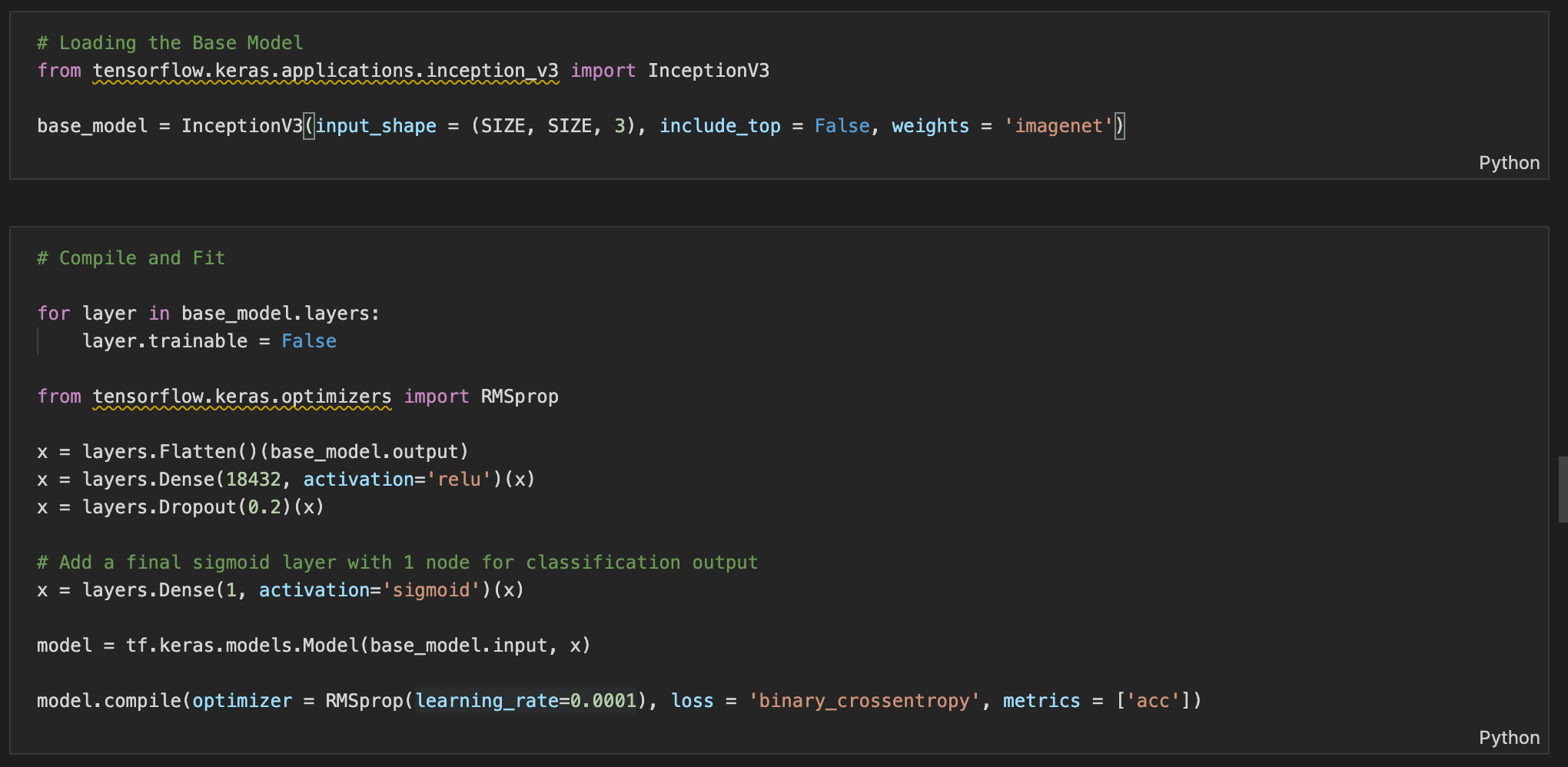
Para aumentar a base de dados foi utilizado o *ImageDataGenerator*, construído pela biblioteca TensorFlow e Keras. É utilizado para gerar lotes de imagens em formato de dados de imagem *tensor* (próprio para modelos de machine learning baseados em TensorFlow), contendo dados aumentados em tempo real. Este é um passo bastante importante pois uma etapa com esta pode aumentar consideravelmente a acurácia do modelo e ao mesmo tempo reduzir possíveis vieses dos dados.

### 5.2.2 Separação de bases de treinamento e validação

Fonte: elaborado pelo autor em 2022.

Etapa simples onde separamos as bases de treino e validação, através dos diretórios definidos na camada de Análise e Exploração dos Dados. Foi utilizado a função de *flow\_from\_directory*, classificando as imagens de maneira binária. A base de treino contém 2236 imagens, classificadas entre melanoma ou nevus*.* A base de validação, por outro lado, contém 560 imagens, também classificadas em duas classes: melanoma ou nevus*.*

### 5.2.3 Carregamento do modelo e compilação

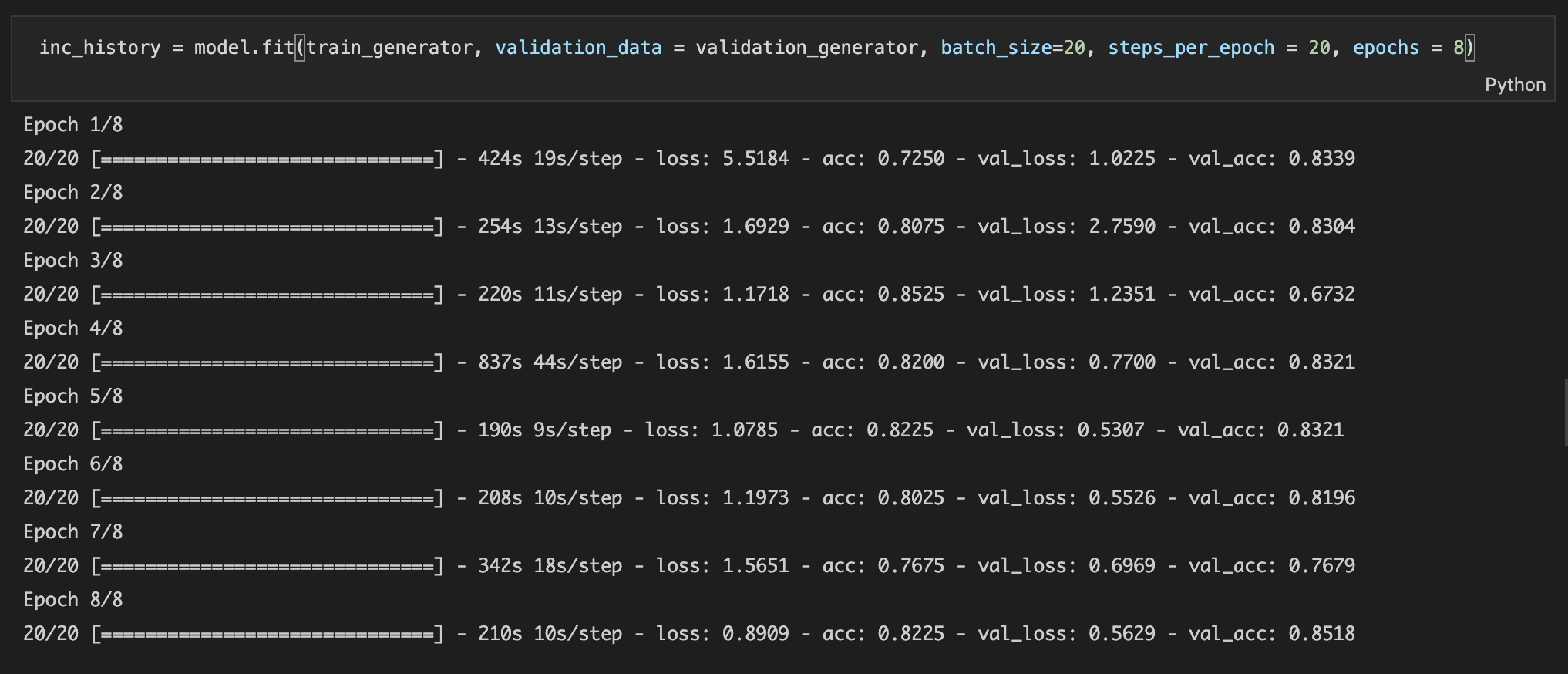
Fonte: elaborado pelo autor em 2022.

Nesta etapa, é feito o carregamento do modelo *Inceptionv3*, mais moderno modelo de *Inception*. Para isso seguimos com o tamanho de imagem de 150 pixels por 150 pixels. Para Otimização deste modelo foi utilizado o mais novo *RMSprop* com índice de aprendizado de 10ˆ-4. Permitindo um leve crescimento de aprendizado na variação de parâmetros, tornando os ajustes ao longo das interações mais refinados. O otimizador *RMSprop* implementa o algoritmo de *RMSprop*, que tem como base os seguintes conceitos matemáticos:

* Manter a média móvel (descontada) dos gradientes quadráticos.
* Divide o gradiente pela raiz da média.

*RMSprop*, faz uso de momentum plano, ao invés do momentum *Nesterov*. A versão centralizada deste otimizador mantém ainda as médias móveis dos gradientes, e usa essa média para estimar a variação.

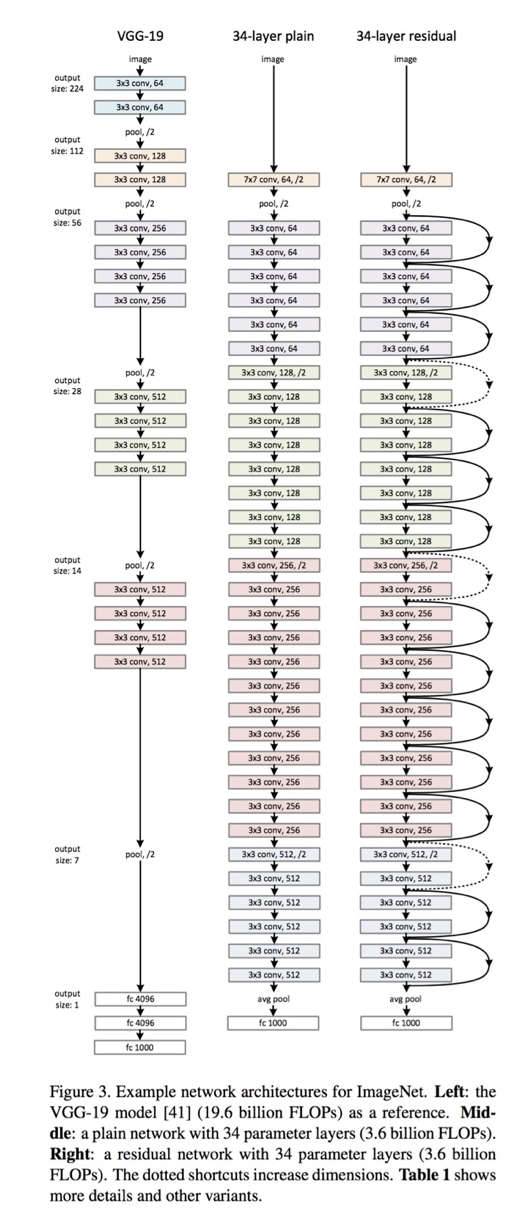
### 5.2.4 Treinamento

Fonte: elaborado pelo autor em 2022.

O treinamento deste modelo faz uso da função fit, inteirando por lotes de 20. Após alguns testes locais, ficou claro que 20 passos para cada *epoch* e 8 *epochs,* foram mais do que suficientes para atingir uma acurácia de 85,18%.

## 5.3 ResNet50

Também conhecido como Residual Net (Rede Residual em tradução livre) ou ResNet, foi lançado em 2015. A versão que será testada abaixo é uma versão melhorada ao longo dos anos. A principal motivação por trás deste modelo foi evitar uma acurácia baixa, considerando que o modelo se tornou uma rede neural ainda mais profunda. Adicionalmente, este modelo ResNet foi criado mirando a solução de um problema crônico dos modelos *Gradient Descent* e *Vanishing Gradient*. Abaixo segue uma imagem com exemplo de arquitetura abordada pelo ResNet34, bastante semelhante ao ResNet50 usado neste projeto.



Fonte: Coletado por Purva91 em 2020.

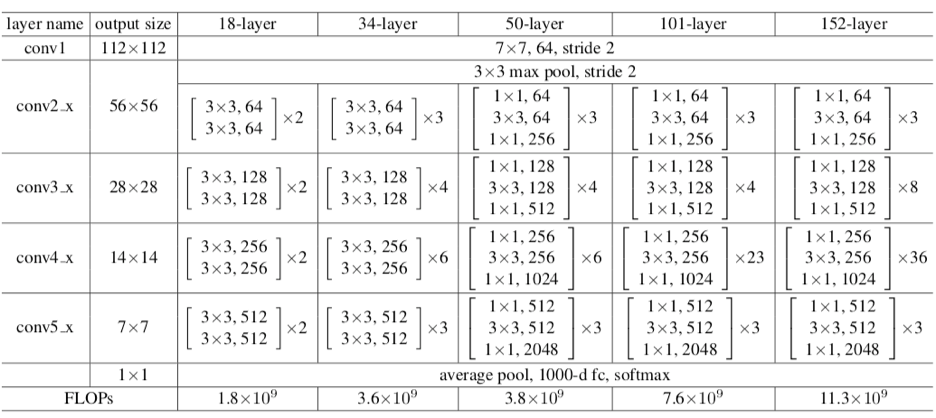
Como detalhado acima, após iniciar com uma única camada convolucional e *MAX Pooling* , vem 4 camadas similares com apenas uma variação de filtros, todos usando operações convolucionais 3 \* 3. Além disso, sempre que ocorrem duas convoluções, uma camada intermediária é pulada ou de certo modo ignorada. Este é o conceito principal por trás dos modelos ResNet. Essas conexões ignoradas são chamadas de 'conexões de atalho de identidade' e usa o que é chamado de blocos residuais:



Fonte: Coletado por Purva91 em 2020.

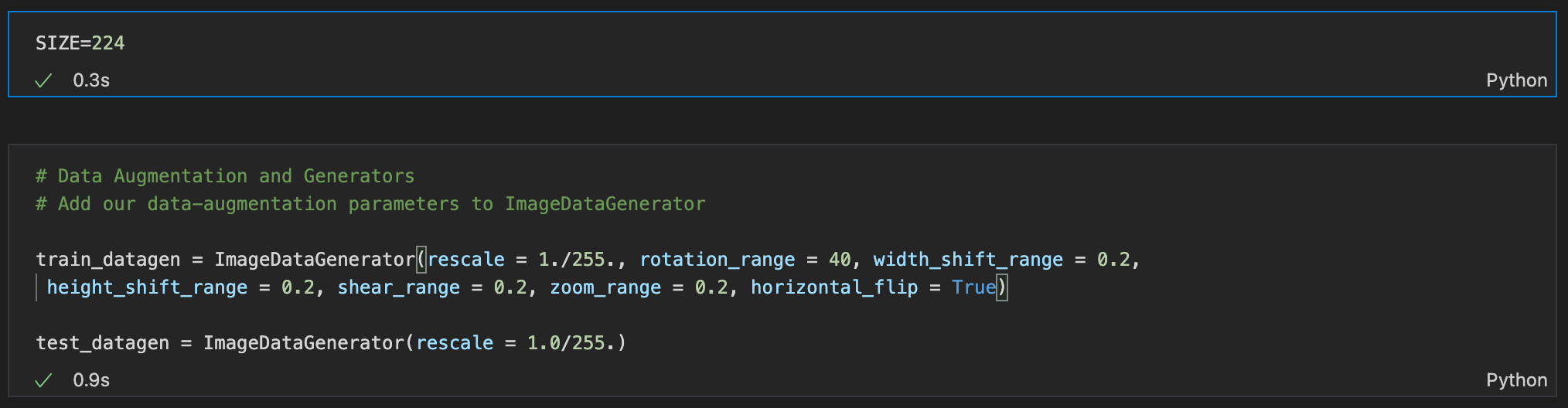
Em outras palavras, os autores do ResNet propõem que ajustar um mapeamento residual é muito mais fácil do que ajustar o mapeamento real e, assim, aplicá-lo em todas as camadas. Outro ponto interessante a ser observado é que os autores do ResNet são da opinião de que quanto mais camadas empilhamos, o modelo não deve ter um desempenho pior.

Ao contrário do visto no *Inception*, é quase semelhante ao VGG16 no sentido de que está apenas empilhando camadas umas sobre as outras. ResNet apenas altera o mapeamento subjacente. O modelo ResNet tem muitas variantes, das quais a mais recente é ResNet152. A seguir está a arquitetura da família ResNet em termos de camadas usadas:

Fonte: Coletado por Purva91 em 2020.

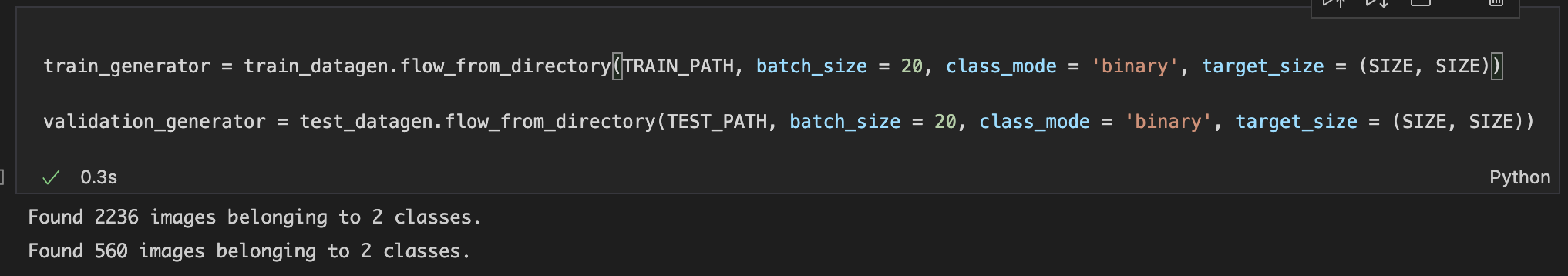
Seguindo agora com a aplicação no contexto deste projeto:

### 5.3.1 Aumentação de dados

Fonte: elaborado pelo autor em 2022.

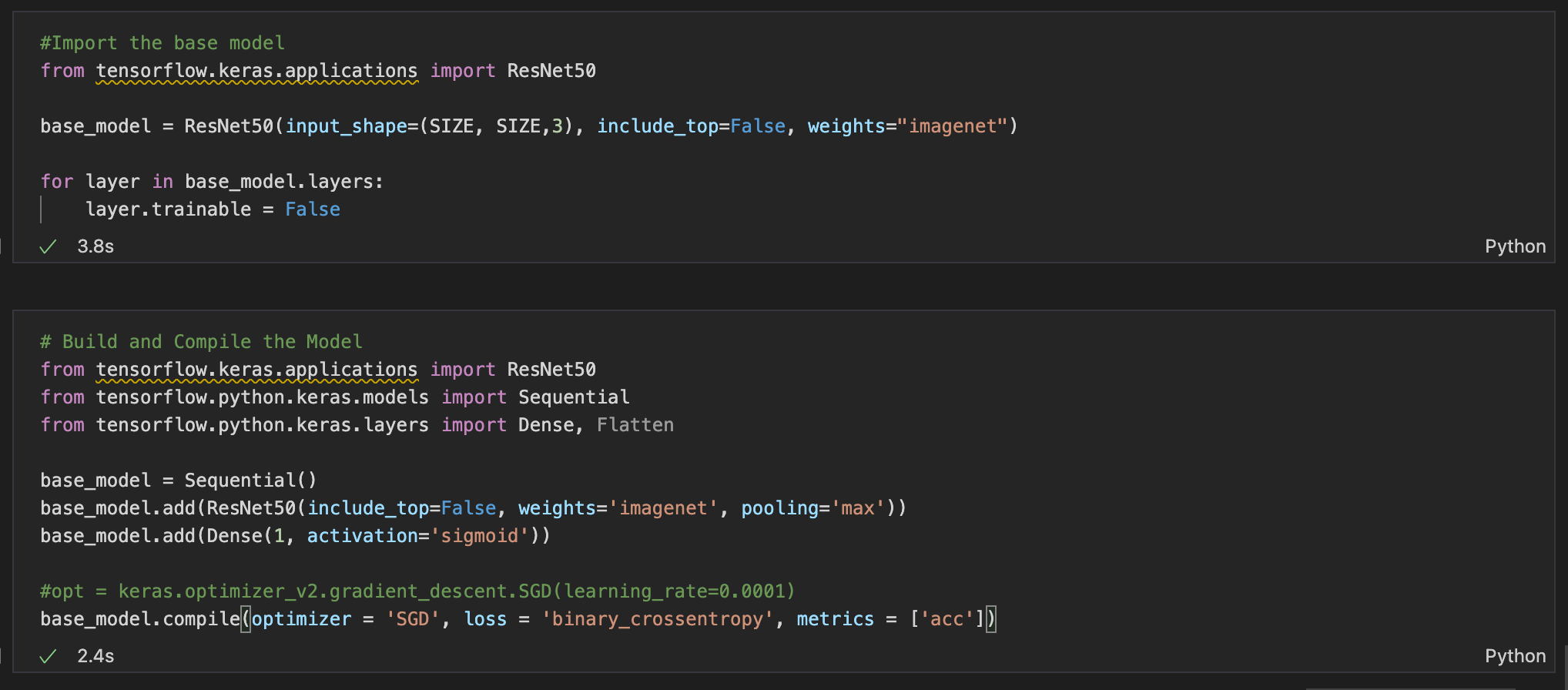
Para aumentar a base de dados foi utilizado o *ImageDataGenerator*, construído pela biblioteca TensorFlow e Keras. É utilizado para gerar lotes de imagens em formato de dados de imagem *tensor* (próprio para modelos de machine learning baseados em TensorFlow), contendo dados aumentados em tempo real. Semelhante às mesmas etapas do modelo VGG16 e Inception.

### 5.3.2 Separação de bases de treinamento e validação

Fonte: elaborado pelo autor em 2022.

Etapa simples onde separamos as bases de treino e validação, através dos diretórios definidos na camada de Análise e Exploração dos Dados. Foi utilizado a função de *flow\_from\_directory*, classificando as imagens de maneira binária. A base de treino contém 2236 imagens, classificadas entre melanoma ou nevus*.* A base de validação, por outro lado, contém 560 imagens, também classificadas em duas classes: melanoma ou nevus*.*

### 5.3.3 Carregamento do modelo e compilação

Fonte: elaborado pelo autor em 2022.

Nesta etapa, é feito o carregamento do modelo ResNet50. Para isso, o tamanho da imagem é configurado para 224 pixels por 224 pixels. Para otimização deste modelo foi utilizado SGD, ou Gradiente Estocástico Decrescente (*Stochastic gradient descent*, em inglês), é um método iterativo para otimizar uma função objetiva com propriedades de suavidade adequadas (por exemplo, diferenciável ou sub diferenciável). Especialmente em problemas de otimização de alta dimensão, isso reduz a carga computacional muito alta, alcançando interações mais rápidas na troca para uma taxa de convergência mais baixa.

### 5.3.4 Treinamento

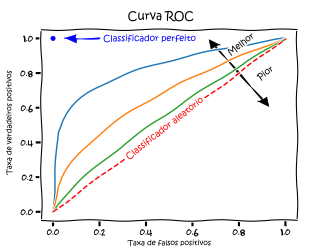
# 

Fonte: elaborado pelo autor em 2022.

O treinamento deste modelo faz uso da função *fit*, inteirando por lotes de 20. Acurácia e valor de perda serão explorados abaixo.

# 6. Interpretação dos Resultados

Conceitos importantes de serem abordados, antes de efetivamente demonstrar os resultados:  
 A **Curva Característica de Operação do Receptor** (Curva COR), ou, do inglês, Receiver Operating Characteristic Curve (ROC curve). Ou, simplesmente, curva ROC, é uma representação gráfica que ilustra o desempenho (ou performance) de um sistema [classificador binário](https://pt.wikipedia.org/w/index.php?title=Classificador_bin%C3%A1rio&action=edit&redlink=1) à medida que o seu limiar de discriminação varia. A curva ROC é também conhecida como curva de característica de operação relativa, porque o seu critério de mudança é resultado da operação de duas características (PV e PF). A curva ROC é obtida pela representação da razão RPV = Positivos Verdadeiros / Positivos Totais versus a razão RPF = Positivos Falsos / Negativos Totais, para vários valores do limiar de classificação. O RPV é também conhecido como sensibilidade (ou taxa de verdadeiros positivos), e RPF = 1-especificidade ou taxa de falsos positivos. A especificidade é conhecida como taxa de verdadeiros negativos (RVN).



Fonte: elaborado por Dcbmariano em Abril/2021

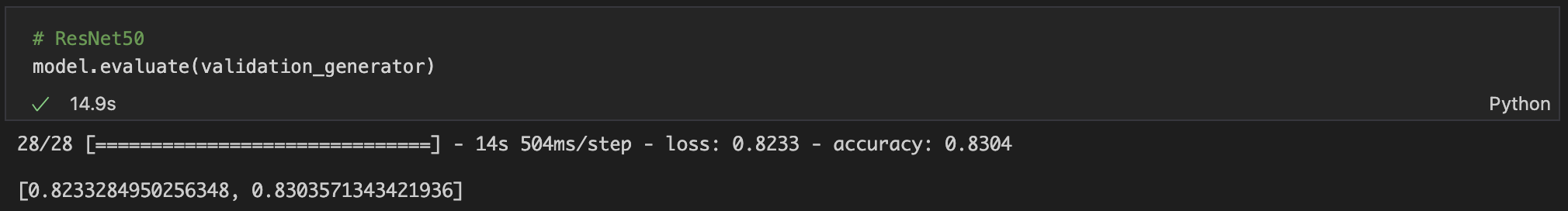
A análise ROC fornece ferramentas para selecionar modelos possivelmente ideais (modelos ótimos) e descartar modelos não tão ótimos, independentemente (e antes de especificar) o contexto de custos ou a distribuição de classe. A análise ROC está relacionada de forma direta e natural com a análise de custo/benefício do diagnóstico.

**Loss** ou **valor de perda**, é usada para otimizar algoritmos de *Machine Learning*. O valor de perda é calculado durante treinamento e validação, sua interpretação está baseada em quão bem o modelo está performando essas duas etapas (treinamento e validação). É a soma de erros feita por cada exemplo durante o treinamento e validação. O valor de perda implica em quão pobre ou bem o modelo se comporta após cada etapa iteração de otimização. Quanto menor a perda, melhor.

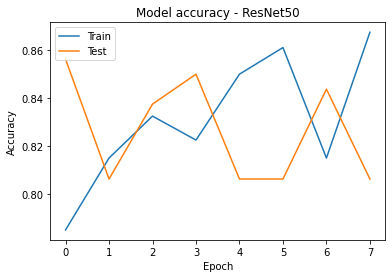
**Accuracy** ou **Precisão**, é a métrica usada para mensurar a performance do algoritmo de uma maneira interpretativa . A precisão do modelo é geralmente determinada após os modelos do parâmetro serem decididos, e é calculada em forma de porcentagem. Essa medida indica quão preciso é ao classificar registros comparando com os dados reais e conhecidos. Quanto maior a precisão, melhor.

### 6.1 ResNet50

O modelo ResNet50, apresentou valor de perda de 82,33% após última iteração e acurácia de 83,04%. Ao longo das iterações de treinamento o modelo teve um pico inferior de 80% de acurácia e pico superior de 85,62%. A curva ROC se mostrou ruim, classificando erroneamente vários registros, talvez com mais iterações e refino fosse possível inverter essa curva. Afinal, se um modelo classifica erroneamente muito bem, ele pode acertar mais se "invertermos" o cálculo. Seguem algumas imagens e gráficos que comprovam os valores informados, como a avaliação do modelo abaixo:

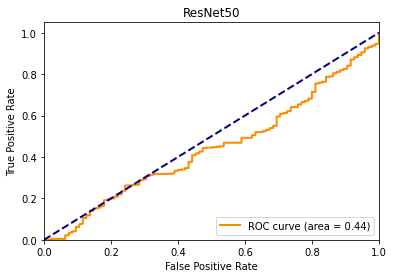
Fonte: elaborado pelo autor em 2022

Acurácia do modelo ao longo das iterações ou *epochs*, comparando base de treino e teste:



Fonte: elaborado pelo autor em 2022

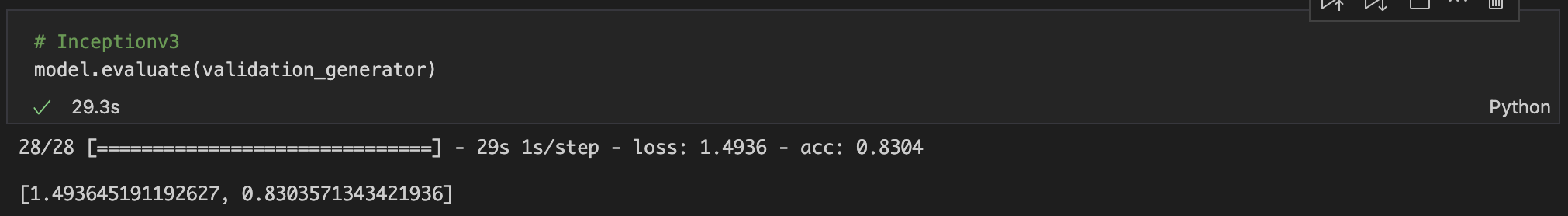
Curva ROC calculada com área de curva:



Fonte: elaborado pelo autor em 2022

### 6.2 Inception

O modelo InceptionV3, apresentou valor de perda de 149,36%, bastante ruim, após última iteração e acurácia de 83,04%. Ao longo das iterações de treinamento o modelo teve um pico inferior a 50% de acurácia, considerando a base de teste e pico superior em torno 85%, também considerando a base de testes. A curva ROC se mostrou bastante ruim, classificando erroneamente muitos registros, talvez com mais iterações de treinamento e refino fosse possível inverter essa curva. Seguem algumas imagens e gráficos que comprovam os valores informados:

Fonte: elaborado pelo autor em 2022 Fonte: elaborado pelo autor em 2022

Acurácia do modelo ao longo das iterações ou *epochs*, comparando base de treino e teste:



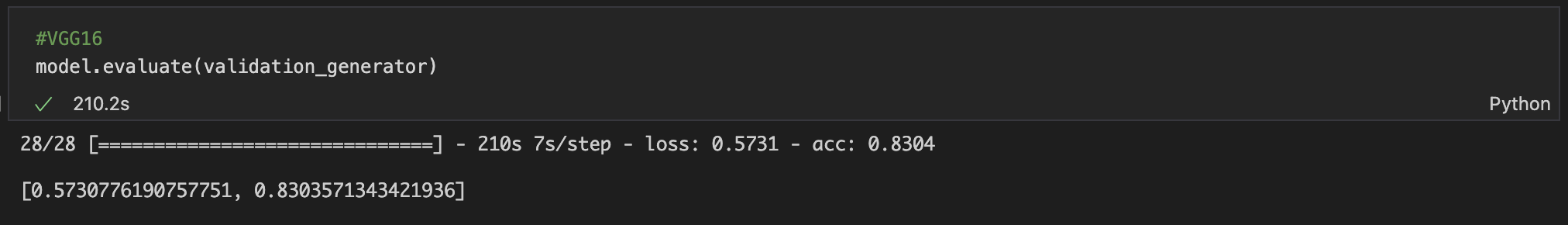
Fonte: Elaborado pelo autor em 2022.

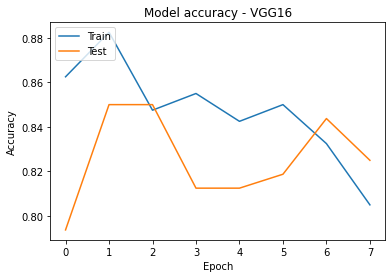
Curva ROC calculada com área de curva:

Fonte: elaborado pelo autor em 2022

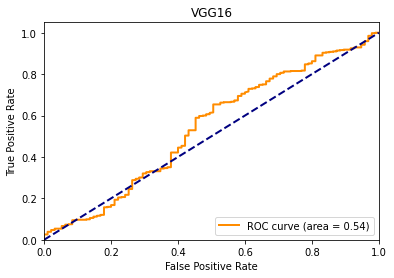
### 6.3 **VGG16**

O modelo VGG16, apresentou valor de perda de 57,31%, bastante bom, após última iteração e acurácia de 83,04%. Ao longo das iterações de treinamento o modelo teve um pico próximo a 80% de acurácia, considerando a base de teste e pico superior em torno 84%, também considerando a base de testes. A curva ROC se mostrou bastante boa, única que classificou corretamente vários registros.Seguem algumas imagens e gráficos que comprovam os valores informados:

Fonte: elaborado pelo autor em 2022

Acurácia do modelo ao longo das iterações ou *epochs*, comparando base de treino e teste:

Fonte: Elaborado pelo autor em 2022.

Curva ROC calculada com área de curva:

Fonte: Elaborado pelo autor em 2022.

### 6.4 Conclusão

Tabela de resultados final:

| Modelo | Ano de Publicação | Número de Parâmetros | Coeficiente de Perda | Acurácia | Área da Curva ROC | Curva ROC |
| --- | --- | --- | --- | --- | --- | --- |
| ResNet50 | 2015 | 25 milhões | 82,33% | 83,04% | 0,44 | Negativa |
| Inceptionv3 | 2015 | 24 milhões | 149,36% | 83,04% | 0,43 | Negativa |
| VGG16 | 2014 | 138 milhões | 57,31% | 83,04% | 0,54 | Positiva |

Fonte: elaborado pelo autor em 2022

Conforme resultados demonstrados acima, o modelo VGG16, foi o modelo mais preciso e com menor perda. Além disso, a Curva ROC se mostrou melhor, aliás, fato importante pois verdadeiros positivos são definitivamente mais importantes que falsos positivos, uma vez que um verdadeiro positivo, pode realmente salvar a vida de milhares, buscando o correto acompanhamento médico. Os modelos tiveram acurácia muito parecida pois todos são, na verdade, variações de CNNs pré-treinados. Entretanto é na perda e curva ROC que podemos perceber como as especificidades de cada modelo se destacam, de maneira positiva ou negativa. No mais, temos 3 modelos usando apenas conjuntos de imagens, informações como idade dos pacientes e localização da lesão (ambas disponibilizadas pela HAM10000), poderiam facilmente elevar a acurácia modelo.

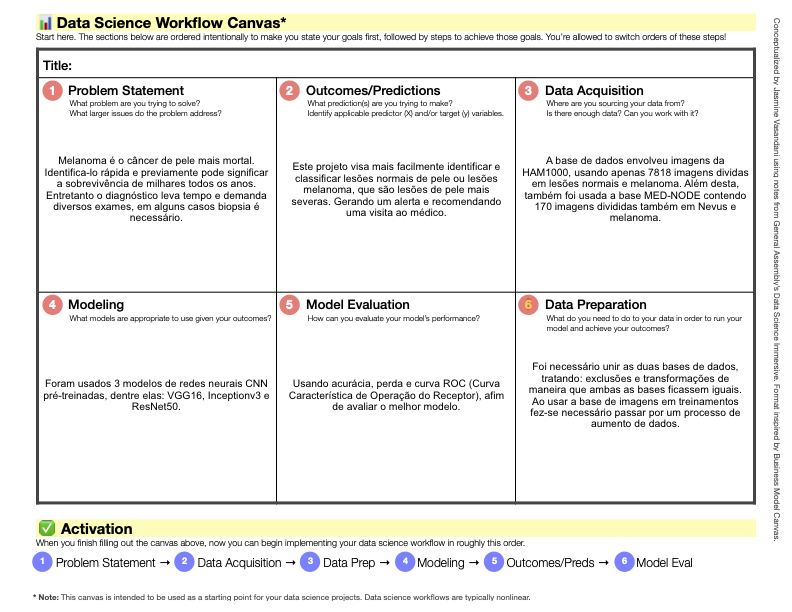
# 7. Apresentação dos Resultados

Este projeto visava o estudo, implantação e análise de um modelo que pudesse auxiliar na identificação de lesões de pele do tipo melanoma, o câncer de pele mais mortal. Separando lesões de pele comuns, como pintas, de lesões de pele severas que requerem cuidados médicos. Para isso foi necessário buscar bases de imagens confiáveis que apresentassem um bom número de registros não só de lesões comuns como também cancerígenas, mais precisamente melanoma.

Com duas bases distintas foi necessário aplicar tratamentos e deleções a fim de tornar ambas as bases homogêneas. Com as bases devidamente unidas, foi necessário separar em base de treino (80%) e validação (20%), para efeito de estudo e testes foram usados apenas 35% dos registros, de um total de 7988 imagens, divididas entre 6805 nevus*,* ou pintas comuns, e 1183 imagens de melanoma; fez-se uso de 2395 imagens de nevus e 401 de melanoma. Toda essa redução se fez necessária por capacidade computacional local, num eventual cenário com mais capacidade computacional, CPU e GPU mais potentes alinhados a mais memória RAM, seria possível considerar a base inteira de imagens, o que por sua vez poderia elevar a acurácia dos modelos.

Dentro deste projeto foram usados 3 modelos do tipo CNN, ou Redes Neurais Convolucionais, pré-treinados e fornecidos pela Tensor Flow, mais especificamente os seguintes modelos: VGG16, InceptionV3 e ResNet50. Destes, o modelo VGG16 obteve a maior acurácia: 83,04%, menor perda de 57,31% e melhor curva de ROC, com área de curva ROC equivalente a 0,54, classificando corretamente casos de verdadeiro positivo. Destaco também que todos os modelos foram treinados e testados com as mesmas 2796 imagens, passando pelos mesmos passos de aumento de dados.

Abaixo, segue uma abstração do *Data Science Workflow Canvas*:



Fonte: elaborado pelo autor em 2022

A acurácia dos modelos pode ter sido fortemente influenciada pela base desbalanceada, num trabalho futuro vejo a necessidade de balancear a base de imagens antes de efetivamente treinar o modelo, além disso, gostaria também de testar outros modelos além das populares CNNs, um dos que já destaco como alternativas possíveis é o random forest, que infelizmente não pode ser implementado aqui por falta de tempo hábil.

# 8. Links

## 8.1. Link Vídeo

youtube.com/...

## 8.2. Link Repositório

<https://github.com/cesaraugusto98/TCC_Stricto_Sensu>

## 8.3. Link Dataset Final

<https://drive.google.com/file/d/1T0ztIuk6ofSJejrSFEK_f4g2y96Xu25R/view?usp=sharing>

# 9.Referências

Instituto Nacional do Câncer. **Introdução.** Disponível em <https://www.inca.gov.br/estimativa/introducao>. Acesso em Abril/2022.

American Cancer Society. **Key Statistics for Melanoma Skin Cancer.** Disponível em <https://www.cancer.org/cancer/melanoma-skin-cancer/about/key-statistics.html>. Acesso em Abril/2022.

Instituto Nacional do Câncer. **Câncer de Pele Melanoma.** Disponível em <https://www.inca.gov.br/tipos-de-cancer/cancer-de-pele-melanoma>. Acesso em Abril/2022.

TSCHANDL, Phillip. **The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions.** Disponível em <https://doi.org/10.7910/DVN/DBW86T>. Acesso em Dez/2021.

I. Giotis, N. Molders, S. Land, M. Biehl, M.F. Jonkman e N. Petkov. **MED-NODE: A computer-assisted melanoma diagnosis system using non-dermoscopic images.**  Expert Systems with Applications, 2015. Base de imagens disponível em [complete\_mednode\_dataset.zip](http://www.cs.rug.nl/~imaging/databases/melanoma_naevi/complete_mednode_dataset.zip). Acesso em Dez/2021.

[KADAMPUR, Mohammad e RIYAEE, Sulaiman](https://www.sciencedirect.com/science/article/pii/S2352914819302047#!). **Skin cancer detection: Applying a deep learning based model driven architecture in the cloud for classifying dermal cell images.**Disponível em <https://www.sciencedirect.com/science/article/pii/S2352914819302047>. Acesso em Jan/2022.

FERREIRA, Pedro. **PH2 Dataset.** Disponível em <https://www.fc.up.pt/addi/ph2%20database.html>. Acesso em Jan/2022.

MAIA, Lucas Bezerra. **Aprendizagem Profunda Aplicada a Identificação de melanoma**. São Luís, Universidade Federal do Maranhão, 2019. Disponível em <https://tedebc.ufma.br/jspui/bitstream/tede/2578/2/LucasMaia.pdf>. Acesso em Fev/2022.

PATEL, Tirth. **Skin Cancer Detection**. Disponível em <https://github.com/Tirth27/Skin-Cancer-Classification-using-Deep-Learning>.Acesso em Fev/2022.

GAUTAM, Tanishq. **Good Keras Guide**. Disponível em https://www.analyticsvidhya.com/blog/2020/10/create-image-classification-model-python-keras/. Acesso em Fev/2022.

GLANDER, Shirin. **TRAIN YOUR OWN IMAGE CLASSIFICATION MODEL WITH KERAS AND TENSORFLOW.** Disponível em <https://mlconference.ai/blog/machine-learning-with-python/>. Acesso em Fev/20222.

Purva91. **Top 4 Pre-Trained Models for Image Classification with Python Code.** Disponível em <https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/>. Acesso em Mar/2022.

VARSHENI, Shri. **Binary Image Classifier using PyTorch**. Disponível em <https://www.analyticsvidhya.com/blog/2021/06/binary-image-classifier-using-pytorch/>. Acesso em Mar/2022.

PHAN, Binh. **CNN Binary Image Classifier in TensorFlow.** Disponível em <https://towardsdatascience.com/10-minutes-to-building-a-cnn-binary-image-classifier-in-tensorflow-4e216b2034aa>. Acesso em Abril/2022.

[SHRIVASTAV, Aman](https://iq.opengenus.org/author/aman-shrivastav/). **Different Types of CNN Models.** Disponível em <https://iq.opengenus.org/different-types-of-cnn-models/>. Acesso em Abril/2022.

VASANDANI, Jasmine. **A Data Science Workflow Canvas to Kickstart Your Projects**. Disponível em <https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0>. Acesso em Abril/2022.

# Apêndice

**Programação/Scripts**

# To add a new cell, type '# %%'

# To add a new markdown cell, type '# %% [markdown]'

# %%

from IPython import get\_ipython

# %% [markdown]

# # TCC Strictu Sensu - Algoritmo de Classificação Binária: Melanoma versus Nevus.

# %% [markdown]

# ## 1. Definição do Problema

#

# \*Por que?\*

# Câncer de pele é um dos cânceres mais comuns na atualidade, se identificado no início, pode ser mais facilmente tratado. A conscientização é muito importante, devemos informar a população de modo geral e facilitar o teste, como por exemplo um simples carregar de uma foto em um aplicativo que já te retornará as chances de ser uma pinta ou mancha na pele se desenvolver em um câncer de pele.

#

# \*Quem?\*

# Este projeto visa ajudar todos que tiverem acesso a internet, ou smartphones, provendo uma maneira fácil de avaliar se pintas ou manchas na pele podem ser um câncer de pele em estágio inicial

#

# \*O que?\*

# Um modelo de machine learning(mais precisamente deep learninng) que permitirá que usuários externos testem pintas ou manchas, buscando por possíveis câncer de pele.

#

# \*Quando?\*

# O modelo deverá responder de maneira instantânea, ou mais próxima ao tempo real, como por exemplo entre 1 a 5 minutos.

#

# \*Onde?\*

# A princípio através de input manual nesse projeto, mas no futuro usuários poderão carregar e testar imagens por um site ou aplicativo de Smartphone.

#

# %% [markdown]

# ## 2. Coleta dos Dados

#

# Segundo os requisitos deste projeto, duas bases de dados são requeridas. O tema deste projeto tem sido muito trabalhado por vários projetos ao longo dos últimos anos, porém existem apenas algumas bases muito populares, praticamente todas providas pela mesma fonte a ISIC que anualmente (desde 2016) tem lançado um desafio de Machine Learning e providenciado datasets com milhares de imagens com este tema. Entretanto como exigência do projeto tive de buscar uma base de dados distinta e após pesquisa encontrei um dataset pequeno mas que me fornece imagens de Nevus normais e melanoma. A fim de unir as bases precisei reduzir o escopo do dataset provido pela HAM10000 (Imagens colhidas pela ISIC em 2018), em apenas Nevus normais ou melanomas.

# %%

import pandas as pd

import shutil

import os

# %%

FINAL\_PATH = "dataset\_final//imagens//"

# %%

path\_HAM10000 = "dataset\_1//HAM10000\_images//"

df\_HAM10000\_csv = "dataset\_1//HAM10000\_metadata.csv"

df\_HAM10000\_csv = pd.read\_csv(df\_HAM10000\_csv)

#Tamanho original

print('Tamanho Original: ', df\_HAM10000\_csv.shape)

#Filtrando apenas melanocytic nevi (begnigno) e melanoma(maligno),

#o resto da base será desconsiderado para este estudo.

df\_HAM10000\_csv\_filtered=df\_HAM10000\_csv[(df\_HAM10000\_csv.dx == "nv") | (df\_HAM10000\_csv.dx == "mel")]

#Tamanho após filtrado

print('Tamanho após aplifcar o filtro: ', df\_HAM10000\_csv\_filtered.shape)

#Movendo os arquivos para Pasta Final

for index, row in df\_HAM10000\_csv\_filtered.iterrows():

shutil.copy2(path\_HAM10000 + row['image\_id'] + '.jpg', FINAL\_PATH)

# %%

df\_mednode = pd.DataFrame(columns=['image\_id','type'])

path\_mednode = "dataset\_2//complete\_mednode\_dataset//"

for diretorio,subpasta, arquivos in os.walk(path\_mednode):

if diretorio !="dataset\_2//complete\_mednode\_dataset//":

for arquivo in arquivos:

if 'melanoma' in diretorio:

df\_mednode=df\_mednode.append({'image\_id': arquivo.replace('.jpg',''), 'type':'mel'}, ignore\_index=True)

shutil.copy2(diretorio +'//'+ arquivo, FINAL\_PATH)

elif 'naevus' in diretorio:

df\_mednode=df\_mednode.append({'image\_id': arquivo.replace('.jpg',''), 'type':'nv'}, ignore\_index=True)

shutil.copy2(diretorio +'//'+ arquivo, FINAL\_PATH)

print(df\_mednode.shape)

df\_mednode.head()

# %%

#Unindo os dataframes de Typagem de cada imagem:

print('Dataframe HAM10000: ', df\_HAM10000\_csv\_filtered.shape)

print('Dataframe HAM10000 colunas: ', df\_HAM10000\_csv\_filtered.columns)

print('Dataframe Med Node: ', df\_mednode.shape)

print('Dataframe Med Node colunas: ', df\_mednode.columns)

df\_HAM10000\_final=df\_HAM10000\_csv\_filtered.drop(columns=['lesion\_id', 'dx\_type', 'age', 'sex', 'localization', 'dataset'])

df\_HAM10000\_final=df\_HAM10000\_final.rename(columns = {'image\_id': 'image\_id', 'dx': 'type'}, inplace = False)

df\_final = pd.DataFrame(columns=['image\_id','type'])

df\_final = df\_final.append(df\_HAM10000\_final, ignore\_index=True)

df\_final = df\_final.append(df\_mednode, ignore\_index=True)

print('Dataframe final: ' , df\_final.shape)

print(df\_final.head())

print(df\_final.tail())

df\_final.to\_csv("dataset\_final//" + "classificacao\_dataset.csv")

# %% [markdown]

# ## 3. Processamento/Tratamento dos Dados

#

# %%

import cv2

import matplotlib.pyplot as plt

import numpy as np

import os

# %% [markdown]

# !cd dataset\_final/imagens/ && ls

# %%

imagens = []

for arquivo in os.listdir(FINAL\_PATH):

img = cv2.imread(os.path.join(FINAL\_PATH,arquivo), 1)

if img is not None:

imagens.append(img)

# %%

print('Quantidade de Imagens: ', len(imagens))

# %%

altura=[]

largura=[]

canal=[]

for i in range(7983,7988):

print("Imagem:",i+1)

x,y,z=imagens[i].shape

largura.append(x)

altura.append(y)

canal.append(z)

print('largura: ' + str(x) + ', altura: ' + str(y) + ', canais: '+ str(z))

plt.imshow(imagens[i])

plt.show()

# %% [markdown]

# ## 4. Análise/Exploração dos Dados

#

#

# %%

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import os

from glob import glob

import seaborn as sns

from PIL import Image

# %%

get\_ipython().system('pwd')

# %%

pele\_df = pd.read\_csv('dataset\_final/classificacao\_dataset.csv')

print(pele\_df.head())

np.random.seed(42)

SIZE=64

# %%

pele\_df\_img = pele\_df.copy()

# %%

image\_path = {os.path.splitext(os.path.basename(x))[0]: x

for x in glob(os.path.join('dataset\_final', '\*', '\*.jpg'))}

#Define the path and add as a new column

pele\_df\_img['path'] = pele\_df['image\_id'].map(image\_path.get)

#Use the path to read images.

pele\_df\_img['imagem'] = pele\_df\_img['path'].map(lambda x: np.asarray(Image.open(x).resize((SIZE,SIZE))))

pele\_df\_img.head()

# %%

num\_exemplos = 3

# Plot

fig, m\_axs = plt.subplots(2, num\_exemplos, figsize = (4\*num\_exemplos, 2\*3))

for n\_axs, (type\_name, type\_rows) in zip(m\_axs,

pele\_df\_img.sort\_values(['type']).groupby('type')):

n\_axs[0].set\_title(type\_name)

for c\_ax, (\_, c\_row) in zip(n\_axs, type\_rows.sample(num\_exemplos, random\_state=1234).iterrows()):

c\_ax.imshow(c\_row['imagem'])

c\_ax.axis('off')

# %% [markdown]

# ## 5. Criação do Modelo de ML

# %% [markdown]

# ### Testando Diversos Modelos

# %%

import os

import pandas as pd

import numpy as np

import tensorflow as tf

import shutil

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import layers

from tensorflow.keras import Model

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from glob import glob

from PIL import Image

import ssl

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

# %%

#Setting Global Variables

FINAL\_PATH = "dataset\_final//imagens//"

TRAIN\_PATH = "dataset\_final//train//"

TEST\_PATH = "dataset\_final//test//"

SIZE=224

# %%

pele\_df = pd.read\_csv('dataset\_final/classificacao\_dataset.csv')

le = LabelEncoder()

le.fit(pele\_df['type'])

LabelEncoder()

print('label 0 é igual á ' + list(le.classes\_)[0])

print('label 1 é igual á ' + list(le.classes\_)[1])

pele\_df['label'] = le.transform(pele\_df["type"])

print(pele\_df['label'].value\_counts())

print(pele\_df.head())

# %%

pele\_df\_final = pele\_df.copy()

image\_path = {os.path.splitext(os.path.basename(x))[0]: x

for x in glob(os.path.join('dataset\_final', '\*', '\*.jpg'))}

#Definindo o diretório como uma nova coluna

pele\_df\_final['path'] = pele\_df['image\_id'].map(image\_path.get)

#Usar o dataframe completo estava levando muito tempo, portanto tive de quebra-lo num fração menor, ainda com mais de 2000 linhas

pele\_df\_final\_frac = pele\_df\_final.sample(frac=0.35)

print(pele\_df\_final\_frac['type'].value\_counts())

X = pele\_df\_final\_frac['path']

Y = pele\_df\_final\_frac['type'] #Assign label values to Y

# %%

print(f'Valores únicos para Y: {list(set(Y))}')

x\_train\_auto, x\_test\_auto, y\_train\_auto, y\_test\_auto = train\_test\_split(X, Y, test\_size=0.20, random\_state=42)

print(f'x\_train\_auto: {x\_train\_auto[0:5]}')

print(f'x\_test\_auto: {x\_test\_auto[0:5]}')

print(f'y\_train\_auto: {y\_train\_auto[0:5]}')

print(f'y\_test\_auto: {y\_test\_auto[0:5]}')

# %%

df\_train = pd.DataFrame(columns=['image\_path','type'])

df\_train['image\_path'] = x\_train\_auto

df\_train['type'] = y\_train\_auto

df\_test = pd.DataFrame(columns=['image\_path','type'])

df\_test['image\_path'] = x\_test\_auto

df\_test['type'] = y\_test\_auto

# %%

#Creating a directory for Train

for n, row in df\_train.iterrows():

if row['type']=='mel':

shutil.copy2(row['image\_path'], TRAIN\_PATH + 'mel//')

else:

shutil.copy2(row['image\_path'], TRAIN\_PATH+ 'nv//')

# %%

#Creating a directory for Test

for n, row in df\_test.iterrows():

if row['type']=='mel':

shutil.copy2(row['image\_path'], TEST\_PATH + 'mel//')

else:

shutil.copy2(row['image\_path'], TEST\_PATH+ 'nv//')

# %%

files = ["{}/{}".format(FINAL\_PATH, fi) for fi in os.listdir(FINAL\_PATH) if fi.endswith("jpg")]

# %%

def get\_image\_data(filename):

img = Image.open(filename)

img.load()

data = np.asarray(img, dtype="int32")

return data

def plot\_N(indices, savename=None):

f, axarr = plt.subplots(len(indices) // 5, 5)

f.set\_size\_inches(14, 14)

f.subplots\_adjust(wspace=0.2, hspace=0, left=0, right=1, top=0.4, bottom=0)

for i in range(len(indices)):

axarr[i // 5, i % 5].axis("off")

axarr[i // 5, i % 5].imshow(get\_image\_data(files[indices[i]]))

if savename is not None:

f.savefig('./imagens\_resultados/' + str(savename))

# %% [markdown]

# ### Testing VGG-16

# %%

SIZE = 224

# %%

# Image Augmentation

# Add our data-augmentation parameters to ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255.,rotation\_range = 40, width\_shift\_range = 0.2, height\_shift\_range = 0.2, shear\_range = 0.2, zoom\_range = 0.2, horizontal\_flip = True)

# Note that the validation data should not be augmented!

test\_datagen = ImageDataGenerator( rescale = 1.0/255. )

# %%

# Training and Validation Sets

# Flow training images in batches of 20 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(TRAIN\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

# Flow validation images in batches of 20 using test\_datagen generator

validation\_generator = test\_datagen.flow\_from\_directory(TEST\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

# %%

# Loading the Base Model

from keras.applications.vgg16 import VGG16

base\_model = VGG16(input\_shape = (SIZE, SIZE, 3), # Shape of our images

include\_top = True, # Leave out the last fully connected layer

weights = 'imagenet')

for layer in base\_model.layers:

layer.trainable = False

# %%

# Flatten the output layer to 1 dimension

x = layers.Flatten()(base\_model.output)

# Add a fully connected layer with 512 hidden units and ReLU activation

x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.5

x = layers.Dropout(0.5)(x)

# Add a final sigmoid layer with 1 node for classification output

x = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(base\_model.input, x)

model.compile(optimizer = tf.keras.optimizers.RMSprop(learning\_rate=0.0001), loss = 'binary\_crossentropy',metrics = ['acc'])

# %%

vgghist = model.fit(train\_generator, validation\_data = validation\_generator, validation\_steps= 8, batch\_size=20, steps\_per\_epoch = 20, epochs = 8)

# %% [markdown]

# ## Testing Inception

# %%

SIZE = 150

# %%

# Data Augmentation

# Add our data-augmentation parameters to ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255., rotation\_range = 40, width\_shift\_range = 0.2,

height\_shift\_range = 0.2,shear\_range = 0.2, zoom\_range = 0.2, horizontal\_flip = True)

test\_datagen = ImageDataGenerator( rescale = 1.0/255. )

# %%

# Training and Validation Generators

# Flow training images in batches of 20 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(TRAIN\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

# Flow validation images in batches of 20 using test\_datagen generator

validation\_generator = test\_datagen.flow\_from\_directory(TEST\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

# %%

# Loading the Base Model

from tensorflow.keras.applications.inception\_v3 import InceptionV3

base\_model = InceptionV3(input\_shape = (SIZE, SIZE, 3), include\_top = False, weights = 'imagenet')

# %%

# Compile and Fit

for layer in base\_model.layers:

layer.trainable = False

from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(base\_model.output)

x = layers.Dense(18432, activation='relu')(x)

x = layers.Dropout(0.2)(x)

# Add a final sigmoid layer with 1 node for classification output

x = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(base\_model.input, x)

model.compile(optimizer = RMSprop(learning\_rate=0.0001), loss = 'binary\_crossentropy', metrics = ['acc'])

# %%

inc\_history = model.fit(train\_generator, validation\_data = validation\_generator, validation\_steps= 8, batch\_size=20, steps\_per\_epoch = 20, epochs = 8)

# %% [markdown]

# ## Testing ResNet50

# %%

SIZE=224

# %%

# Data Augmentation and Generators

# Add our data-augmentation parameters to ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255., rotation\_range = 40, width\_shift\_range = 0.2,

height\_shift\_range = 0.2, shear\_range = 0.2, zoom\_range = 0.2, horizontal\_flip = True)

test\_datagen = ImageDataGenerator(rescale = 1.0/255.)

# %%

train\_generator = train\_datagen.flow\_from\_directory(TRAIN\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

validation\_generator = test\_datagen.flow\_from\_directory(TEST\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

# %%

#Import the base model

from tensorflow.keras.applications import ResNet50

base\_model = ResNet50(input\_shape=(SIZE, SIZE,3), include\_top=False, weights="imagenet")

for layer in base\_model.layers:

layer.trainable = False

# %%

# Build and Compile the Model

from tensorflow.keras.applications import ResNet50

from tensorflow.python.keras.models import Sequential

from tensorflow.python.keras.layers import Dense, Flatten

base\_model = Sequential()

base\_model.add(ResNet50(include\_top=False, weights='imagenet', pooling='max'))

base\_model.add(Dense(1, activation='sigmoid'))

#opt = keras.optimizer\_v2.gradient\_descent.SGD(learning\_rate=0.0001)

base\_model.compile(optimizer = 'SGD', loss = 'binary\_crossentropy', metrics = ['acc'])

# %%

resnet\_history = base\_model.fit(train\_generator, batch\_size=20, validation\_data = validation\_generator, validation\_steps= 8,

steps\_per\_epoch = 20, epochs = 8)

# %% [markdown]

# ### CNN com Tensorflow

# %%

from sklearn.metrics import roc\_curve, auc

from sklearn.model\_selection import train\_test\_split

# %%

SIZE = 224

# %%

# Data Augmentation and Generators

# Add our data-augmentation parameters to ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255., rotation\_range = 40, width\_shift\_range = 0.2, height\_shift\_range = 0.2, shear\_range = 0.2, zoom\_range = 0.2, horizontal\_flip = True)

test\_datagen = ImageDataGenerator(rescale = 1.0/255.)

# %%

train\_generator = train\_datagen.flow\_from\_directory(TRAIN\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

validation\_generator = test\_datagen.flow\_from\_directory(TEST\_PATH, batch\_size = 20, class\_mode = 'binary', target\_size = (SIZE, SIZE))

# %%

model = tf.keras.models.Sequential([

# Note the input shape is the desired size of the image 200x200 with 3 bytes color

# This is the first convolution

tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(SIZE, SIZE, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

# The second convolution

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The third convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The fourth convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# # The fifth convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# Flatten the results to feed into a DNN

tf.keras.layers.Flatten(),

# 512 neuron hidden layer

tf.keras.layers.Dense(512, activation='relu'),

# Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('dandelions') and 1 for the other ('grass')

tf.keras.layers.Dense(1, activation='sigmoid')

])

# %%

from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary\_crossentropy',

optimizer=RMSprop(learning\_rate=0.001),

metrics='accuracy')

# %%

cnn\_history = model.fit(train\_generator, steps\_per\_epoch=20, epochs=8, verbose=1, validation\_data = validation\_generator, validation\_steps=8)

# %% [markdown]

# ## 6. Interpretação dos Resultados

# %% [markdown]

# Para análise do modelo, utilizei dos seguintes conceitos:

#

# \* Coeficiente de Perda

# \* Acurácia

# \* Curva ROC

# %% [markdown]

# ### VGG16

# %%

#VGG16

model.evaluate(validation\_generator)

# %%

plt.plot(vgghist.history['acc'])

plt.plot(vgghist.history['val\_acc'])

plt.title('Model accuracy - VGG16')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# %%

STEP\_SIZE\_TEST=validation\_generator.n//validation\_generator.batch\_size

validation\_generator.reset()

preds = model.predict(validation\_generator,

verbose=1)

# %%

fpr, tpr, \_ = roc\_curve(validation\_generator.classes, preds)

roc\_auc = auc(fpr, tpr)

plt.figure()

lw = 2

plt.plot(fpr, tpr, color='darkorange',

lw=lw, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('VGG16')

plt.legend(loc="lower right")

plt.show()

# %% [markdown]

# ### InceptionV3

# %%

# Inceptionv3

model.evaluate(validation\_generator)

# %%

plt.plot(inc\_history.history['acc'])

plt.plot(inc\_history.history['val\_acc'])

plt.title('Model accuracy - Inception')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# %%

STEP\_SIZE\_TEST=validation\_generator.n//validation\_generator.batch\_size

validation\_generator.reset()

preds = model.predict(validation\_generator,

verbose=1)

# %%

fpr, tpr, \_ = roc\_curve(validation\_generator.classes, preds)

roc\_auc = auc(fpr, tpr)

plt.figure()

lw = 2

plt.plot(fpr, tpr, color='darkorange',

lw=lw, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Inception')

plt.legend(loc="lower right")

plt.show()

# %% [markdown]

# ### ResNet50

# %%

# ResNet50

model.evaluate(validation\_generator)

# %%

plt.plot(resnet\_history.history['acc'])

plt.plot(resnet\_history.history['val\_acc'])

plt.title('Model accuracy - ResNet50')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# %%

STEP\_SIZE\_TEST=validation\_generator.n//validation\_generator.batch\_size

validation\_generator.reset()

preds = model.predict(validation\_generator,

verbose=1)

# %%

fpr, tpr, \_ = roc\_curve(validation\_generator.classes, preds)

roc\_auc = auc(fpr, tpr)

plt.figure()

lw = 2

plt.plot(fpr, tpr, color='darkorange',

lw=lw, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ResNet50')

plt.legend(loc="lower right")

plt.show()

# %% [markdown]

# ### CNN Manualmente Composto

# %%

model.evaluate(validation\_generator)

# %%

STEP\_SIZE\_TEST=validation\_generator.n//validation\_generator.batch\_size

validation\_generator.reset()

preds = model.predict(validation\_generator,

verbose=1)

# %%

fpr, tpr, \_ = roc\_curve(validation\_generator.classes, preds)

roc\_auc = auc(fpr, tpr)

plt.figure()

lw = 2

plt.plot(fpr, tpr, color='darkorange',

lw=lw, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('CNN Manualmente Composto')

plt.legend(loc="lower right")

plt.show()

# %% [markdown]

# ## 7. Comunicação dos Resultados

# %%

get\_ipython().system('pwd')

# %% [markdown]

# Por favor visite o Documento:

#

# https://github.com/cesaraugusto98/tcc\_pos/blob/main/imagens\_resultados/TCC-workflow-canvas.png

# %% [markdown]

# ## 8. Referencias

# Tschandl, Philipp, 2018, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions", https://doi.org/10.7910/DVN/DBW86T, Harvard Dataverse, V3, UNF:6:/APKSsDGVDhwPBWzsStU5A== [fileUNF]

#

# I. Giotis, N. Molders, S. Land, M. Biehl, M.F. Jonkman and N. Petkov: "MED-NODE: A computer-assisted melanoma diagnosis system using non-dermoscopic images", Expert Systems with Applications, 42 (2015), 6578-6585

#

# Skin cancer detection: Applying a deep learning based model driven architecture in the cloud for classifying dermal cell images | https://www.sciencedirect.com/science/article/pii/S2352914819302047

#

# PH2 dataset | https://www.fc.up.pt/addi/ph2%20database.html

#

#

# Aprendizagem Profunda Aplicada a Identificação de melanoma | https://tedebc.ufma.br/jspui/bitstream/tede/2578/2/LucasMaia.pdf

#

# Github Project - Skin Cancer detection |

# https://github.com/Tirth27/Skin-Cancer-Classification-using-Deep-Learning

#

# Good keras gudie | https://www.analyticsvidhya.com/blog/2020/10/create-image-classification-model-python-keras/

#

# MACHINE LEARNING WITH PYTHON: TRAIN YOUR OWN IMAGE CLASSIFICATION MODEL WITH KERAS AND TENSORFLOW | https://mlconference.ai/blog/machine-learning-with-python/

#

# Single Label Imagem Classification | https://blog.workaround.vercel.app/blog/single-label-image-classification-with-keras

#

# Top 4 Pre-Trained Models for Image Classification with Python Code | https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/

#

# Binary Image Classification | https://www.analyticsvidhya.com/blog/2021/06/binary-image-classifier-using-pytorch/

#

# CNN Binary Image Classifier in TensorFlow | https://towardsdatascience.com/10-minutes-to-building-a-cnn-binary-image-classifier-in-tensorflow-4e216b2034aa

#

# Different Types od CNN Models | https://iq.opengenus.org/different-types-of-cnn-models/

#

# A Data Science Workflow Canvas to Kickstart Your Projects | https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0