Aluno: Antoni Vinicius da Silva. Matrícula: 20190031597
Aluno: Filipe Trindade. Matrícula: 20190019912

Introdução

Para o trabalho/projeto da disciplina, optamos por desenvolver um pequeno sistema que coleta imagens na web (o *web scraping),* e a partir dessa coleta de imagens (que pode ocorrer via uma palavra chave de um provedor de busca, tipo o Google, ou um link para um site específico), pode fazer comparações entre uma imagem com uma ou com mais, de forma que elas possam também serem divididas em grupo ou adicionadas a um.

O motivo de optar por algo do tipo, foi que os integrantes do grupo não apenas possuíam experiência prévia com Python, mas também com *web scraping* e algumas das bibliotecas do Python para esta finalidade. E, dado um banco de dados de imagens, uma das possíveis coisas a se fazer com o mesmo seria a organização das imagens por similaridade.

Existem muitas formas de agrupar imagens, não só por similaridade (como tipo, tamanho, etc). Para isso, optamos por usar uma rede neural convolucional e funções que derivam de bibliotecas de Python e algoritmos de similaridade.

Materiais e Métodos

O trabalho foi desenvolvido com utilização da ferramenta de programação Python. As ferramentas usadas foram as diversas bibliotecas de *web scraping* e agrupamento de imagens por similaridade existentes.

Naturalmente, também se usaram diversas bibliotecas comuns a Python, como *numpy*, útil para cálculo envolvendo matrizes e vetores (no caso, relaciona-se o fato de que imagens podem ser interpretadas dessa forma). Para *web scraping*, o Python disponibiliza de várias bibliotecas, uma delas é a *Beautifulsoup*, que a partir de páginas e seu código HTML, pode “encontrar” arquivos do tipo imagem e seus respectivos links. Para acessar os links, se usa a biblioteca *requests*, que acessa a página web e seu conteúdo (também simulando um acesso de um usuário real). Além disso, foi utilizado algumas funções da biblioteca urllib, para separar e juntar partes de urls.

Para captar imagens via provedor de busca web, optamos por usar a biblioteca *simple image download*, que “simula” um acesso ao Google a partir de um certo termo de busca, e a partir disso, coleta os n primeiros resultados de imagens acessados com sucesso. Essa biblioteca também faz uso de outras, como a request. Uma outra possibilidade era a utilização da biblioteca Selenium.

A escolha desta biblioteca ao invés do uso do Selenium se deu pela semelhança entre as duas e a complexidade entre elas. De fato, a biblioteca Selenium possui funcionamento mais complexo, e até mais eficiente em certos casos, contudo, observamos que a biblioteca utilizada desempenha um papel muito similar e de forma mais simples e não precisa de nenhum pré-requisito para ser utilizada. A troca do *simple image download* pelo Selenium pode ser feita sem que o funcionamento do programa seja prejudicado.

Para manipulação de arquivos e diretórios foram utilizadas três bibliotecas, *os*, *glob,* e *shutil*, cada uma dessas bibliotecas possui funções equivalente, porém umas se sobressaem a outras. A biblioteca *os* foi utilizada para manipulação de caminhos, criação e remoção de arquivos, busca de arquivos entre outros, entre as três foi a biblioteca mais

utilizada. A biblioteca *glob* foi utilizada para encontrar imagens de forma recursiva, algo que não conseguimos fazer com a biblioteca *os*. Por fim, a biblioteca *shutil* foi utilizada apenas para copiar arquivo de uma pasta para a outra, essa operação não era realmente necessária, contudo, optamos por fazê-la para fins de demonstração do programa.

Em Python, existem diversas bibliotecas, funções e algoritmos que permitem a comparação de imagens por similaridade. Algumas buscam quantificar a diferença entre imagens (no caso, um resultado de 0 indicaria imagens idênticas), enquanto outras retornam um valor de similaridade entre as imagens de forma percentual. Uma biblioteca que fornece diversas funções desse tipo é a biblioteca *sewar*, que possui diversos algoritmos de comparação de imagem em si.

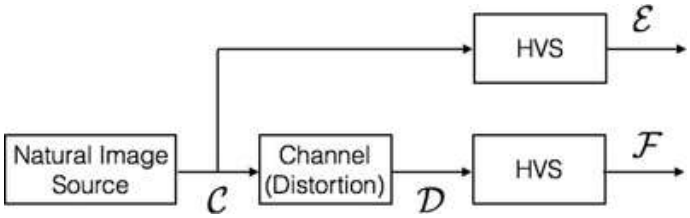
Dos vários algoritmos possíveis fornecidos, decidimos optar por 2 deles, o UQI (Universal Quality Index) e o VIF (Visual Image Fidelity). A razão de escolhermos essas, em vez de opções mais simples, como SSIM(Structural Similarity Index) e Mean Squared Error (MSE) é que ao invés de indicar um "erro" (no caso, diferença) absoluto entre as imagens, esses algoritmos tentam quantificar a semelhança em porcentagem. Ainda sim, a forma de funcionamento entre eles é bastante diferente. Nossa intenção ao usá-los não foi entender tais algoritmos e seu funcionamento perfeitamente, mas apenas exemplificar algumas das diversas formas de se comparar imagens visualmente.

O UQI, diferente de métodos de soma de erros, é um índice modelado a partir de três fatores: perda de correlação, distorção de luminância e distorção de contraste. Ainda que definido matematicamente, ele não utiliza explicitamente de nenhum modelo visual humano. A partir de sua definição original, uma das diversas formas de reescrevê-lo é:

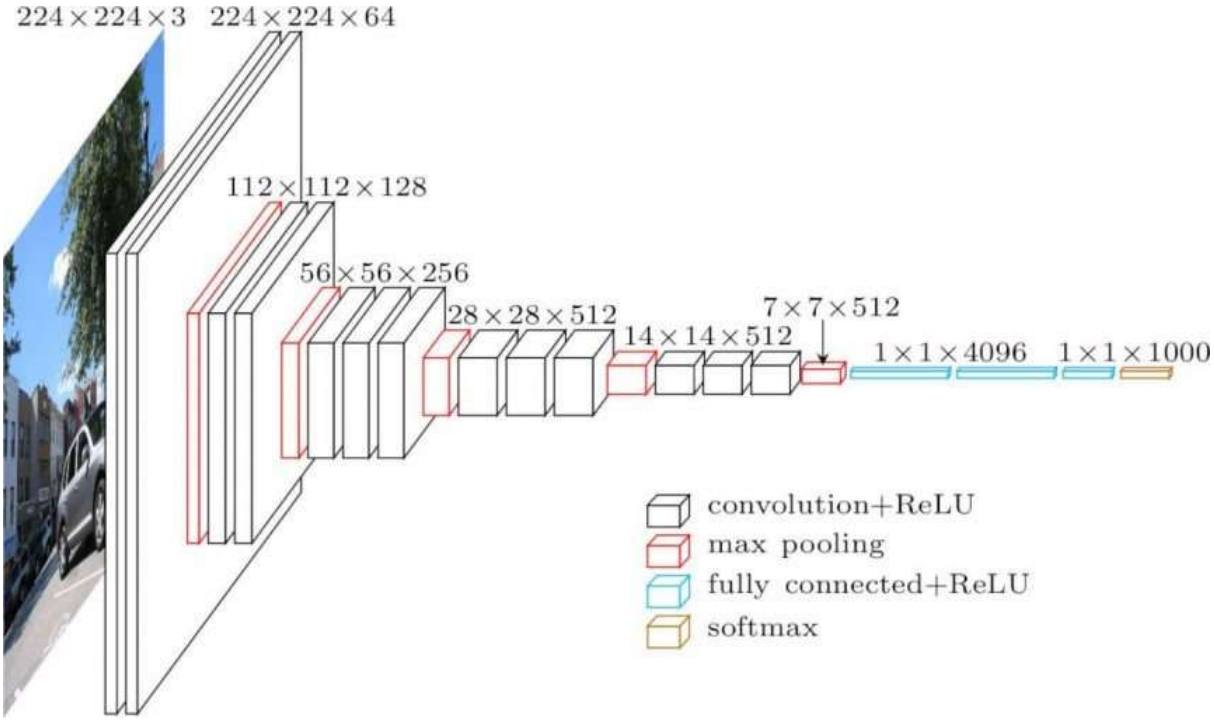
$$Q = \underbrace{\frac{\sigma_{xy}}{\sigma_x \sigma_y}}_{[-1,1]} \times \underbrace{\frac{2\bar{x}\bar{y}}{(\bar{x})^2 + (\bar{y})^2}}_{[0,1]} \times \underbrace{\frac{2\sigma_x \sigma_y}{\sigma_x^2 + \sigma_y^2}}_{[0,1]}$$

Loss of correlation Luminance distortion contrast distortion

Já o VIF (Visual Image Fidelity) busca ser mais ligado à noção e subjetividade humana quanto à percepção de imagens, e também a estatística de cenas naturais. O método leva em conta não apenas as informações que a imagem apresenta, mas a qualidade visual da mesma. Desta forma, o VIF se assemelha a diversos modelos que se baseiam no sistema visual humano. Uma forma simples de interpretar ele seria "o quanto da imagem original é possível interpretar-se na segunda imagem", visto que espera-se que haja distorções nas imagens, uma vez que sejam visualizadas e interpretadas por seres humanos.



As bibliotecas chave para o agrupamento de imagens descritas no artigo foram a Keras e a SKLearn, a biblioteca Keras disponibilizou o uso da rede neural VGG16, uma rede neural bastante conhecida da área e bastante poderosa, ela possui um limite de arquitetura de 224x224, contudo, essa limitação não é um ponto muito baixo desta rede.

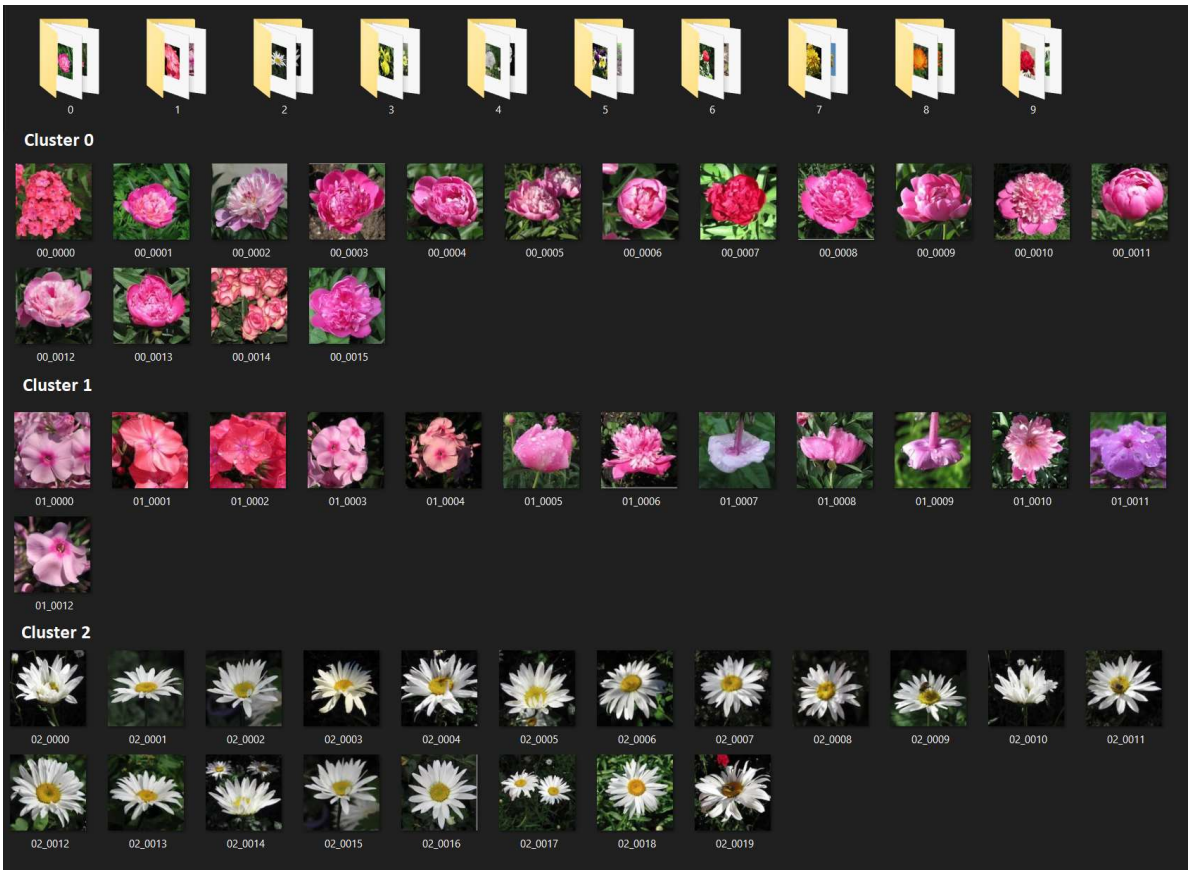


Já a biblioteca SKLearn foi responsável por agrupar as imagens, através da função Kmeans de acordo com características de cada imagem.

O restante das bibliotecas não foram muito utilizadas e possuem propostas diversas, a biblioteca Pickle foi utilizada na função de agrupamento para salvar características caso ocorra algum erro. A biblioteca cv2 foi utilizada para manipulação de imagens e as bibliotecas tqdm e time são utilizadas apenas para interface do programa.

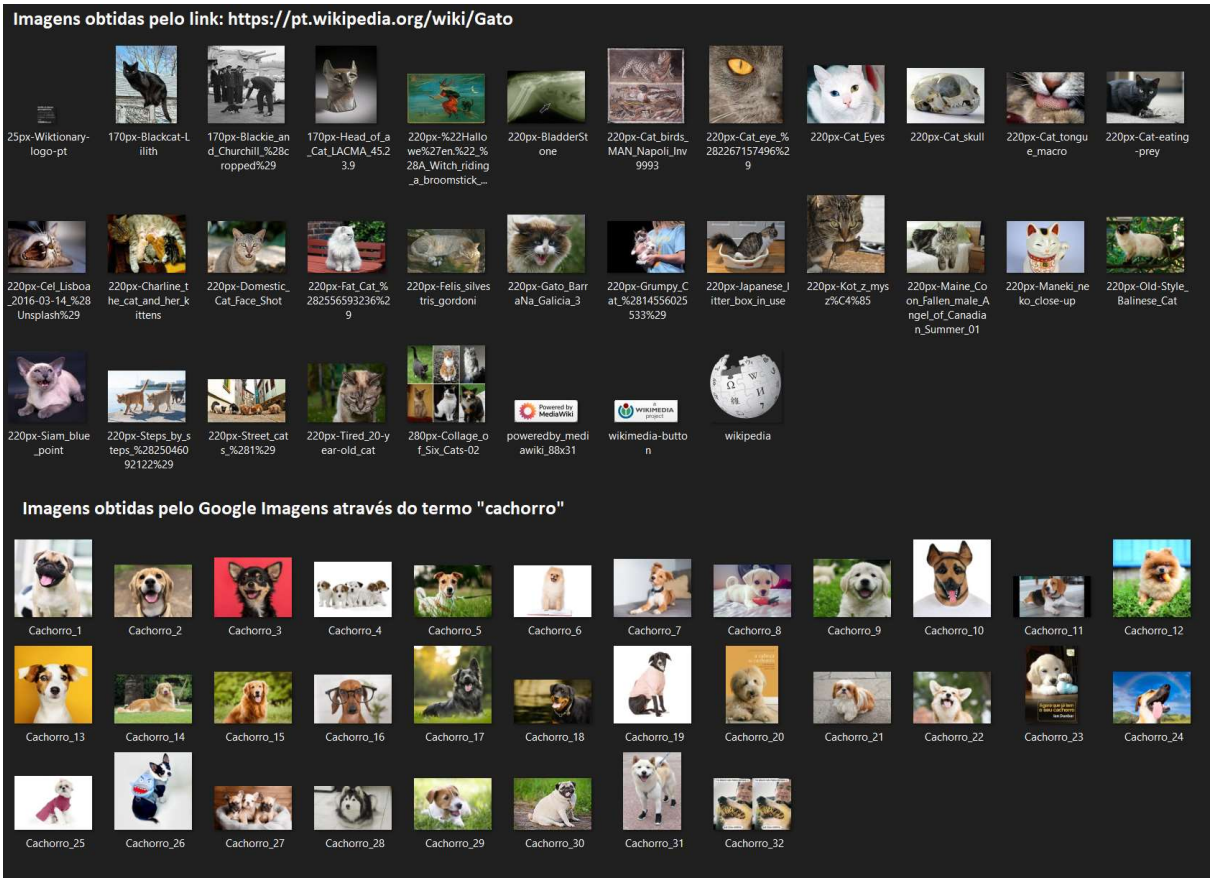
Resultados

O programa desempenhou seu papel como planejado. No artigo original, o autor utiliza um *dataset* de flores, com 10 espécies de flores distintas e 210 imagens, utilizamos esse mesmo *dataset* no programa para verificar o seu funcionamento e o resultado foi esse:



Além deste *dataset* utilizamos outros, um deles foi de um outro artigo que utiliza um *dataset* com 4738 imagens de animais, sendo esses animais divididos em onças, leões, lobos, tigres, raposas e guepardos, todos os grupos foram perfeitamente agrupados em seus respectivos *clusters*, tendo por volta de 400 até 1000 imagens por cluster.

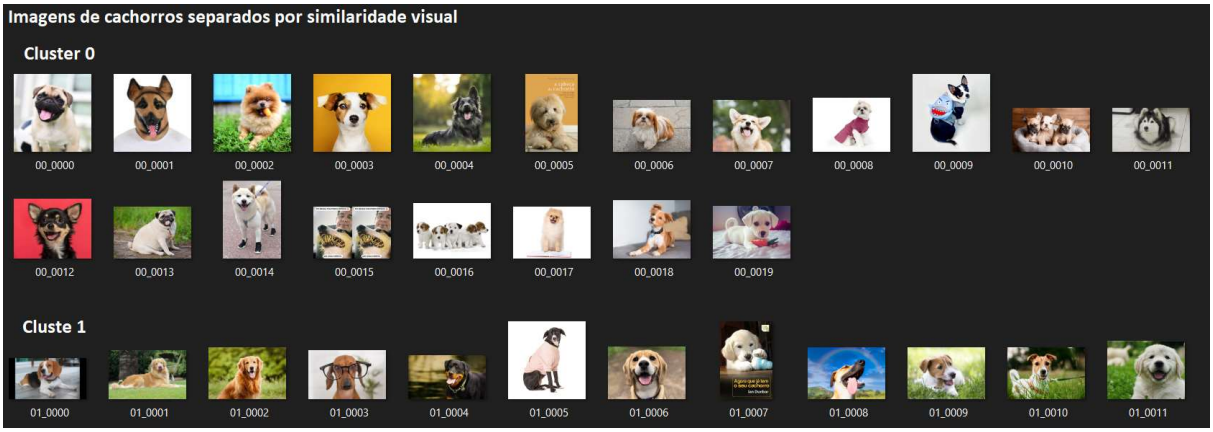
Indo para a parte do *web scraping* demos duas opções para realizá-los. A primeira é através do Google Imagens, onde é dado um termo a ser pesquisado e a quantidade de imagens para que seja feito o download. A outra opção é feita através de um link que o usuário fornece, esse link será analisado pelo algoritmo e ele buscará **todas** as imagens que o site possuir, sem distinção. As imagens salvas por ambos os métodos são salvas em um diretório fornecido pelo usuário.



De forma complementar o algoritmo possui três operações. A primeira delas permite a comparação entre imagens, que mostra o quão elas se parecem através de três algoritmos diferentes(um utilizando keypoints, o VIF e o UQI).

A segunda permite também a comparação entre uma imagem e um *dataset*, dando a possibilidade de adicionar essa imagem ao *dataset* que contenha a imagem mais semelhante a ela.

E por fim, permite também agrupar imagens resultantes do *web scraping* utilizando a rede neural VGG16.



O código do projeto e a biblioteca *simple image download* pode ser encontrada neste link: https://github.com/antonivini1/scrapping

Discussão

Primeiramente, o algoritmo de clustering foi bem complexo de se entender, o algoritmo utiliza a rede neural VGG16 para extrair características das imagens e as armazena em um vetor, essas características são agrupadas pelo KMeans e isso é a chave para formar os clusters, o algoritmo foi basicamente dado no artigo, e com a devida compreensão do mesmo alteramos poucas partes do algoritmo original, a principal mudança foi no método view_cluster, o alteremos para que, ao invés de mostrar as imagens apenas as salvassem em pastas que é pedida no começo do programa.

Para comparação de imagens, como já mencionado anteriormente, utilizamos duas funções da biblioteca sewar, porém, nosso algoritmo principal de comparação foi baseado no código ensinado e disponibilizado do canal Pysource, esse algoritmo mapeia a imagem criando keypoints em características das imagens e os compara com uma certa "distância" entre os pontos selecionando apenas bons pontos. Após selecionar bons pontos, ele divide pelo maior número de pontos obtendo a porcentagem de similaridade entre as imagens. Além disso, adaptamos um dos códigos que ele apresenta em seus vídeos para que nós conseguíssemos comparar uma imagem com um dataset inteiro, e se necessário, mover essa imagem para o dataset.

O código de *web scraping* foi baseado em um código já feito pelos membros do grupo, porém, como esse código era feito puramente em RegEx utilizamos o código do site *thepythoncode* para otimizar o código. Outra adaptação feita foi a biblioteca *simple image download*, pois, originalmente essa biblioteca cria uma pasta para armazenar as imagens,

dessa forma, fizemos uma alteração para que as imagens salvas fossem para uma pasta dada pelo usuário.

O desenvolvimento do projeto teve muitos problemas, mas conseguimos resolver a maioria deles, os problemas que ainda persistem são aqueles que não são consistentes, como por exemplo, ao utilizar a biblioteca *simple image download* existe a possibilidade de baixar imagens "não carregadas". Outro erro inconsistente é que ao utilizar o algoritmo de *web scraping* a imagem pode ser corrompida no processo, dando *crash* no programa.

Finalizando, é necessário dizer que neste projeto limitamos as imagens a serem baixadas apenas se forem das extensões .jpeg, .jpg e .png, para que pudéssemos ter mais controle dos resultados, além disso, no painel de execução, a medida que as funções são utilizadas aparecem *warnings* que deixam a visualização do programa poluída.

Acreditamos que o resultado final do programa/sistema desenvolvido foi bom, pois realiza seu propósito de forma razoável, enquanto se relaciona também a muitos tópicos relacionados aos vistos na disciplina.

Conclusão

A utilização de programas como esses tem diversas finalidades; as operações incluídas permitem que seja possível treinar redes neurais, criar ou gerenciar banco de dados, entre outras aplicações. Uma aplicação mencionada pelo professor que se enquadra nesse projeto é a capacidade de fazer com que uma IA reconheça rosto, placas entre outras coisas, treinando uma rede neural e comparando a imagem da IA com uma da *database*. Existem muitas formas de comparar imagens, ainda que não tenhamos englobado todas elas, e o trabalho esteja longe da perfeição acreditamos ter feito algo que segue bem o caminho para essas finalidades.

Referências

Artigo: Measuring similarity in two images using Python:
https://towardsdatascience.com/measuring-similarity-in-two-images-using-python-b72233eb53c6?gi=9772dc9ed381
Dataset dos animais:
https://medium.com/analytics-vidhya/image-similarity-model-6b89a22e2f1a
Canal Pysource:
https://www.youtube.com/playlist?list=PL6Yc5OUgcoTlQuAdhtnByty15Ea9-cQly
Página thepythoncode:
https://www.thepythoncode.com/code/download-web-page-images-python
simple_image_download:
https://github.com/RiddlerQ/simple_image_download/tree/master/simple_image_download
Apresentação da biblioteca sewar:
https://towardsdatascience.com/measuring-similarity-in-two-images-using-python-b72233eb53c6
Explicação sobre o VGG16:
https://neurohive.io/en/popular-networks/vgg16/
Explicação sobre o KMeans:
https://medium.com/programadores-ajudando-programadores/k-means-o-que-%C3%A9-como-funciona-aplica%C3%A7%C3%B5es-e-exemplo-em-python-6021df6e2572
UQI:
https://ieeexplore.ieee.org/document/995823
Ilustração referente ao UQI usada nos slides:
https://www.researchgate.net/figure/Illustration-of-the-universal-image-quality-index-UIQI-measurement-system_fig2_281293239
Ilustração referente ao UQI:
https://slidetodoc.com/a-universal-image-quality-index-author-zhou-wang/
VIF: https://live.ece.utexas.edu/research/Quality/VIF.htm