# Instruções para o funcionamento e considerações gerais:

### Image Scraping para montar a base de dados:

Primeiramente, para começar o desafio foi preciso encontrar imagens de aeronaves das duas companhias aéreas escolhidas para o desafio, no caso, Azul e Gol.

O processo precisou ser bastante cuidadoso já que o site apresentado como fonte de dados possuía um número considerável de imagens de companhias aéreas distintas das pesquisadas(por exemplo, quando era pesquisada Gol, apareciam algumas imagens da WebJet e quando era pesquisada Azul, apareciam imagens da JetBlue) além de imagens inadequadas para o treinamento do modelo como por exemplo imagens do interior do avião.

Os critérios para a seleção dos dados foram:

. Selecionar o mesmo número de imagens da Azul e da Gol para que quando o modelo da rede neural fosse treinado, não houvesse favorecimento de uma em relação a outra. No caso, a Azul possuía menos foto, então delimitou o número de imagens na base de dados.

. Não utilizar imagens que contivessem aviões da Azul e da Gol simultaneamente.

. Utilizar imagens com avião ou aviões da Azul com avião ou aviões de outras companhias.

. Utilizar imagens com avião ou aviões da Gol com avião ou aviões de outras companhias.

. Utilizar imagens de partes do avião como pedaços de turbinas, ponta da asa.

Obs: Os três últimos critérios foram decidos com o intuito de deixar o modelo mais robusto.

Feita a seleção das imagens com base nos critérios descritos, foram verificadas as imagens do Dataset gerado, uma a uma, com o objetivo de fazer um ajuste fino caso, alguma imagem inadequada nos critérios estabelecidos houvesse passado pela primeira seleção. Após esse processo, foi gerado um Dataset com um número igual de imagens de aviões da Azul e da Gol.

### Divisão de conjunto de dados de treino e de teste:

Levando em consideração que o conjunto de dados não possui algo na ordem de milhões ou mais de dados, foi utilizada uma divisão na proporção 80% dados de treino e 20% dados de teste. Isso foi feito dividindo o conjunto inicial de dados em dois para cada companhia: um de treino e outro de teste, onde o de treino consistia nas 80% primeiras fotos e o de teste, nas 20% últimas fotos, assumindo que as fotos estão aleatoriamente distribuídas pelo conjunto de dados, o que é razoável já que cada foto no site é tirada de um certo ângulo, por um certo fotógrafo, então é seguro assumir que não há risco de se colocar mais fotos de um tipo nos dados de treino em relação aos de teste ou vice-versa.

### Processamento das imagens para treino e teste

Primeiramente, é preciso fazer os seguintes imports:

import numpy as np

import matplotlib.pyplot as plt

import os

import cv2

import random

import pickle

Após isso, foi especificado o diretório, no caso em meu computador, em que se encontram as imagens do Train\_Data e as possíveis classes, Azul e Gol.

A partir disso, foi criado um vetor vazio que vai servir de recipiente para as imagens do Train\_Dataset processadas. Esse processamento se dá lendo as imagens do Train\_Dataset utilizando OpenCV, transformando-as em vetores e após isso as redimensionando para ficarem num tamanho aceitável em termos de processamento para a rede neural, no caso, 64x64. Feito esse processamento em cada imagem, armazenam-se as novas imagens redimensionadas no vetor criado.

Após isso, mais como medida extra de segurança, as imagens são misturadas aleatoriamente dentro desse vetor.

Em seguida, é extrair os dados de input e labels. Criam-se então um vetor de valores de pixel de cada imagem e um vetor de label de cada imagem e converte-se esses vetores para vetores numpy. Como último passo, é preciso salvar esses vetores de dados que se dividem em X\_training e y\_training, o primeiro contendo vetores numpy das imagens e o segundo seus labels. Salvamos esses vetores utilizando a biblioteca pickle na pasta do projeto como explicitado no código.

Para o conjunto de teste foi feito basicamente o mesmo processo, exceto o comando para organizar aleatoriamente os dados no testing\_data.

### Construção da rede neural

Em primeiro lugar, é preciso fazer os seguintes imports:

import tensorflow as tf

from tensorflow.keras.models import Sequential[]

from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D

import pickle

import matplotlib.pyplot as plt

Após o teste de diversas variações, o modelo que atendeu melhor às necessidades desse problema foi uma rede neural convolucional consistindo de oito camadas, sendo as três primeiras, camadas convolucionais de 64 neurons e foi adicionado Dropout de 0.25 com o objtivo de fazer a regularização de cada uma delas, a quarta, uma camada convolucional de 32 neurons e a quinta, uma camada convolucional de 16 neurons. Todas as camadas de convolução possuem uma ativação “relu”, é aplicado MaxPooling com Pool size de 2x2 em cada uma delas e as quatro primeiras possuem filtros de convolução 3x3, enquanto a quinta possui um filtro de convolução 1x1. Passando para as camadas densas, a sexta camada consiste de 32 neurons e ativação “relu”, a sétima camada possui 8 neurons e ativação “relu” e a oitava e última camada é composta por um único neuron com ativação “sigmoid”.

A perda desse modelo, é Binary Crossentropy, seu otimizador, Adam e sua métrica é accuracy.

Para o treinamento do modelo foi utilizado o comando model.fit e após o treinamento e exibição de seu resultado, o modelo é salvo.

### Verificando a eficiência do modelo:

A eficiência do modelo foi verificada no código Avaliação do modelo.py onde foi carregado o modelo e os dados do conjunto de teste e usando o comando model.evaluate, que retorna um vetor [loss, accuracy]. Foram obtidos como resultados:

Loss = 0.10174

Accuracy = 0.982

### Retornando o output pedido

Por último, foi feito um código de aplicação onde, uma vez que o usuário tenha escolhido as fotos que quer que sejam classificadas pela rede neural e as salvo em um diretório, as imagens são pré-processadas e normalizadas e após esses passos, são usadas como input para a rede neural, que após analisa-las, mostra as imagens com suas respectivas classificações abaixo. Para que esse teste fosse realizado com a maior fidelidade possível, foram selecionadas imagens que não vieram do site de fonte de dados, para que com certeza, as imagens a serem testadas não fizessem parte do conjunto de treino nem de teste. Após isso, as fotos foram desorganizadas dentro do conjunto de dados de forma que não houvesse padrão entre fotos da Azul e da Gol. Foram testadas apenas 20 fotos, sendo 10 da Azul e 10 da Gol, apenas para ter uma confirmação extra do resultado, no entanto, o usuário pode inserir quantas imagens precisar.

No drive existem três pastas: Train\_data, que consiste nas imagens usadas para treinar a rede neural, Test\_Data, que consiste nas imagens usadas para verificar a perda e a precisão do modelo e por fim, Real\_Data, que consiste de imagens para serem usadas como uma confirmação de imagens do mundo real, como se a rede neural estivesse ativa e trabalhando com imagens que fossem capturadas por algumas câmera.

Link do drive: https://drive.google.com/drive/folders/1qQ44EzRd8AzfVoRFJabKc8QLOcnngKj3?usp=sharing

### Script demo do Código de definição e treinamento do modelo

import tensorflow as tf

from tensorflow.keras.models import Sequential[]

from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D

import pickle

import matplotlib.pyplot as plt

# Carregando os dados de treino

X\_training = pickle.load(open("X\_training.pickle", "rb"))

y\_training = pickle.load(open("y\_training.pickle", "rb"))

# normalização das imagens

X\_training = X\_training/255.0

# Definição do modelo

model = Sequential()

model.add(Conv2D(64, (3,3), input\_shape = X\_training.shape[1:]))

model.add(Activation("relu"))

model.add(MaxPooling2D(pool\_size = (2,2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3,3)))

model.add(Activation("relu"))

model.add(MaxPooling2D(pool\_size = (2,2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3,3)))

model.add(Activation("relu"))

model.add(MaxPooling2D(pool\_size = (2,2)))

model.add(Dropout(0.25))

model.add(Conv2D(32, (3,3)))

model.add(Activation("relu"))

model.add(MaxPooling2D(pool\_size = (2,2)))

model.add(Conv2D(16, (1,1)))

model.add(Activation("relu"))

model.add(MaxPooling2D(pool\_size = (2,2)))

model.add(Flatten())

model.add(Dense(32))

model.add(Activation("relu"))

model.add(Dense(8))

model.add(Activation("relu"))

model.add(Dense(1))

model.add(Activation("sigmoid"))

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

optimizer = "adam",

metrics = ["accuracy"])

# Treinamento dividindo o conjunto de treino em Train\_Data e Dev\_Data numa razão de 25% de Dev Data

history = model.fit(X\_training, y\_training, validation\_split=0.25, epochs=100, batch\_size=16, verbose=1)

# Plota valores de training e validation accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

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

plt.show()

# Plota valores de training e validation loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

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

plt.show()

# Salvando o modelo

model.save("Conv Net Desafio CyberLabs")