Técnicas de Deep Learning para classificar imagens da Marvel

Grupo:

- Marcelo Reis Bohrer
- John Alexander Castillo

Emails:

- mrbohrer@edu.unisinos.br
- johnlo@edu.unisinos.br

O presente trabalho foi desenvolvido utilizando-se uma base de dados de imagens de um grupo de super-heróis da Marvel.

Contexto

Os super-heróis estão na cultura popular há muito tempo e agora mais do que nunca. Este conjunto de dados (dataset) tem por objetivo fornecer imagens de um grupo de oito super-heróis.

Dataset escolhido:

https://www.kaggle.com/hchen13/marvel-heroes

Caracteristicas do dataset:

Arquivos de imagens: 3035 Classes: 8 (conforme os diretórios)

Após baixarmos o Dataset, todas as imagens foram colocadas na estrutura abaixo e classificadas conforme o nome das pastas que correspondem ao nome do super-herói em questão.

Carregar dependencias

Por se tratar apenas de imagens, o Dataset baixado não possui variáveis, ou seja, não é um arquivo de dados tradicional, são apenas arquivos de imagens separados em pastas. Após baixarmos todos os arquivos de imagens, os mesmos foram estruturados em pastas conforme o nome de cada super-herói. Estas pastas irão gerar as classes utilizadas como saída para a classificação das imagens.

Diretórios:

DeepLearning/Marvel/train

- black_widow
- captain_america
- doctor strange
- hulk
- ironman
- loki
- spider_man
- thano

Carregar base da Marvel disponibilizada no GitHub público

```
!git clone https://github.com/johncastillodc/DeepLearning.git
```

```
Cloning into 'DeepLearning'...
remote: Enumerating objects: 3037, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 3037 (delta 2), reused 0 (delta 0), pack-reused 3028
Receiving objects: 100% (3037/3037), 564.50 MiB | 40.56 MiB/s, done.
Resolving deltas: 100% (14/14), done.
Checking out files: 100% (3036/3036), done.
```

Importar bibliotecas utilizadas

```
import cv2 as cv
import numpy as np
import pandas as pd
import os
import seaborn as sns

import matplotlib.pyplot as plt
from matplotlib import image
```

```
from numpy import asarray

import PIL
from PIL import Image

from matplotlib import pyplot
from keras.preprocessing.image import load_img
from keras.preprocessing.image import save_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.mobilenet_v2 import preprocess_input


from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization, LeakyReLU
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
from keras.models import Model, Sequential

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
```

Verificando GPU estão ativadas para o tensorflow

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
print(device_name)
```

```
     /device:GPU:0
```

Método para importar arquivos de Treinamento para o Google Colab

Procedimento para carregar todos os arquivos do disco e criar uma dataframe.

A partir da leitura das pastas estruturadas, foi criado um Data Frame com os seguintes campos:

- ARQUIVO: Possui a descrição do caminho em disco onde se encontra o arquivo de imagem
- ROTULO: Possui o nome de cada pasta correspondente a cada nome de super-herói
- ALVO: Número inteiro de 1 a 8 correspondente a cada pasta com nomes de super-heróis

```
def carrega_dataframe():
```

```
dados = {
    "ARQUIVO": [],
    "ROTULO": [],
    "ALVO": [],
}
root = 'DeepLearning/Marvel/train/'
caminho_black_widow     = root+"black_widow"
caminho_captain_america = root+"captain_america"
caminho_doctor_strange  = root+"doctor_strange"
caminho_hulk            = root+"hulk"
caminho_ironman         = root+"ironman"
caminho_loki            = root+"loki"
caminho_spider_man      = root+"spider_man"
caminho_thanos          = root+"thanos"


black_widow     = os.listdir(caminho_black_widow )
captain_america = os.listdir(caminho_captain_america )
doctor_strange  = os.listdir(caminho_doctor_strange )
hulk            = os.listdir(caminho_hulk )
spider_man      = os.listdir(caminho_spider_man )
ironman         = os.listdir(caminho_ironman )
loki            = os.listdir(caminho_loki )
thanos          = os.listdir(caminho_thanos )


for arquivo in black_widow:
    dados["ARQUIVO"].append(f"{caminho_black_widow}{os.sep}{arquivo}")
    dados["ROTULO"].append(f"black_widow")
    dados["ALVO"].append(1)

for arquivo in captain_america:
    dados["ARQUIVO"].append(f"{caminho_captain_america}{os.sep}{arquivo}")
    dados["ROTULO"].append(f"captain_america")
    dados["ALVO"].append(2)

for arquivo in doctor_strange:
    dados["ARQUIVO"].append(f"{caminho_doctor_strange}{os.sep}{arquivo}")
    dados["ROTULO"].append(f"doctor_strange")
    dados["ALVO"].append(3)

for arquivo in hulk:
    dados["ARQUIVO"].append(f"{caminho_hulk}{os.sep}{arquivo}")
    dados["ROTULO"].append(f"hulk")
    dados["ALVO"].append(4)

for arquivo in spider_man:
    dados["ARQUIVO"].append(f"{caminho_spider_man}{os.sep}{arquivo}")
    dados["ROTULO"].append(f"spider_man")
    dados["ALVO"].append(5)

for arquivo in ironman:
```

```
    for arquivo in ironman:
        dados["ARQUIVO"].append(f"{caminho_ironman}{os.sep}{arquivo}")
        dados["ROTULO"].append(f"ironman")
        dados["ALVO"].append(6)

    for arquivo in loki:
        dados["ARQUIVO"].append(f"{caminho_loki}{os.sep}{arquivo}")
        dados["ROTULO"].append(f"loki")
        dados["ALVO"].append(7)

    for arquivo in thanos:
        dados["ARQUIVO"].append(f"{caminho_thanos}{os.sep}{arquivo}")
        dados["ROTULO"].append(f"thanos")
        dados["ALVO"].append(8)


    dataframe = pd.DataFrame(dados)

    return dataframe


dados = carrega_dataframe()


dados
```

	ARQUIVO	ROTULO	ALVO
0	DeepLearning/Marvel/train/black_widow/pic_259.jpg	black_widow	1
1	DeepLearning/Marvel/train/black_widow/pic_203.jpg	black_widow	1
2	DeepLearning/Marvel/train/black_widow/pic_206.jpg	black_widow	1
3	DeepLearning/Marvel/train/black_widow/pic_377.jpg	black_widow	1
4	DeepLearning/Marvel/train/black_widow/pic_082.jpg	black_widow	1
...	...	...	...
2579	DeepLearning/Marvel/train/thanos/pic_051.jpg	thanos	8
2580	DeepLearning/Marvel/train/thanos/pic_068.jpg	thanos	8
2581	DeepLearning/Marvel/train/thanos/pic_113.jpg	thanos	8
2582	DeepLearning/Marvel/train/thanos/pic_249.jpg	thanos	8
2583	DeepLearning/Marvel/train/thanos/pic_179.jpg	thanos	8

2584 rows × 3 columns

```
dados['ROTULO'].value_counts()
```

```
doctor_strange     345
spider_man         326
```

```
captain_america    324
thanos             323
hulk               321
black_widow        320
ironman            318
loki               307
Name: ROTULO, dtype: int64
```

```
dados['ROTULO'].unique()
```

```
array(['black_widow', 'captain_america', 'doctor_strange', 'hulk',
       'spider_man', 'ironman', 'loki', 'thanos'], dtype=object)
```

```
dados.columns
```

```
Index(['ARQUIVO', 'ROTULO', 'ALVO'], dtype='object')
```

Salvar o dataframe em arquivo csv

```
dados.to_csv("DeepLearning/Marvel/train/imagens-marvel.csv")
```

Como o dataframe já foi salvo em arquivo .csv não há necessidade de rodar os procedimentos para criação do dataframe. Basta ler o arquivo .csv criado e seguir deste ponto.

Leitura dos dados

```
dados = pd.read_csv("DeepLearning/Marvel/train/imagens-marvel.csv")
```

```
dados.head(10000)
```

	Unnamed: 0	ARQUIVO	ROTULO	ALVO
0	0	DeepLearning/Marvel/train/black_widow/pic_259.jpg	black_widow	1
1	1	DeepLearning/Marvel/train/black_widow/pic_203.jpg	black_widow	1
2	2	DeepLearning/Marvel/train/black_widow/pic_206.jpg	black_widow	1
3	3	DeepLearning/Marvel/train/black_widow/pic_377.jpg	black_widow	1

```
dados['ALVO'].unique()
```

```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

2579	2579	DeepLearning/Marvel/train/thanos/pic_051.jpg	thanos	8

Procedimento para seleção de dados aleatoriamente

2581	2581	DeepLearning/Marvel/train/thanos/pic_113.jpg	thanos	8

Cria dataframe com 200 exemplos aleatorios

2583	2583	DeepLearning/Marvel/train/thanos/pic_179.jpg	thanos	8

```
num_samples = 200
samples = []

for category in dados['ROTULO'].unique():
    category_slice = dados.query("ROTULO == @category")
    samples.append(category_slice.sample( num_samples, random_state=1, replace=True  ))

dados = pd.concat(samples, axis=0).sample(frac=1.0, random_state=1).reset_index(drop=True)
dados['ROTULO'].value_counts()
```

```
spider_man         200
doctor_strange     200
hulk               200
loki               200
captain_america    200
black_widow        200
thanos             200
ironman            200
Name: ROTULO, dtype: int64
```

```
print('Training data shape : ', dados.shape)
print('Testing data shape. : ', dados.shape)
```

```
Training data shape :  (1600, 4)
Testing data shape. :  (1600, 4)
```

```
dados["ROTULO"].unique()
```

```
array(['black_widow', 'loki', 'doctor_strange', 'thanos', 'ironman',
       'captain_america', 'spider_man', 'hulk'], dtype=object)
```

```
dados.head(4)
```

	Unnamed: 0	ARQUIVO	ROTULO	ALVO
0	302	DeepLearning/Marvel/train/black_widow/pic_394.jpg	black_widow	1
1	1997	DeepLearning/Marvel/train/loki/pic_118.jpg	loki	7
2	773	DeepLearning/Marvel/train/doctor_strange/pic_1...	doctor_strange	3
3	2154	DeepLearning/Marvel/train/loki/pic_280.jpg	loki	7

Método para importar arquivos de Válidos para o Google Colab

```
def carrega_Validdataframe():

    dadosv = {
        "ARQUIVO": [],
        "ROTULO": [],
        "ALVO": [],
    }
    rootv = 'DeepLearning/Marvel/valid/'
    caminho_black_widow     = rootv+"black_widow"
    caminho_captain_america = rootv+"captain_america"
    caminho_doctor_strange  = rootv+"doctor_strange"
    caminho_hulk            = rootv+"hulk"
    caminho_ironman         = rootv+"ironman"
    caminho_loki            = rootv+"loki"
    caminho_spider_man      = rootv+"spider_man"
    caminho_thanos          = rootv+"thanos"


    black_widow     = os.listdir(caminho_black_widow )
    captain_america = os.listdir(caminho_captain_america )
    doctor_strange  = os.listdir(caminho_doctor_strange )
    hulk            = os.listdir(caminho_hulk )
    spider_man      = os.listdir(caminho_spider_man )
    ironman         = os.listdir(caminho_ironman )
    loki            = os.listdir(caminho_loki )
    thanos          = os.listdir(caminho_thanos )


    for arquivo in black_widow:
        dadosv["ARQUIVO"].append(f"{caminho_black_widow}{os.sep}{arquivo}")
        dadosv["ROTULO"].append(f"black_widow")
        dadosv["ALVO"].append(1)

    for arquivo in captain_america:
        dadosv["ARQUIVO"].append(f"{caminho_captain_america}{os.sep}{arquivo}")
```

```
            dadosv["ARQUIVO"].append(f"{caminho_captain_america}{os.sep}{arquivo}")
            dadosv["ROTULO"].append(f"captain_america")
            dadosv["ALVO"].append(2)

        for arquivo in doctor_strange:
            dadosv["ARQUIVO"].append(f"{caminho_doctor_strange}{os.sep}{arquivo}")
            dadosv["ROTULO"].append(f"doctor_strange")
            dadosv["ALVO"].append(3)

        for arquivo in hulk:
            dadosv["ARQUIVO"].append(f"{caminho_hulk}{os.sep}{arquivo}")
            dadosv["ROTULO"].append(f"hulk")
            dadosv["ALVO"].append(4)

        for arquivo in spider_man:
            dadosv["ARQUIVO"].append(f"{caminho_spider_man}{os.sep}{arquivo}")
            dadosv["ROTULO"].append(f"spider_man")
            dadosv["ALVO"].append(5)

        for arquivo in ironman:
            dadosv["ARQUIVO"].append(f"{caminho_ironman}{os.sep}{arquivo}")
            dadosv["ROTULO"].append(f"ironman")
            dadosv["ALVO"].append(6)

        for arquivo in loki:
            dadosv["ARQUIVO"].append(f"{caminho_loki}{os.sep}{arquivo}")
            dadosv["ROTULO"].append(f"loki")
            dadosv["ALVO"].append(7)

        for arquivo in thanos:
            dadosv["ARQUIVO"].append(f"{caminho_thanos}{os.sep}{arquivo}")
            dadosv["ROTULO"].append(f"thanos")
            dadosv["ALVO"].append(8)


        dataframev = pd.DataFrame(dadosv)

        return dataframev


    dadosv = carrega_Validdataframe()


    dadosv
```

	ARQUIVO	ROTULO	ALVO
0	DeepLearning/Marvel/valid/black_widow/pic_053.jpg	black_widow	1
1	DeepLearning/Marvel/valid/black_widow/pic_357.jpg	black_widow	1
2	DeepLearning/Marvel/valid/black_widow/pic_336.jpg	black_widow	1
3	DeepLearning/Marvel/valid/black_widow/pic_191.jpg	black_widow	1
4	DeepLearning/Marvel/valid/black_widow/pic_006.jpg	black_widow	1
...	...	...	...
446	DeepLearning/Marvel/valid/thanos/pic_009.jpg	thanos	8
447	DeepLearning/Marvel/valid/thanos/pic_350.jpg	thanos	8

```
dadosv['ROTULO'].value_counts()
```

```
doctor_strange     61
spider_man         57
captain_america    57
hulk               56
ironman            56
thanos             55
black_widow        55
loki               54
Name: ROTULO, dtype: int64
```

Separação dos dados em treinamento e teste

```
train_df=dados
test_df=dadosv
```

```
train_df['ALVO'].count()
```

```
1600
```

```
test_df['ALVO'].count()
```

```
451
```

Image Augmentation on the fly using Keras ImageDataGenerator!

https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/

```
data_path = './'
```

```
datagen   = ImageDataGenerator(preprocessing_function=preprocess_input,validation_split=0.2)

train_generator=datagen.flow_from_dataframe(dataframe=train_df,
                                            directory=(data_path),
                                            x_col='ARQUIVO',
                                            y_col='ROTULO',
                                            subset='training',
                                            batch_size=32,
                                            seed=42,
                                            shuffle=True,
                                            class_mode='categorical',
                                            target_size=(224,224),
                                            color_mode='rgb',
                                            validate_filenames=False)

valid_generator=datagen.flow_from_dataframe(dataframe=train_df,
                                            directory=(data_path),
                                            x_col='ARQUIVO',
                                            y_col='ROTULO',
                                            subset='validation',
                                            classes=None,
                                            batch_size=32,
                                            seed=42,
                                            class_mode='categorical',
                                            target_size=(224,224),
                                            color_mode='rgb',
                                            validate_filenames=False)


test_generator=datagen.flow_from_dataframe(dataframe=test_df,
                                            directory=(data_path),
                                            x_col='ARQUIVO',
                                            y_col='ROTULO',
                                            subset='validation',
                                            classes=None,
                                            batch_size=32,
                                            seed=42,
                                            class_mode='categorical',
                                            target_size=(224,224),
                                            color_mode='rgb',
                                            validate_filenames=False)
```

```
Found 1280 non-validated image filenames belonging to 8 classes.
Found 320 non-validated image filenames belonging to 8 classes.
Found 90 non-validated image filenames belonging to 8 classes.
```

Problema

Dada uma imagem de um super-herói qualquer, a aplicação deverá classificar esta imagem entre as oito classes treinadas para tal.

Modelagem

Por se tratar de classificação de imagens foi escolhida a técnica de CNN - Convolutional Neural Network, utilizando-se as bibliotécas de Deep Learning do Keras. https://keras.io/getting_started/.

Também foi utilizado o algoritmo de árvore de decisão na perspectiva do Machining Learnig.

A partir do Data Frame montado, foram selecionados aleatoriamente um conjunto x imagens para cada classe e separadas em treinamento e teste. Neste caso utilizamos as folder previamente separadas pelo repositorio do kaggle.

A partir das bases de treinamento e teste, utilizamos a técnica de geração de imagens utilizando a bibliotéca “ImageDataGenerator” do Keras. A técnica de aumento de imagem é uma ótima maneira de expandir o tamanho do seu conjunto de dados. Você pode criar novas imagens transformadas de seu conjunto de dados original. Algumas transformações são: rotação, deslocamento, brilho, zoom, etc.

A partir dos dados de treinamento e teste, foram geradas 1280 imagens de treinamento, 320 imagens de validação e 90 imagens para testes.

- Found 1280 non-validated image filenames belonging to 8 classes.
- Found 320 non-validated image filenames belonging to 8 classes.
- Found 90 non-validated image filenames belonging to 8 classes.

Árvore de Decisão

```
labels = dados["ROTULO"].unique()
labels.shape
```

```
(8,)
```

```
#creating bag of words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 150000)
X= cv.fit_transform(train_df.iloc[:,1]).toarray()
y= train_df.iloc[:,3]
X.shape, y.shape
```

```
((1600, 416), (1600,))
```

Separar dados em treino e teste para Árvore de decisão

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((1120, 416), (1120,), (480, 416), (480,))
```

```
from sklearn.ensemble import ExtraTreesClassifier
# criação do modelo Árvode de decisão
modelo = ExtraTreesClassifier( criterion='gini')
```

```
modelo.fit(X_train, y_train)
```

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

```
# Acurácia do modelo
# y_teste estão as classificações corretas
# score = compara os dados de testes com os dados de treino (modelo rodou para os dados de tr
resultado = modelo.score(X_test, y_test)

print('Acurácia =', resultado)
```

```
Acurácia = 1.0
```

```
y_pred = modelo.predict(X_test)
```

Fazendo a matriz de confusão

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[64,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 66,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 69,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 56,  0,  0,  0,  0],
```

```
[ 0,  0,  0,  0, 61,  0,  0,  0],
[ 0,  0,  0,  0,  0, 52,  0,  0],
[ 0,  0,  0,  0,  0,  0, 51,  0],
[ 0,  0,  0,  0,  0,  0,  0, 61]])
```

Fazendo a matriz de confusão detalhada

```
print(pd.crosstab(y_test, y_pred, rownames=['Reais'],colnames=['Preditos'], margins=True))
```

Preditos	1	2	3	4	5	6	7	8	All
Reais									
1	64	0	0	0	0	0	0	0	64
2	0	66	0	0	0	0	0	0	66
3	0	0	69	0	0	0	0	0	69
4	0	0	0	56	0	0	0	0	56
5	0	0	0	0	61	0	0	0	61
6	0	0	0	0	0	52	0	0	52
7	0	0	0	0	0	0	51	0	51
8	0	0	0	0	0	0	0	61	61
All	64	66	69	56	61	52	51	61	480

CNN - Deep Learning - Arquitetura A

Criamos duas arquiteturas de modelos A e B Se executar a arquitetura A não executar a arquitetura B

Se executar a arquitetura B não executar a arquitetura A (célula abaixo), pular para célula da arq. A

```
model = Sequential()

# Primeira camada convolution
model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=train_generator.image
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2),padding='same'))

# Segunda camada convolution
model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# terceira camada convolution
model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

model.add(Flatten())
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=0.1))
```

```
model.add(Dense(8, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 224, 224, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 112, 112, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 128)       73856
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 56, 56, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
dense (Dense)                (None, 128)               12845184
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 1032
=================================================================
Total params: 12,939,464
Trainable params: 12,939,464
Non-trainable params: 0
_________________________________________________________________
```

Otimizador escolhido:

RMSprop

```
model.compile(optimizers.RMSprop(lr=0.0001, decay=1e-6),loss='categorical_crossentropy',metri
#model.compile( optimizers.Adam(learning_rate=0.001,beta_1=0.9,beta_2=0.999,epsilon=1e-07,ams
#model.compile( optimizers.SGD(learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD"),
#model.compile( optimizers.Adamax(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-0
```

```
# importar biblioteca para calculo de métricas
from sklearn.metrics import mean_squared_error
```

Criação do modelo

Funções de ativação

https://keras.io/api/layers/activations/#available-activations

Otimizadores utilizados na compilação do modelo

https://keras.io/api/optimizers/

Métricas

https://keras.io/api/metrics/

CNN - Deep Learning - Arquitetura B

Se a arquitetura A foi criada, não executar a célula abaixo

```
model = Sequential()

# camada de entrada
model.add(Conv2D(32, (3, 3), padding='same', input_shape=train_generator.image_shape))
model.add(Activation('relu'))

# primeira camada oculta
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))

# segunda camanda oculta
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Flatten())

# camada de saida (deve ter número de classes definida)
model.add(Dense(8, activation='softmax'))



model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 224, 224, 32)      896
_________________________________________________________________
activation (Activation)      (None, 224, 224, 32)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 222, 222, 32)      9248
_________________________________________________________________
activation_1 (Activation)    (None, 222, 222, 32)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 222, 222, 64)      18496
_________________________________________________________________
activation_2 (Activation)    (None, 222, 222, 64)      0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 111, 111, 64)      0
_________________________________________________________________
dropout (Dropout)            (None, 111, 111, 64)      0
_________________________________________________________________
dense_2 (Dense)              (None, 111, 111, 512)     33280
_________________________________________________________________
activation_3 (Activation)    (None, 111, 111, 512)     0
_________________________________________________________________
dropout_1 (Dropout)          (None, 111, 111, 512)     0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6308352)           0
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 50466824
=================================================================
Total params: 50,528,744
Trainable params: 50,528,744
Non-trainable params: 0
_________________________________________________________________
```

Compilando o modelos com outros otimizadores

```
#  https://keras.io/api/optimizers/

model.compile(optimizers.RMSprop(lr=0.0001, decay=1e-6),loss='categorical_crossentropy',metri
#model.compile( optimizers.Adam(learning_rate=0.001,beta_1=0.9,beta_2=0.999,epsilon=1e-07,ams
#model.compile( optimizers.SGD(learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD"),
#model.compile( optimizers.Adamax(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-0



STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size

print(STEP_SIZE_TRAIN, STEP_SIZE_VALID)
```

```
40 10
```

```
history = model.fit(train_generator, y=None, steps_per_epoch=STEP_SIZE_TRAIN, validation_data
```

```
Epoch 1/10
40/40 [==============================] - 59s 620ms/step - loss: 10.4247 - accuracy: 0.1
Epoch 2/10
40/40 [==============================] - 24s 606ms/step - loss: 2.0254 - accuracy: 0.24
Epoch 3/10
40/40 [==============================] - 24s 607ms/step - loss: 1.5827 - accuracy: 0.46
Epoch 4/10
40/40 [==============================] - 24s 604ms/step - loss: 1.0426 - accuracy: 0.73
Epoch 5/10
40/40 [==============================] - 24s 608ms/step - loss: 0.6298 - accuracy: 0.87
Epoch 6/10
40/40 [==============================] - 25s 607ms/step - loss: 0.3360 - accuracy: 0.95
Epoch 7/10
40/40 [==============================] - 24s 610ms/step - loss: 0.1927 - accuracy: 0.97
Epoch 8/10
40/40 [==============================] - 24s 603ms/step - loss: 0.1939 - accuracy: 0.96
Epoch 9/10
40/40 [==============================] - 24s 610ms/step - loss: 0.1063 - accuracy: 0.97
Epoch 10/10
40/40 [==============================] - 25s 615ms/step - loss: 0.0518 - accuracy: 0.99
```

```
test_eval = model.evaluate(test_generator, verbose=1)

print('Test loss:'    , test_eval[0])
print('Test accuracy:', test_eval[1])
```

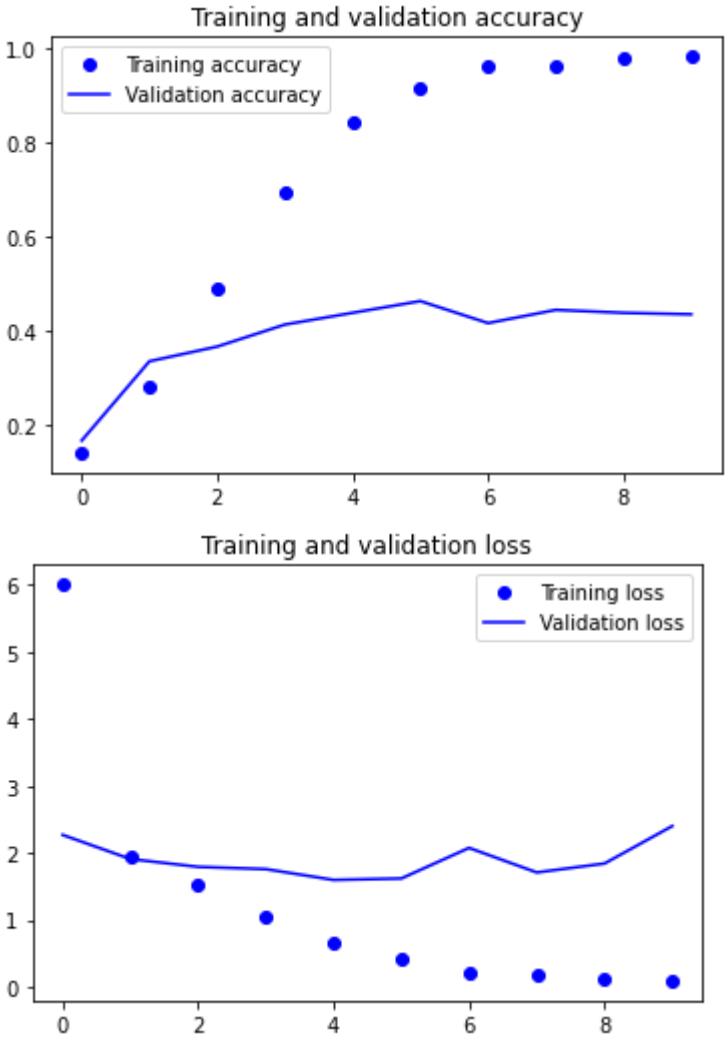
```
3/3 [==============================] - 2s 865ms/step - loss: 1.9842 - accuracy: 0.4444
Test loss: 1.9841973781585693
Test accuracy: 0.4444444477558136
```

Observamos ocorrência de Overffiting

O algoritmo “presta muita atenção” nas particularidades dos dados de treinamento e não consegue generalizar muito bem.

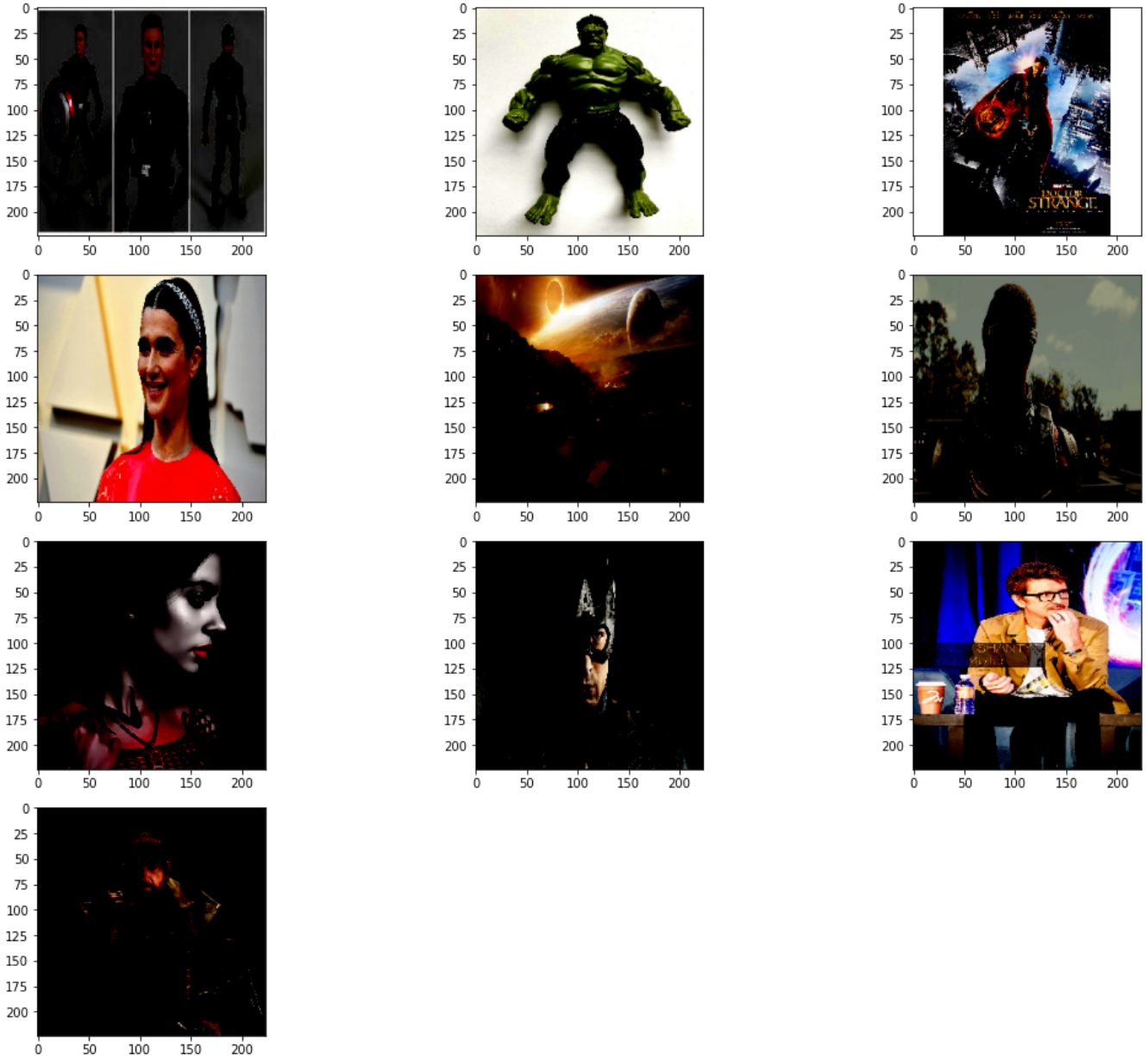
```
accuracy     = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

loss      = history.history['loss']
val_loss  = history.history['val_loss']
epochs    = range(len(accuracy))


plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
```

```
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
    plt.show()
```



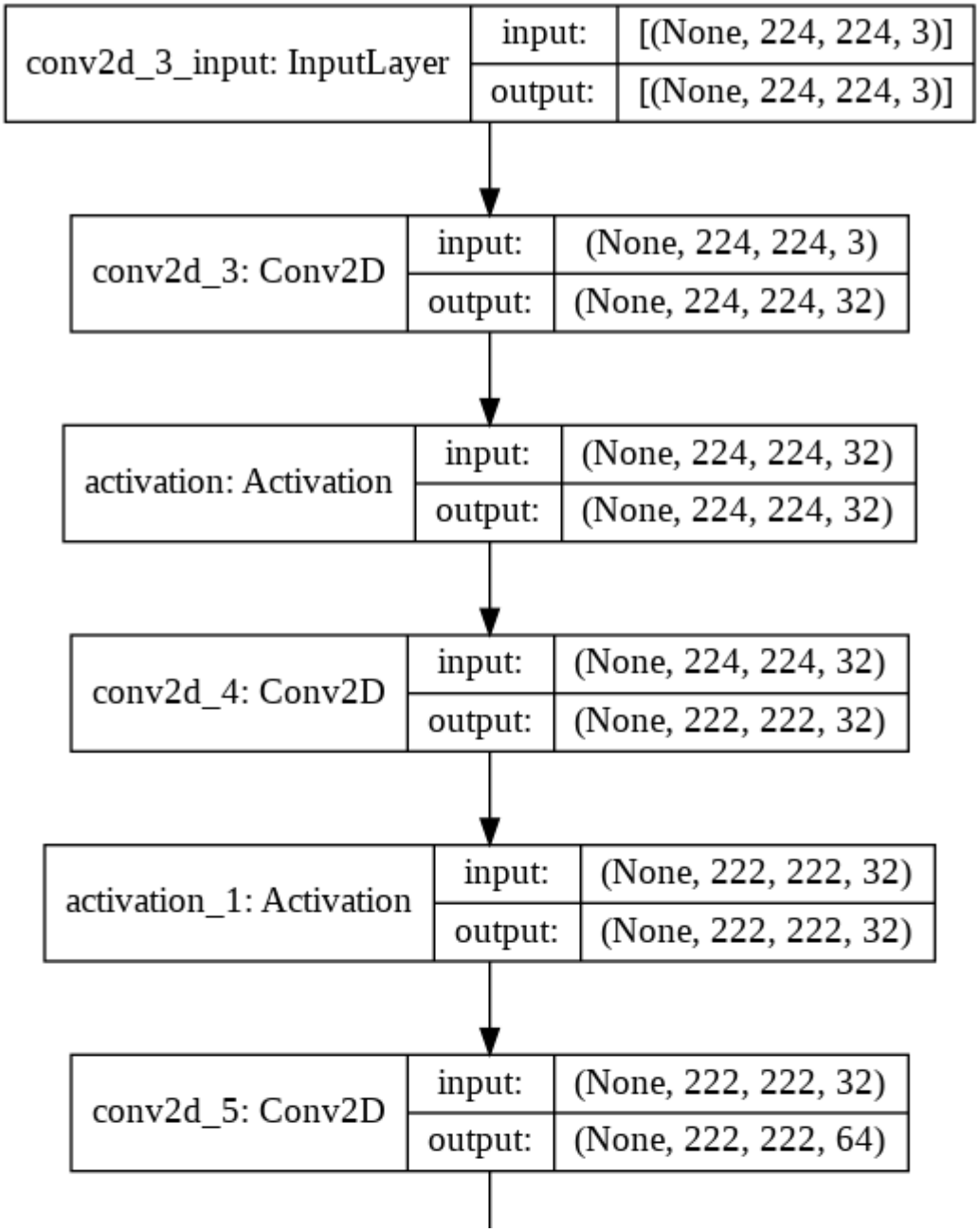

```
    plt.figure(figsize=(15,15))
    for i in range(0,10):
        plt.subplot(5,3,i+1)
        for X_batch, Y_batch in train_generator:
            image=X_batch[0]
            plt.imshow(image)
            break
    plt.tight_layout()
    plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
```



Salvar Arquitetura do Modelo em arquivo de imagem

```
dot_img_file = 'DeepLearning/Marvel/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```



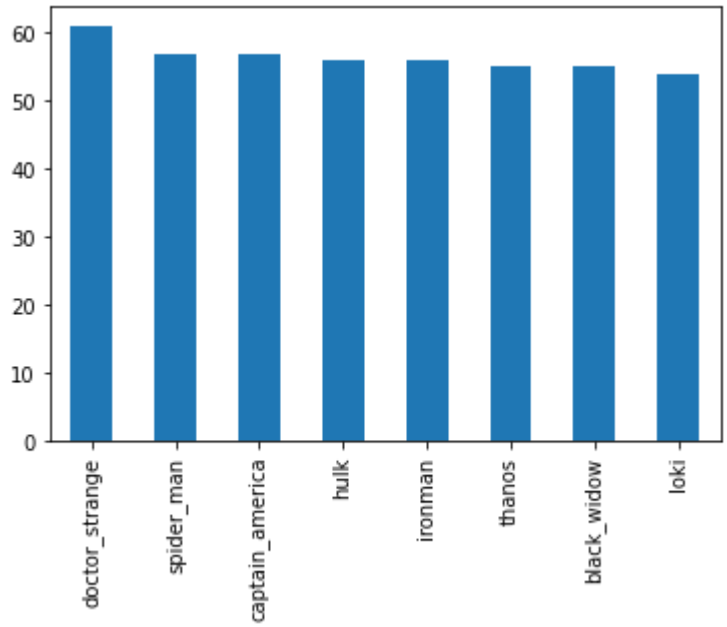

```
results = model.evaluate( test_generator, verbose=1)
predictions = np.argmax(model.predict(test_generator), axis=1)

class_names = list(test_generator.class_indices.keys())

clr = classification_report(test_generator.labels, predictions, labels=np.arange(8), target_r

print("Test Accuracy: {:.2f}%".format(results[1] * 100))
print('')
print("Classification Report:\n----------------------\n", clr)
```

```
3/3 [==============================] - 1s 357ms/step - loss: 1.9842 - accuracy: 0.4444
Test Accuracy: 44.44%

Classification Report:
----------------------
```

```
                    precision    recall  f1-score   support

      black_widow       0.68      0.62      0.65        55
   captain_america       0.60      0.17      0.27        35
    doctor_strange       0.00      0.00      0.00         0
              hulk       0.00      0.00      0.00         0
           ironman       0.00      0.00      0.00         0
              loki       0.00      0.00      0.00         0
        spider_man       0.00      0.00      0.00         0
            thanos       0.00      0.00      0.00         0

         micro avg       0.44      0.44      0.44        90
         macro avg       0.16      0.10      0.11        90
      weighted avg       0.65      0.44      0.50        90

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
```

```
dadosv['ROTULO'].value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5b112533d0>
```




```
y_pred = model.predict(test_generator)
y_pred
```

```
[4.78109094e-01, 1.07249222e-01, 1.85004182e-01, 2.09636959e-02,
 5.62564172e-02, 2.96069626e-02, 2.50506345e-02, 9.77673456e-02],

[5.18862426e-01, 7.21441284e-02, 1.53164372e-01, 2.11478514e-03,
 1.45619502e-02, 3.81174870e-02, 4.54988256e-02, 1.55536011e-01],
[8.02758262e-02, 3.16855013e-02, 2.04653498e-02, 4.16038372e-02,
 4.63211030e-01, 1.39552755e-02, 5.11197448e-02, 2.97683448e-01],
[8.06939509e-03, 1.46551132e-01, 6.07578026e-04, 2.70889583e-03,
 1.74603574e-02, 2.66176579e-03, 1.42165605e-04, 8.21798682e-01],
[4.79382873e-01, 8.52511376e-02, 4.13937643e-02, 1.25805229e-01,
```

```
 6.38647676e-02, 4.67221439e-02, 1.50277501e-03, 1.56077251e-01],
[9.99434650e-01, 3.14160343e-06, 4.38334480e-09, 3.27587477e-05,
 1.40782311e-06, 1.45144395e-05, 1.98939733e-06, 5.11550170e-04],
[6.54664934e-01, 8.19307640e-02, 3.62388603e-02, 3.20240520e-02,
 4.42065969e-02, 3.74630541e-02, 4.31895517e-02, 7.02822283e-02],
[4.38827515e-01, 5.69855934e-03, 3.39906604e-04, 3.97095978e-02,
 6.15685433e-03, 5.09251282e-03, 1.09959580e-02, 4.93179113e-01],
[2.07964946e-02, 5.28445169e-02, 6.31733937e-03, 3.97366732e-01,
 6.66618496e-02, 1.54987231e-01, 4.29045362e-03, 2.96735376e-01],
[5.71089983e-03, 2.23578700e-05, 6.27874215e-07, 1.63220335e-04,
 1.99851888e-06, 9.93812561e-01, 1.11895204e-07, 2.88234151e-04],
[1.52909383e-01, 5.46504438e-01, 1.19389733e-02, 7.69732147e-02,
 3.06106471e-02, 5.23144333e-03, 1.16069131e-01, 5.97627684e-02],
[6.09154165e-01, 5.83762676e-03, 1.40468618e-02, 3.99183389e-03,
 5.95839192e-05, 1.13757059e-03, 2.85454690e-01, 8.03177208e-02],
[2.33662091e-02, 8.86184536e-03, 6.48251807e-05, 1.25024922e-03,
 1.11906236e-04, 2.56911130e-06, 1.01514335e-03, 9.65327144e-01],
[2.46684641e-01, 1.36322647e-01, 4.86921519e-02, 1.63360173e-03,
 1.06504358e-01, 3.96860205e-03, 8.70851427e-02, 3.69108826e-01],
[5.19015133e-01, 3.06871012e-02, 2.62897164e-01, 3.70311737e-02,
 3.92608568e-02, 5.90067022e-02, 1.51344733e-02, 3.69673818e-02],
[6.27246499e-01, 2.03271373e-03, 1.47979136e-03, 2.84062512e-02,
 1.68615559e-04, 2.30683727e-05, 5.03638876e-04, 3.40139359e-01],
[9.55334902e-02, 3.69780153e-01, 8.90097767e-03, 1.68994442e-01,
 1.11362383e-01, 1.98600963e-02, 1.36034012e-01, 8.95344317e-02],
[8.98855329e-01, 1.67749877e-05, 4.84009661e-06, 5.86049509e-06,
 3.04324232e-04, 3.82154985e-06, 1.00345686e-01, 4.63271426e-04],
[1.07250195e-02, 1.31657743e-03, 5.83472662e-04, 9.81029868e-01,
 5.87026938e-04, 7.37395778e-04, 3.19842889e-04, 4.70090052e-03],
[3.25601429e-01, 1.33106604e-01, 8.57301131e-02, 6.59205168e-02,
 7.19580203e-02, 5.23038656e-02, 3.30812968e-02, 2.32298180e-01],
[9.19116974e-01, 3.82796687e-04, 3.45596845e-06, 1.03321602e-03,
 1.37581330e-04, 1.41656456e-05, 1.72842265e-05, 7.92944804e-02],
[5.67833543e-01, 7.55054206e-02, 6.92996904e-02, 3.97000909e-02,
 8.00038502e-03, 1.88020349e-01, 6.16207533e-03, 4.54784557e-02],
[9.69099328e-02, 2.08734393e-01, 3.00259772e-03, 2.14875758e-01,
 1.30049577e-02, 5.29866219e-02, 1.38051016e-02, 3.96680623e-01],
[4.35793668e-01, 1.00848721e-02, 7.25367106e-03, 2.15378143e-02,
 6.19526766e-02, 3.42089130e-04, 7.57895759e-05, 4.62959409e-01],
[4.95570838e-01, 1.28450543e-01, 5.85435927e-02, 6.51397854e-02,
 5.00478502e-03, 6.03275858e-02, 1.84909776e-02, 1.68471888e-01],
[3.72342646e-01, 4.80908811e-01, 3.83158942e-04, 1.10592037e-01,
 1.23055130e-02, 5.37382998e-03, 1.85840647e-04, 1.79082416e-02],
[6.79098606e-01, 1.15423582e-01, 3.24917659e-02, 6.02383800e-02,
 2.90715937e-02, 2.74009705e-02, 4.09565680e-02, 1.53184552e-02],
[8.21647048e-01, 2.01984658e-03, 2.85738468e-04, 4.92596738e-02,
 4.09412343e-04, 2.44194572e-03, 9.27586691e-04, 1.23008743e-01],
[2.50563025e-01, 1.43063003e-02, 1.51212991e-03, 2.60079592e-01,
 1.11399792e-01, 8.38473730e-04, 3.41240775e-05, 3.61266613e-01],
[2.39245936e-01, 8.86363164e-02, 5.68774471e-04, 1.36941513e-02,
```

Avaliação dos resultados:

Árvore de Decisão – Machine Learning

Nro. Exemplos	Épocas	loss	% Acuracidade
150	10	2,9446	33,33
150	20	4,3842	30,56
150	5	4,7769	27,78

Convolutional Neural Network (CNN) – Deep Learning

Nro. Exemplos	Épocas	Arquitetura Rede	Otimizador	loss	% Acuracidade
100	20	Arquitetura A	RMSprop	2,3337	37,50
150	20	Arquitetura A	RMSprop	1,5851	59,72
200	20	Arquitetura A	RMSprop	0,6319	86,46
200	20	Arquitetura A	Adam	0,5010	89,58
200	20	Arquitetura A	SGD	2,0689	15,62
200	20	Arquitetura A	Adamax	0,6431	89,58
200	20	Arquitetura B	RMSprop	0,7138	83,33
200	10	Arquitetura B	RMSprop	0,8624	72,92

Conclusão

Observou-se que o trabalho prático é uma excelente ferramenta para o aprendizado desta disciplina. Em relação aos dados, temos um grande desafio que é sua preparação antes de submetê-los a qualquer algoritmo e/ou técnicas de aprendizado de máquina. O processo, como um todo, é laborioso haja vista que são necessários várias iterações e vários ajustes nos vários parâmetros dos algoritmos. Ainda, é importante salientar que sem o devido conhecimento conceitual fica mais complexo a escolha do algoritmo, bem como os ajustes em seus parâmetros.

✓ 1s conclusão: 19:54