Classify_imgs

October 7, 2020

[13]:
```python
from tensorflow import keras
from imutils import paths
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD
import numpy as np
```

[2]:
```python
# importe de pacotes
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import ZeroPadding2D
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import add
from tensorflow.keras.regularizers import l2
from tensorflow.keras import backend as K


class ResNet:
    @staticmethod
    def residual_module(data, K, stride, chanDim, red=False,
                        reg=0.0001, bnEps=2e-5, bnMom=0.9):
        # the shortcut branch of the ResNet module should be
        # initialize as the input (identity) data
        shortcut = data

        # the first block of the ResNet module are the 1x1 CONVs
        bn1 = BatchNormalization(axis=chanDim, epsilon=bnEps,
                                 momentum=bnMom)(data)
        act1 = Activation("relu")(bn1)
        conv1 = Conv2D(int(K * 0.25), (1, 1), use_bias=False,
                       kernel_regularizer=l2(reg))(act1)
```

```
        # the second block of the ResNet module are the 3x3 CONVs
        bn2 = BatchNormalization(axis=chanDim, epsilon=bnEps,
                                   momentum=bnMom)(conv1)
        act2 = Activation("relu")(bn2)
        conv2 = Conv2D(int(K * 0.25), (3, 3), strides=stride,
                       padding="same", use_bias=False,
                       kernel_regularizer=l2(reg))(act2)

        # the third block of the ResNet module is another set of 1x1
        # CONVs
        bn3 = BatchNormalization(axis=chanDim, epsilon=bnEps,
                                   momentum=bnMom)(conv2)
        act3 = Activation("relu")(bn3)
        conv3 = Conv2D(K, (1, 1), use_bias=False,
                       kernel_regularizer=l2(reg))(act3)

        # if we are to reduce the spatial size, apply a CONV layer to
        # the shortcut
        if red:
            shortcut = Conv2D(K, (1, 1), strides=stride,
                              use_bias=False, kernel_regularizer=l2(reg))(act1)

        # add together the shortcut and the final CONV
        x = add([conv3, shortcut])

        # return the addition as the output of the ResNet module
        return x

    @staticmethod
    def build(width, height, depth, classes, stages, filters,
              reg=0.0001, bnEps=2e-5, bnMom=0.9):
        # initialize the input shape to be "channels last" and the
        # channels dimension itself
        inputShape = (height, width, depth)
        chanDim = -1

        # if we are using "channels first", update the input shape
        # and channels dimension
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)
            chanDim = 1

        # set the input and apply BN
        inputs = Input(shape=inputShape)
        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
                               momentum=bnMom)(inputs)
```

```
        # apply CONV => BN => ACT => POOL to reduce spatial size
        x = Conv2D(filters[0], (5, 5), use_bias=False,
                   padding="same", kernel_regularizer=l2(reg))(x)
        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
                               momentum=bnMom)(x)
        x = Activation("relu")(x)
        x = ZeroPadding2D((1, 1))(x)
        x = MaxPooling2D((3, 3), strides=(2, 2))(x)

        # loop over the number of stages
        for i in range(0, len(stages)):
            # initialize the stride, then apply a residual module
            # used to reduce the spatial size of the input volume
            stride = (1, 1) if i == 0 else (2, 2)
            x = ResNet.residual_module(x, filters[i + 1], stride,
                                       chanDim, red=True, bnEps=bnEps,␣
↪bnMom=bnMom)

            # loop over the number of layers in the stage
            for j in range(0, stages[i] - 1):
                # apply a ResNet module
                x = ResNet.residual_module(x, filters[i + 1],
                                           (1, 1), chanDim, bnEps=bnEps,␣
↪bnMom=bnMom)

        # apply BN => ACT => POOL
        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
                               momentum=bnMom)(x)
        x = Activation("relu")(x)
        x = AveragePooling2D((8, 8))(x)

        # sigmoid classifier
        x = Flatten()(x)
        x = Dense(classes, kernel_regularizer=l2(reg))(x)
        x = Activation("sigmoid")(x)

        # create the model
        model = Model(inputs, x, name="resnet")

        # return the constructed network architecture
        return model
```

[3]:
```
height, width = 224, 224
```

[4]:
```
# initialize the number of training epochs and batch size
NUM_EPOCHS = 50
BS = 32
```

```
TRAIN_PATH = '../dados/'
# determine the total number of image paths in training, validation,
# and testing directories
totalTrain = len(list(paths.list_images(TRAIN_PATH)))
```

[5]:
```
# initialize the training training data augmentation object
trainAug = ImageDataGenerator(
    rescale=1 / 255.0,
    rotation_range=20,
    zoom_range=0.05,
    width_shift_range=0.05,
    height_shift_range=0.05,
    shear_range=0.05,
    horizontal_flip=True,
    validation_split=0.1)
```

[6]:
```
# initialize the testing data augmentation object
testAug = ImageDataGenerator(rescale=1 / 255.0, validation_split=0.1)
```

[7]:
```
# initialize the training generator
trainGen = trainAug.flow_from_directory(
    TRAIN_PATH,
    class_mode="categorical",
    target_size=(height, width),
    color_mode="rgb",
    shuffle=True,
    seed=123,
    batch_size=BS,
    subset='training')
```

```
Found 3200 images belonging to 2 classes.
```

[8]:
```
# initialize the testing generator
testGen = testAug.flow_from_directory(
    TRAIN_PATH,
    class_mode="categorical",
    target_size=(height, width),
    color_mode="rgb",
    shuffle=False,
    batch_size=BS,
    subset='validation')
```

```
Found 355 images belonging to 2 classes.
```

[9]:
```
model = ResNet.build(height, width, 3, 2, (2, 2, 3),
                     (32, 64, 128, 256), reg=0.0005)
```

[10]:
```
opt = SGD(lr=1e-1, momentum=0.9, decay=1e-1 / NUM_EPOCHS)
model.compile(loss="binary_crossentropy",
              optimizer=opt,
              metrics=["accuracy",
                       keras.metrics.AUC(),
                       keras.metrics.Precision(),
                       keras.metrics.Recall()])
```

[11]:
```
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

# train our Keras model
H = model.fit(
    trainGen,
    validation_data=testGen,
    epochs=NUM_EPOCHS)
```

```
Epoch 1/50
100/100 [==============================] - 293s 3s/step - loss: 0.5292 -
accuracy: 0.8944 - auc: 0.9509 - precision: 0.8944 - recall: 0.8922 - val_loss:
0.6406 - val_accuracy: 0.8789 - val_auc: 0.9368 - val_precision: 0.8789 -
val_recall: 0.8789
Epoch 2/50
100/100 [==============================] - 304s 3s/step - loss: 0.3887 -
accuracy: 0.9350 - auc: 0.9785 - precision: 0.9353 - recall: 0.9356 - val_loss:
0.5573 - val_accuracy: 0.8901 - val_auc: 0.9472 - val_precision: 0.8901 -
val_recall: 0.8901
Epoch 3/50
100/100 [==============================] - 299s 3s/step - loss: 0.3592 -
accuracy: 0.9447 - auc: 0.9821 - precision: 0.9450 - recall: 0.9447 - val_loss:
0.3716 - val_accuracy: 0.9324 - val_auc: 0.9797 - val_precision: 0.9298 -
val_recall: 0.9324
Epoch 4/50
100/100 [==============================] - 302s 3s/step - loss: 0.3387 -
accuracy: 0.9466 - auc: 0.9844 - precision: 0.9466 - recall: 0.9466 - val_loss:
0.3940 - val_accuracy: 0.9127 - val_auc: 0.9763 - val_precision: 0.9129 -
val_recall: 0.9155
Epoch 5/50
100/100 [==============================] - 296s 3s/step - loss: 0.3332 -
accuracy: 0.9475 - auc: 0.9838 - precision: 0.9475 - recall: 0.9478 - val_loss:
0.3413 - val_accuracy: 0.9408 - val_auc: 0.9838 - val_precision: 0.9408 -
val_recall: 0.9408
Epoch 6/50
100/100 [==============================] - 295s 3s/step - loss: 0.3153 -
accuracy: 0.9519 - auc: 0.9861 - precision: 0.9516 - recall: 0.9519 - val_loss:
0.3414 - val_accuracy: 0.9352 - val_auc: 0.9824 - val_precision: 0.9352 -
val_recall: 0.9352
```

```
Epoch 7/50
100/100 [==============================] - 290s 3s/step - loss: 0.3008 -
accuracy: 0.9559 - auc: 0.9881 - precision: 0.9563 - recall: 0.9566 - val_loss:
0.3147 - val_accuracy: 0.9521 - val_auc: 0.9869 - val_precision: 0.9494 -
val_recall: 0.9521
Epoch 8/50
100/100 [==============================] - 299s 3s/step - loss: 0.3029 -
accuracy: 0.9525 - auc: 0.9872 - precision: 0.9522 - recall: 0.9528 - val_loss:
0.3235 - val_accuracy: 0.9465 - val_auc: 0.9845 - val_precision: 0.9466 -
val_recall: 0.9493
Epoch 9/50
100/100 [==============================] - 295s 3s/step - loss: 0.2968 -
accuracy: 0.9569 - auc: 0.9876 - precision: 0.9569 - recall: 0.9569 - val_loss:
0.3086 - val_accuracy: 0.9465 - val_auc: 0.9869 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 10/50
100/100 [==============================] - 308s 3s/step - loss: 0.2869 -
accuracy: 0.9553 - auc: 0.9884 - precision: 0.9562 - recall: 0.9550 - val_loss:
0.3248 - val_accuracy: 0.9493 - val_auc: 0.9838 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 11/50
100/100 [==============================] - 307s 3s/step - loss: 0.2845 -
accuracy: 0.9572 - auc: 0.9889 - precision: 0.9575 - recall: 0.9572 - val_loss:
0.3348 - val_accuracy: 0.9296 - val_auc: 0.9803 - val_precision: 0.9322 -
val_recall: 0.9296
Epoch 12/50
100/100 [==============================] - 304s 3s/step - loss: 0.2738 -
accuracy: 0.9594 - auc: 0.9900 - precision: 0.9585 - recall: 0.9591 - val_loss:
0.2905 - val_accuracy: 0.9437 - val_auc: 0.9890 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 13/50
100/100 [==============================] - 299s 3s/step - loss: 0.2785 -
accuracy: 0.9541 - auc: 0.9892 - precision: 0.9544 - recall: 0.9541 - val_loss:
0.3087 - val_accuracy: 0.9324 - val_auc: 0.9851 - val_precision: 0.9324 -
val_recall: 0.9324
Epoch 14/50
100/100 [==============================] - 303s 3s/step - loss: 0.2618 -
accuracy: 0.9609 - auc: 0.9912 - precision: 0.9609 - recall: 0.9609 - val_loss:
0.2941 - val_accuracy: 0.9493 - val_auc: 0.9879 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 15/50
100/100 [==============================] - 294s 3s/step - loss: 0.2678 -
accuracy: 0.9594 - auc: 0.9896 - precision: 0.9591 - recall: 0.9597 - val_loss:
0.3215 - val_accuracy: 0.9211 - val_auc: 0.9831 - val_precision: 0.9211 -
val_recall: 0.9211
Epoch 16/50
100/100 [==============================] - 297s 3s/step - loss: 0.2619 -
accuracy: 0.9572 - auc: 0.9908 - precision: 0.9572 - recall: 0.9572 - val_loss:
```

```
0.3454 - val_accuracy: 0.9239 - val_auc: 0.9762 - val_precision: 0.9239 -
val_recall: 0.9239
Epoch 17/50
100/100 [==============================] - 296s 3s/step - loss: 0.2586 -
accuracy: 0.9641 - auc: 0.9909 - precision: 0.9641 - recall: 0.9641 - val_loss:
0.3015 - val_accuracy: 0.9380 - val_auc: 0.9854 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 18/50
100/100 [==============================] - 295s 3s/step - loss: 0.2530 -
accuracy: 0.9631 - auc: 0.9919 - precision: 0.9631 - recall: 0.9628 - val_loss:
0.3146 - val_accuracy: 0.9380 - val_auc: 0.9849 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 19/50
100/100 [==============================] - 298s 3s/step - loss: 0.2511 -
accuracy: 0.9600 - auc: 0.9914 - precision: 0.9600 - recall: 0.9600 - val_loss:
0.3051 - val_accuracy: 0.9296 - val_auc: 0.9836 - val_precision: 0.9296 -
val_recall: 0.9296
Epoch 20/50
100/100 [==============================] - 291s 3s/step - loss: 0.2452 -
accuracy: 0.9641 - auc: 0.9926 - precision: 0.9641 - recall: 0.9641 - val_loss:
0.2878 - val_accuracy: 0.9521 - val_auc: 0.9888 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 21/50
100/100 [==============================] - 298s 3s/step - loss: 0.2508 -
accuracy: 0.9597 - auc: 0.9915 - precision: 0.9597 - recall: 0.9600 - val_loss:
0.2917 - val_accuracy: 0.9408 - val_auc: 0.9858 - val_precision: 0.9408 -
val_recall: 0.9408
Epoch 22/50
100/100 [==============================] - 300s 3s/step - loss: 0.2433 -
accuracy: 0.9638 - auc: 0.9921 - precision: 0.9632 - recall: 0.9641 - val_loss:
0.2887 - val_accuracy: 0.9380 - val_auc: 0.9874 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 23/50
100/100 [==============================] - 299s 3s/step - loss: 0.2427 -
accuracy: 0.9622 - auc: 0.9914 - precision: 0.9622 - recall: 0.9622 - val_loss:
0.3109 - val_accuracy: 0.9352 - val_auc: 0.9818 - val_precision: 0.9352 -
val_recall: 0.9352
Epoch 24/50
100/100 [==============================] - 284s 3s/step - loss: 0.2470 -
accuracy: 0.9616 - auc: 0.9911 - precision: 0.9616 - recall: 0.9613 - val_loss:
0.2460 - val_accuracy: 0.9577 - val_auc: 0.9928 - val_precision: 0.9577 -
val_recall: 0.9577
Epoch 25/50
100/100 [==============================] - 285s 3s/step - loss: 0.2374 -
accuracy: 0.9613 - auc: 0.9921 - precision: 0.9613 - recall: 0.9616 - val_loss:
0.2649 - val_accuracy: 0.9465 - val_auc: 0.9899 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 26/50
```

```
100/100 [==============================] - 283s 3s/step - loss: 0.2324 -
accuracy: 0.9625 - auc: 0.9929 - precision: 0.9628 - recall: 0.9634 - val_loss:
0.2767 - val_accuracy: 0.9352 - val_auc: 0.9882 - val_precision: 0.9350 -
val_recall: 0.9324
Epoch 27/50
100/100 [==============================] - 281s 3s/step - loss: 0.2233 -
accuracy: 0.9675 - auc: 0.9935 - precision: 0.9672 - recall: 0.9675 - val_loss:
0.2689 - val_accuracy: 0.9408 - val_auc: 0.9894 - val_precision: 0.9408 -
val_recall: 0.9408
Epoch 28/50
100/100 [==============================] - 284s 3s/step - loss: 0.2283 -
accuracy: 0.9659 - auc: 0.9928 - precision: 0.9662 - recall: 0.9656 - val_loss:
0.2708 - val_accuracy: 0.9521 - val_auc: 0.9870 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 29/50
100/100 [==============================] - 285s 3s/step - loss: 0.2249 -
accuracy: 0.9663 - auc: 0.9935 - precision: 0.9669 - recall: 0.9666 - val_loss:
0.2484 - val_accuracy: 0.9577 - val_auc: 0.9917 - val_precision: 0.9577 -
val_recall: 0.9577
Epoch 30/50
100/100 [==============================] - 286s 3s/step - loss: 0.2341 -
accuracy: 0.9613 - auc: 0.9919 - precision: 0.9622 - recall: 0.9613 - val_loss:
0.2487 - val_accuracy: 0.9549 - val_auc: 0.9911 - val_precision: 0.9549 -
val_recall: 0.9549
Epoch 31/50
100/100 [==============================] - 285s 3s/step - loss: 0.2276 -
accuracy: 0.9672 - auc: 0.9931 - precision: 0.9672 - recall: 0.9672 - val_loss:
0.2562 - val_accuracy: 0.9465 - val_auc: 0.9905 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 32/50
100/100 [==============================] - 283s 3s/step - loss: 0.2194 -
accuracy: 0.9641 - auc: 0.9940 - precision: 0.9641 - recall: 0.9641 - val_loss:
0.2529 - val_accuracy: 0.9380 - val_auc: 0.9908 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 33/50
100/100 [==============================] - 295s 3s/step - loss: 0.2171 -
accuracy: 0.9672 - auc: 0.9935 - precision: 0.9672 - recall: 0.9669 - val_loss:
0.2908 - val_accuracy: 0.9268 - val_auc: 0.9841 - val_precision: 0.9268 -
val_recall: 0.9268
Epoch 34/50
100/100 [==============================] - 298s 3s/step - loss: 0.2165 -
accuracy: 0.9666 - auc: 0.9942 - precision: 0.9663 - recall: 0.9666 - val_loss:
0.2580 - val_accuracy: 0.9437 - val_auc: 0.9898 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 35/50
100/100 [==============================] - 296s 3s/step - loss: 0.2188 -
accuracy: 0.9678 - auc: 0.9939 - precision: 0.9681 - recall: 0.9678 - val_loss:
0.2287 - val_accuracy: 0.9549 - val_auc: 0.9937 - val_precision: 0.9549 -
```

```
val_recall: 0.9549
Epoch 36/50
100/100 [==============================] - 291s 3s/step - loss: 0.2198 -
accuracy: 0.9650 - auc: 0.9937 - precision: 0.9653 - recall: 0.9650 - val_loss:
0.2404 - val_accuracy: 0.9549 - val_auc: 0.9923 - val_precision: 0.9522 -
val_recall: 0.9549
Epoch 37/50
100/100 [==============================] - 290s 3s/step - loss: 0.2165 -
accuracy: 0.9678 - auc: 0.9944 - precision: 0.9681 - recall: 0.9678 - val_loss:
0.3116 - val_accuracy: 0.9268 - val_auc: 0.9837 - val_precision: 0.9268 -
val_recall: 0.9268
Epoch 38/50
100/100 [==============================] - 283s 3s/step - loss: 0.2100 -
accuracy: 0.9691 - auc: 0.9950 - precision: 0.9691 - recall: 0.9691 - val_loss:
0.2366 - val_accuracy: 0.9521 - val_auc: 0.9926 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 39/50
100/100 [==============================] - 285s 3s/step - loss: 0.2104 -
accuracy: 0.9691 - auc: 0.9947 - precision: 0.9688 - recall: 0.9688 - val_loss:
0.2405 - val_accuracy: 0.9493 - val_auc: 0.9923 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 40/50
100/100 [==============================] - 292s 3s/step - loss: 0.2057 -
accuracy: 0.9694 - auc: 0.9953 - precision: 0.9694 - recall: 0.9697 - val_loss:
0.2908 - val_accuracy: 0.9408 - val_auc: 0.9835 - val_precision: 0.9408 -
val_recall: 0.9408
Epoch 41/50
100/100 [==============================] - 293s 3s/step - loss: 0.2077 -
accuracy: 0.9688 - auc: 0.9948 - precision: 0.9684 - recall: 0.9688 - val_loss:
0.2563 - val_accuracy: 0.9549 - val_auc: 0.9888 - val_precision: 0.9549 -
val_recall: 0.9549
Epoch 42/50
100/100 [==============================] - 292s 3s/step - loss: 0.2027 -
accuracy: 0.9684 - auc: 0.9957 - precision: 0.9691 - recall: 0.9688 - val_loss:
0.2443 - val_accuracy: 0.9521 - val_auc: 0.9911 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 43/50
100/100 [==============================] - 294s 3s/step - loss: 0.2095 -
accuracy: 0.9688 - auc: 0.9942 - precision: 0.9688 - recall: 0.9691 - val_loss:
0.2357 - val_accuracy: 0.9549 - val_auc: 0.9919 - val_precision: 0.9549 -
val_recall: 0.9549
Epoch 44/50
100/100 [==============================] - 297s 3s/step - loss: 0.2021 -
accuracy: 0.9712 - auc: 0.9952 - precision: 0.9713 - recall: 0.9716 - val_loss:
0.2470 - val_accuracy: 0.9521 - val_auc: 0.9912 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 45/50
100/100 [==============================] - 305s 3s/step - loss: 0.2041 -
```

```
accuracy: 0.9688 - auc: 0.9949 - precision: 0.9688 - recall: 0.9688 - val_loss:
0.2538 - val_accuracy: 0.9493 - val_auc: 0.9902 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 46/50
100/100 [==============================] - 301s 3s/step - loss: 0.2070 -
accuracy: 0.9712 - auc: 0.9941 - precision: 0.9706 - recall: 0.9712 - val_loss:
0.2430 - val_accuracy: 0.9521 - val_auc: 0.9918 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 47/50
100/100 [==============================] - 301s 3s/step - loss: 0.1982 -
accuracy: 0.9719 - auc: 0.9951 - precision: 0.9719 - recall: 0.9716 - val_loss:
0.2429 - val_accuracy: 0.9465 - val_auc: 0.9911 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 48/50
100/100 [==============================] - 305s 3s/step - loss: 0.2042 -
accuracy: 0.9703 - auc: 0.9945 - precision: 0.9703 - recall: 0.9700 - val_loss:
0.2299 - val_accuracy: 0.9521 - val_auc: 0.9927 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 49/50
100/100 [==============================] - 304s 3s/step - loss: 0.1975 -
accuracy: 0.9700 - auc: 0.9955 - precision: 0.9694 - recall: 0.9700 - val_loss:
0.2574 - val_accuracy: 0.9380 - val_auc: 0.9884 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 50/50
100/100 [==============================] - 298s 3s/step - loss: 0.1952 -
accuracy: 0.9709 - auc: 0.9960 - precision: 0.9706 - recall: 0.9706 - val_loss:
0.2542 - val_accuracy: 0.9493 - val_auc: 0.9899 - val_precision: 0.9493 -
val_recall: 0.9493
```

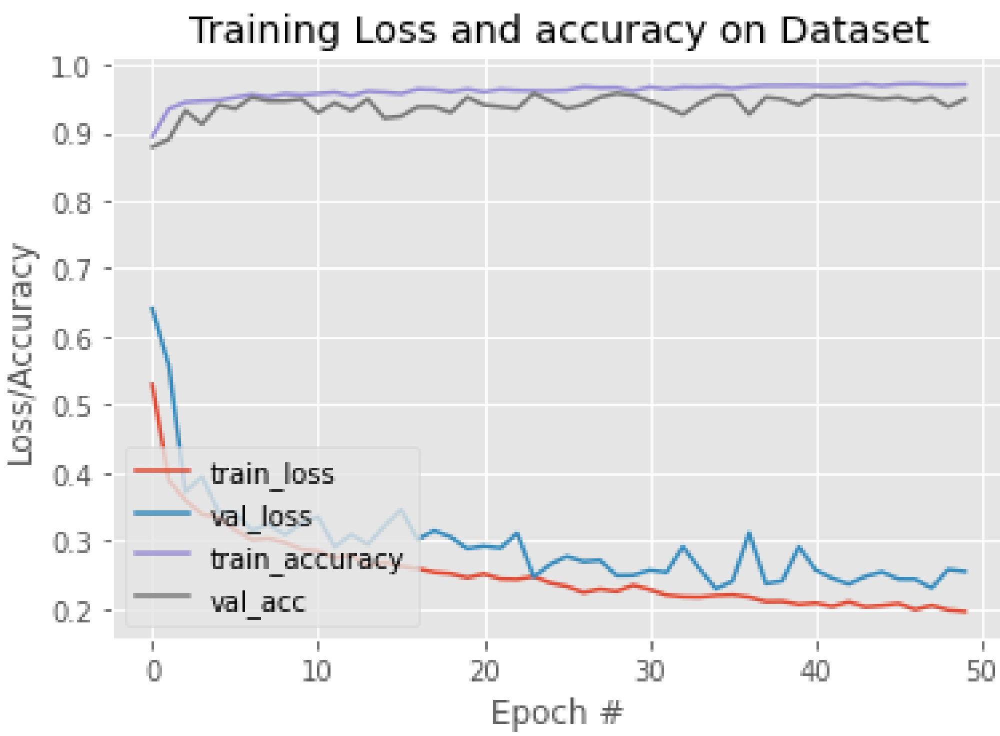
[25]:

```
import matplotlib.pyplot as plt

N = NUM_EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_accuracy")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig('Training Loss and accuracy on Dataset')
H.history.keys()
```

[25]:

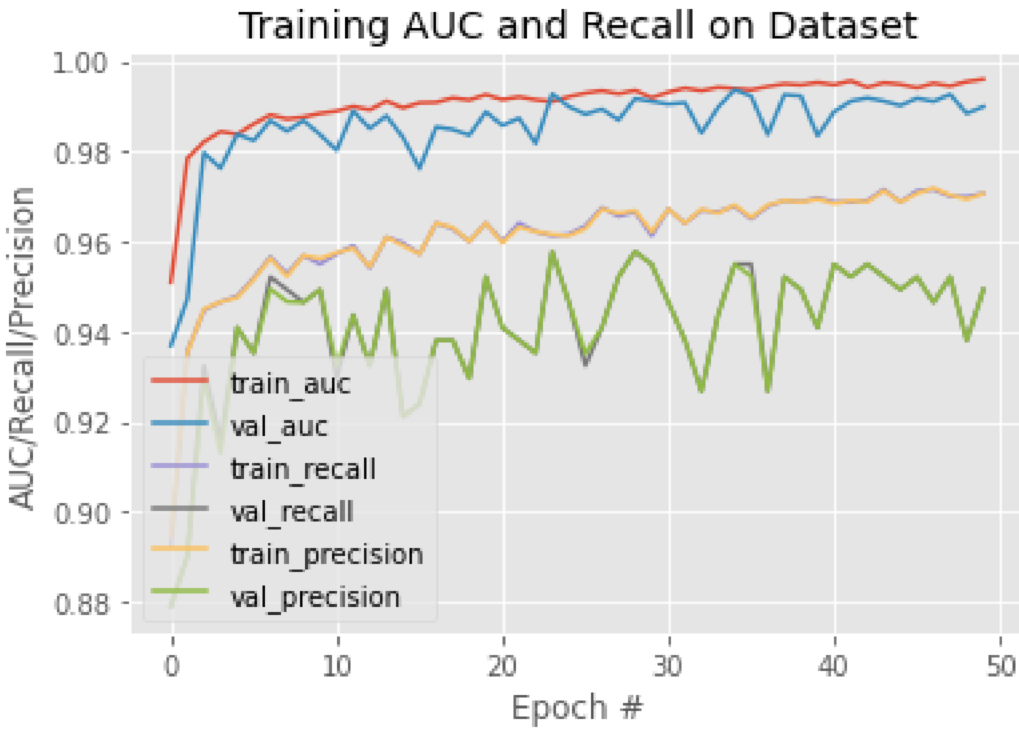
```
dict_keys([&#39;loss&#39;, &#39;accuracy&#39;, &#39;auc&#39;,
&#39;precision&#39;, &#39;recall&#39;, &#39;val_loss&#39;,
```

```
&#39;val_accuracy&#39;, &#39;val_auc&#39;, &#39;val_precision&#39;,
&#39;val_recall&#39;])
```



[27]:

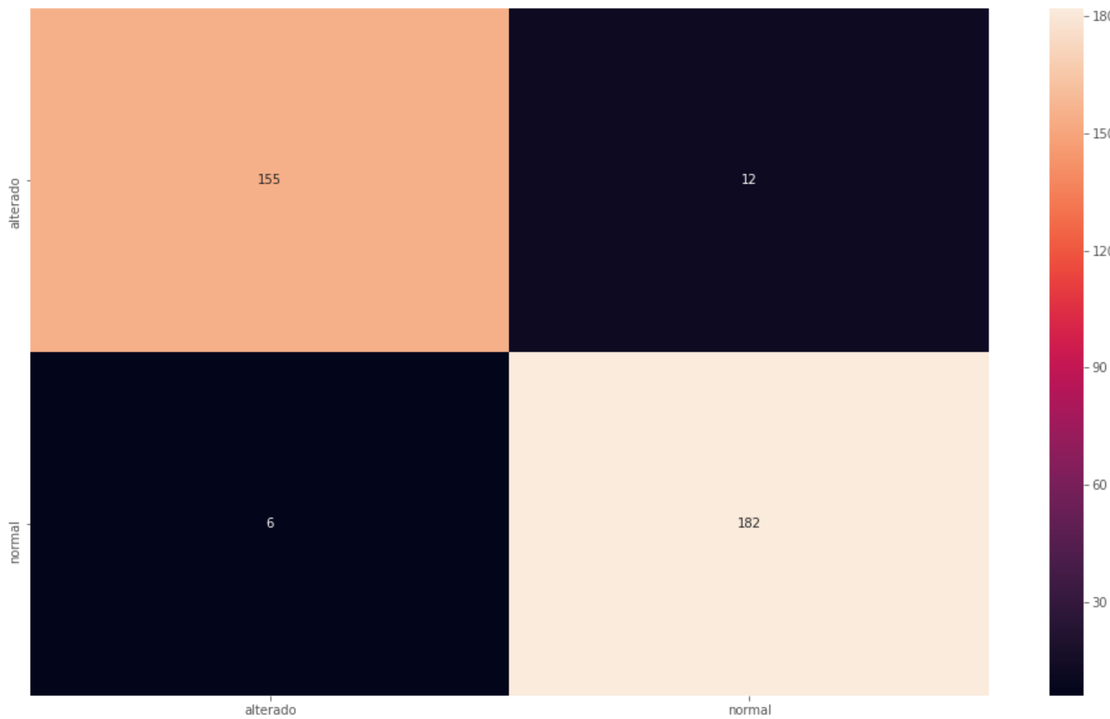
```python
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["auc"], label="train_auc")
plt.plot(np.arange(0, N), H.history["val_auc"], label="val_auc")
plt.plot(np.arange(0, N), H.history["recall"], label="train_recall")
plt.plot(np.arange(0, N), H.history["val_recall"], label="val_recall")
plt.plot(np.arange(0, N), H.history["precision"], label="train_precision")
plt.plot(np.arange(0, N), H.history["val_precision"], label="val_precision")

plt.title("Training AUC and Recall on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("AUC/Recall/Precision")
plt.legend(loc="lower left")
plt.savefig('Training AUC, Recall and Precision on Dataset')
```



[32]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sns

testGen.reset()
predIdxs = model.predict(testGen, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)


conf_mat = confusion_matrix(testGen.classes, predIdxs)

class_names = ['alterado', 'normal']
fig = plt.figure(figsize=(17,10))
df_cm = pd.DataFrame(conf_mat, index=class_names, columns=class_names)
heatmap = sns.heatmap(df_cm, annot=True, fmt='d')
heatmap

# show a nicely formatted classification report
print(classification_report(testGen.classes, predIdxs,
```

```
                    target_names=testGen.class_indices.keys()))
```

```
                   precision    recall  f1-score   support

rx-alterado-anonim       0.96      0.93      0.95       167
  rx-normal-anonim       0.94      0.97      0.95       188

          accuracy                           0.95       355
         macro avg       0.95      0.95      0.95       355
      weighted avg       0.95      0.95      0.95       355
```



[29]:
```
model.save('Models/H{}W{}.h5'.format(height, width))
```