Classify_imgs

October 8, 2020

[1]:
```
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]:
```
# 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)
```

```python
        # 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 = 128, 128
```

[4]:
```
# initialize the number of training epochs and batch size
NUM_EPOCHS = 50
BS = 64
```

```
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
50/50 [==============================] - 349s 7s/step - loss: 0.4235 - accuracy:
0.9109 - auc: 0.9595 - precision: 0.9078 - recall: 0.9050 - val_loss: 3.9052 -
val_accuracy: 0.5775 - val_auc: 0.6206 - val_precision: 0.5767 - val_recall:
0.5718
Epoch 2/50
50/50 [==============================] - 332s 7s/step - loss: 0.3448 - accuracy:
0.9397 - auc: 0.9765 - precision: 0.9400 - recall: 0.9400 - val_loss: 0.7864 -
val_accuracy: 0.7690 - val_auc: 0.8684 - val_precision: 0.7692 - val_recall:
0.7606
Epoch 3/50
50/50 [==============================] - 304s 6s/step - loss: 0.3119 - accuracy:
0.9475 - auc: 0.9838 - precision: 0.9480 - recall: 0.9459 - val_loss: 1.2012 -
val_accuracy: 0.5155 - val_auc: 0.7024 - val_precision: 0.5181 - val_recall:
0.5239
Epoch 4/50
50/50 [==============================] - 299s 6s/step - loss: 0.3086 - accuracy:
0.9466 - auc: 0.9837 - precision: 0.9466 - recall: 0.9478 - val_loss: 0.3472 -
val_accuracy: 0.9324 - val_auc: 0.9764 - val_precision: 0.9326 - val_recall:
0.9352
Epoch 5/50
50/50 [==============================] - 296s 6s/step - loss: 0.2862 - accuracy:
0.9600 - auc: 0.9872 - precision: 0.9603 - recall: 0.9600 - val_loss: 0.3204 -
val_accuracy: 0.9437 - val_auc: 0.9829 - val_precision: 0.9437 - val_recall:
0.9437
Epoch 6/50
50/50 [==============================] - 303s 6s/step - loss: 0.2692 - accuracy:
0.9594 - auc: 0.9906 - precision: 0.9593 - recall: 0.9588 - val_loss: 0.3208 -
val_accuracy: 0.9408 - val_auc: 0.9843 - val_precision: 0.9382 - val_recall:
0.9408
```

```
Epoch 7/50
50/50 [==============================] - 318s 6s/step - loss: 0.2658 - accuracy:
0.9634 - auc: 0.9899 - precision: 0.9631 - recall: 0.9628 - val_loss: 0.2762 -
val_accuracy: 0.9521 - val_auc: 0.9906 - val_precision: 0.9521 - val_recall:
0.9521
Epoch 8/50
50/50 [==============================] - 307s 6s/step - loss: 0.2613 - accuracy:
0.9616 - auc: 0.9910 - precision: 0.9616 - recall: 0.9616 - val_loss: 0.2792 -
val_accuracy: 0.9408 - val_auc: 0.9890 - val_precision: 0.9408 - val_recall:
0.9408
Epoch 9/50
50/50 [==============================] - 316s 6s/step - loss: 0.2554 - accuracy:
0.9603 - auc: 0.9915 - precision: 0.9597 - recall: 0.9600 - val_loss: 0.2823 -
val_accuracy: 0.9493 - val_auc: 0.9874 - val_precision: 0.9493 - val_recall:
0.9493
Epoch 10/50
50/50 [==============================] - 304s 6s/step - loss: 0.2593 - accuracy:
0.9594 - auc: 0.9910 - precision: 0.9587 - recall: 0.9584 - val_loss: 0.3196 -
val_accuracy: 0.9296 - val_auc: 0.9815 - val_precision: 0.9270 - val_recall:
0.9296
Epoch 11/50
50/50 [==============================] - 318s 6s/step - loss: 0.2458 - accuracy:
0.9609 - auc: 0.9928 - precision: 0.9612 - recall: 0.9609 - val_loss: 0.2903 -
val_accuracy: 0.9521 - val_auc: 0.9847 - val_precision: 0.9521 - val_recall:
0.9521
Epoch 12/50
50/50 [==============================] - 363s 7s/step - loss: 0.2407 - accuracy:
0.9659 - auc: 0.9933 - precision: 0.9653 - recall: 0.9659 - val_loss: 0.2879 -
val_accuracy: 0.9521 - val_auc: 0.9854 - val_precision: 0.9521 - val_recall:
0.9521
Epoch 13/50
50/50 [==============================] - 325s 6s/step - loss: 0.2303 - accuracy:
0.9672 - auc: 0.9944 - precision: 0.9669 - recall: 0.9672 - val_loss: 0.2903 -
val_accuracy: 0.9352 - val_auc: 0.9865 - val_precision: 0.9379 - val_recall:
0.9352
Epoch 14/50
50/50 [==============================] - 328s 7s/step - loss: 0.2438 - accuracy:
0.9616 - auc: 0.9921 - precision: 0.9618 - recall: 0.9609 - val_loss: 0.3624 -
val_accuracy: 0.9183 - val_auc: 0.9712 - val_precision: 0.9157 - val_recall:
0.9183
Epoch 15/50
50/50 [==============================] - 331s 7s/step - loss: 0.2480 - accuracy:
0.9591 - auc: 0.9921 - precision: 0.9588 - recall: 0.9591 - val_loss: 0.2478 -
val_accuracy: 0.9549 - val_auc: 0.9927 - val_precision: 0.9549 - val_recall:
0.9549
Epoch 16/50
50/50 [==============================] - 336s 7s/step - loss: 0.2352 - accuracy:
0.9638 - auc: 0.9937 - precision: 0.9638 - recall: 0.9638 - val_loss: 0.2433 -
```

```
val_accuracy: 0.9577 - val_auc: 0.9931 - val_precision: 0.9576 - val_recall:
0.9549
Epoch 17/50
50/50 [==============================] - 343s 7s/step - loss: 0.2324 - accuracy:
0.9644 - auc: 0.9932 - precision: 0.9644 - recall: 0.9647 - val_loss: 0.2767 -
val_accuracy: 0.9408 - val_auc: 0.9886 - val_precision: 0.9408 - val_recall:
0.9408
Epoch 18/50
50/50 [==============================] - 338s 7s/step - loss: 0.2272 - accuracy:
0.9644 - auc: 0.9940 - precision: 0.9644 - recall: 0.9644 - val_loss: 0.2635 -
val_accuracy: 0.9577 - val_auc: 0.9903 - val_precision: 0.9577 - val_recall:
0.9577
Epoch 19/50
50/50 [==============================] - 344s 7s/step - loss: 0.2301 - accuracy:
0.9666 - auc: 0.9930 - precision: 0.9663 - recall: 0.9666 - val_loss: 0.2599 -
val_accuracy: 0.9380 - val_auc: 0.9904 - val_precision: 0.9380 - val_recall:
0.9380
Epoch 20/50
50/50 [==============================] - 354s 7s/step - loss: 0.2189 - accuracy:
0.9688 - auc: 0.9949 - precision: 0.9688 - recall: 0.9688 - val_loss: 0.2387 -
val_accuracy: 0.9577 - val_auc: 0.9939 - val_precision: 0.9605 - val_recall:
0.9577
Epoch 21/50
50/50 [==============================] - 345s 7s/step - loss: 0.2187 - accuracy:
0.9678 - auc: 0.9949 - precision: 0.9681 - recall: 0.9675 - val_loss: 0.2388 -
val_accuracy: 0.9465 - val_auc: 0.9932 - val_precision: 0.9465 - val_recall:
0.9465
Epoch 22/50
50/50 [==============================] - 323s 6s/step - loss: 0.2115 - accuracy:
0.9694 - auc: 0.9955 - precision: 0.9691 - recall: 0.9700 - val_loss: 0.2375 -
val_accuracy: 0.9634 - val_auc: 0.9932 - val_precision: 0.9634 - val_recall:
0.9634
Epoch 23/50
50/50 [==============================] - 315s 6s/step - loss: 0.2143 - accuracy:
0.9712 - auc: 0.9956 - precision: 0.9709 - recall: 0.9709 - val_loss: 0.2600 -
val_accuracy: 0.9549 - val_auc: 0.9888 - val_precision: 0.9549 - val_recall:
0.9549
Epoch 24/50
50/50 [==============================] - 346s 7s/step - loss: 0.2089 - accuracy:
0.9697 - auc: 0.9958 - precision: 0.9703 - recall: 0.9697 - val_loss: 0.2490 -
val_accuracy: 0.9606 - val_auc: 0.9920 - val_precision: 0.9579 - val_recall:
0.9606
Epoch 25/50
50/50 [==============================] - 349s 7s/step - loss: 0.2024 - accuracy:
0.9728 - auc: 0.9964 - precision: 0.9725 - recall: 0.9728 - val_loss: 0.2410 -
val_accuracy: 0.9521 - val_auc: 0.9927 - val_precision: 0.9521 - val_recall:
0.9521
Epoch 26/50
```

```
50/50 [==============================] - 339s 7s/step - loss: 0.2078 - accuracy:
0.9684 - auc: 0.9960 - precision: 0.9684 - recall: 0.9688 - val_loss: 0.2595 -
val_accuracy: 0.9521 - val_auc: 0.9914 - val_precision: 0.9521 - val_recall:
0.9521
Epoch 27/50
50/50 [==============================] - 361s 7s/step - loss: 0.1985 - accuracy:
0.9728 - auc: 0.9965 - precision: 0.9728 - recall: 0.9725 - val_loss: 0.2606 -
val_accuracy: 0.9521 - val_auc: 0.9904 - val_precision: 0.9521 - val_recall:
0.9521
Epoch 28/50
50/50 [==============================] - 441s 9s/step - loss: 0.2067 - accuracy:
0.9709 - auc: 0.9958 - precision: 0.9710 - recall: 0.9716 - val_loss: 0.2312 -
val_accuracy: 0.9606 - val_auc: 0.9944 - val_precision: 0.9606 - val_recall:
0.9606
Epoch 29/50
50/50 [==============================] - 387s 8s/step - loss: 0.2086 - accuracy:
0.9684 - auc: 0.9953 - precision: 0.9688 - recall: 0.9691 - val_loss: 0.2315 -
val_accuracy: 0.9521 - val_auc: 0.9932 - val_precision: 0.9524 - val_recall:
0.9577
Epoch 30/50
50/50 [==============================] - 377s 8s/step - loss: 0.1987 - accuracy:
0.9725 - auc: 0.9968 - precision: 0.9725 - recall: 0.9725 - val_loss: 0.2687 -
val_accuracy: 0.9352 - val_auc: 0.9872 - val_precision: 0.9352 - val_recall:
0.9352
Epoch 31/50
50/50 [==============================] - 386s 8s/step - loss: 0.1914 - accuracy:
0.9753 - auc: 0.9973 - precision: 0.9756 - recall: 0.9756 - val_loss: 0.2482 -
val_accuracy: 0.9437 - val_auc: 0.9911 - val_precision: 0.9437 - val_recall:
0.9437
Epoch 32/50
50/50 [==============================] - 390s 8s/step - loss: 0.2024 - accuracy:
0.9703 - auc: 0.9961 - precision: 0.9706 - recall: 0.9706 - val_loss: 0.2417 -
val_accuracy: 0.9521 - val_auc: 0.9925 - val_precision: 0.9518 - val_recall:
0.9465
Epoch 33/50
50/50 [==============================] - 396s 8s/step - loss: 0.1929 - accuracy:
0.9747 - auc: 0.9970 - precision: 0.9744 - recall: 0.9747 - val_loss: 0.2772 -
val_accuracy: 0.9493 - val_auc: 0.9862 - val_precision: 0.9493 - val_recall:
0.9493
Epoch 34/50
50/50 [==============================] - 338s 7s/step - loss: 0.1948 - accuracy:
0.9750 - auc: 0.9967 - precision: 0.9753 - recall: 0.9753 - val_loss: 0.2378 -
val_accuracy: 0.9606 - val_auc: 0.9925 - val_precision: 0.9605 - val_recall:
0.9577
Epoch 35/50
50/50 [==============================] - 344s 7s/step - loss: 0.1904 - accuracy:
0.9750 - auc: 0.9969 - precision: 0.9753 - recall: 0.9750 - val_loss: 0.2197 -
val_accuracy: 0.9606 - val_auc: 0.9951 - val_precision: 0.9606 - val_recall:
```

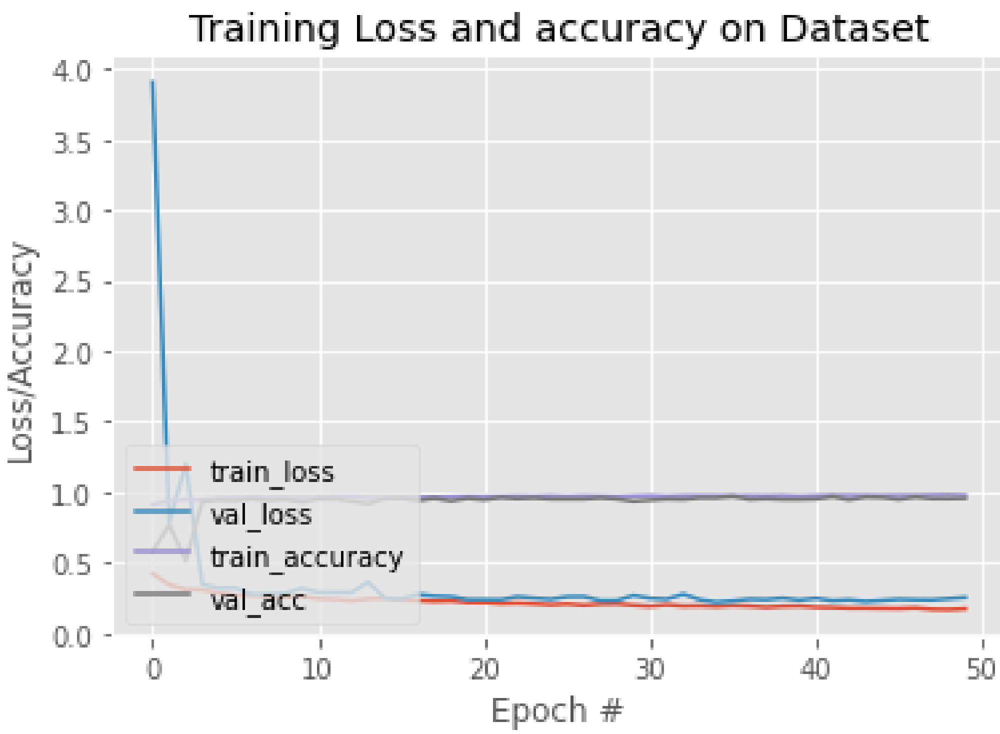
```
0.9606
Epoch 36/50
50/50 [==============================] - 295s 6s/step - loss: 0.1965 - accuracy:
0.9722 - auc: 0.9963 - precision: 0.9728 - recall: 0.9722 - val_loss: 0.2292 -
val_accuracy: 0.9718 - val_auc: 0.9930 - val_precision: 0.9718 - val_recall:
0.9718
Epoch 37/50
50/50 [==============================] - 271s 5s/step - loss: 0.1928 - accuracy:
0.9744 - auc: 0.9964 - precision: 0.9744 - recall: 0.9750 - val_loss: 0.2420 -
val_accuracy: 0.9493 - val_auc: 0.9901 - val_precision: 0.9496 - val_recall:
0.9549
Epoch 38/50
50/50 [==============================] - 264s 5s/step - loss: 0.1847 - accuracy:
0.9734 - auc: 0.9976 - precision: 0.9740 - recall: 0.9734 - val_loss: 0.2390 -
val_accuracy: 0.9549 - val_auc: 0.9926 - val_precision: 0.9549 - val_recall:
0.9549
Epoch 39/50
50/50 [==============================] - 272s 5s/step - loss: 0.1911 - accuracy:
0.9756 - auc: 0.9969 - precision: 0.9753 - recall: 0.9756 - val_loss: 0.2492 -
val_accuracy: 0.9493 - val_auc: 0.9912 - val_precision: 0.9493 - val_recall:
0.9493
Epoch 40/50
50/50 [==============================] - 266s 5s/step - loss: 0.1936 - accuracy:
0.9709 - auc: 0.9967 - precision: 0.9709 - recall: 0.9706 - val_loss: 0.2353 -
val_accuracy: 0.9493 - val_auc: 0.9926 - val_precision: 0.9493 - val_recall:
0.9493
Epoch 41/50
50/50 [==============================] - 270s 5s/step - loss: 0.1849 - accuracy:
0.9747 - auc: 0.9975 - precision: 0.9753 - recall: 0.9747 - val_loss: 0.2481 -
val_accuracy: 0.9521 - val_auc: 0.9906 - val_precision: 0.9521 - val_recall:
0.9521
Epoch 42/50
50/50 [==============================] - 264s 5s/step - loss: 0.1828 - accuracy:
0.9781 - auc: 0.9969 - precision: 0.9781 - recall: 0.9781 - val_loss: 0.2307 -
val_accuracy: 0.9690 - val_auc: 0.9929 - val_precision: 0.9663 - val_recall:
0.9690
Epoch 43/50
50/50 [==============================] - 267s 5s/step - loss: 0.1781 - accuracy:
0.9778 - auc: 0.9979 - precision: 0.9781 - recall: 0.9778 - val_loss: 0.2369 -
val_accuracy: 0.9465 - val_auc: 0.9897 - val_precision: 0.9438 - val_recall:
0.9465
Epoch 44/50
50/50 [==============================] - 266s 5s/step - loss: 0.1783 - accuracy:
0.9766 - auc: 0.9979 - precision: 0.9769 - recall: 0.9766 - val_loss: 0.2246 -
val_accuracy: 0.9662 - val_auc: 0.9942 - val_precision: 0.9662 - val_recall:
0.9662
Epoch 45/50
50/50 [==============================] - 268s 5s/step - loss: 0.1762 - accuracy:
```

```
0.9756 - auc: 0.9980 - precision: 0.9756 - recall: 0.9762 - val_loss: 0.2327 -
val_accuracy: 0.9634 - val_auc: 0.9924 - val_precision: 0.9634 - val_recall:
0.9634
Epoch 46/50
50/50 [==============================] - 266s 5s/step - loss: 0.1753 - accuracy:
0.9778 - auc: 0.9976 - precision: 0.9781 - recall: 0.9778 - val_loss: 0.2410 -
val_accuracy: 0.9493 - val_auc: 0.9925 - val_precision: 0.9493 - val_recall:
0.9493
Epoch 47/50
50/50 [==============================] - 264s 5s/step - loss: 0.1798 - accuracy:
0.9769 - auc: 0.9975 - precision: 0.9766 - recall: 0.9769 - val_loss: 0.2376 -
val_accuracy: 0.9662 - val_auc: 0.9918 - val_precision: 0.9662 - val_recall:
0.9662
Epoch 48/50
50/50 [==============================] - 275s 6s/step - loss: 0.1698 - accuracy:
0.9797 - auc: 0.9983 - precision: 0.9800 - recall: 0.9797 - val_loss: 0.2364 -
val_accuracy: 0.9549 - val_auc: 0.9920 - val_precision: 0.9549 - val_recall:
0.9549
Epoch 49/50
50/50 [==============================] - 279s 6s/step - loss: 0.1677 - accuracy:
0.9812 - auc: 0.9984 - precision: 0.9813 - recall: 0.9819 - val_loss: 0.2453 -
val_accuracy: 0.9549 - val_auc: 0.9921 - val_precision: 0.9549 - val_recall:
0.9549
Epoch 50/50
50/50 [==============================] - 272s 5s/step - loss: 0.1738 - accuracy:
0.9781 - auc: 0.9981 - precision: 0.9778 - recall: 0.9784 - val_loss: 0.2543 -
val_accuracy: 0.9577 - val_auc: 0.9883 - val_precision: 0.9577 - val_recall:
0.9577
```

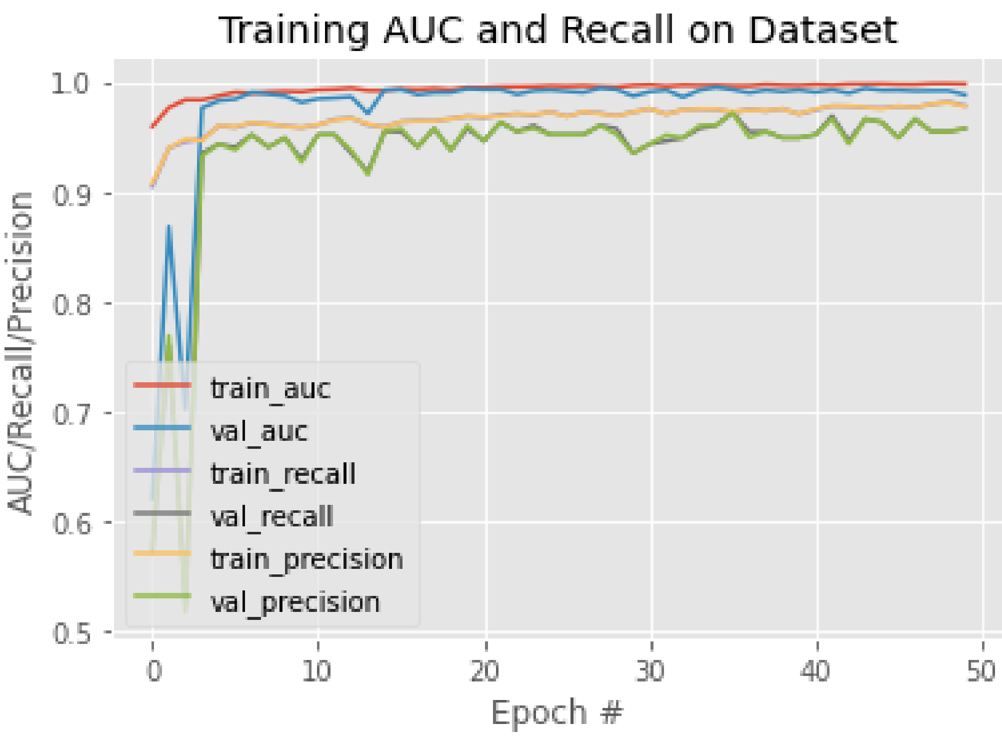
[12]:

```python
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()
```

[12]: dict_keys(['loss', 'accuracy', 'auc', 'precision', 'recall', 'val_loss', 'val_accuracy', 'val_auc', 'val_precision', 'val_recall'])



[13]:
```
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')
```

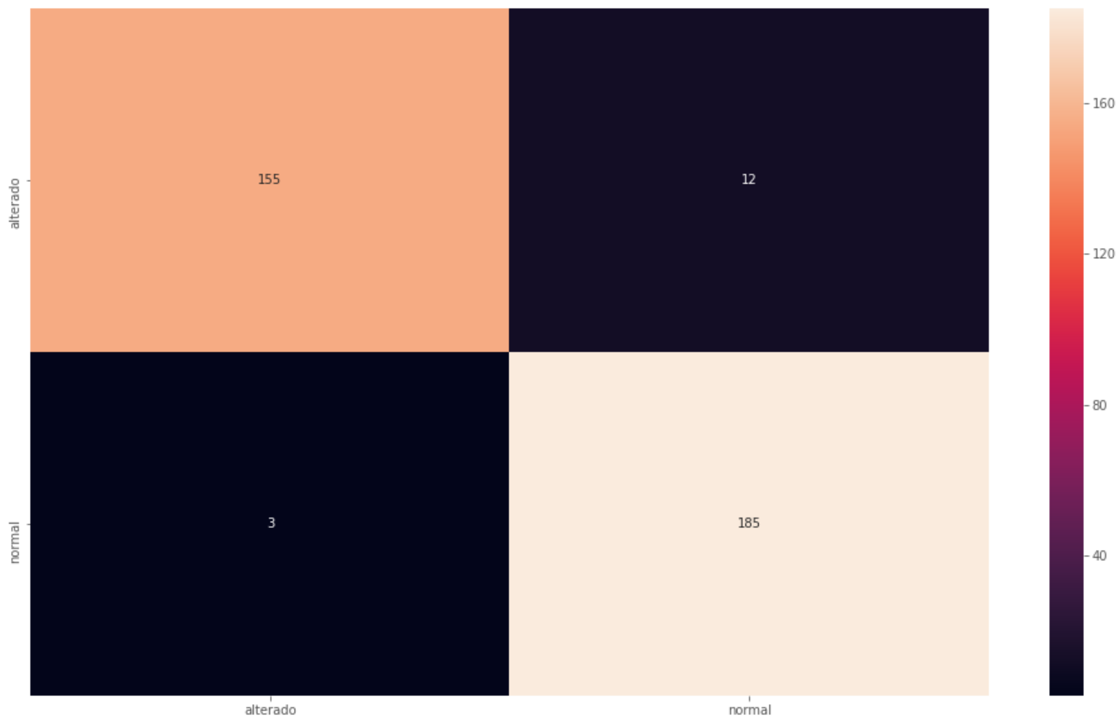


[14]:

```
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       0.98      0.93      0.95       167
  rx-normal-anonim       0.94      0.98      0.96       188

          accuracy                           0.96       355
         macro avg       0.96      0.96      0.96       355
      weighted avg       0.96      0.96      0.96       355
```



[15]:
```
model.save('Models/H{}W{}.h5'.format(height, width))
```