Classify_imgs

October 7, 2020

[1]:
```
from tensorflow import keras
from imutils import paths
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
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 = 512, 512
```

[4]:
```
# initialize the number of training epochs and batch size
NUM_EPOCHS = 50
BS = 8
```

```
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]:
```
model.compile(loss="binary_crossentropy",
              optimizer='Adam',
              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
400/400 [==============================] - 421s 1s/step - loss: 0.4535 -
accuracy: 0.8950 - auc: 0.9547 - precision: 0.8965 - recall: 0.8931 - val_loss:
0.4838 - val_accuracy: 0.8817 - val_auc: 0.9352 - val_precision: 0.8817 -
val_recall: 0.8817
Epoch 2/50
400/400 [==============================] - 417s 1s/step - loss: 0.3393 -
accuracy: 0.9287 - auc: 0.9752 - precision: 0.9299 - recall: 0.9287 - val_loss:
0.3122 - val_accuracy: 0.9380 - val_auc: 0.9794 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 3/50
400/400 [==============================] - 417s 1s/step - loss: 0.2968 -
accuracy: 0.9422 - auc: 0.9802 - precision: 0.9416 - recall: 0.9422 - val_loss:
0.4291 - val_accuracy: 0.8845 - val_auc: 0.9577 - val_precision: 0.8792 -
val_recall: 0.8817
Epoch 4/50
400/400 [==============================] - 417s 1s/step - loss: 0.2698 -
accuracy: 0.9444 - auc: 0.9827 - precision: 0.9441 - recall: 0.9453 - val_loss:
0.2743 - val_accuracy: 0.9380 - val_auc: 0.9781 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 5/50
400/400 [==============================] - 415s 1s/step - loss: 0.2703 -
accuracy: 0.9359 - auc: 0.9794 - precision: 0.9356 - recall: 0.9359 - val_loss:
0.2427 - val_accuracy: 0.9408 - val_auc: 0.9845 - val_precision: 0.9382 -
val_recall: 0.9408
Epoch 6/50
400/400 [==============================] - 414s 1s/step - loss: 0.2382 -
accuracy: 0.9444 - auc: 0.9836 - precision: 0.9441 - recall: 0.9444 - val_loss:
0.3909 - val_accuracy: 0.8845 - val_auc: 0.9462 - val_precision: 0.8845 -
val_recall: 0.8845
Epoch 7/50
```

```
400/400 [==============================] - 418s 1s/step - loss: 0.2309 -
accuracy: 0.9466 - auc: 0.9826 - precision: 0.9471 - recall: 0.9450 - val_loss:
0.2590 - val_accuracy: 0.9239 - val_auc: 0.9818 - val_precision: 0.9239 -
val_recall: 0.9239
Epoch 8/50
400/400 [==============================] - 418s 1s/step - loss: 0.2153 -
accuracy: 0.9466 - auc: 0.9838 - precision: 0.9480 - recall: 0.9466 - val_loss:
0.2608 - val_accuracy: 0.9268 - val_auc: 0.9817 - val_precision: 0.9268 -
val_recall: 0.9268
Epoch 9/50
400/400 [==============================] - 418s 1s/step - loss: 0.2066 -
accuracy: 0.9522 - auc: 0.9844 - precision: 0.9516 - recall: 0.9519 - val_loss:
0.2177 - val_accuracy: 0.9437 - val_auc: 0.9850 - val_precision: 0.9438 -
val_recall: 0.9465
Epoch 10/50
400/400 [==============================] - 426s 1s/step - loss: 0.1944 -
accuracy: 0.9488 - auc: 0.9860 - precision: 0.9488 - recall: 0.9500 - val_loss:
0.2018 - val_accuracy: 0.9352 - val_auc: 0.9866 - val_precision: 0.9352 -
val_recall: 0.9352
Epoch 11/50
400/400 [==============================] - 420s 1s/step - loss: 0.1975 -
accuracy: 0.9463 - auc: 0.9849 - precision: 0.9463 - recall: 0.9466 - val_loss:
0.5527 - val_accuracy: 0.8000 - val_auc: 0.8937 - val_precision: 0.8000 -
val_recall: 0.8000
Epoch 12/50
400/400 [==============================] - 463s 1s/step - loss: 0.1877 -
accuracy: 0.9513 - auc: 0.9862 - precision: 0.9521 - recall: 0.9513 - val_loss:
0.1944 - val_accuracy: 0.9437 - val_auc: 0.9855 - val_precision: 0.9438 -
val_recall: 0.9465
Epoch 13/50
400/400 [==============================] - 542s 1s/step - loss: 0.1855 -
accuracy: 0.9500 - auc: 0.9858 - precision: 0.9503 - recall: 0.9500 - val_loss:
0.2569 - val_accuracy: 0.9183 - val_auc: 0.9701 - val_precision: 0.9209 -
val_recall: 0.9183
Epoch 14/50
400/400 [==============================] - 456s 1s/step - loss: 0.1879 -
accuracy: 0.9506 - auc: 0.9850 - precision: 0.9500 - recall: 0.9506 - val_loss:
0.2336 - val_accuracy: 0.9239 - val_auc: 0.9769 - val_precision: 0.9239 -
val_recall: 0.9239
Epoch 15/50
400/400 [==============================] - 419s 1s/step - loss: 0.1698 -
accuracy: 0.9494 - auc: 0.9881 - precision: 0.9493 - recall: 0.9488 - val_loss:
0.2713 - val_accuracy: 0.8986 - val_auc: 0.9681 - val_precision: 0.8986 -
val_recall: 0.8986
Epoch 16/50
400/400 [==============================] - 419s 1s/step - loss: 0.1771 -
accuracy: 0.9566 - auc: 0.9858 - precision: 0.9566 - recall: 0.9572 - val_loss:
0.2053 - val_accuracy: 0.9465 - val_auc: 0.9812 - val_precision: 0.9465 -
```

```
val_recall: 0.9465
Epoch 17/50
400/400 [==============================] - 415s 1s/step - loss: 0.1737 -
accuracy: 0.9559 - auc: 0.9865 - precision: 0.9559 - recall: 0.9556 - val_loss:
0.1708 - val_accuracy: 0.9521 - val_auc: 0.9882 - val_precision: 0.9494 -
val_recall: 0.9521
Epoch 18/50
400/400 [==============================] - 413s 1s/step - loss: 0.1608 -
accuracy: 0.9547 - auc: 0.9884 - precision: 0.9556 - recall: 0.9553 - val_loss:
0.1945 - val_accuracy: 0.9493 - val_auc: 0.9863 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 19/50
400/400 [==============================] - 415s 1s/step - loss: 0.1666 -
accuracy: 0.9544 - auc: 0.9878 - precision: 0.9547 - recall: 0.9544 - val_loss:
0.1823 - val_accuracy: 0.9465 - val_auc: 0.9855 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 20/50
400/400 [==============================] - 402s 1s/step - loss: 0.1793 -
accuracy: 0.9519 - auc: 0.9848 - precision: 0.9519 - recall: 0.9519 - val_loss:
0.1972 - val_accuracy: 0.9465 - val_auc: 0.9841 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 21/50
400/400 [==============================] - 400s 999ms/step - loss: 0.1765 -
accuracy: 0.9522 - auc: 0.9861 - precision: 0.9525 - recall: 0.9519 - val_loss:
0.1893 - val_accuracy: 0.9437 - val_auc: 0.9837 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 22/50
400/400 [==============================] - 402s 1s/step - loss: 0.1679 -
accuracy: 0.9538 - auc: 0.9873 - precision: 0.9537 - recall: 0.9534 - val_loss:
0.1932 - val_accuracy: 0.9437 - val_auc: 0.9838 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 23/50
400/400 [==============================] - 412s 1s/step - loss: 0.1626 -
accuracy: 0.9547 - auc: 0.9881 - precision: 0.9547 - recall: 0.9550 - val_loss:
0.1906 - val_accuracy: 0.9465 - val_auc: 0.9832 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 24/50
400/400 [==============================] - 422s 1s/step - loss: 0.1622 -
accuracy: 0.9569 - auc: 0.9878 - precision: 0.9566 - recall: 0.9572 - val_loss:
0.1838 - val_accuracy: 0.9493 - val_auc: 0.9846 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 25/50
400/400 [==============================] - 422s 1s/step - loss: 0.1520 -
accuracy: 0.9566 - auc: 0.9896 - precision: 0.9566 - recall: 0.9566 - val_loss:
0.2166 - val_accuracy: 0.9380 - val_auc: 0.9788 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 26/50
400/400 [==============================] - 414s 1s/step - loss: 0.1608 -
```

```
accuracy: 0.9534 - auc: 0.9887 - precision: 0.9532 - recall: 0.9538 - val_loss:
0.1935 - val_accuracy: 0.9465 - val_auc: 0.9840 - val_precision: 0.9492 -
val_recall: 0.9465
Epoch 27/50
400/400 [==============================] - 415s 1s/step - loss: 0.1620 -
accuracy: 0.9516 - auc: 0.9883 - precision: 0.9516 - recall: 0.9516 - val_loss:
0.2545 - val_accuracy: 0.9239 - val_auc: 0.9762 - val_precision: 0.9239 -
val_recall: 0.9239
Epoch 28/50
400/400 [==============================] - 416s 1s/step - loss: 0.1618 -
accuracy: 0.9556 - auc: 0.9879 - precision: 0.9556 - recall: 0.9553 - val_loss:
0.1885 - val_accuracy: 0.9493 - val_auc: 0.9849 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 29/50
400/400 [==============================] - 418s 1s/step - loss: 0.1548 -
accuracy: 0.9569 - auc: 0.9887 - precision: 0.9569 - recall: 0.9566 - val_loss:
0.1616 - val_accuracy: 0.9521 - val_auc: 0.9893 - val_precision: 0.9521 -
val_recall: 0.9521
Epoch 30/50
400/400 [==============================] - 408s 1s/step - loss: 0.1564 -
accuracy: 0.9559 - auc: 0.9886 - precision: 0.9557 - recall: 0.9563 - val_loss:
0.2085 - val_accuracy: 0.9268 - val_auc: 0.9824 - val_precision: 0.9266 -
val_recall: 0.9239
Epoch 31/50
400/400 [==============================] - 415s 1s/step - loss: 0.1622 -
accuracy: 0.9581 - auc: 0.9881 - precision: 0.9581 - recall: 0.9578 - val_loss:
0.1977 - val_accuracy: 0.9465 - val_auc: 0.9840 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 32/50
400/400 [==============================] - 417s 1s/step - loss: 0.1636 -
accuracy: 0.9544 - auc: 0.9878 - precision: 0.9544 - recall: 0.9544 - val_loss:
0.2542 - val_accuracy: 0.9155 - val_auc: 0.9695 - val_precision: 0.9155 -
val_recall: 0.9155
Epoch 33/50
400/400 [==============================] - 414s 1s/step - loss: 0.1613 -
accuracy: 0.9559 - auc: 0.9877 - precision: 0.9559 - recall: 0.9559 - val_loss:
0.5106 - val_accuracy: 0.8901 - val_auc: 0.9329 - val_precision: 0.8904 -
val_recall: 0.8930
Epoch 34/50
400/400 [==============================] - 415s 1s/step - loss: 0.1581 -
accuracy: 0.9594 - auc: 0.9886 - precision: 0.9594 - recall: 0.9594 - val_loss:
0.1927 - val_accuracy: 0.9465 - val_auc: 0.9815 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 35/50
400/400 [==============================] - 415s 1s/step - loss: 0.1537 -
accuracy: 0.9569 - auc: 0.9897 - precision: 0.9569 - recall: 0.9566 - val_loss:
0.1812 - val_accuracy: 0.9493 - val_auc: 0.9841 - val_precision: 0.9493 -
val_recall: 0.9493
```

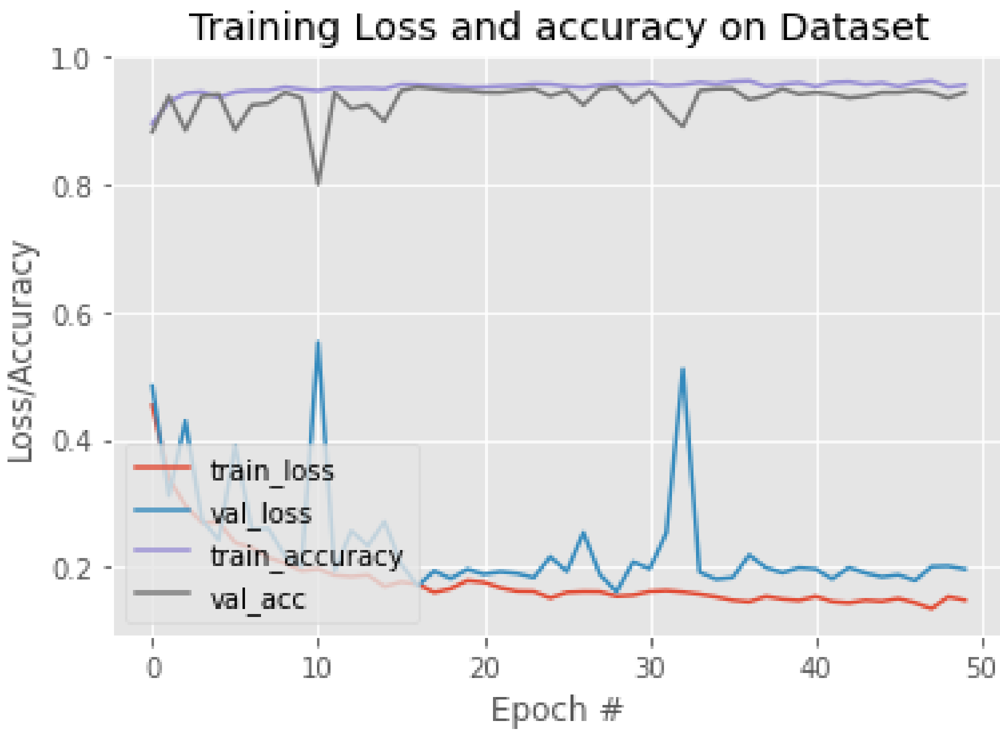
```
Epoch 36/50
400/400 [==============================] - 409s 1s/step - loss: 0.1483 -
accuracy: 0.9606 - auc: 0.9899 - precision: 0.9606 - recall: 0.9600 - val_loss:
0.1841 - val_accuracy: 0.9493 - val_auc: 0.9861 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 37/50
400/400 [==============================] - 411s 1s/step - loss: 0.1459 -
accuracy: 0.9619 - auc: 0.9903 - precision: 0.9619 - recall: 0.9622 - val_loss:
0.2193 - val_accuracy: 0.9324 - val_auc: 0.9735 - val_precision: 0.9324 -
val_recall: 0.9324
Epoch 38/50
400/400 [==============================] - 416s 1s/step - loss: 0.1546 -
accuracy: 0.9531 - auc: 0.9891 - precision: 0.9534 - recall: 0.9531 - val_loss:
0.1995 - val_accuracy: 0.9380 - val_auc: 0.9811 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 39/50
400/400 [==============================] - 411s 1s/step - loss: 0.1503 -
accuracy: 0.9566 - auc: 0.9886 - precision: 0.9566 - recall: 0.9566 - val_loss:
0.1916 - val_accuracy: 0.9493 - val_auc: 0.9826 - val_precision: 0.9493 -
val_recall: 0.9493
Epoch 40/50
400/400 [==============================] - 414s 1s/step - loss: 0.1480 -
accuracy: 0.9588 - auc: 0.9886 - precision: 0.9590 - recall: 0.9588 - val_loss:
0.1993 - val_accuracy: 0.9408 - val_auc: 0.9803 - val_precision: 0.9408 -
val_recall: 0.9408
Epoch 41/50
400/400 [==============================] - 412s 1s/step - loss: 0.1545 -
accuracy: 0.9531 - auc: 0.9885 - precision: 0.9531 - recall: 0.9531 - val_loss:
0.1973 - val_accuracy: 0.9437 - val_auc: 0.9835 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 42/50
400/400 [==============================] - 412s 1s/step - loss: 0.1463 -
accuracy: 0.9591 - auc: 0.9898 - precision: 0.9591 - recall: 0.9591 - val_loss:
0.1818 - val_accuracy: 0.9408 - val_auc: 0.9838 - val_precision: 0.9408 -
val_recall: 0.9408
Epoch 43/50
400/400 [==============================] - 411s 1s/step - loss: 0.1442 -
accuracy: 0.9603 - auc: 0.9894 - precision: 0.9603 - recall: 0.9606 - val_loss:
0.1995 - val_accuracy: 0.9352 - val_auc: 0.9838 - val_precision: 0.9352 -
val_recall: 0.9352
Epoch 44/50
400/400 [==============================] - 411s 1s/step - loss: 0.1483 -
accuracy: 0.9563 - auc: 0.9893 - precision: 0.9563 - recall: 0.9563 - val_loss:
0.1912 - val_accuracy: 0.9380 - val_auc: 0.9821 - val_precision: 0.9380 -
val_recall: 0.9380
Epoch 45/50
400/400 [==============================] - 429s 1s/step - loss: 0.1466 -
accuracy: 0.9588 - auc: 0.9879 - precision: 0.9588 - recall: 0.9588 - val_loss:
```

```
0.1849 - val_accuracy: 0.9437 - val_auc: 0.9847 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 46/50
400/400 [==============================] - 411s 1s/step - loss: 0.1512 -
accuracy: 0.9531 - auc: 0.9892 - precision: 0.9531 - recall: 0.9531 - val_loss:
0.1883 - val_accuracy: 0.9437 - val_auc: 0.9828 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 47/50
400/400 [==============================] - 405s 1s/step - loss: 0.1444 -
accuracy: 0.9588 - auc: 0.9901 - precision: 0.9588 - recall: 0.9588 - val_loss:
0.1792 - val_accuracy: 0.9465 - val_auc: 0.9852 - val_precision: 0.9465 -
val_recall: 0.9465
Epoch 48/50
400/400 [==============================] - 402s 1s/step - loss: 0.1353 -
accuracy: 0.9619 - auc: 0.9914 - precision: 0.9622 - recall: 0.9619 - val_loss:
0.2012 - val_accuracy: 0.9437 - val_auc: 0.9827 - val_precision: 0.9437 -
val_recall: 0.9437
Epoch 49/50
400/400 [==============================] - 405s 1s/step - loss: 0.1539 -
accuracy: 0.9522 - auc: 0.9888 - precision: 0.9522 - recall: 0.9522 - val_loss:
0.2019 - val_accuracy: 0.9352 - val_auc: 0.9798 - val_precision: 0.9352 -
val_recall: 0.9352
Epoch 50/50
400/400 [==============================] - 402s 1s/step - loss: 0.1486 -
accuracy: 0.9556 - auc: 0.9889 - precision: 0.9556 - recall: 0.9559 - val_loss:
0.1971 - val_accuracy: 0.9437 - val_auc: 0.9829 - val_precision: 0.9437 -
val_recall: 0.9437
```

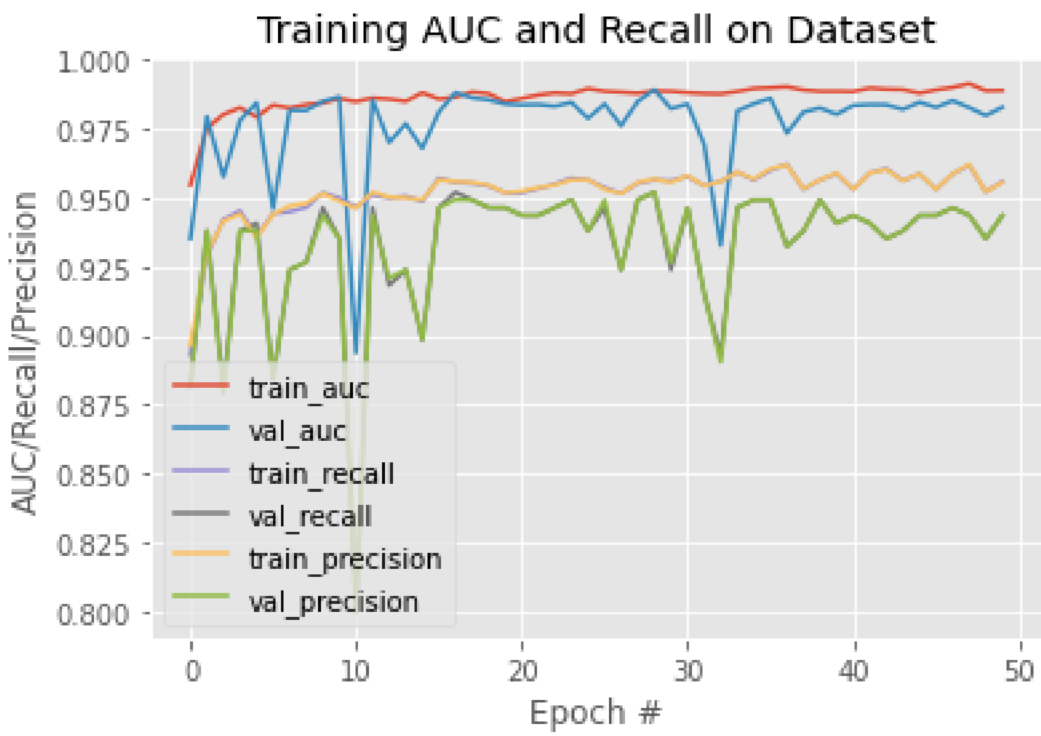
[12]:
```python
import matplotlib.pyplot as plt

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([&#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;])
```



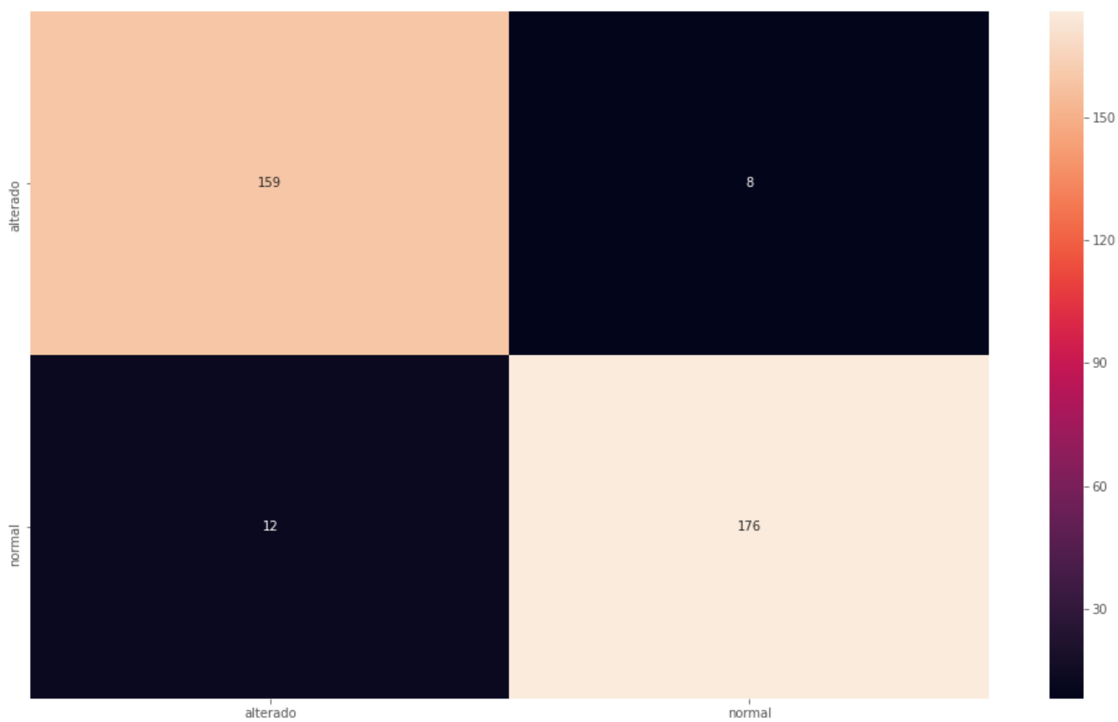
[13]:
```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["auc"], label="train_auc")
plt.plot(np.arange(0, N), H.history["val_auc"], label="val_auc")
plt.plot(np.arange(0, N), H.history["recall"], label="train_recall")
plt.plot(np.arange(0, N), H.history["val_recall"], label="val_recall")
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]:
```python
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       0.93      0.95      0.94       167
  rx-normal-anonim       0.96      0.94      0.95       188

          accuracy                           0.94       355
         macro avg       0.94      0.94      0.94       355
      weighted avg       0.94      0.94      0.94       355
```



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