differentiating-buildings-from-food-cnn

December 23, 2022

[1]:
```python
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load
from fastai.imports import *
from fastai.vision.all import *
import shutil
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras import regularizers
import keras
from tensorflow.keras import *


# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

1 differentiating buildings from food with a CNN

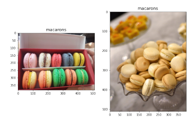
1.1 preparing our dataset

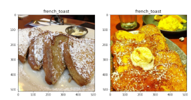
in this notebook we use a subset of the Food-101 dataset and the House Rooms & Streets Image Dataset

first look at some sample images from our datasets

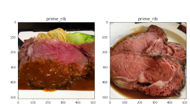
[2]:
```
food_path=Path("../input/food41/images")
building_path=Path("../input/house-rooms-streets-image-dataset/
↪kaggle_room_street_data")
```

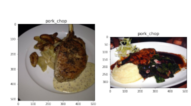
[3]:
```
fig, ax = plt.subplots(figsize=(10,128),nrows=6, ncols=2, ) #make a figure to␣
↪plot
for i,category in enumerate(os.listdir(food_path)): #loop over image categories
    for j, img in enumerate(os.listdir(os.path.join(food_path,category))): #␣
↪loop over images in each category
        ax[i,j].imshow(PILImage.create(os.path.
↪join(food_path,category,img)),label=category) #plot image
        ax[i,j].set_title(category,fontsize = 14)
        if(j==1):break
    if i==5:break
```


macarons
macarons


french_toast
french_toast


lobster_bisque
lobster_bisque


prime_rib
prime_rib


pork_chop
pork_chop

guacamole
guacamole

these were some food images, now let's look at some non-food images

[4]:
```python
fig, ax = plt.subplots(figsize=(10,10),nrows=2, ncols=3, ) #make a figure to plot
for i,category in enumerate(os.listdir(building_path)): #loop over street data categories
    for j, img in enumerate(os.listdir(os.path.join(building_path,category))): # loop over images in each category
        ax[i,j].imshow(PILImage.create(os.path.join(building_path,category,img)),label=category) #plot image
        ax[i,j].set_title(category,fontsize = 14)
        if(j==2):break
```

we will make our custom dataset based on the street and food data

[5]:
```
num_food_class=len(os.listdir(food_path))
print(f"there are {num_food_class} food classes" )
print(f"there are 2 non-food classes" )
```

```
there are 101 food classes
there are 2 non-food classes
```

we will make a dataset consisting of 505 food images (5 images from each class) and 500 non-food images

[6]:
```
dataset_root_path=Path("/kaggle/working/dataset")
```

we have 101 food classes, to make our model robust and don't give it too much information we will not give it all types of food during training. So in the test set there will be different categories of food that the model has never seen before. We do this because the model has to be robust and recognise food, not specific dishes

[7]:
```
i=0
for category_dir, _, images in os.walk(food_path):
    if(i%2):
        for img in images[:20]:
            dest=(dataset_root_path/"train/food") #dest will be /kaggle/working/
 ↪dataset/food/train
            dest.mkdir(exist_ok=True, parents=True)
            shutil.copy(os.path.join(category_dir, img), dest)
    else:
        for img in images[:10]:
            dest=(dataset_root_path/"test/food") #dest will be /kaggle/working/
 ↪dataset/food/train
            dest.mkdir(exist_ok=True, parents=True)
            shutil.copy(os.path.join(category_dir, img), dest)
    i+=1
```

now do the same with the non-food images

[8]:
```
for category_dir, _, images in os.walk(building_path):
    for img in images[:510]:
        dest=(dataset_root_path/"train/not_food") #dest will be /kaggle/working/
 ↪dataset/train/not_food
        dest.mkdir(exist_ok=True, parents=True)
        shutil.copy(os.path.join(category_dir, img), dest)
    for img in images[505:758]:
        dest=(dataset_root_path/"test/not_food") #dest will be /kaggle/working/
 ↪dataset/test/not_food
        dest.mkdir(exist_ok=True, parents=True)
        shutil.copy(os.path.join(category_dir, img), dest)
```

our dataset looks something like this

[9]:
```
!tree "/kaggle/working/dataset" -d
```

```
/kaggle/working/dataset
  test
     food
     not_food
  train
     food
     not_food

6 directories
```

[10]:
```
for dir ,_ ,files in os.walk("/kaggle/working/dataset"):
    if(len(_)==0):
        print(dir + "\tnum of files: "+str(len(files)))
```

```
/kaggle/working/dataset/test/food       num of files: 500
/kaggle/working/dataset/test/not_food   num of files: 506
/kaggle/working/dataset/train/food      num of files: 1020
/kaggle/working/dataset/train/not_food  num of files: 1020
```

now that our dataset is prepared we will load them using **tf.keras.utils.image_dataset_from_directory** utility

[11]:
```
data_dir=dataset_root_path/"train"
test_dir=dataset_root_path/"test"
```

our images have different sizes, we will resize them to 128x128 so they are the same as the tripadvisor dataset

[12]:
```
batch_size = 32
img_height = 128
img_width = 128
```

the labels will be inferred from the directory structure

[13]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=47,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 2040 files belonging to 2 classes.
Using 1632 files for training.
```

```
2022-12-10 19:54:40.574294: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
```

```
node, so returning NUMA node zero
2022-12-10 19:54:40.575402: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.576578: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.577410: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.578253: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.579143: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.583622: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-12-10 19:54:40.832798: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.833685: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.834471: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.835214: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.835945: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:40.836655: I
```

```
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:50.214399: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:50.215316: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:50.216038: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:50.216734: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:50.217450: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:50.218204: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13349 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
2022-12-10 19:54:50.222435: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-10 19:54:50.223126: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13349 MB memory:  -> device:
1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5
```

[14]:
```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=47,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 2040 files belonging to 2 classes.
Using 408 files for validation.
```

[15]:
```
test_ds = tf.keras.utils.image_dataset_from_directory(
  test_dir,
  seed=47,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 1006 files belonging to 2 classes.
```

[16]:
```
resize_and_rescale = keras.Sequential([
  layers.Resizing(img_width, img_height),
  layers.Rescaling(1./255)
])
```

we made ssure to only include relevant augmentations

[17]:
```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.03),
    layers.RandomZoom(height_factor=(-0.1, 0.1)),
#     layers.RandomTranslation(height_factor=(-0.1, 0.1), width_factor=(-0.1, 0.
 ↪1)),
    layers.RandomContrast(factor=0.5),
#     keras.layers.RandomBrightness(factor=0.1)
])
```

take a look at our augmentations

[18]:
```
image= PILImage.create("/kaggle/input/food41/images/apple_pie/1014775.jpg")
image
```

[18]:

[19]:
```
image = tf.cast(tf.expand_dims(image, 0), 'uint8')
plt.figure(figsize=(10, 10))
for i in range(9):
    augmented_image = data_augmentation(image)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_image[0].numpy().astype('uint8'))
    plt.axis("off")
```

[20]:
```python
def prepare_dataset(ds, batch_size=128, b_shuffle=True,augment=True):
    # transform input data into tf.data

    ds = ds.map(map_func = preprocessing ,num_parallel_calls = tf.data.experimental.AUTOTUNE)
    # normally you only need to shuffle the training data
    if b_shuffle == True:
        ds = ds.shuffle(len(ds))
    # normally you only need to shuffle the training data
    if augment:
        ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

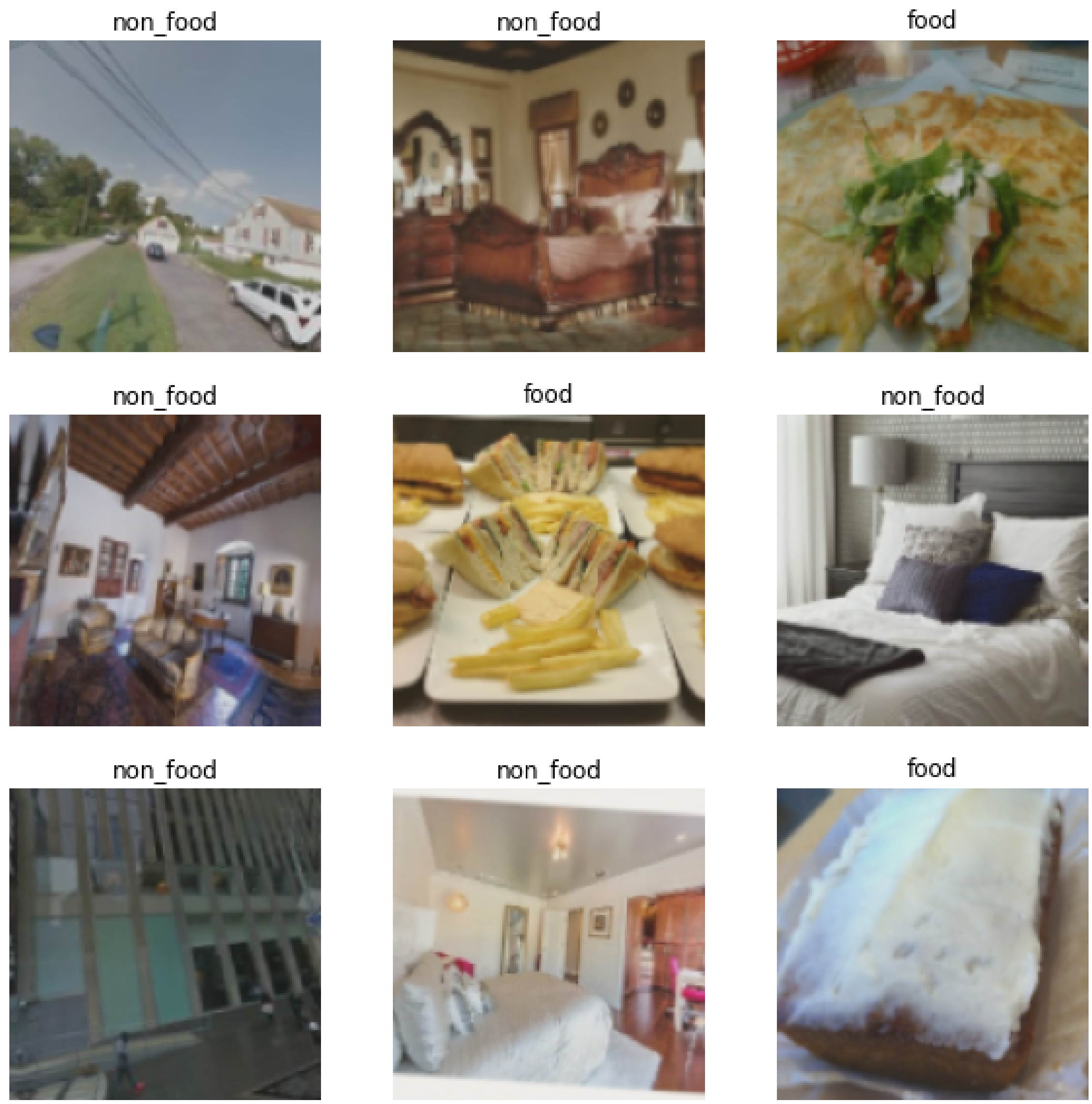
```
    ds = ds.prefetch(buffer_size = tf.data.experimental.AUTOTUNE)
    return ds



def preprocessing(image, label):
    image = resize_and_rescale(image)
    return image, label

batch_size = 128

train_ds = prepare_dataset(train_ds,augment=True)
val_ds = prepare_dataset(val_ds, b_shuffle = False,augment=False)
test_ds = prepare_dataset(test_ds, b_shuffle = False,augment=False)
```

we will configure our datasets for performance

[21]:
```
# class_names = train_ds.class_names
# print(class_names)
class_names=["food","non_food"]
```

[22]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

```
2022-12-10 19:54:54.918344: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
```



The RGB channel values are in the [0, 255] range. This is not ideal for a neural network; Here, we will standardize values to be in the [0, 1] range by using tf.keras.layers.Rescaling

1.2 making our cutom neural net

1.3 training our model

1.3.1 building our model

we will make our own custom neural network

[29]:
```
def build_model():
    model=Sequential()
    # model.add(keras.layers.Resizing(img_width, img_height))
```

```
    # model.add(keras.layers.Rescaling(1./255))
    model.add(keras.layers.InputLayer((img_width,img_height,3)))
    model.add(Conv2D(64, (3, 3), padding='same',kernel_regularizer=regularizers.
↪l2(0.01)))
    model.add(Activation('leaky_relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))

    model.add(Conv2D(64, (3, 3), padding='same',kernel_regularizer=regularizers.
↪l2(0.01),activation="relu"))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(128, (3, 3),␣
↪padding='same',kernel_regularizer=regularizers.l1(0.
↪001),activation="leaky_relu"))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3, 3),activation="relu"))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(256, (3, 3),␣
↪padding='same',kernel_regularizer=regularizers.l2(0.001)))
    model.add(Activation('leaky_relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.4))

    model.add(Conv2D(256, (3, 3),␣
↪padding='same',kernel_regularizer=regularizers.l2(0.03)))
    model.add(BatchNormalization())
    model.add(Dropout(0.4))

    #now add our fully connected layers on top
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    #now our output layer
    model.add(keras.layers.Dense(1,activation="sigmoid")) # we will give out a␣
↪single propability predicting if it is food or not
    # a high number means a high propability of a non-food image
    return model
```

[30]:

```
model_overfit=build_model()
model_overfit.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 128, 128, 64)      1792      
_________________________________________________________________
activation (Activation)      (None, 128, 128, 64)      0         
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 128, 64)      256       
_________________________________________________________________
dropout (Dropout)            (None, 128, 128, 64)      0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      36928     
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 64)      256       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       73856     
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 128)       512       
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 64, 128)       0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 62, 62, 128)       147584    
_________________________________________________________________
batch_normalization_3 (Batch (None, 62, 62, 128)       512       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 31, 31, 256)       295168    
_________________________________________________________________
activation_1 (Activation)    (None, 31, 31, 256)       0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 31, 31, 256)       1024      
_________________________________________________________________
dropout_2 (Dropout)          (None, 31, 31, 256)       0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 31, 31, 256)       590080    
_________________________________________________________________
batch_normalization_5 (Batch (None, 31, 31, 256)       1024      
_________________________________________________________________
dropout_3 (Dropout)          (None, 31, 31, 256)       0         
_________________________________________________________________
flatten (Flatten)            (None, 246016)            0         
_________________________________________________________________
dense (Dense)                (None, 64)                15745088  
_________________________________________________________________
```

```
dense_1 (Dense)             (None, 1)                 65
=================================================================
Total params: 16,894,145
Trainable params: 16,892,353
Non-trainable params: 1,792
_________________________________________________________________
```

1.3.2 first letting our model overfit to see if it is complex enough to distinguish food and non-food

[31]:
```
model_overfit.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001,rho=0.
 ↪9,momentum=0.005),

#     optimizer=tf.keras.optimizers.Adagrad(0.7),#as stated in the original␣
 ↪paper, Adagrad benefits from an initial high lr
#     optimizer=tf.keras.optimizers.Adam(),
    metrics=[tf.keras.metrics.BinaryAccuracy(), tf.keras.metrics.AUC()]
)
```

[32]:
```
history = model_overfit.fit(train_ds,
                        validation_data=val_ds,
                        epochs=50,
                        verbose=1)
```

```
Epoch 1/50

2022-12-10 19:55:05.852482: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
Loaded cuDNN version 8005

51/51 [==============================] - 26s 179ms/step - loss: 50.1324 -
binary_accuracy: 0.7138 - auc: 0.7367 - val_loss: 68.0182 - val_binary_accuracy:
0.5074 - val_auc: 0.5000
Epoch 2/50
51/51 [==============================] - 10s 154ms/step - loss: 20.5330 -
binary_accuracy: 0.7862 - auc: 0.8444 - val_loss: 106.5546 -
val_binary_accuracy: 0.5074 - val_auc: 0.5000
Epoch 3/50
51/51 [==============================] - 9s 154ms/step - loss: 9.6593 -
binary_accuracy: 0.7708 - auc: 0.8547 - val_loss: 20.1697 - val_binary_accuracy:
0.5074 - val_auc: 0.5000
Epoch 4/50
51/51 [==============================] - 9s 154ms/step - loss: 4.6222 -
binary_accuracy: 0.7947 - auc: 0.8549 - val_loss: 45.4185 - val_binary_accuracy:
0.5074 - val_auc: 0.5000
Epoch 5/50
51/51 [==============================] - 9s 146ms/step - loss: 2.6374 -
binary_accuracy: 0.8272 - auc: 0.8941 - val_loss: 2.5021 - val_binary_accuracy:
```

```
0.4926 - val_auc: 0.8284
Epoch 6/50
51/51 [==============================] - 9s 150ms/step - loss: 1.9345 -
binary_accuracy: 0.8119 - auc: 0.9041 - val_loss: 37.1951 - val_binary_accuracy:
0.5074 - val_auc: 0.5000
Epoch 7/50
51/51 [==============================] - 9s 144ms/step - loss: 2.3418 -
binary_accuracy: 0.8382 - auc: 0.9006 - val_loss: 1.8446 - val_binary_accuracy:
0.4926 - val_auc: 0.8730
Epoch 8/50
51/51 [==============================] - 10s 154ms/step - loss: 1.9336 -
binary_accuracy: 0.7629 - auc: 0.8659 - val_loss: 1.6956 - val_binary_accuracy:
0.7721 - val_auc: 0.9373
Epoch 9/50
51/51 [==============================] - 10s 165ms/step - loss: 1.2826 -
binary_accuracy: 0.8646 - auc: 0.9358 - val_loss: 1.6450 - val_binary_accuracy:
0.7426 - val_auc: 0.8364
Epoch 10/50
51/51 [==============================] - 9s 148ms/step - loss: 1.1251 -
binary_accuracy: 0.8315 - auc: 0.9266 - val_loss: 1.2624 - val_binary_accuracy:
0.6936 - val_auc: 0.9143
Epoch 11/50
51/51 [==============================] - 9s 154ms/step - loss: 0.9490 -
binary_accuracy: 0.8499 - auc: 0.9238 - val_loss: 1.0884 - val_binary_accuracy:
0.5417 - val_auc: 0.9053
Epoch 12/50
51/51 [==============================] - 9s 154ms/step - loss: 1.2067 -
binary_accuracy: 0.8511 - auc: 0.9325 - val_loss: 4.0496 - val_binary_accuracy:
0.7696 - val_auc: 0.8120
Epoch 13/50
51/51 [==============================] - 9s 153ms/step - loss: 0.9224 -
binary_accuracy: 0.8591 - auc: 0.9334 - val_loss: 0.8976 - val_binary_accuracy:
0.8407 - val_auc: 0.9290
Epoch 14/50
51/51 [==============================] - 9s 149ms/step - loss: 0.7350 -
binary_accuracy: 0.8591 - auc: 0.9384 - val_loss: 0.6576 - val_binary_accuracy:
0.8873 - val_auc: 0.9615
Epoch 15/50
51/51 [==============================] - 10s 160ms/step - loss: 0.7644 -
binary_accuracy: 0.8793 - auc: 0.9436 - val_loss: 0.6849 - val_binary_accuracy:
0.9069 - val_auc: 0.9649
Epoch 16/50
51/51 [==============================] - 9s 150ms/step - loss: 1.5850 -
binary_accuracy: 0.8719 - auc: 0.9291 - val_loss: 150.5060 -
val_binary_accuracy: 0.4926 - val_auc: 0.4999
Epoch 17/50
51/51 [==============================] - 9s 153ms/step - loss: 1.1110 -
binary_accuracy: 0.8824 - auc: 0.9454 - val_loss: 1.2520 - val_binary_accuracy:
```

```
0.8260 - val_auc: 0.8935
Epoch 18/50
51/51 [==============================] - 9s 148ms/step - loss: 0.7698 -
binary_accuracy: 0.8738 - auc: 0.9425 - val_loss: 0.5665 - val_binary_accuracy:
0.9142 - val_auc: 0.9771
Epoch 19/50
51/51 [==============================] - 10s 148ms/step - loss: 0.8608 -
binary_accuracy: 0.8793 - auc: 0.9413 - val_loss: 7.4865 - val_binary_accuracy:
0.6961 - val_auc: 0.7408
Epoch 20/50
51/51 [==============================] - 10s 154ms/step - loss: 0.9034 -
binary_accuracy: 0.8854 - auc: 0.9464 - val_loss: 12.1273 - val_binary_accuracy:
0.5760 - val_auc: 0.7389
Epoch 21/50
51/51 [==============================] - 9s 152ms/step - loss: 0.8487 -
binary_accuracy: 0.8830 - auc: 0.9473 - val_loss: 8.0094 - val_binary_accuracy:
0.7132 - val_auc: 0.7952
Epoch 22/50
51/51 [==============================] - 10s 150ms/step - loss: 0.7498 -
binary_accuracy: 0.8958 - auc: 0.9559 - val_loss: 0.6190 - val_binary_accuracy:
0.8799 - val_auc: 0.9525
Epoch 23/50
51/51 [==============================] - 10s 161ms/step - loss: 1.0178 -
binary_accuracy: 0.8811 - auc: 0.9391 - val_loss: 0.8475 - val_binary_accuracy:
0.8725 - val_auc: 0.9400
Epoch 24/50
51/51 [==============================] - 9s 152ms/step - loss: 0.6238 -
binary_accuracy: 0.8952 - auc: 0.9594 - val_loss: 0.7726 - val_binary_accuracy:
0.7083 - val_auc: 0.9386
Epoch 25/50
51/51 [==============================] - 10s 162ms/step - loss: 0.5886 -
binary_accuracy: 0.8964 - auc: 0.9570 - val_loss: 0.5467 - val_binary_accuracy:
0.9216 - val_auc: 0.9732
Epoch 26/50
51/51 [==============================] - 10s 156ms/step - loss: 0.6419 -
binary_accuracy: 0.9013 - auc: 0.9549 - val_loss: 0.6313 - val_binary_accuracy:
0.8971 - val_auc: 0.9546
Epoch 27/50
51/51 [==============================] - 9s 151ms/step - loss: 0.5268 -
binary_accuracy: 0.9105 - auc: 0.9662 - val_loss: 0.5211 - val_binary_accuracy:
0.9020 - val_auc: 0.9600
Epoch 28/50
51/51 [==============================] - 10s 160ms/step - loss: 0.9580 -
binary_accuracy: 0.8873 - auc: 0.9459 - val_loss: 1.6421 - val_binary_accuracy:
0.7917 - val_auc: 0.8704
Epoch 29/50
51/51 [==============================] - 10s 156ms/step - loss: 0.7218 -
binary_accuracy: 0.9026 - auc: 0.9572 - val_loss: 0.7268 - val_binary_accuracy:
```

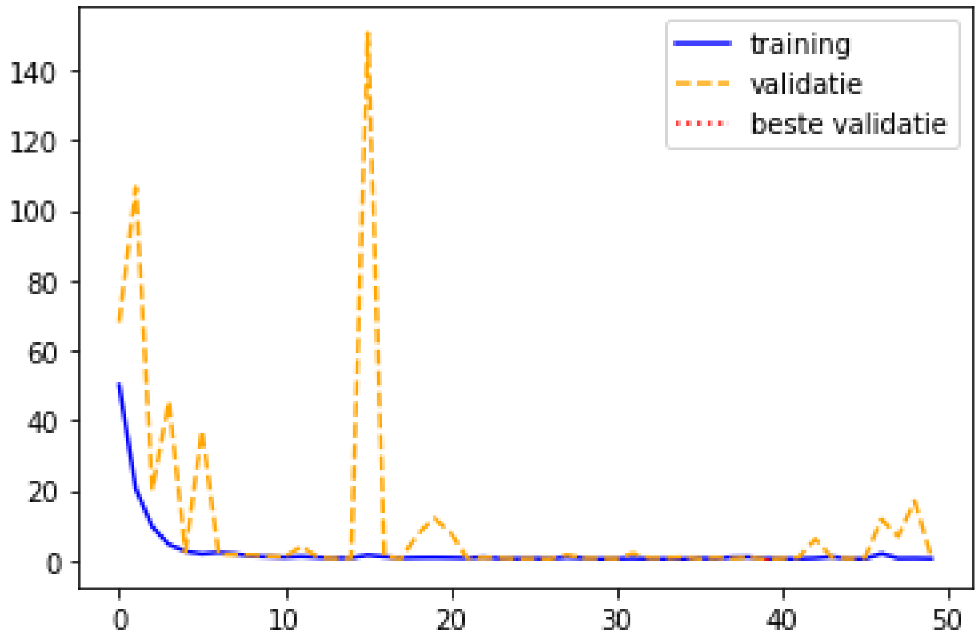
```
0.9142 - val_auc: 0.9659
Epoch 30/50
51/51 [==============================] - 9s 151ms/step - loss: 0.5001 -
binary_accuracy: 0.8989 - auc: 0.9601 - val_loss: 0.7008 - val_binary_accuracy:
0.8824 - val_auc: 0.9555
Epoch 31/50
51/51 [==============================] - 10s 154ms/step - loss: 0.5667 -
binary_accuracy: 0.8946 - auc: 0.9605 - val_loss: 0.6025 - val_binary_accuracy:
0.8652 - val_auc: 0.9666
Epoch 32/50
51/51 [==============================] - 9s 146ms/step - loss: 0.6647 -
binary_accuracy: 0.9118 - auc: 0.9648 - val_loss: 2.0653 - val_binary_accuracy:
0.8627 - val_auc: 0.9628
Epoch 33/50
51/51 [==============================] - 10s 159ms/step - loss: 0.5017 -
binary_accuracy: 0.9136 - auc: 0.9640 - val_loss: 0.7232 - val_binary_accuracy:
0.9265 - val_auc: 0.9642
Epoch 34/50
51/51 [==============================] - 10s 163ms/step - loss: 0.5292 -
binary_accuracy: 0.8964 - auc: 0.9626 - val_loss: 0.7425 - val_binary_accuracy:
0.8431 - val_auc: 0.9454
Epoch 35/50
51/51 [==============================] - 9s 151ms/step - loss: 0.4816 -
binary_accuracy: 0.9191 - auc: 0.9729 - val_loss: 0.9986 - val_binary_accuracy:
0.6936 - val_auc: 0.9564
Epoch 36/50
51/51 [==============================] - 9s 151ms/step - loss: 0.6956 -
binary_accuracy: 0.9075 - auc: 0.9628 - val_loss: 0.5515 - val_binary_accuracy:
0.9167 - val_auc: 0.9686
Epoch 37/50
51/51 [==============================] - 10s 165ms/step - loss: 0.5527 -
binary_accuracy: 0.9185 - auc: 0.9688 - val_loss: 0.8372 - val_binary_accuracy:
0.8676 - val_auc: 0.9480
Epoch 38/50
51/51 [==============================] - 9s 151ms/step - loss: 1.0071 -
binary_accuracy: 0.8885 - auc: 0.9374 - val_loss: 0.5907 - val_binary_accuracy:
0.9289 - val_auc: 0.9749
Epoch 39/50
51/51 [==============================] - 10s 159ms/step - loss: 1.0350 -
binary_accuracy: 0.8824 - auc: 0.9456 - val_loss: 0.9389 - val_binary_accuracy:
0.8578 - val_auc: 0.9444
Epoch 40/50
51/51 [==============================] - 9s 147ms/step - loss: 0.6201 -
binary_accuracy: 0.9056 - auc: 0.9637 - val_loss: 0.4589 - val_binary_accuracy:
0.9436 - val_auc: 0.9822
Epoch 41/50
51/51 [==============================] - 10s 156ms/step - loss: 0.5302 -
binary_accuracy: 0.9081 - auc: 0.9623 - val_loss: 0.6387 - val_binary_accuracy:
```

```
0.8750 - val_auc: 0.9494
Epoch 42/50
51/51 [==============================] - 9s 147ms/step - loss: 0.5416 -
binary_accuracy: 0.9099 - auc: 0.9656 - val_loss: 0.8971 - val_binary_accuracy:
0.8873 - val_auc: 0.9567
Epoch 43/50
51/51 [==============================] - 9s 148ms/step - loss: 0.7140 -
binary_accuracy: 0.8775 - auc: 0.9428 - val_loss: 6.1441 - val_binary_accuracy:
0.7525 - val_auc: 0.8740
Epoch 44/50
51/51 [==============================] - 10s 170ms/step - loss: 1.1011 -
binary_accuracy: 0.8805 - auc: 0.9394 - val_loss: 0.9495 - val_binary_accuracy:
0.8113 - val_auc: 0.9178
Epoch 45/50
51/51 [==============================] - 9s 153ms/step - loss: 0.5411 -
binary_accuracy: 0.9148 - auc: 0.9711 - val_loss: 0.5982 - val_binary_accuracy:
0.9069 - val_auc: 0.9618
Epoch 46/50
51/51 [==============================] - 10s 165ms/step - loss: 0.5171 -
binary_accuracy: 0.9020 - auc: 0.9596 - val_loss: 0.8442 - val_binary_accuracy:
0.8897 - val_auc: 0.9604
Epoch 47/50
51/51 [==============================] - 10s 158ms/step - loss: 2.0250 -
binary_accuracy: 0.8536 - auc: 0.9305 - val_loss: 11.8447 - val_binary_accuracy:
0.8603 - val_auc: 0.8975
Epoch 48/50
51/51 [==============================] - 9s 156ms/step - loss: 0.6459 -
binary_accuracy: 0.9050 - auc: 0.9630 - val_loss: 6.9037 - val_binary_accuracy:
0.8015 - val_auc: 0.8109
Epoch 49/50
51/51 [==============================] - 10s 166ms/step - loss: 0.6860 -
binary_accuracy: 0.9050 - auc: 0.9622 - val_loss: 17.0119 - val_binary_accuracy:
0.7892 - val_auc: 0.8211
Epoch 50/50
51/51 [==============================] - 10s 157ms/step - loss: 0.6015 -
binary_accuracy: 0.8995 - auc: 0.9611 - val_loss: 1.0881 - val_binary_accuracy:
0.9338 - val_auc: 0.9682
```

let's look at the loss curves to see if it overfitted

[33]:
```python
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',
  ls='--')
```

```
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',␣
 ↪color='red', ls=':')
plt.legend()
plt.show()
```



[34]:
```
model_overfit.save_weights('model_overfit.h5')
```

To our surprize it did not overfit, we think this is because of the regularisation and dropout layers and batchnormalisation layers that reduce overfitting

1.3.3 Training our model (for real this time)

[35]:
```
model=build_model()
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001,rho=0.
 ↪9,momentum=0.005),
#     optimizer=tf.keras.optimizers.Adagrad(0.7),#as stated in the original␣
 ↪paper, Adagrad benefits from an initial high lr
#     optimizer=tf.keras.optimizers.Adam(),#as stated in the original paper,␣
 ↪Adagrad benefits from an initial high lr
    metrics=[tf.keras.metrics.BinaryAccuracy(), tf.keras.metrics.AUC()]
)
```

1.3.4 callbacks

Early stopping will interrupt training when meaningful improvements are no longer observed on the validation data, as this indicates that the model may have reached its peak. The second callback will lower RMSprop's learning rate at appropriate times to try to prevent training from stopping prematurely.

[36]:
```
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=20,
    min_delta=1e-4,
    restore_best_weights=True,
)

plateau = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=10,
    min_delta=1e-4,
    cooldown=0,
    verbose=1
)
```

1.3.5 Model fitting

now we can finally start training our model

[37]:
```
history = model.fit(train_ds,
                         validation_data=val_ds,
                         epochs=100,
                         callbacks=[early_stopping, plateau],
                         verbose=1)
```

```
Epoch 1/100
51/51 [==============================] - 11s 155ms/step - loss: 51.5082 -
binary_accuracy: 0.7365 - auc_1: 0.7548 - val_loss: 574.0889 -
val_binary_accuracy: 0.5074 - val_auc_1: 0.5000
Epoch 2/100
51/51 [==============================] - 10s 151ms/step - loss: 32.1776 -
binary_accuracy: 0.7892 - auc_1: 0.8340 - val_loss: 56.0913 -
val_binary_accuracy: 0.4926 - val_auc_1: 0.4976
Epoch 3/100
51/51 [==============================] - 9s 153ms/step - loss: 23.5494 -
binary_accuracy: 0.8364 - auc_1: 0.8675 - val_loss: 81.4421 -
val_binary_accuracy: 0.5074 - val_auc_1: 0.5000
Epoch 4/100
51/51 [==============================] - 10s 162ms/step - loss: 11.8490 -
binary_accuracy: 0.8585 - auc_1: 0.9010 - val_loss: 58.1839 -
val_binary_accuracy: 0.5270 - val_auc_1: 0.5274
```

```
Epoch 5/100
51/51 [==============================] - 10s 161ms/step - loss: 7.2184 -
binary_accuracy: 0.8529 - auc_1: 0.9124 - val_loss: 15.9715 -
val_binary_accuracy: 0.4926 - val_auc_1: 0.4689
Epoch 6/100
51/51 [==============================] - 9s 151ms/step - loss: 4.1229 -
binary_accuracy: 0.8333 - auc_1: 0.9086 - val_loss: 3.0592 -
val_binary_accuracy: 0.4926 - val_auc_1: 0.9623
Epoch 7/100
51/51 [==============================] - 9s 148ms/step - loss: 7.1070 -
binary_accuracy: 0.8064 - auc_1: 0.8711 - val_loss: 2.9850 -
val_binary_accuracy: 0.5490 - val_auc_1: 0.8663
Epoch 8/100
51/51 [==============================] - 10s 159ms/step - loss: 2.4865 -
binary_accuracy: 0.8241 - auc_1: 0.8925 - val_loss: 3.4295 -
val_binary_accuracy: 0.4926 - val_auc_1: 0.5269
Epoch 9/100
51/51 [==============================] - 9s 153ms/step - loss: 1.7414 -
binary_accuracy: 0.8621 - auc_1: 0.9275 - val_loss: 1.5582 -
val_binary_accuracy: 0.5833 - val_auc_1: 0.9268
Epoch 10/100
51/51 [==============================] - 9s 148ms/step - loss: 1.5717 -
binary_accuracy: 0.8419 - auc_1: 0.9093 - val_loss: 1.3714 -
val_binary_accuracy: 0.8676 - val_auc_1: 0.9507
Epoch 11/100
51/51 [==============================] - 10s 161ms/step - loss: 1.0736 -
binary_accuracy: 0.8652 - auc_1: 0.9287 - val_loss: 1.0638 -
val_binary_accuracy: 0.7377 - val_auc_1: 0.9376
Epoch 12/100
51/51 [==============================] - 9s 151ms/step - loss: 1.4188 -
binary_accuracy: 0.8321 - auc_1: 0.9118 - val_loss: 0.9648 -
val_binary_accuracy: 0.9020 - val_auc_1: 0.9735
Epoch 13/100
51/51 [==============================] - 10s 157ms/step - loss: 1.3386 -
binary_accuracy: 0.8670 - auc_1: 0.9315 - val_loss: 1.2214 -
val_binary_accuracy: 0.7108 - val_auc_1: 0.9659
Epoch 14/100
51/51 [==============================] - 10s 163ms/step - loss: 1.5615 -
binary_accuracy: 0.8536 - auc_1: 0.9265 - val_loss: 1.1179 -
val_binary_accuracy: 0.8873 - val_auc_1: 0.9546
Epoch 15/100
51/51 [==============================] - 10s 167ms/step - loss: 0.9715 -
binary_accuracy: 0.8781 - auc_1: 0.9461 - val_loss: 0.7848 -
val_binary_accuracy: 0.8725 - val_auc_1: 0.9619
Epoch 16/100
51/51 [==============================] - 10s 162ms/step - loss: 0.9859 -
binary_accuracy: 0.8658 - auc_1: 0.9354 - val_loss: 1.0598 -
val_binary_accuracy: 0.8260 - val_auc_1: 0.9540
```

```
Epoch 17/100
51/51 [==============================] - 10s 160ms/step - loss: 0.7074 -
binary_accuracy: 0.8983 - auc_1: 0.9549 - val_loss: 0.7481 -
val_binary_accuracy: 0.8627 - val_auc_1: 0.9483
Epoch 18/100
51/51 [==============================] - 9s 149ms/step - loss: 0.7909 -
binary_accuracy: 0.8750 - auc_1: 0.9338 - val_loss: 0.9387 -
val_binary_accuracy: 0.8701 - val_auc_1: 0.9533
Epoch 19/100
51/51 [==============================] - 9s 151ms/step - loss: 0.7386 -
binary_accuracy: 0.8873 - auc_1: 0.9515 - val_loss: 0.7613 -
val_binary_accuracy: 0.9044 - val_auc_1: 0.9656
Epoch 20/100
51/51 [==============================] - 9s 148ms/step - loss: 0.6479 -
binary_accuracy: 0.9081 - auc_1: 0.9603 - val_loss: 0.6244 -
val_binary_accuracy: 0.8922 - val_auc_1: 0.9565
Epoch 21/100
51/51 [==============================] - 10s 151ms/step - loss: 10.5663 -
binary_accuracy: 0.8603 - auc_1: 0.9232 - val_loss: 0.9113 -
val_binary_accuracy: 0.8775 - val_auc_1: 0.9560
Epoch 22/100
51/51 [==============================] - 9s 152ms/step - loss: 1.4167 -
binary_accuracy: 0.8303 - auc_1: 0.9215 - val_loss: 1.7301 -
val_binary_accuracy: 0.7353 - val_auc_1: 0.9124
Epoch 23/100
51/51 [==============================] - 9s 150ms/step - loss: 0.8918 -
binary_accuracy: 0.8640 - auc_1: 0.9379 - val_loss: 1.0113 -
val_binary_accuracy: 0.6250 - val_auc_1: 0.9511
Epoch 24/100
51/51 [==============================] - 10s 151ms/step - loss: 0.7885 -
binary_accuracy: 0.8971 - auc_1: 0.9566 - val_loss: 1.1455 -
val_binary_accuracy: 0.9142 - val_auc_1: 0.9693
Epoch 25/100
51/51 [==============================] - 9s 153ms/step - loss: 0.6035 -
binary_accuracy: 0.9001 - auc_1: 0.9615 - val_loss: 0.5334 -
val_binary_accuracy: 0.9020 - val_auc_1: 0.9721
Epoch 26/100
51/51 [==============================] - 9s 155ms/step - loss: 0.6049 -
binary_accuracy: 0.8854 - auc_1: 0.9523 - val_loss: 0.6705 -
val_binary_accuracy: 0.9093 - val_auc_1: 0.9502
Epoch 27/100
51/51 [==============================] - 10s 164ms/step - loss: 0.5648 -
binary_accuracy: 0.8977 - auc_1: 0.9598 - val_loss: 0.7786 -
val_binary_accuracy: 0.8039 - val_auc_1: 0.9686
Epoch 28/100
51/51 [==============================] - 10s 155ms/step - loss: 0.7654 -
binary_accuracy: 0.9056 - auc_1: 0.9601 - val_loss: 0.7210 -
val_binary_accuracy: 0.8480 - val_auc_1: 0.9705
```

```
Epoch 29/100
51/51 [==============================] - 9s 145ms/step - loss: 0.6746 -
binary_accuracy: 0.8946 - auc_1: 0.9560 - val_loss: 0.7557 -
val_binary_accuracy: 0.9044 - val_auc_1: 0.9742
Epoch 30/100
51/51 [==============================] - 10s 156ms/step - loss: 0.4786 -
binary_accuracy: 0.9044 - auc_1: 0.9642 - val_loss: 2.0528 -
val_binary_accuracy: 0.5049 - val_auc_1: 0.8031
Epoch 31/100
51/51 [==============================] - 10s 162ms/step - loss: 0.9736 -
binary_accuracy: 0.8989 - auc_1: 0.9506 - val_loss: 2.6088 -
val_binary_accuracy: 0.8848 - val_auc_1: 0.9420
Epoch 32/100
51/51 [==============================] - 9s 154ms/step - loss: 0.5634 -
binary_accuracy: 0.9148 - auc_1: 0.9663 - val_loss: 0.5676 -
val_binary_accuracy: 0.8799 - val_auc_1: 0.9553
Epoch 33/100
51/51 [==============================] - 10s 158ms/step - loss: 0.5951 -
binary_accuracy: 0.8964 - auc_1: 0.9624 - val_loss: 0.6247 -
val_binary_accuracy: 0.8824 - val_auc_1: 0.9597
Epoch 34/100
51/51 [==============================] - 9s 154ms/step - loss: 0.4922 -
binary_accuracy: 0.9032 - auc_1: 0.9640 - val_loss: 0.5139 -
val_binary_accuracy: 0.9118 - val_auc_1: 0.9668
Epoch 35/100
51/51 [==============================] - 10s 158ms/step - loss: 0.5902 -
binary_accuracy: 0.8922 - auc_1: 0.9572 - val_loss: 0.7975 -
val_binary_accuracy: 0.7917 - val_auc_1: 0.9236
Epoch 36/100
51/51 [==============================] - 9s 152ms/step - loss: 0.5597 -
binary_accuracy: 0.9013 - auc_1: 0.9649 - val_loss: 0.6008 -
val_binary_accuracy: 0.9191 - val_auc_1: 0.9730
Epoch 37/100
51/51 [==============================] - 9s 148ms/step - loss: 0.5656 -
binary_accuracy: 0.9179 - auc_1: 0.9708 - val_loss: 0.5958 -
val_binary_accuracy: 0.8456 - val_auc_1: 0.9484
Epoch 38/100
51/51 [==============================] - 10s 160ms/step - loss: 1.0902 -
binary_accuracy: 0.8670 - auc_1: 0.9282 - val_loss: 0.8757 -
val_binary_accuracy: 0.7892 - val_auc_1: 0.9598
Epoch 39/100
51/51 [==============================] - 10s 157ms/step - loss: 0.4898 -
binary_accuracy: 0.9154 - auc_1: 0.9690 - val_loss: 0.8442 -
val_binary_accuracy: 0.7966 - val_auc_1: 0.9558
Epoch 40/100
51/51 [==============================] - 9s 147ms/step - loss: 0.4591 -
binary_accuracy: 0.9124 - auc_1: 0.9685 - val_loss: 1.0758 -
val_binary_accuracy: 0.8578 - val_auc_1: 0.9473
```

```
Epoch 41/100
51/51 [==============================] - 9s 149ms/step - loss: 0.4646 -
binary_accuracy: 0.9161 - auc_1: 0.9666 - val_loss: 2.4073 -
val_binary_accuracy: 0.7843 - val_auc_1: 0.9050
Epoch 42/100
51/51 [==============================] - 10s 150ms/step - loss: 0.4939 -
binary_accuracy: 0.9179 - auc_1: 0.9644 - val_loss: 0.4814 -
val_binary_accuracy: 0.9118 - val_auc_1: 0.9667
Epoch 43/100
51/51 [==============================] - 10s 154ms/step - loss: 0.4806 -
binary_accuracy: 0.9185 - auc_1: 0.9676 - val_loss: 1.0877 -
val_binary_accuracy: 0.8039 - val_auc_1: 0.9735
Epoch 44/100
51/51 [==============================] - 10s 159ms/step - loss: 0.4726 -
binary_accuracy: 0.9124 - auc_1: 0.9675 - val_loss: 0.4319 -
val_binary_accuracy: 0.9093 - val_auc_1: 0.9751
Epoch 45/100
51/51 [==============================] - 10s 173ms/step - loss: 0.5787 -
binary_accuracy: 0.9087 - auc_1: 0.9635 - val_loss: 0.6641 -
val_binary_accuracy: 0.9240 - val_auc_1: 0.9607
Epoch 46/100
51/51 [==============================] - 9s 150ms/step - loss: 0.3933 -
binary_accuracy: 0.9308 - auc_1: 0.9781 - val_loss: 0.4554 -
val_binary_accuracy: 0.9118 - val_auc_1: 0.9734
Epoch 47/100
51/51 [==============================] - 10s 160ms/step - loss: 0.4174 -
binary_accuracy: 0.9289 - auc_1: 0.9719 - val_loss: 0.5330 -
val_binary_accuracy: 0.8505 - val_auc_1: 0.9796
Epoch 48/100
51/51 [==============================] - 10s 154ms/step - loss: 0.3913 -
binary_accuracy: 0.9185 - auc_1: 0.9762 - val_loss: 0.6825 -
val_binary_accuracy: 0.8015 - val_auc_1: 0.9776
Epoch 49/100
51/51 [==============================] - 9s 149ms/step - loss: 0.3876 -
binary_accuracy: 0.9350 - auc_1: 0.9789 - val_loss: 0.4051 -
val_binary_accuracy: 0.9216 - val_auc_1: 0.9755
Epoch 50/100
51/51 [==============================] - 10s 164ms/step - loss: 0.7914 -
binary_accuracy: 0.9265 - auc_1: 0.9711 - val_loss: 0.5478 -
val_binary_accuracy: 0.9216 - val_auc_1: 0.9875
Epoch 51/100
51/51 [==============================] - 9s 150ms/step - loss: 0.4024 -
binary_accuracy: 0.9455 - auc_1: 0.9826 - val_loss: 0.3348 -
val_binary_accuracy: 0.9412 - val_auc_1: 0.9827
Epoch 52/100
51/51 [==============================] - 10s 151ms/step - loss: 0.3636 -
binary_accuracy: 0.9381 - auc_1: 0.9831 - val_loss: 0.3065 -
val_binary_accuracy: 0.9412 - val_auc_1: 0.9848
```

```
Epoch 53/100
51/51 [==============================] - 10s 154ms/step - loss: 0.3822 -
binary_accuracy: 0.9381 - auc_1: 0.9793 - val_loss: 0.4454 -
val_binary_accuracy: 0.8971 - val_auc_1: 0.9801
Epoch 54/100
51/51 [==============================] - 9s 155ms/step - loss: 0.4632 -
binary_accuracy: 0.9491 - auc_1: 0.9854 - val_loss: 0.6530 -
val_binary_accuracy: 0.9461 - val_auc_1: 0.9729
Epoch 55/100
51/51 [==============================] - 9s 148ms/step - loss: 0.3370 -
binary_accuracy: 0.9559 - auc_1: 0.9888 - val_loss: 0.3944 -
val_binary_accuracy: 0.9191 - val_auc_1: 0.9817
Epoch 56/100
51/51 [==============================] - 9s 155ms/step - loss: 0.3303 -
binary_accuracy: 0.9498 - auc_1: 0.9849 - val_loss: 0.6332 -
val_binary_accuracy: 0.8824 - val_auc_1: 0.9742
Epoch 57/100
51/51 [==============================] - 10s 162ms/step - loss: 0.3258 -
binary_accuracy: 0.9461 - auc_1: 0.9836 - val_loss: 2.2624 -
val_binary_accuracy: 0.7892 - val_auc_1: 0.9223
Epoch 58/100
51/51 [==============================] - 9s 152ms/step - loss: 0.3366 -
binary_accuracy: 0.9461 - auc_1: 0.9864 - val_loss: 0.2816 -
val_binary_accuracy: 0.9657 - val_auc_1: 0.9901
Epoch 59/100
51/51 [==============================] - 9s 150ms/step - loss: 0.3107 -
binary_accuracy: 0.9473 - auc_1: 0.9883 - val_loss: 0.4804 -
val_binary_accuracy: 0.9338 - val_auc_1: 0.9818
Epoch 60/100
51/51 [==============================] - 10s 153ms/step - loss: 0.3300 -
binary_accuracy: 0.9467 - auc_1: 0.9841 - val_loss: 0.9782 -
val_binary_accuracy: 0.8578 - val_auc_1: 0.9217
Epoch 61/100
51/51 [==============================] - 9s 147ms/step - loss: 0.3855 -
binary_accuracy: 0.9387 - auc_1: 0.9805 - val_loss: 0.4509 -
val_binary_accuracy: 0.9412 - val_auc_1: 0.9851
Epoch 62/100
51/51 [==============================] - 9s 147ms/step - loss: 0.3018 -
binary_accuracy: 0.9498 - auc_1: 0.9881 - val_loss: 0.6213 -
val_binary_accuracy: 0.8162 - val_auc_1: 0.9712
Epoch 63/100
51/51 [==============================] - 10s 158ms/step - loss: 0.3229 -
binary_accuracy: 0.9491 - auc_1: 0.9866 - val_loss: 0.5618 -
val_binary_accuracy: 0.9093 - val_auc_1: 0.9646
Epoch 64/100
51/51 [==============================] - 9s 149ms/step - loss: 0.3060 -
binary_accuracy: 0.9547 - auc_1: 0.9884 - val_loss: 0.4258 -
val_binary_accuracy: 0.9020 - val_auc_1: 0.9746
```

```
Epoch 65/100
51/51 [==============================] - 9s 151ms/step - loss: 0.3563 -
binary_accuracy: 0.9510 - auc_1: 0.9849 - val_loss: 0.3486 -
val_binary_accuracy: 0.9583 - val_auc_1: 0.9755
Epoch 66/100
51/51 [==============================] - 10s 163ms/step - loss: 0.2936 -
binary_accuracy: 0.9583 - auc_1: 0.9891 - val_loss: 0.5826 -
val_binary_accuracy: 0.9436 - val_auc_1: 0.9824
Epoch 67/100
51/51 [==============================] - 9s 148ms/step - loss: 0.2698 -
binary_accuracy: 0.9614 - auc_1: 0.9927 - val_loss: 0.5825 -
val_binary_accuracy: 0.8848 - val_auc_1: 0.9700
Epoch 68/100
51/51 [==============================] - 9s 152ms/step - loss: 0.3076 -
binary_accuracy: 0.9498 - auc_1: 0.9871 - val_loss: 0.3109 -
val_binary_accuracy: 0.9559 - val_auc_1: 0.9810

Epoch 00068: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 69/100
51/51 [==============================] - 9s 154ms/step - loss: 0.1618 -
binary_accuracy: 0.9773 - auc_1: 0.9970 - val_loss: 0.2453 -
val_binary_accuracy: 0.9730 - val_auc_1: 0.9821
Epoch 70/100
51/51 [==============================] - 10s 156ms/step - loss: 0.1461 -
binary_accuracy: 0.9804 - auc_1: 0.9961 - val_loss: 0.2440 -
val_binary_accuracy: 0.9632 - val_auc_1: 0.9807
Epoch 71/100
51/51 [==============================] - 9s 145ms/step - loss: 0.1382 -
binary_accuracy: 0.9749 - auc_1: 0.9969 - val_loss: 0.2162 -
val_binary_accuracy: 0.9657 - val_auc_1: 0.9829
Epoch 72/100
51/51 [==============================] - 10s 160ms/step - loss: 0.1292 -
binary_accuracy: 0.9792 - auc_1: 0.9969 - val_loss: 0.2133 -
val_binary_accuracy: 0.9681 - val_auc_1: 0.9839
Epoch 73/100
51/51 [==============================] - 10s 154ms/step - loss: 0.1272 -
binary_accuracy: 0.9798 - auc_1: 0.9979 - val_loss: 0.2860 -
val_binary_accuracy: 0.9461 - val_auc_1: 0.9833
Epoch 74/100
51/51 [==============================] - 9s 146ms/step - loss: 0.1387 -
binary_accuracy: 0.9779 - auc_1: 0.9963 - val_loss: 0.2874 -
val_binary_accuracy: 0.9681 - val_auc_1: 0.9785
Epoch 75/100
51/51 [==============================] - 10s 166ms/step - loss: 0.1291 -
binary_accuracy: 0.9786 - auc_1: 0.9971 - val_loss: 0.2186 -
val_binary_accuracy: 0.9681 - val_auc_1: 0.9842
Epoch 76/100
51/51 [==============================] - 10s 157ms/step - loss: 0.1312 -
```

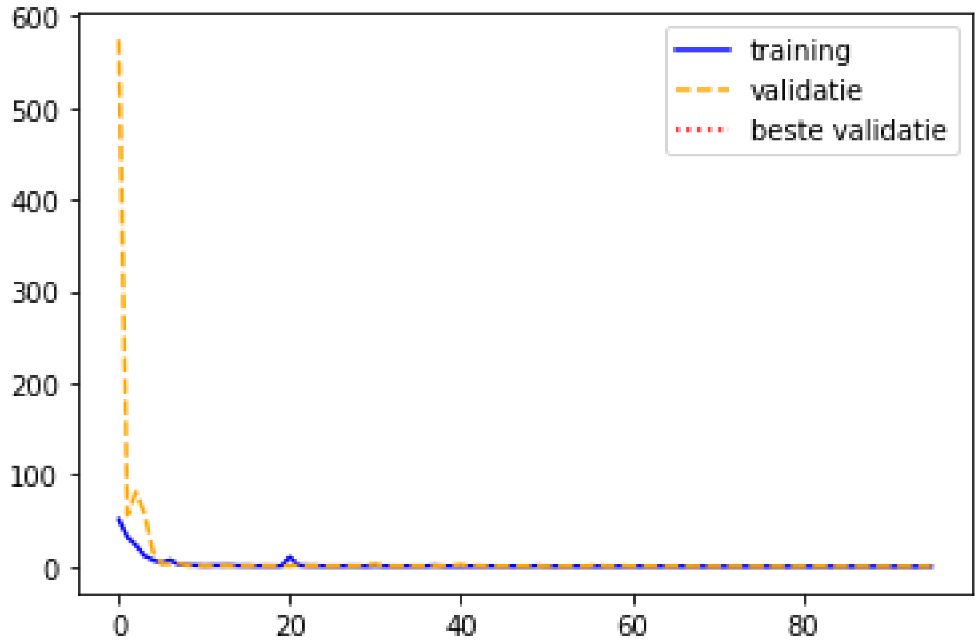
```
binary_accuracy: 0.9804 - auc_1: 0.9952 - val_loss: 0.1822 -
val_binary_accuracy: 0.9755 - val_auc_1: 0.9862
Epoch 77/100
51/51 [==============================] - 9s 150ms/step - loss: 0.1219 -
binary_accuracy: 0.9798 - auc_1: 0.9970 - val_loss: 0.2769 -
val_binary_accuracy: 0.9632 - val_auc_1: 0.9795
Epoch 78/100
51/51 [==============================] - 9s 148ms/step - loss: 0.1289 -
binary_accuracy: 0.9792 - auc_1: 0.9964 - val_loss: 0.2448 -
val_binary_accuracy: 0.9730 - val_auc_1: 0.9843
Epoch 79/100
51/51 [==============================] - 10s 165ms/step - loss: 0.1190 -
binary_accuracy: 0.9828 - auc_1: 0.9973 - val_loss: 0.2447 -
val_binary_accuracy: 0.9706 - val_auc_1: 0.9816
Epoch 80/100
51/51 [==============================] - 9s 148ms/step - loss: 0.1377 -
binary_accuracy: 0.9737 - auc_1: 0.9958 - val_loss: 0.2182 -
val_binary_accuracy: 0.9755 - val_auc_1: 0.9862
Epoch 81/100
51/51 [==============================] - 9s 148ms/step - loss: 0.1345 -
binary_accuracy: 0.9773 - auc_1: 0.9952 - val_loss: 0.3386 -
val_binary_accuracy: 0.9608 - val_auc_1: 0.9798
Epoch 82/100
51/51 [==============================] - 10s 160ms/step - loss: 0.1247 -
binary_accuracy: 0.9755 - auc_1: 0.9970 - val_loss: 0.2477 -
val_binary_accuracy: 0.9657 - val_auc_1: 0.9856
Epoch 83/100
51/51 [==============================] - 9s 144ms/step - loss: 0.1235 -
binary_accuracy: 0.9767 - auc_1: 0.9968 - val_loss: 0.5354 -
val_binary_accuracy: 0.9265 - val_auc_1: 0.9799
Epoch 84/100
51/51 [==============================] - 10s 160ms/step - loss: 0.1239 -
binary_accuracy: 0.9767 - auc_1: 0.9965 - val_loss: 0.4080 -
val_binary_accuracy: 0.9338 - val_auc_1: 0.9824
Epoch 85/100
51/51 [==============================] - 10s 166ms/step - loss: 0.1090 -
binary_accuracy: 0.9816 - auc_1: 0.9980 - val_loss: 0.2341 -
val_binary_accuracy: 0.9706 - val_auc_1: 0.9843
Epoch 86/100
51/51 [==============================] - 10s 152ms/step - loss: 0.1200 -
binary_accuracy: 0.9822 - auc_1: 0.9974 - val_loss: 0.3156 -
val_binary_accuracy: 0.9461 - val_auc_1: 0.9774

Epoch 00086: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
Epoch 87/100
51/51 [==============================] - 9s 150ms/step - loss: 0.0978 -
binary_accuracy: 0.9853 - auc_1: 0.9971 - val_loss: 0.2591 -
val_binary_accuracy: 0.9632 - val_auc_1: 0.9821
```

```
Epoch 88/100
51/51 [==============================] - 10s 157ms/step - loss: 0.0848 -
binary_accuracy: 0.9871 - auc_1: 0.9992 - val_loss: 0.2254 -
val_binary_accuracy: 0.9706 - val_auc_1: 0.9840
Epoch 89/100
51/51 [==============================] - 9s 152ms/step - loss: 0.0936 -
binary_accuracy: 0.9810 - auc_1: 0.9980 - val_loss: 0.2281 -
val_binary_accuracy: 0.9706 - val_auc_1: 0.9843
Epoch 90/100
51/51 [==============================] - 9s 151ms/step - loss: 0.0804 -
binary_accuracy: 0.9890 - auc_1: 0.9985 - val_loss: 0.2159 -
val_binary_accuracy: 0.9681 - val_auc_1: 0.9824
Epoch 91/100
51/51 [==============================] - 9s 146ms/step - loss: 0.0764 -
binary_accuracy: 0.9877 - auc_1: 0.9987 - val_loss: 0.2128 -
val_binary_accuracy: 0.9681 - val_auc_1: 0.9841
Epoch 92/100
51/51 [==============================] - 9s 151ms/step - loss: 0.0741 -
binary_accuracy: 0.9884 - auc_1: 0.9988 - val_loss: 0.2149 -
val_binary_accuracy: 0.9706 - val_auc_1: 0.9868
Epoch 93/100
51/51 [==============================] - 10s 157ms/step - loss: 0.0906 -
binary_accuracy: 0.9798 - auc_1: 0.9989 - val_loss: 0.2773 -
val_binary_accuracy: 0.9632 - val_auc_1: 0.9805
Epoch 94/100
51/51 [==============================] - 9s 150ms/step - loss: 0.0787 -
binary_accuracy: 0.9877 - auc_1: 0.9985 - val_loss: 0.2520 -
val_binary_accuracy: 0.9657 - val_auc_1: 0.9848
Epoch 95/100
51/51 [==============================] - 10s 150ms/step - loss: 0.0745 -
binary_accuracy: 0.9884 - auc_1: 0.9995 - val_loss: 0.2501 -
val_binary_accuracy: 0.9632 - val_auc_1: 0.9850
Epoch 96/100
51/51 [==============================] - 9s 149ms/step - loss: 0.0752 -
binary_accuracy: 0.9865 - auc_1: 0.9989 - val_loss: 0.2697 -
val_binary_accuracy: 0.9657 - val_auc_1: 0.9809

Epoch 00096: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
```

look at the loss curves

[38]:
```python
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
```

```
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
↪ls='--')
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',␣
↪color='red', ls=':')
plt.legend()
plt.show()
```



[39]:
```
loss, binary_accuracy, auc = model.evaluate(test_ds, verbose=1)
loss, binary_accuracy, auc
```

```
32/32 [==============================] - 2s 63ms/step - loss: 0.1271 -
binary_accuracy: 0.9791 - auc_1: 0.9969
```

[39]:
```
(0.12712262570858002, 0.9791252613067627, 0.9968735575675964)
```

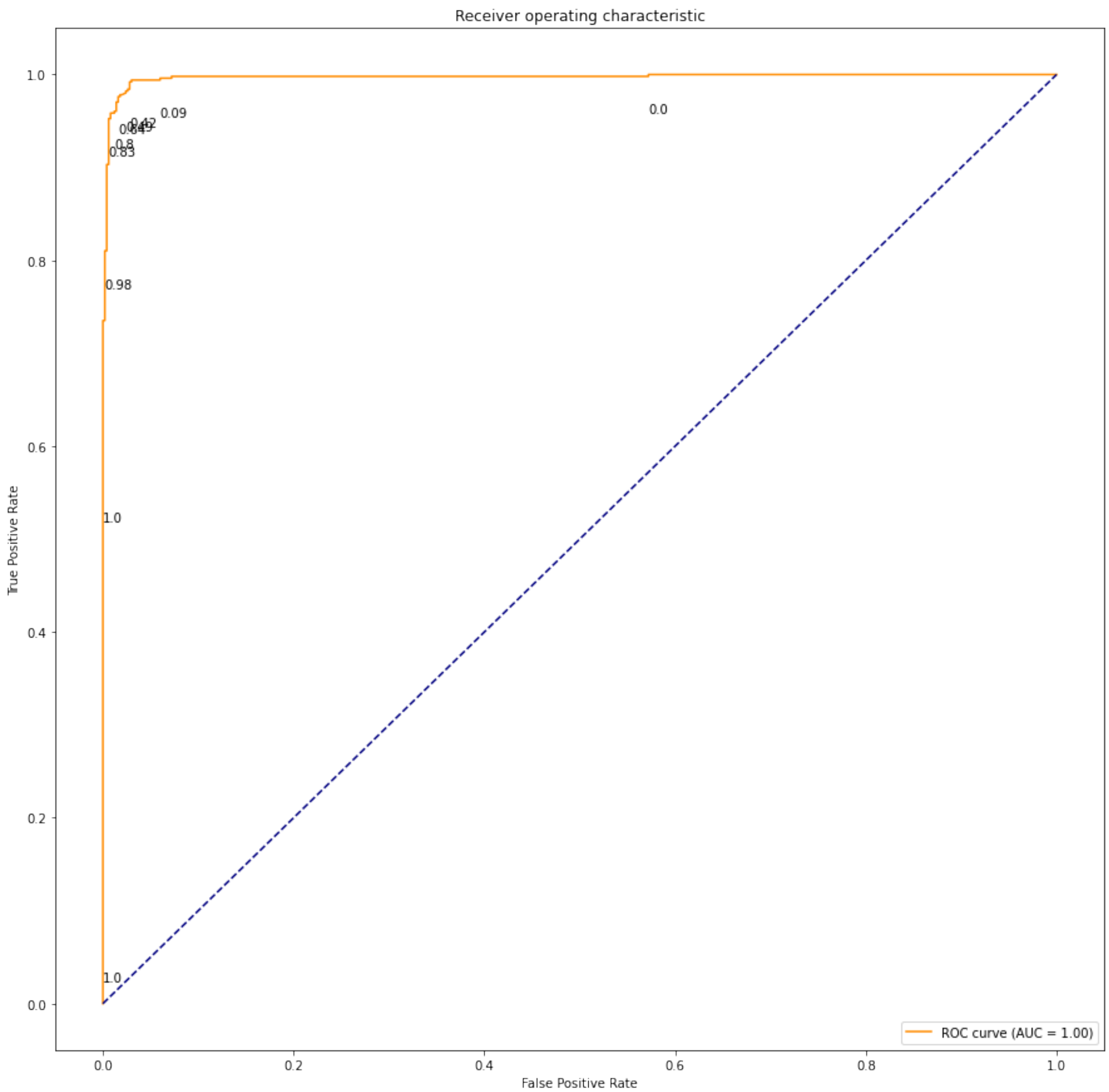
let's look at the ROC curve

[40]:
```
scores, labels = [], []
for b, (x_batch, y_batch) in enumerate((test_ds)):
    batch_scores = model.predict(x_batch)
    scores.append(batch_scores)
    labels.append(y_batch)

    if b >= len(test_ds):
        break
scores = np.concatenate(scores).squeeze()
```

```
labels = np.concatenate(labels).squeeze()
```

[41]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import roc_curve, auc, accuracy_score


fpr, tpr, thresholds = roc_curve(labels, scores)
roc_auc = auc(fpr, tpr)
# accuracy = accuracy_score(labels, scores)
plt.figure(figsize=(15, 15))

plt.plot(fpr, tpr, color='darkorange', label='ROC curve (AUC = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
i=0
for x, y, txt in zip(fpr, tpr, thresholds):
    i+=1
    if i%4==0:
        plt.annotate(np.round(txt,2), (x, y-0.04))

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
```

[41]:
```
<matplotlib.legend.Legend at 0x7f34497bf410>
```



Our model bacame really good at differentiating buildings and street images from food. We will nog apply this model on our trip advisor dataset to remove the non food images.

[42]:
```
model.save_weights('model.h5')
```