fc_project_li (copy)

August 15, 2021

[51]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly
import plotly.express as px
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, plot, iplot
import cv2
import random
import os
import glob
import tensorflow.keras
import seaborn as sns
from tensorflow.keras.applications.resnet50 import ResNet50
from keras_applications import xception
from tqdm.notebook import tqdm
import albumentations as A
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense,␣
↪Dropout , BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg16 import VGG16
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import roc_curve, roc_auc_score, auc

from sklearn.utils.multiclass import unique_labels
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, fbeta_score, classification_report
from sklearn.metrics import roc_curve, precision_recall_curve, roc_auc_score
from tensorflow.keras.callbacks import ReduceLROnPlateau
import torch
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
import time
```

[178]:
```
device=torch.device('cuda')
print('Num GPUs available', len(tf.config.experimental.
 ↪list_physical_devices('GPU')))
```

```
Num GPUs available 1
```

[53]:
```
cwd=os.getcwd()
wd=cwd+'/chest_xray'
```

[54]:
```
train_data=glob.glob(wd+'/train/**/*.jpeg')
test_data=glob.glob(wd+'/test/**/*.jpeg')
val_data=glob.glob(wd+'/val/**/*.jpeg')
```

[55]:
```
print("//"*20)
print(f"Training Set has: {len(train_data)} images")
print(f"Testing Set has: {len(test_data)} images")
print(f"Validation Set has: {len(val_data)} images")
print("//"*20)
```

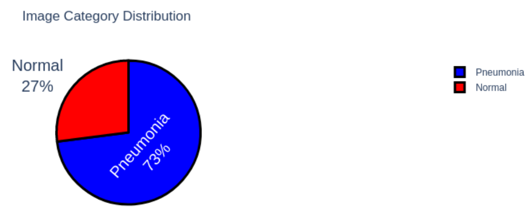
```
////////////////////////////////////////
Training Set has: 5216 images
Testing Set has: 624 images
Validation Set has: 16 images
////////////////////////////////////////
```

[56]:
```
sets = ["train", "test", "val"]
all_pneumonia = []
all_normal = []
results={}
results['model']=[]
results['total_param']=[]
results['time']=[]
results['score']=[]

for cat in sets:
    path = os.path.join(wd, cat)
    norm = glob.glob(os.path.join(path, "NORMAL/*.jpeg"))
    pneu = glob.glob(os.path.join(path, "PNEUMONIA/*.jpeg"))
    all_normal.extend(norm)
    all_pneumonia.extend(pneu)
print(" "*20)
print(f"Total Pneumonia Images: {len(all_pneumonia)}")
print(f"Total Normal Images: {len(all_normal)}")
print(" "*20)
```

```
Total Pneumonia Images: 4273
Total Normal Images: 1583
```

[57]:

```
## Plotly chart for Class distribution
labels = ["Normal",'Pneumonia ']
values = [len(all_normal), len(all_pneumonia)]
colors = ['red', 'blue']
fig = go.Figure(data=[go.Pie(labels=labels,
                             values=values)])
fig.update_traces(hoverinfo='value', textinfo='label+percent', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='#000000',␣
 ↪width=3)))
fig.update_layout(title="Image Category Distribution", title_x=0.5)
iplot(fig)
```

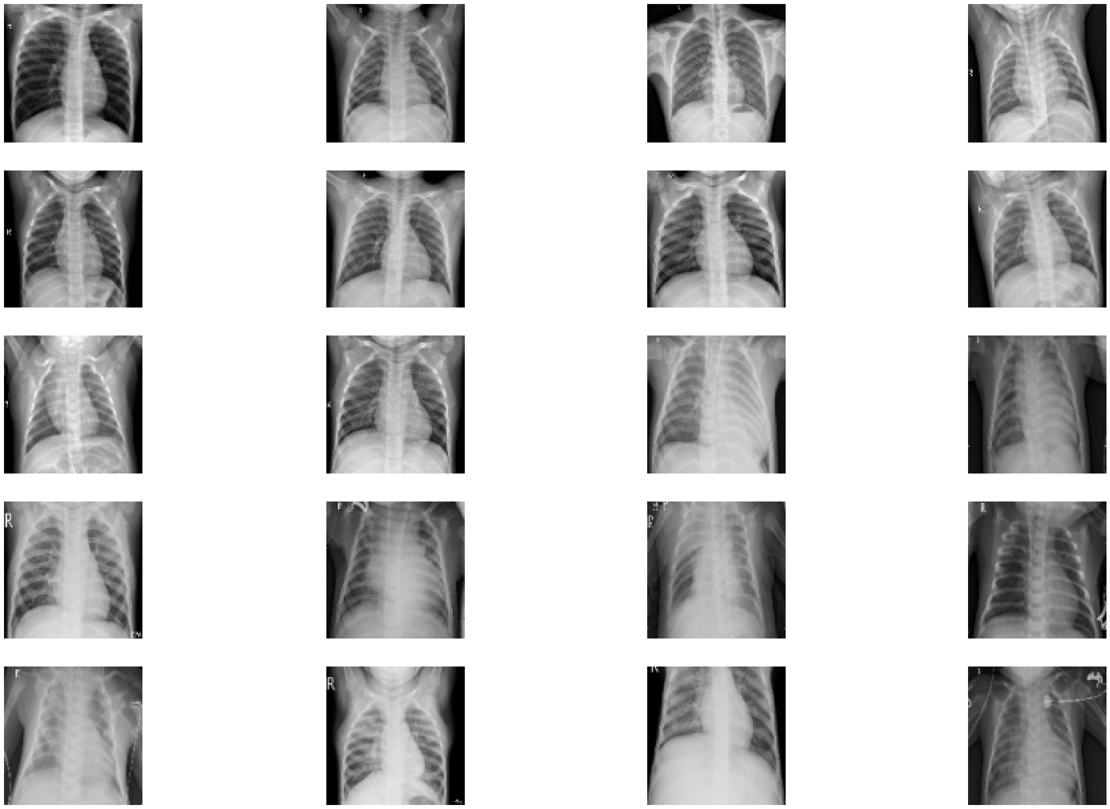


[58]:

```
random.shuffle(all_normal)
random.shuffle(all_pneumonia)
images = all_normal[:11] + all_pneumonia[:10]
```

[59]:

```
fig=plt.figure(figsize=(15, 10))
columns = 4; rows = 5
for i in range(1, columns*rows+1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (128, 128))
    fig.add_subplot(rows, columns, i)
    plt.imshow(img)
    plt.axis(False)
```



1 Base Model

[60]:
```
#data augmentation
train_gen = ImageDataGenerator(
        rescale=1/255.,
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the␣
 ↪dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range (degrees, 0␣
 ↪to 180)
        zoom_range = 0.4, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction␣
 ↪of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction␣
 ↪of total height)
        horizontal_flip = True,  # randomly flip images
        vertical_flip=False,
```

```
        validation_split=0.2) #validation size
test_gen=ImageDataGenerator(
        rescale=1/255.,
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the␣
 ↪dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        horizontal_flip = False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

[77]:
```
batch_size=64
nb_epochs=90
train_generator = train_gen.flow_from_directory(
    wd+"/train",
    batch_size=batch_size,
    target_size=(224, 224), #class_mode="binary"
  #  class_mode='binary',
    subset='training'
)

validation_generator = train_gen.flow_from_directory(
    wd+"/train", # same directory as training data
    batch_size=batch_size,
    target_size=(224, 224),
 #   class_mode='binary',
    subset='validation') # set as validation data
test_generator=test_gen.flow_from_directory(
    wd+"/test", # same directory as training data
    batch_size=1,
    target_size=(224, 224),
    shuffle=False)
```

```
Found 4173 images belonging to 2 classes.
Found 1043 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

[62]:
```
model = Sequential()
model.add(Conv2D(32,(3,3),strides=(1, 1),activation='relu',padding='same',␣
 ↪input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3),strides=(1, 1) ,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(128,(3,3),strides=(1, 1),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(256,(3,3),strides=(1, 1),padding='same', activation='relu'))
```

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 112, 112, 64)      18496
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 56, 56, 128)       73856
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 28, 28, 256)       295168
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 14, 14, 256)       0
_________________________________________________________________
flatten_4 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_12 (Dense)             (None, 128)               6422656
_________________________________________________________________
dense_13 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_14 (Dense)             (None, 2)                 130
=================================================================
Total params: 6,819,458
Trainable params: 6,819,458
Non-trainable params: 0
_________________________________________________________________
```

[64]:
```
## Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[65]:
```
early_stopping_cb = tf.keras.callbacks.
↪EarlyStopping(patience=5,restore_best_weights=True)
```

```
start=time.time()
history_base = model.fit_generator(
    train_generator,
    epochs = nb_epochs,
    validation_data=validation_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_steps = validation_generator.samples // batch_size,
    callbacks=[early_stopping_cb],
    verbose=1)
run_time=time.time()-start
```

```
Epoch 1/90
65/65 [==============================] - 73s 1s/step - loss: 0.6386 - accuracy:
0.6670 - val_loss: 0.4465 - val_accuracy: 0.7695
Epoch 2/90
65/65 [==============================] - 72s 1s/step - loss: 0.4128 - accuracy:
0.8038 - val_loss: 0.4406 - val_accuracy: 0.8076
Epoch 3/90
65/65 [==============================] - 72s 1s/step - loss: 0.3380 - accuracy:
0.8420 - val_loss: 0.3415 - val_accuracy: 0.8281
Epoch 4/90
65/65 [==============================] - 72s 1s/step - loss: 0.3384 - accuracy:
0.8502 - val_loss: 0.3706 - val_accuracy: 0.8125
Epoch 5/90
65/65 [==============================] - 72s 1s/step - loss: 0.3088 - accuracy:
0.8628 - val_loss: 0.3360 - val_accuracy: 0.8369
Epoch 6/90
65/65 [==============================] - 72s 1s/step - loss: 0.2778 - accuracy:
0.8751 - val_loss: 0.3408 - val_accuracy: 0.8281
Epoch 7/90
65/65 [==============================] - 73s 1s/step - loss: 0.2877 - accuracy:
0.8818 - val_loss: 0.2947 - val_accuracy: 0.8721
Epoch 8/90
65/65 [==============================] - 72s 1s/step - loss: 0.2526 - accuracy:
0.8893 - val_loss: 0.2901 - val_accuracy: 0.8750
Epoch 9/90
65/65 [==============================] - 72s 1s/step - loss: 0.2605 - accuracy:
0.8899 - val_loss: 0.2696 - val_accuracy: 0.8740
Epoch 10/90
65/65 [==============================] - 72s 1s/step - loss: 0.2233 - accuracy:
0.9071 - val_loss: 0.2904 - val_accuracy: 0.8643
Epoch 11/90
65/65 [==============================] - 73s 1s/step - loss: 0.2166 - accuracy:
0.9127 - val_loss: 0.3759 - val_accuracy: 0.8672
Epoch 12/90
65/65 [==============================] - 72s 1s/step - loss: 0.2131 - accuracy:
0.9134 - val_loss: 0.2718 - val_accuracy: 0.8701
```

```
Epoch 13/90
65/65 [==============================] - 72s 1s/step - loss: 0.1961 - accuracy:
0.9254 - val_loss: 0.2658 - val_accuracy: 0.8896
Epoch 14/90
65/65 [==============================] - 73s 1s/step - loss: 0.2063 - accuracy:
0.9152 - val_loss: 0.2518 - val_accuracy: 0.8887
Epoch 15/90
65/65 [==============================] - 73s 1s/step - loss: 0.1735 - accuracy:
0.9290 - val_loss: 0.2071 - val_accuracy: 0.8994
Epoch 16/90
65/65 [==============================] - 72s 1s/step - loss: 0.1903 - accuracy:
0.9255 - val_loss: 0.2388 - val_accuracy: 0.8916
Epoch 17/90
65/65 [==============================] - 73s 1s/step - loss: 0.1674 - accuracy:
0.9362 - val_loss: 0.1923 - val_accuracy: 0.9189
Epoch 18/90
65/65 [==============================] - 73s 1s/step - loss: 0.1703 - accuracy:
0.9323 - val_loss: 0.2189 - val_accuracy: 0.9189
Epoch 19/90
65/65 [==============================] - 73s 1s/step - loss: 0.1608 - accuracy:
0.9299 - val_loss: 0.1756 - val_accuracy: 0.9248
Epoch 20/90
65/65 [==============================] - 72s 1s/step - loss: 0.1528 - accuracy:
0.9384 - val_loss: 0.1793 - val_accuracy: 0.9229
Epoch 21/90
65/65 [==============================] - 73s 1s/step - loss: 0.1474 - accuracy:
0.9442 - val_loss: 0.1797 - val_accuracy: 0.9297
Epoch 22/90
65/65 [==============================] - 72s 1s/step - loss: 0.1444 - accuracy:
0.9414 - val_loss: 0.1848 - val_accuracy: 0.9219
Epoch 23/90
65/65 [==============================] - 73s 1s/step - loss: 0.1515 - accuracy:
0.9418 - val_loss: 0.1706 - val_accuracy: 0.9209
Epoch 24/90
65/65 [==============================] - 72s 1s/step - loss: 0.1417 - accuracy:
0.9494 - val_loss: 0.1822 - val_accuracy: 0.9238
Epoch 25/90
65/65 [==============================] - 72s 1s/step - loss: 0.1363 - accuracy:
0.9457 - val_loss: 0.2172 - val_accuracy: 0.9131
Epoch 26/90
65/65 [==============================] - 73s 1s/step - loss: 0.1395 - accuracy:
0.9458 - val_loss: 0.3130 - val_accuracy: 0.8750
Epoch 27/90
65/65 [==============================] - 72s 1s/step - loss: 0.1778 - accuracy:
0.9325 - val_loss: 0.1891 - val_accuracy: 0.9131
Epoch 28/90
65/65 [==============================] - 72s 1s/step - loss: 0.1400 - accuracy:
0.9406 - val_loss: 0.1916 - val_accuracy: 0.9092
```

[66]:

```
score = model.evaluate_generator(test_generator,  verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.4663000702857971
Test accuracy: 0.807692289352417
```

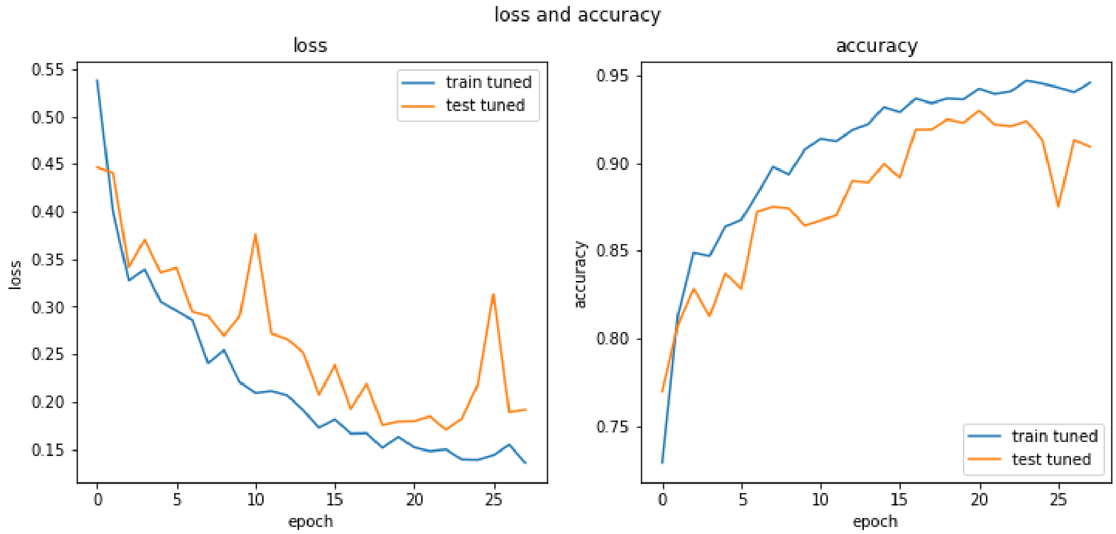
[67]:

```
#saving the model and histories
#tf.keras.models.save_model(filepath=cwd, model=model)
model.save(cwd+'/assets/model_base')
np.save(cwd+'/assets/history_base.npy', history_base.history)
results['model'].append('base_model')
results['total_param'].append(6819458)
results['time'].append(run_time)
results['score'].append(score[1])
```

```
INFO:tensorflow:Assets written to:
/home/jovyan/work/fc_project/assets/model_base/assets
```

[68]:

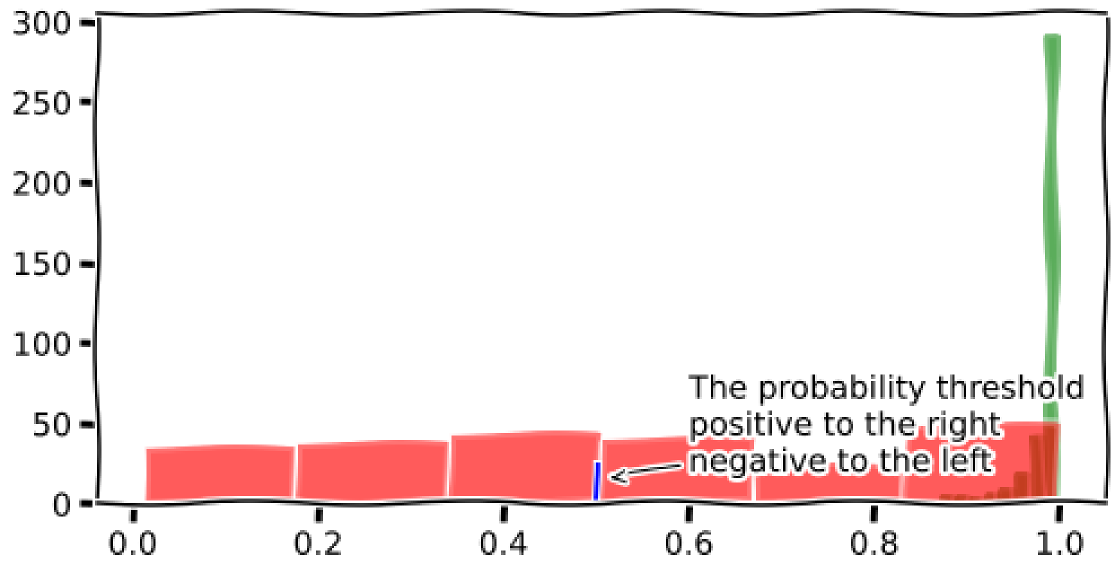
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))

ax1.plot(history_base.history['loss'])
ax1.plot(history_base.history['val_loss'])
ax1.set_title('loss')
ax1.set_ylabel('loss')
ax1.set_xlabel('epoch')
ax1.legend(['train tuned', 'test tuned','train','test'], loc='upper right')
ax2.plot(history_base.history['accuracy'])
ax2.plot(history_base.history['val_accuracy'])

ax2.set_title('accuracy')
ax2.set_ylabel('accuracy')
ax2.set_xlabel('epoch')
ax2.legend(['train tuned', 'test tuned','train','test'], loc='lower right')
fig.suptitle('loss and accuracy')
plt.show()
```




```
[78]: STEP_SIZE_TEST=624
test_generator.reset()
preds = model.predict(test_generator,
steps=STEP_SIZE_TEST,
verbose=1)
```

624/624 [==============================] - 7s 7ms/step

```
[79]: #scale the output of the model to [0,1]
#scaled_preds=(preds[:,1]-min(preds[:,1]))/(max(preds[:,1])-min(preds[:,1]))
pos = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 1]
neg = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 0]

with plt.xkcd():
  fig = plt.figure(figsize=(8, 4))

  sns.distplot(pos, hist = True, kde = False, color='g',
               kde_kws = {'shade': True, 'linewidth': 3})

  sns.distplot(neg, hist = True, kde = False, color='r',
               kde_kws = {'shade': True, 'linewidth': 3})

  plt.plot([0.5, 0.5], [0, 25], '-b')
  plt.annotate(
        'The probability threshold\npositive to the right\nnegative to the␣
↪left',
        xy=(0.51, 15), arrowprops=dict(arrowstyle='->'), xytext=(0.6, 20))

plt.show()
```



[80]:
```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    Given a scikit-learn confusion matrix (CM), make a nice plot.

    Arguments

   ---------
    cm:           Confusion matrix from sklearn.metrics.confusion_matrix

    target_names: Given classification classes, such as [0, 1, 2]
                  The class names, for example, ['high', 'medium', 'low']

    title:        The text to display at the top of the matrix

    cmap:         The gradient of the values displayed from matplotlib.pyplot.cm
                  See http://matplotlib.org/examples/color/colormaps_reference.
↪html
                  `plt.get_cmap('jet')` or `plt.cm.Blues`

    normalize:    If `False`, plot the raw numbers
                  If `True`, plot the proportions

    Usage
    -----
```

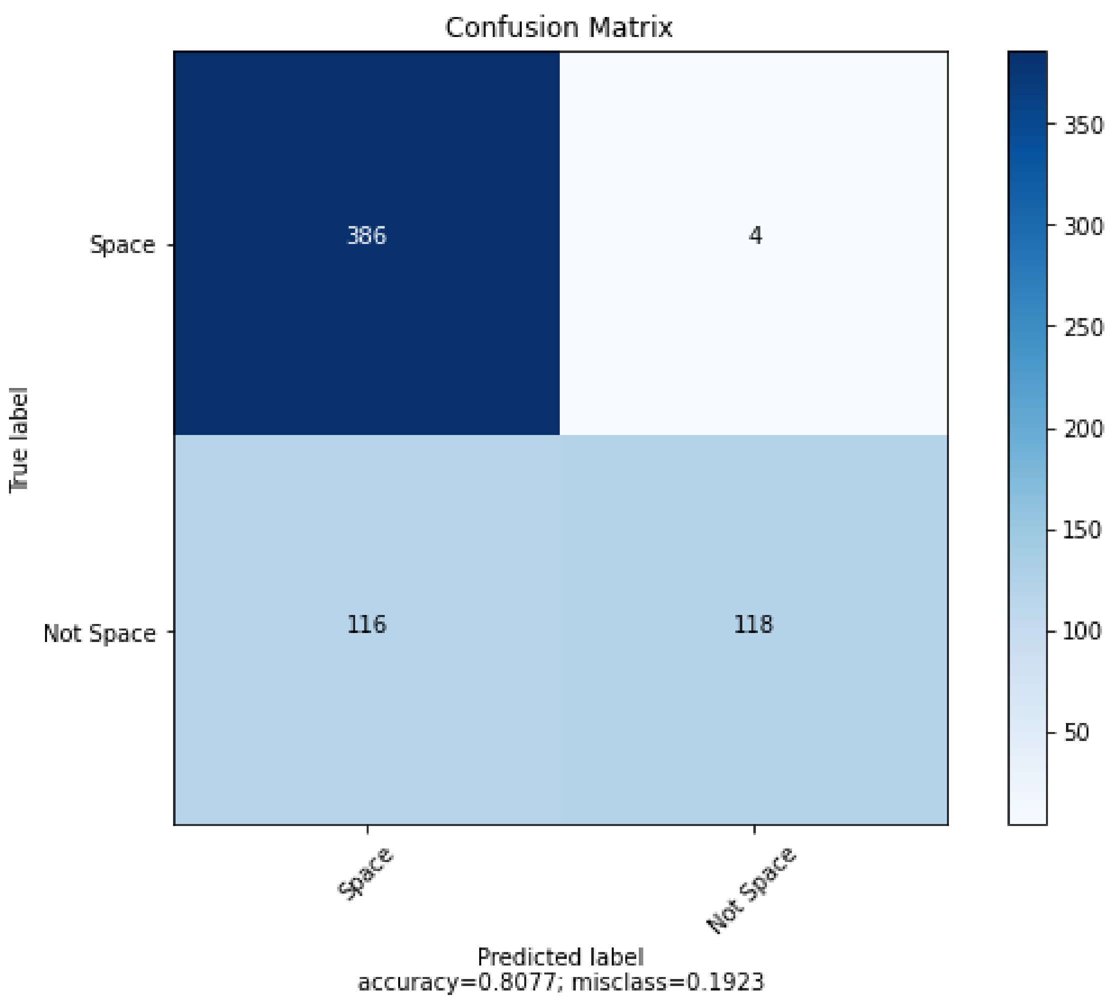
```
    plot_confusion_matrix(cm           = cm,                  # Confusion␣
↪matrix created by
                                                              # `sklearn.
↪metrics.confusion_matrix`
                          normalize    = True,                # Show proportions
                          target_names = y_labels_vals,       # List of names␣
↪of the classes
                          title        = best_estimator_name) # Title of graph

    Citation
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/
↪plot_confusion_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
```

```
                    color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.
 ↪format(accuracy, misclass))
    plt.show()
```

[81]:
```
confusion = confusion_matrix(test_generator.classes, np.where(preds[:,1]>0.
 ↪5,1,0), labels=[1, 0])
print(confusion)
```

```
[[386   4]
 [116 118]]
```

[82]:
```
plot_confusion_matrix(cm=confusion, target_names = ['Space', 'Not Space'],␣
 ↪title = 'Confusion Matrix',normalize=False)
```



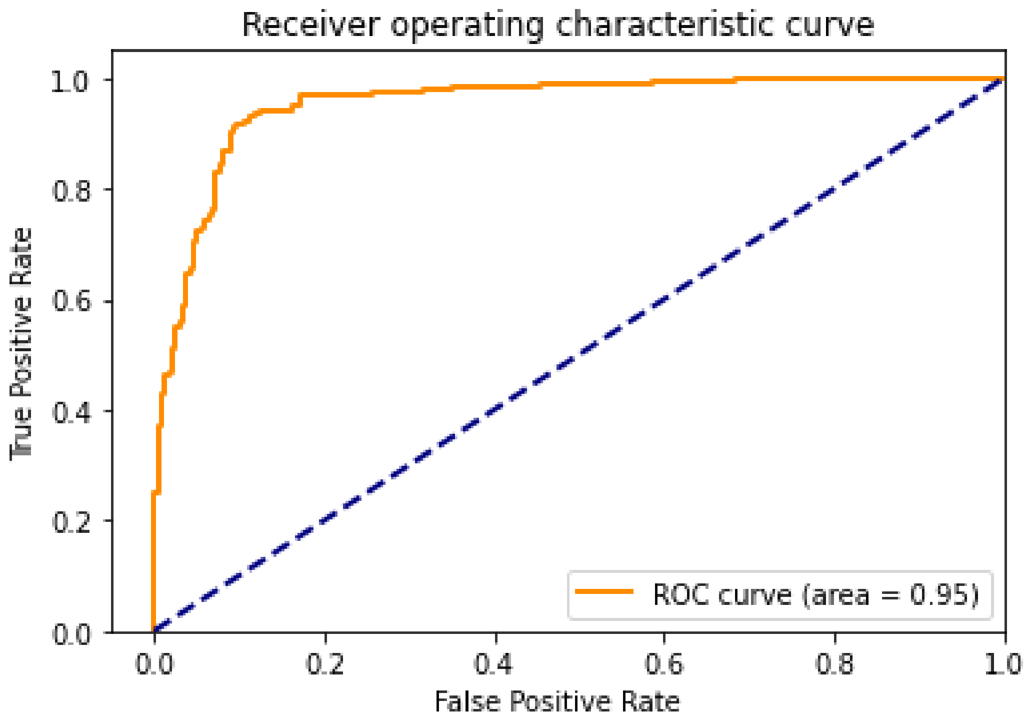
[83]:

```
from sklearn.metrics import classification_report

target_names = ['negative', 'positive']
print(classification_report(test_generator.classes, np.where(preds[:,1]>0.
↪5,1,0) , target_names=target_names))
```

```
              precision    recall  f1-score   support

    negative       0.97      0.50      0.66       234
    positive       0.77      0.99      0.87       390

    accuracy                           0.81       624
   macro avg       0.87      0.75      0.76       624
weighted avg       0.84      0.81      0.79       624
```

[84]:

```
fpr, tpr, _ = roc_curve(test_generator.classes, preds[:,1])
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```



[ ]:

2 Xception

[121]:
```
#data augmentation
train_gen = ImageDataGenerator(
        rescale=1/255.,
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the␣
 ↪dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range (degrees, 0␣
 ↪to 180)
        zoom_range = 0.4, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction␣
 ↪of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction␣
 ↪of total height)
        horizontal_flip = True,  # randomly flip images
        vertical_flip=False,
```

```
        validation_split=0.2) #validation size
test_gen=ImageDataGenerator(
        rescale=1/255.,
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the␣
 ↪dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        horizontal_flip = False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

[122]:
```
batch_size=16
nb_epochs=90
train_generator = train_gen.flow_from_directory(
    wd+"/train",
    batch_size=batch_size,
    target_size=(224, 224), #class_mode="binary"
  #  class_mode='binary',
    subset='training'
)

validation_generator = train_gen.flow_from_directory(
    wd+"/train", # same directory as training data
    batch_size=batch_size,
    target_size=(224, 224),
 #   class_mode='binary',
    subset='validation') # set as validation data
test_generator=test_gen.flow_from_directory(
    wd+"/test", # same directory as training data
    batch_size=1,
    target_size=(224, 224),
    shuffle=False)
```

```
Found 4173 images belonging to 2 classes.
Found 1043 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

[123]:
```
##First, instantiate a base model with pre-trained weights.
base_model = tf.keras.applications.Xception(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(224, 224, 3),
    include_top=False,
)  # Do not include the ImageNet classifier at the top.

# Freeze the base_model
base_model.trainable = False
```

```
# Create new model on top
inputs = tf.keras.Input(shape=(224, 224, 3))
#x = tf.data_augmentation(inputs)  # Apply random data augmentation

# Pre-trained Xception weights requires that input be normalized
# from (0, 255) to a range (-1., +1.), the normalization layer
# does the following, outputs = (inputs - mean) / sqrt(var)
norm_layer = tf.keras.layers.experimental.preprocessing.Normalization()
mean = np.array([0.5] * 3)
var = mean ** 2
# Scale inputs to [-1, +1]
x = norm_layer(inputs)
norm_layer.set_weights([mean, var])
# The base model contains batchnorm layers. We want to keep them in inference␣
↪mode
# when we unfreeze the base model for fine-tuning, so we make sure that the
# base_model is running in inference mode here.
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.2)(x)  # Regularize with dropout
outputs = tf.keras.layers.Dense(1)(x)
model_base = tf.keras.Model(inputs, outputs)

early_stopping_cb = tf.keras.callbacks.
↪EarlyStopping(patience=10,restore_best_weights=True)
model=Sequential()
model.add(model_base)
model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(2,activation='softmax'))

## Freezing the layers
#for layer in base_model.layers:
 #   layer.trainable=False

model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

model_base.summary()
start=time.time()
history_xception = model.
↪fit_generator(train_generator,epochs=90,validation_data=validation_generator,steps_per_epoc
run_1=time.time()-start
```

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
normalization_4 (Normalizati (None, 224, 224, 3)       7
_________________________________________________________________
xception (Functional)        (None, 7, 7, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_5 ( (None, 2048)              0
_________________________________________________________________
dropout_8 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_34 (Dense)             (None, 1)                 2049
=================================================================
Total params: 20,863,536
Trainable params: 2,049
Non-trainable params: 20,861,487
_________________________________________________________________
Epoch 1/90
260/260 [==============================] - 74s 280ms/step - loss: 0.3883 -
accuracy: 0.8114 - val_loss: 0.2270 - val_accuracy: 0.8993
Epoch 2/90
260/260 [==============================] - 72s 278ms/step - loss: 0.2231 -
accuracy: 0.9030 - val_loss: 0.2072 - val_accuracy: 0.9166
Epoch 3/90
260/260 [==============================] - 72s 277ms/step - loss: 0.2061 -
accuracy: 0.9141 - val_loss: 0.2020 - val_accuracy: 0.9166
Epoch 4/90
260/260 [==============================] - 72s 279ms/step - loss: 0.2014 -
accuracy: 0.9199 - val_loss: 0.1801 - val_accuracy: 0.9319
Epoch 5/90
260/260 [==============================] - 72s 279ms/step - loss: 0.1951 -
accuracy: 0.9223 - val_loss: 0.1834 - val_accuracy: 0.9204
Epoch 6/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1550 -
accuracy: 0.9383 - val_loss: 0.1791 - val_accuracy: 0.9175
Epoch 7/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1779 -
accuracy: 0.9296 - val_loss: 0.1696 - val_accuracy: 0.9281
Epoch 8/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1833 -
accuracy: 0.9245 - val_loss: 0.1814 - val_accuracy: 0.9204
Epoch 9/90
260/260 [==============================] - 73s 282ms/step - loss: 0.1806 -
accuracy: 0.9264 - val_loss: 0.1994 - val_accuracy: 0.9156
Epoch 10/90
```

```
260/260 [==============================] - 74s 284ms/step - loss: 0.1701 -
accuracy: 0.9354 - val_loss: 0.1800 - val_accuracy: 0.9214
Epoch 11/90
260/260 [==============================] - 72s 279ms/step - loss: 0.1630 -
accuracy: 0.9407 - val_loss: 0.1774 - val_accuracy: 0.9300
Epoch 12/90
260/260 [==============================] - 73s 279ms/step - loss: 0.1637 -
accuracy: 0.9342 - val_loss: 0.1658 - val_accuracy: 0.9386
Epoch 13/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1683 -
accuracy: 0.9394 - val_loss: 0.1547 - val_accuracy: 0.9415
Epoch 14/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1629 -
accuracy: 0.9370 - val_loss: 0.1572 - val_accuracy: 0.9434
Epoch 15/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1587 -
accuracy: 0.9352 - val_loss: 0.1608 - val_accuracy: 0.9367
Epoch 16/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1715 -
accuracy: 0.9311 - val_loss: 0.1636 - val_accuracy: 0.9204
Epoch 17/90
260/260 [==============================] - 73s 279ms/step - loss: 0.1804 -
accuracy: 0.9261 - val_loss: 0.1833 - val_accuracy: 0.9243
Epoch 18/90
260/260 [==============================] - 72s 279ms/step - loss: 0.1780 -
accuracy: 0.9300 - val_loss: 0.1638 - val_accuracy: 0.9319
Epoch 19/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1598 -
accuracy: 0.9418 - val_loss: 0.1752 - val_accuracy: 0.9348
Epoch 20/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1675 -
accuracy: 0.9396 - val_loss: 0.1545 - val_accuracy: 0.9415
Epoch 21/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1610 -
accuracy: 0.9427 - val_loss: 0.1649 - val_accuracy: 0.9358
Epoch 22/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1597 -
accuracy: 0.9393 - val_loss: 0.1645 - val_accuracy: 0.9338
Epoch 23/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1448 -
accuracy: 0.9432 - val_loss: 0.1347 - val_accuracy: 0.9521
Epoch 24/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1644 -
accuracy: 0.9336 - val_loss: 0.1722 - val_accuracy: 0.9233
Epoch 25/90
260/260 [==============================] - 72s 276ms/step - loss: 0.1801 -
accuracy: 0.9311 - val_loss: 0.1673 - val_accuracy: 0.9291
Epoch 26/90
```

```
260/260 [==============================] - 72s 277ms/step - loss: 0.1579 -
accuracy: 0.9398 - val_loss: 0.1403 - val_accuracy: 0.9473
Epoch 27/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1618 -
accuracy: 0.9419 - val_loss: 0.1684 - val_accuracy: 0.9358
Epoch 28/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1597 -
accuracy: 0.9343 - val_loss: 0.1443 - val_accuracy: 0.9415
Epoch 29/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1673 -
accuracy: 0.9296 - val_loss: 0.1438 - val_accuracy: 0.9406
Epoch 30/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1594 -
accuracy: 0.9352 - val_loss: 0.1320 - val_accuracy: 0.9463
Epoch 31/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1595 -
accuracy: 0.9363 - val_loss: 0.1655 - val_accuracy: 0.9348
Epoch 32/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1606 -
accuracy: 0.9398 - val_loss: 0.1575 - val_accuracy: 0.9396
Epoch 33/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1592 -
accuracy: 0.9370 - val_loss: 0.1595 - val_accuracy: 0.9348
Epoch 34/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1608 -
accuracy: 0.9434 - val_loss: 0.1371 - val_accuracy: 0.9501
Epoch 35/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1644 -
accuracy: 0.9386 - val_loss: 0.1613 - val_accuracy: 0.9386
Epoch 36/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1666 -
accuracy: 0.9291 - val_loss: 0.1588 - val_accuracy: 0.9386
Epoch 37/90
260/260 [==============================] - 72s 279ms/step - loss: 0.1475 -
accuracy: 0.9425 - val_loss: 0.1609 - val_accuracy: 0.9367
Epoch 38/90
260/260 [==============================] - 73s 280ms/step - loss: 0.1427 -
accuracy: 0.9425 - val_loss: 0.1511 - val_accuracy: 0.9348
Epoch 39/90
260/260 [==============================] - 73s 279ms/step - loss: 0.1589 -
accuracy: 0.9378 - val_loss: 0.1521 - val_accuracy: 0.9425
Epoch 40/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1570 -
accuracy: 0.9420 - val_loss: 0.1627 - val_accuracy: 0.9329
```

[124]:
```
# Unfreeze the base_model. Note that it keeps running in inference mode
# since we passed `training=False` when calling it. This means that
```

```
# the batchnorm layers will not update their batch statistics.
# This prevents the batchnorm layers from undoing all the training
# we've done so far.
base_model.trainable = True
model.summary()

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),  # Low learning rate
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

epochs = 10
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_4 (Functional)         (None, 1)                 20863536
_________________________________________________________________
flatten_10 (Flatten)         (None, 1)                 0
_________________________________________________________________
dense_35 (Dense)             (None, 128)               256
_________________________________________________________________
dense_36 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_37 (Dense)             (None, 2)                 130
=================================================================
Total params: 20,872,178
Trainable params: 20,817,643
Non-trainable params: 54,535
_________________________________________________________________
```

[125]:
```
start=time.time()
history_xception_tuning = model.
↪fit_generator(train_generator,epochs=10,validation_data=validation_generator)
runtime=time.time()-start+run_1
```

```
Epoch 1/10
261/261 [==============================] - 90s 335ms/step - loss: 0.1605 -
accuracy: 0.9443 - val_loss: 0.1123 - val_accuracy: 0.9501
Epoch 2/10
261/261 [==============================] - 88s 335ms/step - loss: 0.1147 -
accuracy: 0.9554 - val_loss: 0.1328 - val_accuracy: 0.9463
Epoch 3/10
261/261 [==============================] - 88s 335ms/step - loss: 0.0929 -
accuracy: 0.9653 - val_loss: 0.0914 - val_accuracy: 0.9664
```

```
Epoch 4/10
261/261 [==============================] - 88s 337ms/step - loss: 0.0837 -
accuracy: 0.9693 - val_loss: 0.0811 - val_accuracy: 0.9703
Epoch 5/10
261/261 [==============================] - 88s 338ms/step - loss: 0.0854 -
accuracy: 0.9664 - val_loss: 0.0745 - val_accuracy: 0.9693
Epoch 6/10
261/261 [==============================] - 88s 337ms/step - loss: 0.0775 -
accuracy: 0.9691 - val_loss: 0.0560 - val_accuracy: 0.9789
Epoch 7/10
261/261 [==============================] - 88s 338ms/step - loss: 0.0596 -
accuracy: 0.9765 - val_loss: 0.0603 - val_accuracy: 0.9770
Epoch 8/10
261/261 [==============================] - 88s 337ms/step - loss: 0.0555 -
accuracy: 0.9781 - val_loss: 0.0725 - val_accuracy: 0.9732
Epoch 9/10
261/261 [==============================] - 88s 336ms/step - loss: 0.0521 -
accuracy: 0.9813 - val_loss: 0.0701 - val_accuracy: 0.9770
Epoch 10/10
261/261 [==============================] - 88s 337ms/step - loss: 0.0464 -
accuracy: 0.9844 - val_loss: 0.0714 - val_accuracy: 0.9760
```

[126]:
```
score = model.evaluate(test_generator,  verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

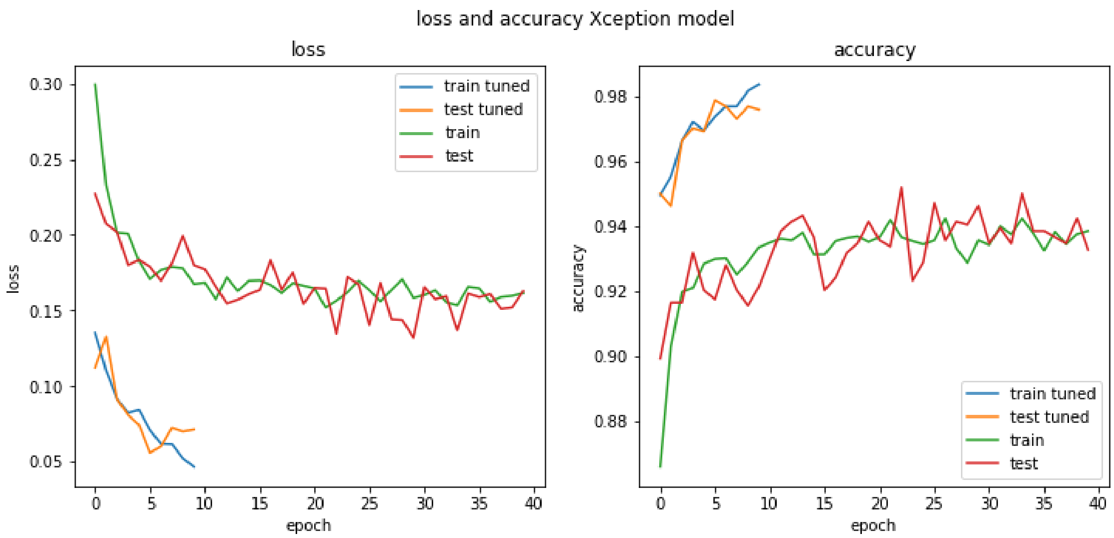
```
Test score: 0.1737084835767746
Test accuracy: 0.932692289352417
```

[127]:
```
#saving the model and histories
#tf.keras.models.save_model(filepath=cwd, model=model)
model.save(cwd+'/assets/model_xception')
np.save(cwd+'/assets/history_xception.npy', history_xception.history)
np.save(cwd+'/assets/history_xception_tuning.npy', history_xception_tuning.
 ↪history)
results['model'].append('Xception2')
results['total_param'].append(20863536)
results['time'].append(runtime)
results['score'].append(score[1])
np.save(cwd+'/assets/results.npy', results)
```

```
INFO:tensorflow:Assets written to:
/home/jovyan/work/fc_project/assets/model_xception/assets
```

[128]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
ax1.plot(history_xception_tuning.history['loss'])
ax1.plot(history_xception_tuning.history['val_loss'])
ax1.plot(history_xception.history['loss'])
```

```
ax1.plot(history_xception.history['val_loss'])
ax1.set_title('loss')
ax1.set_ylabel('loss')
ax1.set_xlabel('epoch')
ax1.legend(['train tuned', 'test tuned','train','test'], loc='upper right')
ax2.plot(history_xception_tuning.history['accuracy'])
ax2.plot(history_xception_tuning.history['val_accuracy'])
ax2.plot(history_xception.history['accuracy'])
ax2.plot(history_xception.history['val_accuracy'])

ax2.set_title('accuracy')
ax2.set_ylabel('accuracy')
ax2.set_xlabel('epoch')
ax2.legend(['train tuned', 'test tuned','train','test'], loc='lower right')
fig.suptitle('loss and accuracy Xception model')
plt.show()
```



[ ]:

[ ]:

[ ]:

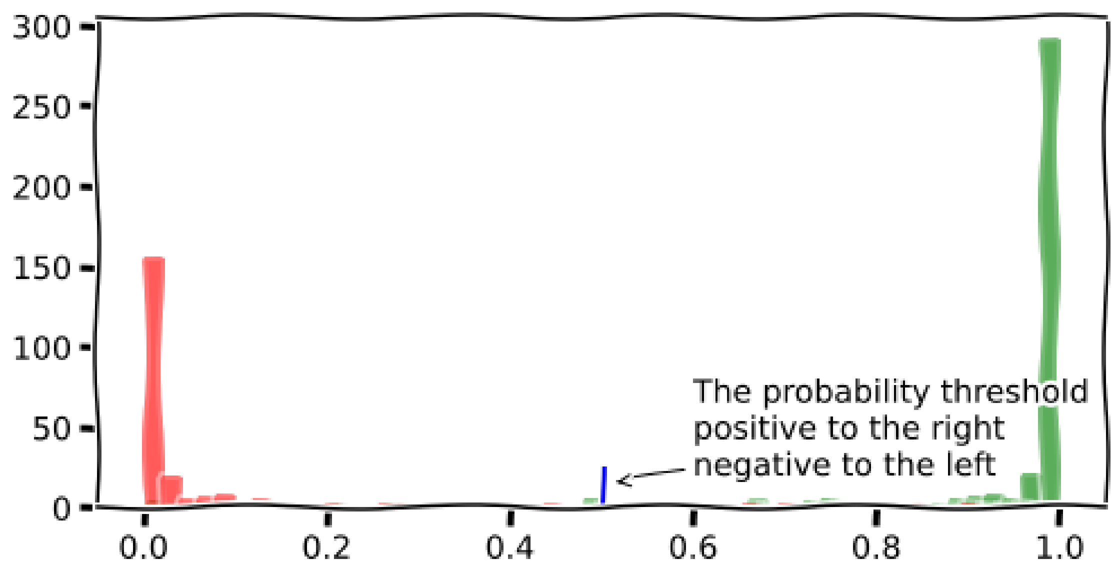
[ ]:

[129]:
```
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
test_generator.reset()
preds = model.predict(test_generator,
steps=STEP_SIZE_TEST,
```

```
verbose=1)
```

624/624 [==============================] - 6s 9ms/step

[130]:
```
#scale the output of the model to [0,1]
#scaled_preds=(preds[:,1]-min(preds[:,1]))/(max(preds[:,1])-min(preds[:,1]))
pos = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 1]
neg = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 0]

with plt.xkcd():
  fig = plt.figure(figsize=(8, 4))

  sns.distplot(pos, hist = True, kde = False, color='g',
               kde_kws = {'shade': True, 'linewidth': 3})

  sns.distplot(neg, hist = True, kde = False, color='r',
               kde_kws = {'shade': True, 'linewidth': 3})

  plt.plot([0.5, 0.5], [0, 25], '-b')
  plt.annotate(
        'The probability threshold\npositive to the right\nnegative to the␣
↪left',
        xy=(0.51, 15), arrowprops=dict(arrowstyle='->'), xytext=(0.6, 20))

plt.show()
```



[ ]:

```
[ ]:
```

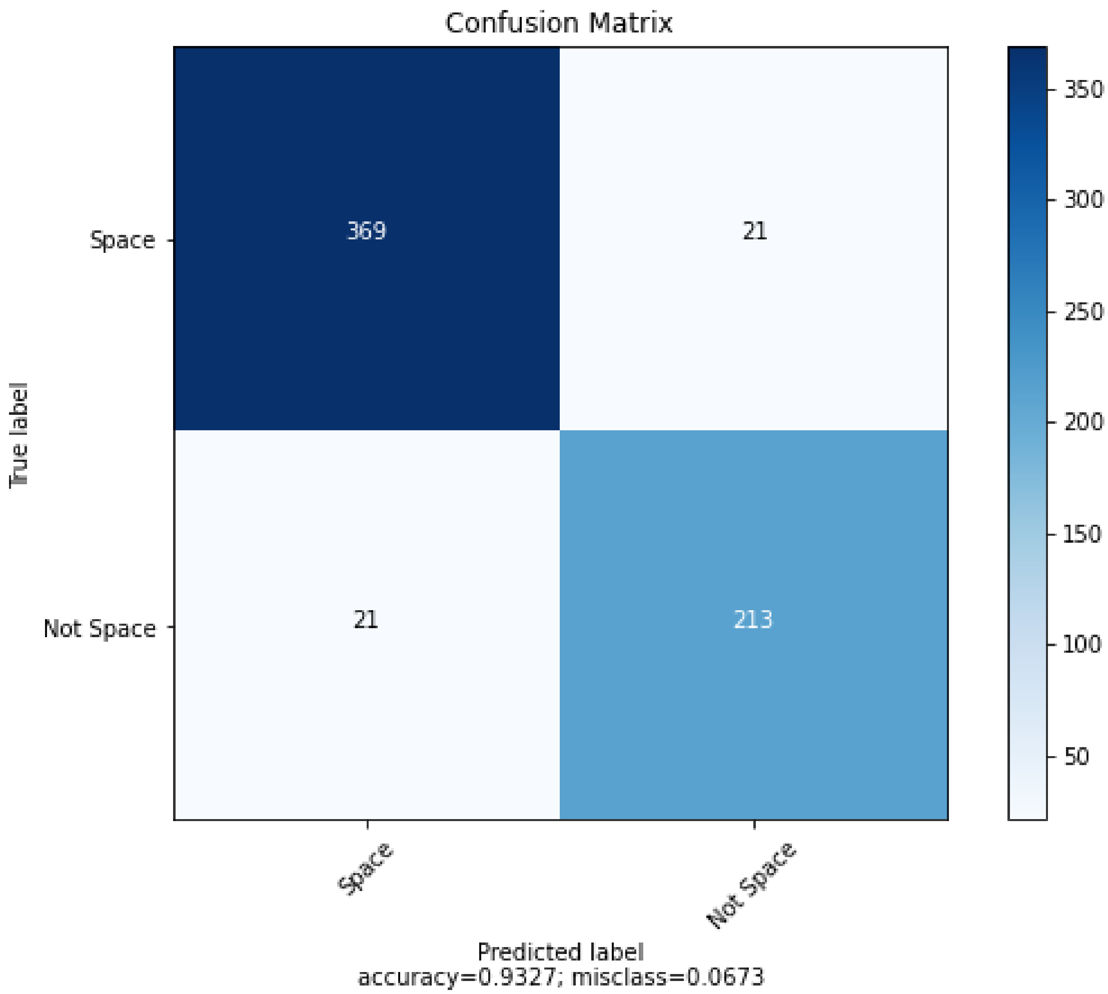
```
[131]: confusion = confusion_matrix(test_generator.classes, np.where(preds[:,1]>0.
        ↪5,1,0), labels=[1, 0])
       print(confusion)
```

```
[[369  21]
 [ 21 213]]
```

```
[132]: def plot_confusion_matrix(cm,
                                 target_names,
                                 title='Confusion matrix',
                                 cmap=None,
                                 normalize=True):
           """
           Given a scikit-learn confusion matrix (CM), make a nice plot.

           Arguments

          ---------
           cm:           Confusion matrix from sklearn.metrics.confusion_matrix

           target_names: Given classification classes, such as [0, 1, 2]
                         The class names, for example, ['high', 'medium', 'low']

           title:        The text to display at the top of the matrix

           cmap:         The gradient of the values displayed from matplotlib.pyplot.cm
                         See http://matplotlib.org/examples/color/colormaps_reference.
        ↪html
                         `plt.get_cmap('jet')` or `plt.cm.Blues`

           normalize:    If `False`, plot the raw numbers
                         If `True`, plot the proportions

           Usage
           -----
           plot_confusion_matrix(cm           = cm,                  # Confusion␣
        ↪matrix created by
                                                                     # `sklearn.
        ↪metrics.confusion_matrix`
                                 normalize    = True,                # Show proportions
                                 target_names = y_labels_vals,       # List of names␣
        ↪of the classes
                                 title        = best_estimator_name) # Title of graph

           Citation
```

```
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/
↪plot_confusion_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.
↪format(accuracy, misclass))
    plt.show()
```

[133]:
```
plot_confusion_matrix(cm=confusion, target_names = ['Space', 'Not Space'],␣
↪title = 'Confusion Matrix',normalize=False)
```



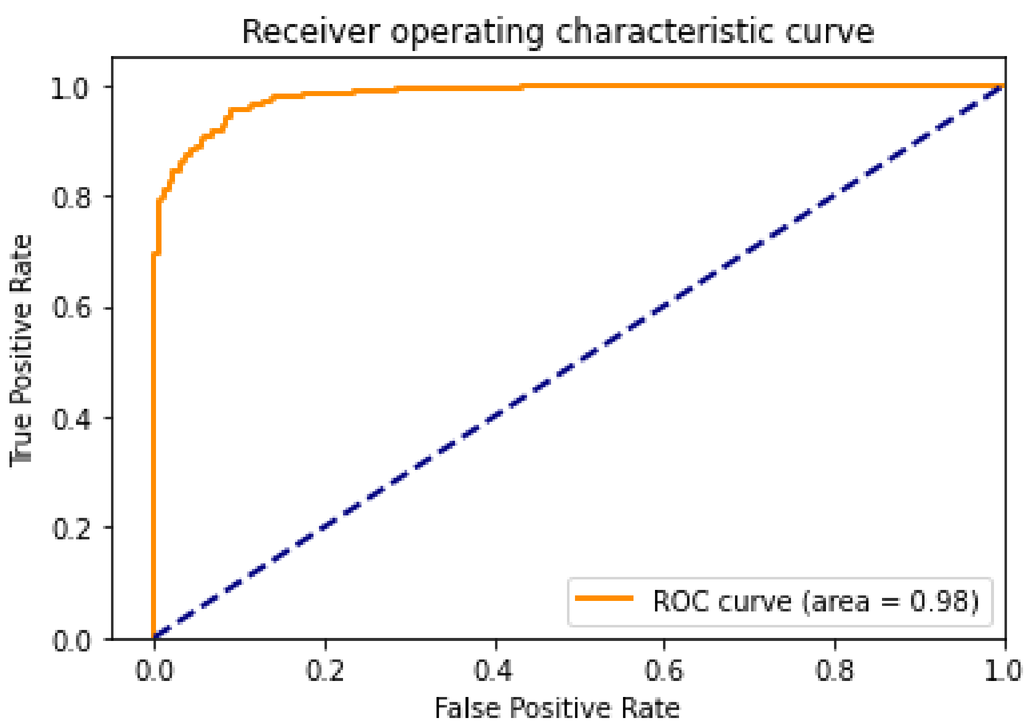
[134]:
```
from sklearn.metrics import classification_report

target_names = ['negative', 'positive']
print(classification_report(test_generator.classes, np.where(preds[:,1]>0.
↪5,1,0) , target_names=target_names))
```

```
              precision    recall  f1-score   support

    negative       0.91      0.91      0.91       234
    positive       0.95      0.95      0.95       390

    accuracy                           0.93       624
   macro avg       0.93      0.93      0.93       624
```

```
weighted avg       0.93      0.93      0.93       624
```

[135]:

```
fpr, tpr, _ = roc_curve(test_generator.classes, preds[:,1])
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```



3 RESNET50

[186]:
```
#data augmentation
train_gen = ImageDataGenerator(
    #    rescale=1/255,
        dtype = 'float32',
   #    featurewise_center=False,  # set input mean to 0 over the dataset
   #    samplewise_center=False,  # set each sample mean to 0
   #    featurewise_std_normalization=False,  # divide inputs by std of the␣
↪dataset
   #    samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range (degrees, 0␣
↪to 180)
        zoom_range = 0.4, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction␣
↪of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction␣
↪of total height)
        horizontal_flip = True,  # randomly flip images
        vertical_flip=False,
        preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
        validation_split=0.2) #validation size
test_gen=ImageDataGenerator(
   #     rescale=1/255,
        dtype = 'float32',
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the␣
↪dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        horizontal_flip = False,  # randomly flip images
        preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
        vertical_flip=False)  # randomly flip images
```

[187]:
```
batch_size=16
nb_epochs=90
train_generator = train_gen.flow_from_directory(
    wd+"/train",
    batch_size=batch_size,
    target_size=(224, 224), #class_mode="binary"
  #  class_mode='binary',
    subset='training'
)

validation_generator = train_gen.flow_from_directory(
```

```
    wd+"/train", # same directory as training data
    batch_size=batch_size,
    target_size=(224, 224),
 #   class_mode='binary',
    subset='validation') # set as validation data
test_generator=test_gen.flow_from_directory(
    wd+"/test", # same directory as training data
    batch_size=1,
    target_size=(224, 224),
    shuffle=False)
```

```
Found 4173 images belonging to 2 classes.
Found 1043 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

[188]:
```
##First, instantiate a base model with pre-trained weights.
base_model = tf.keras.applications.ResNet50(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(224, 224, 3),
    include_top=False,
)  # Do not include the ImageNet classifier at the top.

# Freeze the base_model
base_model.trainable = False

# Create new model on top
inputs = tf.keras.Input(shape=(224, 224, 3))
#x = tf.data_augmentation(inputs)  # Apply random data augmentation


x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.2)(x)  # Regularize with dropout
outputs = tf.keras.layers.Dense(1)(x)
model_base = tf.keras.Model(inputs, outputs)

early_stopping_cb = tf.keras.callbacks.
 ↪EarlyStopping(patience=10,restore_best_weights=True)
model=Sequential()
model.add(model_base)
model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(2,activation='softmax'))

## Freezing the layers
```

```python
#for layer in base_model.layers:
 #   layer.trainable=False

model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

model_base.summary()
start=time.time()
history_resnet50 = model.
 ↪fit_generator(train_generator,epochs=90,validation_data=validation_generator,steps_per_epoc
run_1=time.time()-start
```

```
Model: "model_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_39 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
resnet50 (Functional)        (None, 7, 7, 2048)        23587712
_________________________________________________________________
global_average_pooling2d_11  (None, 2048)              0
_________________________________________________________________
dropout_14 (Dropout)         (None, 2048)              0
_________________________________________________________________
dense_56 (Dense)             (None, 1)                 2049
=================================================================
Total params: 23,589,761
Trainable params: 2,049
Non-trainable params: 23,587,712
_________________________________________________________________
Epoch 1/90
260/260 [==============================] - 75s 282ms/step - loss: 0.4017 -
accuracy: 0.7947 - val_loss: 0.1992 - val_accuracy: 0.9252
Epoch 2/90
260/260 [==============================] - 72s 279ms/step - loss: 0.1732 -
accuracy: 0.9302 - val_loss: 0.1591 - val_accuracy: 0.9348
Epoch 3/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1661 -
accuracy: 0.9344 - val_loss: 0.1506 - val_accuracy: 0.9300
Epoch 4/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1392 -
accuracy: 0.9486 - val_loss: 0.1261 - val_accuracy: 0.9511
Epoch 5/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1552 -
accuracy: 0.9397 - val_loss: 0.1194 - val_accuracy: 0.9549
Epoch 6/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1338 -
```

```
accuracy: 0.9452 - val_loss: 0.1170 - val_accuracy: 0.9569
Epoch 7/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1255 -
accuracy: 0.9525 - val_loss: 0.1055 - val_accuracy: 0.9578
Epoch 8/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1263 -
accuracy: 0.9524 - val_loss: 0.1044 - val_accuracy: 0.9626
Epoch 9/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1212 -
accuracy: 0.9532 - val_loss: 0.1061 - val_accuracy: 0.9530
Epoch 10/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1177 -
accuracy: 0.9603 - val_loss: 0.1085 - val_accuracy: 0.9607
Epoch 11/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1375 -
accuracy: 0.9427 - val_loss: 0.1081 - val_accuracy: 0.9626
Epoch 12/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1136 -
accuracy: 0.9562 - val_loss: 0.0913 - val_accuracy: 0.9712
Epoch 13/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1153 -
accuracy: 0.9568 - val_loss: 0.1119 - val_accuracy: 0.9540
Epoch 14/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1083 -
accuracy: 0.9545 - val_loss: 0.1041 - val_accuracy: 0.9540
Epoch 15/90
260/260 [==============================] - 73s 281ms/step - loss: 0.1169 -
accuracy: 0.9524 - val_loss: 0.0971 - val_accuracy: 0.9616
Epoch 16/90
260/260 [==============================] - 74s 286ms/step - loss: 0.1179 -
accuracy: 0.9560 - val_loss: 0.1026 - val_accuracy: 0.9578
Epoch 17/90
260/260 [==============================] - 73s 280ms/step - loss: 0.1209 -
accuracy: 0.9541 - val_loss: 0.1350 - val_accuracy: 0.9492
Epoch 18/90
260/260 [==============================] - 73s 282ms/step - loss: 0.1088 -
accuracy: 0.9575 - val_loss: 0.1516 - val_accuracy: 0.9367
Epoch 19/90
260/260 [==============================] - 73s 282ms/step - loss: 0.1153 -
accuracy: 0.9591 - val_loss: 0.1133 - val_accuracy: 0.9521
Epoch 20/90
260/260 [==============================] - 73s 279ms/step - loss: 0.1120 -
accuracy: 0.9548 - val_loss: 0.1071 - val_accuracy: 0.9607
Epoch 21/90
260/260 [==============================] - 72s 276ms/step - loss: 0.1207 -
accuracy: 0.9535 - val_loss: 0.1096 - val_accuracy: 0.9559
Epoch 22/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1090 -
```

```
accuracy: 0.9548 - val_loss: 0.0897 - val_accuracy: 0.9674
Epoch 23/90
260/260 [==============================] - 71s 274ms/step - loss: 0.1131 -
accuracy: 0.9553 - val_loss: 0.1333 - val_accuracy: 0.9463
Epoch 24/90
260/260 [==============================] - 72s 278ms/step - loss: 0.1126 -
accuracy: 0.9558 - val_loss: 0.0963 - val_accuracy: 0.9569
Epoch 25/90
260/260 [==============================] - 72s 277ms/step - loss: 0.1278 -
accuracy: 0.9467 - val_loss: 0.1056 - val_accuracy: 0.9588
Epoch 26/90
260/260 [==============================] - 73s 279ms/step - loss: 0.1069 -
accuracy: 0.9557 - val_loss: 0.0894 - val_accuracy: 0.9645
Epoch 27/90
260/260 [==============================] - 73s 282ms/step - loss: 0.1020 -
accuracy: 0.9625 - val_loss: 0.0924 - val_accuracy: 0.9655
Epoch 28/90
260/260 [==============================] - 72s 276ms/step - loss: 0.1020 -
accuracy: 0.9627 - val_loss: 0.0959 - val_accuracy: 0.9626
Epoch 29/90
260/260 [==============================] - 72s 276ms/step - loss: 0.1032 -
accuracy: 0.9657 - val_loss: 0.0810 - val_accuracy: 0.9732
Epoch 30/90
260/260 [==============================] - 71s 273ms/step - loss: 0.1011 -
accuracy: 0.9595 - val_loss: 0.0853 - val_accuracy: 0.9684
Epoch 31/90
260/260 [==============================] - 71s 273ms/step - loss: 0.1289 -
accuracy: 0.9537 - val_loss: 0.1050 - val_accuracy: 0.9655
Epoch 32/90
260/260 [==============================] - 73s 280ms/step - loss: 0.1137 -
accuracy: 0.9569 - val_loss: 0.0862 - val_accuracy: 0.9655
Epoch 33/90
260/260 [==============================] - 71s 275ms/step - loss: 0.0990 -
accuracy: 0.9623 - val_loss: 0.1008 - val_accuracy: 0.9664
Epoch 34/90
260/260 [==============================] - 71s 272ms/step - loss: 0.1069 -
accuracy: 0.9598 - val_loss: 0.0972 - val_accuracy: 0.9626
Epoch 35/90
260/260 [==============================] - 71s 273ms/step - loss: 0.1059 -
accuracy: 0.9596 - val_loss: 0.1137 - val_accuracy: 0.9607
Epoch 36/90
260/260 [==============================] - 72s 275ms/step - loss: 0.1183 -
accuracy: 0.9567 - val_loss: 0.0892 - val_accuracy: 0.9655
Epoch 37/90
260/260 [==============================] - 71s 273ms/step - loss: 0.1100 -
accuracy: 0.9548 - val_loss: 0.0984 - val_accuracy: 0.9626
Epoch 38/90
260/260 [==============================] - 71s 272ms/step - loss: 0.1005 -
```

```
accuracy: 0.9624 - val_loss: 0.1070 - val_accuracy: 0.9588
Epoch 39/90
260/260 [==============================] - 71s 273ms/step - loss: 0.1125 -
accuracy: 0.9593 - val_loss: 0.0789 - val_accuracy: 0.9693
Epoch 40/90
260/260 [==============================] - 71s 273ms/step - loss: 0.1026 -
accuracy: 0.9590 - val_loss: 0.1092 - val_accuracy: 0.9569
Epoch 41/90
260/260 [==============================] - 73s 281ms/step - loss: 0.1042 -
accuracy: 0.9648 - val_loss: 0.1241 - val_accuracy: 0.9453
Epoch 42/90
260/260 [==============================] - 72s 276ms/step - loss: 0.1018 -
accuracy: 0.9645 - val_loss: 0.0802 - val_accuracy: 0.9703
Epoch 43/90
260/260 [==============================] - 71s 274ms/step - loss: 0.1135 -
accuracy: 0.9566 - val_loss: 0.0964 - val_accuracy: 0.9655
Epoch 44/90
260/260 [==============================] - 71s 273ms/step - loss: 0.1082 -
accuracy: 0.9627 - val_loss: 0.0842 - val_accuracy: 0.9703
Epoch 45/90
260/260 [==============================] - 71s 275ms/step - loss: 0.1004 -
accuracy: 0.9635 - val_loss: 0.0860 - val_accuracy: 0.9664
Epoch 46/90
260/260 [==============================] - 71s 274ms/step - loss: 0.1029 -
accuracy: 0.9586 - val_loss: 0.1046 - val_accuracy: 0.9607
Epoch 47/90
260/260 [==============================] - 72s 275ms/step - loss: 0.1147 -
accuracy: 0.9573 - val_loss: 0.0965 - val_accuracy: 0.9626
Epoch 48/90
260/260 [==============================] - 71s 273ms/step - loss: 0.0997 -
accuracy: 0.9633 - val_loss: 0.1320 - val_accuracy: 0.9434
Epoch 49/90
260/260 [==============================] - 71s 273ms/step - loss: 0.0923 -
accuracy: 0.9602 - val_loss: 0.0954 - val_accuracy: 0.9626
```

[189]:
```
#saving the model and histories
#tf.keras.models.save_model(filepath=cwd, model=model)
model.save(cwd+'/assets/model_resnet50')
np.save(cwd+'/assets/history_resnet50.npy', history_resnet50.history)
#np.save(cwd+'/assets/history_xception_tuning.npy', history_xception_tuning.
 ↪history)
```

```
INFO:tensorflow:Assets written to:
/home/jovyan/work/fc_project/assets/model_resnet50/assets
```

[190]:
```
# Unfreeze the base_model. Note that it keeps running in inference mode
# since we passed `training=False` when calling it. This means that
```

```
# the batchnorm layers will not update their batch statistics.
# This prevents the batchnorm layers from undoing all the training
# we've done so far.
base_model.trainable = True
model.summary()

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),  # Low learning rate
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

epochs = 10
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_8 (Functional)         (None, 1)                 23589761
_________________________________________________________________
flatten_16 (Flatten)         (None, 1)                 0
_________________________________________________________________
dense_57 (Dense)             (None, 128)               256
_________________________________________________________________
dense_58 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_59 (Dense)             (None, 2)                 130
=================================================================
Total params: 23,598,403
Trainable params: 23,545,283
Non-trainable params: 53,120
_________________________________________________________________
```

[191]:

```
start=time.time()
history_resnet50_tuning = model.
↪fit_generator(train_generator,epochs=10,validation_data=validation_generator)
runtime=time.time()-start+run_1
```

```
Epoch 1/10
261/261 [==============================] - 77s 279ms/step - loss: 0.1480 -
accuracy: 0.9447 - val_loss: 0.1170 - val_accuracy: 0.9569
Epoch 2/10
261/261 [==============================] - 72s 276ms/step - loss: 0.1225 -
accuracy: 0.9509 - val_loss: 0.0821 - val_accuracy: 0.9693
Epoch 3/10
261/261 [==============================] - 72s 276ms/step - loss: 0.0821 -
accuracy: 0.9696 - val_loss: 0.0694 - val_accuracy: 0.9732
```

```
Epoch 4/10
261/261 [==============================] - 72s 277ms/step - loss: 0.0779 -
accuracy: 0.9692 - val_loss: 0.0741 - val_accuracy: 0.9712
Epoch 5/10
261/261 [==============================] - 73s 277ms/step - loss: 0.0531 -
accuracy: 0.9805 - val_loss: 0.1266 - val_accuracy: 0.9540
Epoch 6/10
261/261 [==============================] - 73s 278ms/step - loss: 0.0736 -
accuracy: 0.9712 - val_loss: 0.0534 - val_accuracy: 0.9789
Epoch 7/10
261/261 [==============================] - 73s 277ms/step - loss: 0.0492 -
accuracy: 0.9839 - val_loss: 0.1070 - val_accuracy: 0.9540
Epoch 8/10
261/261 [==============================] - 73s 279ms/step - loss: 0.0347 -
accuracy: 0.9866 - val_loss: 0.0537 - val_accuracy: 0.9808
Epoch 9/10
261/261 [==============================] - 73s 278ms/step - loss: 0.0617 -
accuracy: 0.9789 - val_loss: 0.0469 - val_accuracy: 0.9818
Epoch 10/10
261/261 [==============================] - 73s 278ms/step - loss: 0.0470 -
accuracy: 0.9844 - val_loss: 0.0750 - val_accuracy: 0.9760
```

[192]:
```python
#saving the model and histories
#tf.keras.models.save_model(filepath=cwd, model=model)
model.save(cwd+'/assets/model_resnet50_tuning')
np.save(cwd+'/assets/history_resnet50_tuning.npy', history_resnet50_tuning.
 ↪history)
#np.save(cwd+'/assets/history_xception_tuning.npy', history_xception_tuning.
 ↪history)
```

```
INFO:tensorflow:Assets written to:
/home/jovyan/work/fc_project/assets/model_resnet50_tuning/assets
```

[201]:
```python
model=tf.keras.models.load_model(cwd+'/assets/model_resnet50_tuning')
history_resnet50_tuning=np.load(cwd+'/assets/history_resnet50_tuning.npy',␣
 ↪allow_pickle=True)
history_resnet50=np.load(cwd+'/assets/history_resnet50.npy', allow_pickle=True)
```

[194]:
```python

```

[202]:
```python
score = model.evaluate(test_generator,  verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

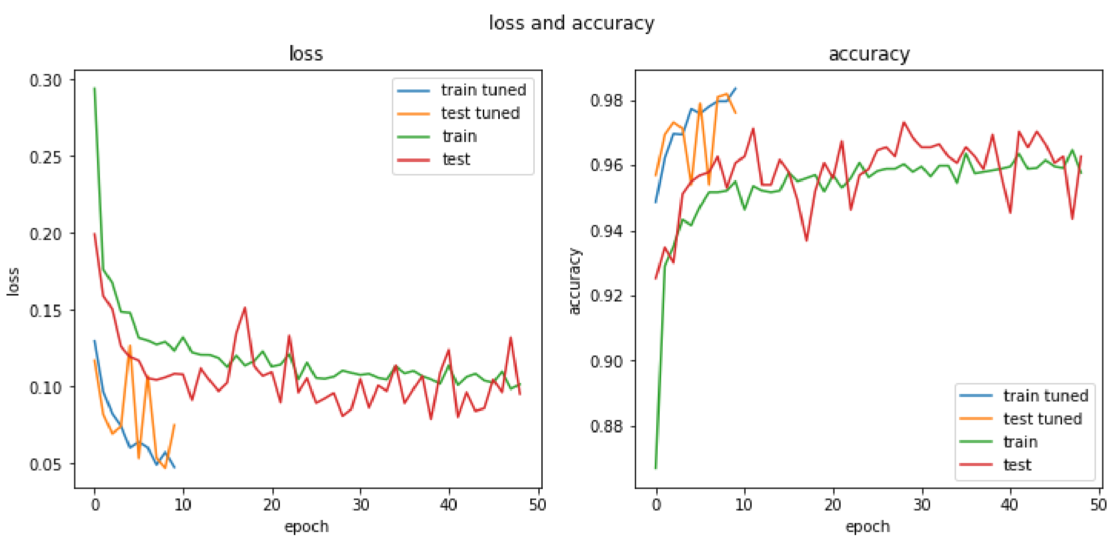
```
Test score: 0.2513222396373749
Test accuracy: 0.9070512652397156
```

[203]:
```
score = model_resnet50.evaluate(test_generator,  verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.21405178308486938
Test accuracy: 0.9086538553237915
```

[204]:
```
history_resnet50_tuning=history_resnet50_tuning.tolist()
history_resnet50=history_resnet50.tolist()
```

[205]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
ax1.plot(history_resnet50_tuning['loss'])
ax1.plot(history_resnet50_tuning['val_loss'])
ax1.plot(history_resnet50['loss'])
ax1.plot(history_resnet50['val_loss'])
ax1.set_title('loss')
ax1.set_ylabel('loss')
ax1.set_xlabel('epoch')
ax1.legend(['train tuned', 'test tuned','train','test'], loc='upper right')
ax2.plot(history_resnet50_tuning['accuracy'])
ax2.plot(history_resnet50_tuning['val_accuracy'])
ax2.plot(history_resnet50['accuracy'])
ax2.plot(history_resnet50['val_accuracy'])

ax2.set_title('accuracy')
ax2.set_ylabel('accuracy')
ax2.set_xlabel('epoch')
ax2.legend(['train tuned', 'test tuned','train','test'], loc='lower right')
fig.suptitle('loss and accuracy')
plt.show()
```

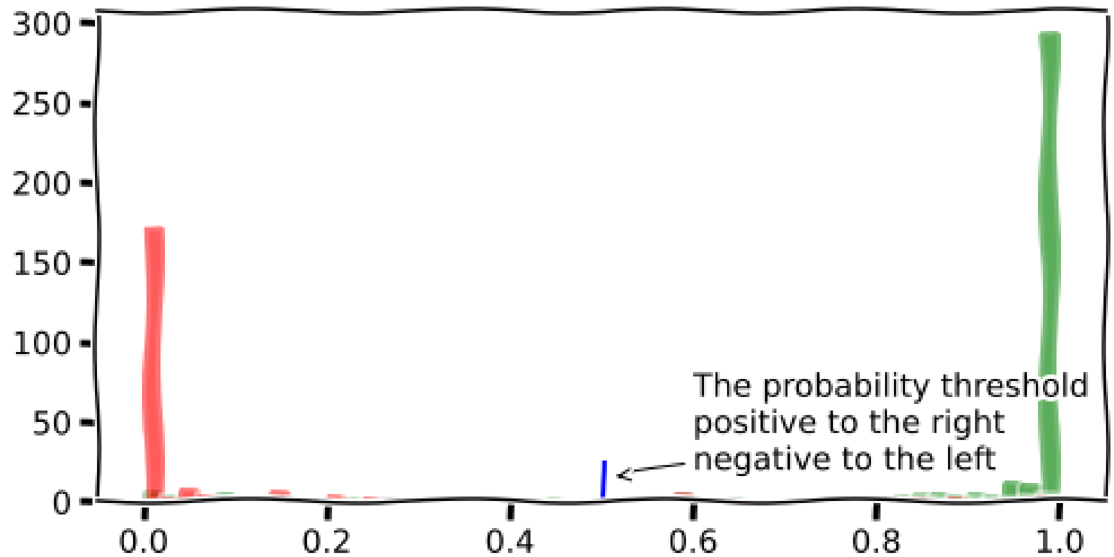


[206]:
```
results['model'].append('resnet50')
results['total_param'].append(23598403)
results['time'].append(runtime)
results['score'].append(score[1])
np.save(cwd+'/assets/results.npy', results)
```

[207]:
```
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
test_generator.reset()
preds = model.predict(test_generator,
steps=STEP_SIZE_TEST,
verbose=1)
```

```
624/624 [==============================] - 6s 9ms/step
```

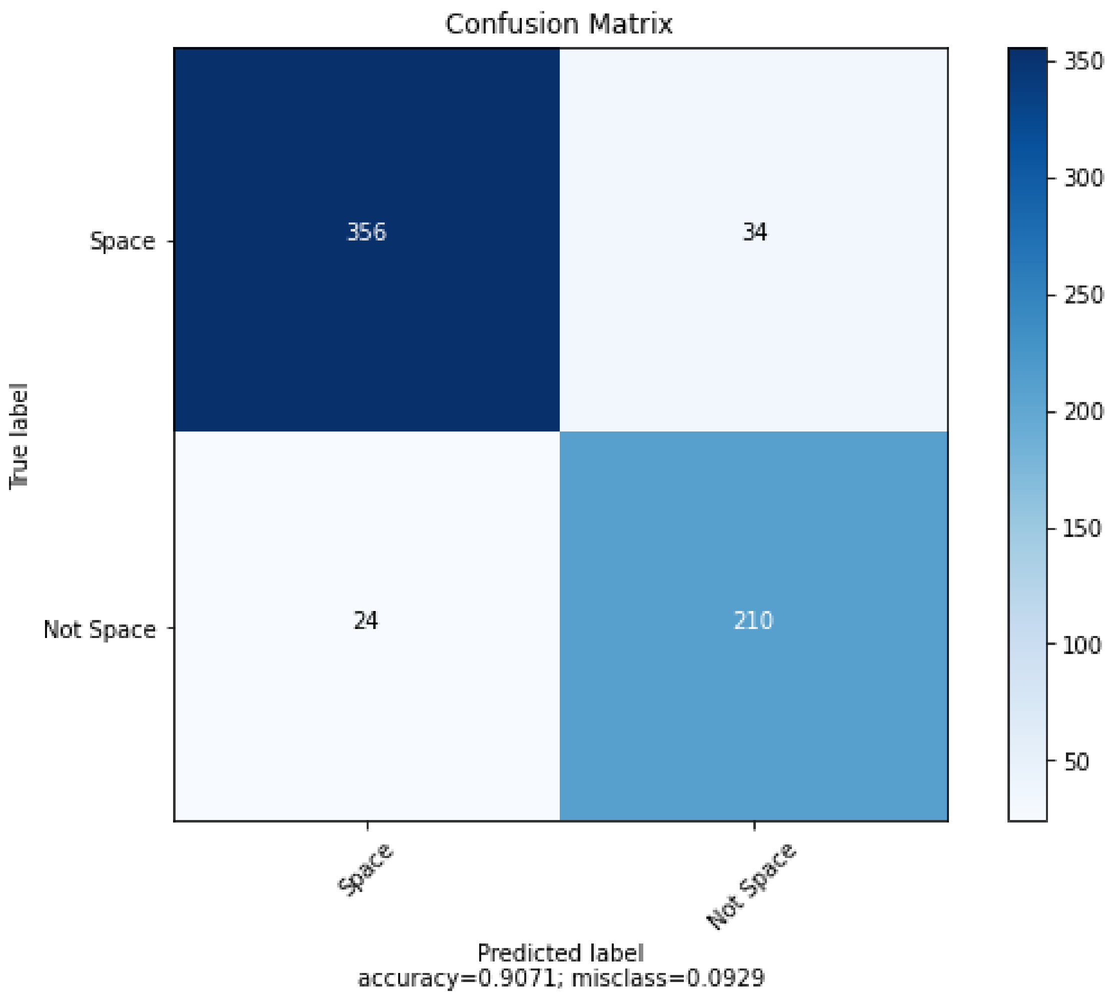
[208]:
```
#scale the output of the model to [0,1]
#scaled_preds=(preds[:,1]-min(preds[:,1]))/(max(preds[:,1])-min(preds[:,1]))
pos = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 1]
neg = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 0]

with plt.xkcd():
  fig = plt.figure(figsize=(8, 4))

  sns.distplot(pos, hist = True, kde = False, color='g',
               kde_kws = {'shade': True, 'linewidth': 3})

  sns.distplot(neg, hist = True, kde = False, color='r',
               kde_kws = {'shade': True, 'linewidth': 3})

  plt.plot([0.5, 0.5], [0, 25], '-b')
  plt.annotate(
        'The probability threshold\npositive to the right\nnegative to the␣
↪left',
        xy=(0.51, 15), arrowprops=dict(arrowstyle='->'), xytext=(0.6, 20))

plt.show()
```



[209]:
```
confusion = confusion_matrix(test_generator.classes, np.where(preds[:,1]>0.
↪5,1,0), labels=[1, 0])
print(confusion)
```

```
[[356  34]
 [ 24 210]]
```

[210]:
```
plot_confusion_matrix(cm=confusion, target_names = ['Space', 'Not Space'],␣
↪title = 'Confusion Matrix',normalize=False)
```

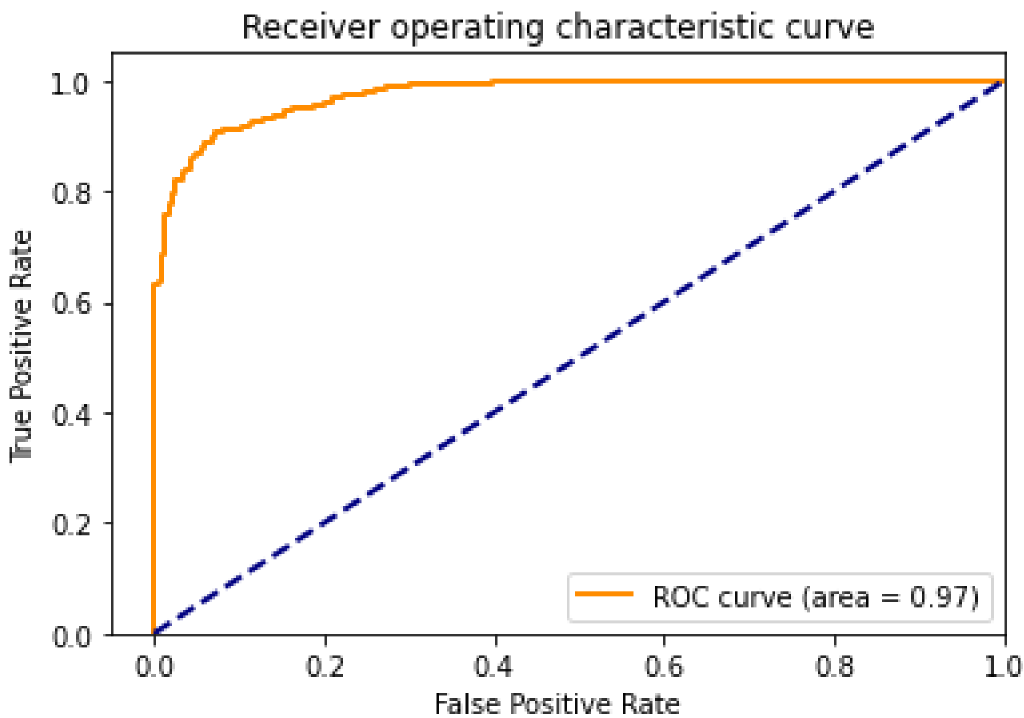


[211]:
```
target_names = ['negative', 'positive']
print(classification_report(test_generator.classes, np.where(preds[:,1]>0.
↪5,1,0) , target_names=target_names))
```

```
              precision    recall  f1-score   support

    negative       0.86      0.90      0.88       234
    positive       0.94      0.91      0.92       390

    accuracy                           0.91       624
   macro avg       0.90      0.91      0.90       624
weighted avg       0.91      0.91      0.91       624
```

[212]:
```
fpr, tpr, _ = roc_curve(test_generator.classes, preds[:,1])
roc_auc = auc(fpr, tpr)
```

```
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```



4 VGG16

[213]:
```
#data augmentation
train_gen = ImageDataGenerator(
        dtype = 'float32',
   #    featurewise_center=False,  # set input mean to 0 over the dataset
   #    samplewise_center=False,  # set each sample mean to 0
   #    featurewise_std_normalization=False,  # divide inputs by std of the␣
 ↪dataset
```

```
  #    samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range (degrees, 0␣
↪to 180)
        zoom_range = 0.4, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction␣
↪of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction␣
↪of total height)
        horizontal_flip = True,  # randomly flip images
        vertical_flip=False,
        preprocessing_function=tf.keras.applications.vgg16.preprocess_input,
        validation_split=0.2) #validation size
test_gen=ImageDataGenerator(
        dtype = 'float32',
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the␣
↪dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        horizontal_flip = False,  # randomly flip images
        preprocessing_function=tf.keras.applications.vgg16.preprocess_input,
        vertical_flip=False)  # randomly flip images
```

[214]:
```
batch_size=64
nb_epochs=90
train_generator = train_gen.flow_from_directory(
    wd+"/train",
    batch_size=batch_size,
    target_size=(224, 224), #class_mode="binary"
  #  class_mode='binary',
    subset='training'
)

validation_generator = train_gen.flow_from_directory(
    wd+"/train", # same directory as training data
    batch_size=batch_size,
    target_size=(224, 224),
 #   class_mode='binary',
    subset='validation') # set as validation data
test_generator=test_gen.flow_from_directory(
    wd+"/test", # same directory as training data
    batch_size=1,
    target_size=(224, 224),
    shuffle=False)
```

```
Found 4173 images belonging to 2 classes.
Found 1043 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

[215]:

```
inputs = tf.keras.Input(shape=(224,224,3))
##First, instantiate a base model with pre-trained weights.
base_model = tf.keras.applications.VGG16(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(224,224,3),
    include_top=False,
    pooling='avg',
    input_tensor=inputs
)  # Do not include the ImageNet classifier at the top.

# Freeze the base_model
base_model.trainable = False

# Create new model on top

early_stopping_cb = tf.keras.callbacks.
 ↪EarlyStopping(patience=10,restore_best_weights=True)
model=Sequential()
model.add(base_model)
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(2,activation='softmax'))

model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
start=time.time()
history_vgg16 = model.
 ↪fit_generator(train_generator,epochs=90,validation_data=validation_generator,callbacks=[ear
runtime=time.time()-start
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 138s 2us/step
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
vgg16 (Functional)           (None, 512)              14714688
_________________________________________________________________
flatten_17 (Flatten)         (None, 512)              0
_________________________________________________________________
batch_normalization_33 (Batc (None, 512)              2048
_________________________________________________________________
dense_60 (Dense)             (None, 128)              65664
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)              0
_________________________________________________________________
batch_normalization_34 (Batc (None, 128)              512
_________________________________________________________________
dense_61 (Dense)             (None, 64)               8256
_________________________________________________________________
dropout_16 (Dropout)         (None, 64)               0
_________________________________________________________________
batch_normalization_35 (Batc (None, 64)               256
_________________________________________________________________
dense_62 (Dense)             (None, 2)                130
=================================================================
Total params: 14,791,554
Trainable params: 75,458
Non-trainable params: 14,716,096
_________________________________________________________________
Epoch 1/90

2021-08-15 16:30:57.639008: W
tensorflow/core/common_runtime/bfc_allocator.cc:248] Allocator (GPU_0_bfc) ran
out of memory trying to allocate 3.46GiB with freed_by_count=0. The caller
indicates that this is not a failure, but may mean that there could be
performance gains if more memory were available.

66/66 [==============================] - 106s 1s/step - loss: 0.7699 - accuracy:
0.6566 - val_loss: 1.1946 - val_accuracy: 0.6261
Epoch 2/90
66/66 [==============================] - 73s 1s/step - loss: 0.3752 - accuracy:
0.8584 - val_loss: 0.3569 - val_accuracy: 0.8792
Epoch 3/90
66/66 [==============================] - 74s 1s/step - loss: 0.2759 - accuracy:
0.8960 - val_loss: 0.2283 - val_accuracy: 0.9271
Epoch 4/90
66/66 [==============================] - 74s 1s/step - loss: 0.2371 - accuracy:
0.9103 - val_loss: 0.1751 - val_accuracy: 0.9348
Epoch 5/90
66/66 [==============================] - 73s 1s/step - loss: 0.2212 - accuracy:
0.9171 - val_loss: 0.1488 - val_accuracy: 0.9473
Epoch 6/90
66/66 [==============================] - 72s 1s/step - loss: 0.2149 - accuracy:
```

```
0.9177 - val_loss: 0.1332 - val_accuracy: 0.9511
Epoch 7/90
66/66 [==============================] - 72s 1s/step - loss: 0.1989 - accuracy:
0.9281 - val_loss: 0.1420 - val_accuracy: 0.9425
Epoch 8/90
66/66 [==============================] - 72s 1s/step - loss: 0.1838 - accuracy:
0.9296 - val_loss: 0.1315 - val_accuracy: 0.9511
Epoch 9/90
66/66 [==============================] - 72s 1s/step - loss: 0.1782 - accuracy:
0.9325 - val_loss: 0.1467 - val_accuracy: 0.9396
Epoch 10/90
66/66 [==============================] - 72s 1s/step - loss: 0.1616 - accuracy:
0.9404 - val_loss: 0.1388 - val_accuracy: 0.9530
Epoch 11/90
66/66 [==============================] - 72s 1s/step - loss: 0.1633 - accuracy:
0.9388 - val_loss: 0.1388 - val_accuracy: 0.9453
Epoch 12/90
66/66 [==============================] - 73s 1s/step - loss: 0.1535 - accuracy:
0.9431 - val_loss: 0.1284 - val_accuracy: 0.9530
Epoch 13/90
66/66 [==============================] - 72s 1s/step - loss: 0.1680 - accuracy:
0.9376 - val_loss: 0.1217 - val_accuracy: 0.9540
Epoch 14/90
66/66 [==============================] - 72s 1s/step - loss: 0.1665 - accuracy:
0.9398 - val_loss: 0.1325 - val_accuracy: 0.9549
Epoch 15/90
66/66 [==============================] - 73s 1s/step - loss: 0.1621 - accuracy:
0.9418 - val_loss: 0.1224 - val_accuracy: 0.9549
Epoch 16/90
66/66 [==============================] - 73s 1s/step - loss: 0.1530 - accuracy:
0.9406 - val_loss: 0.1328 - val_accuracy: 0.9559
Epoch 17/90
66/66 [==============================] - 72s 1s/step - loss: 0.1418 - accuracy:
0.9470 - val_loss: 0.1255 - val_accuracy: 0.9511
Epoch 18/90
66/66 [==============================] - 72s 1s/step - loss: 0.1408 - accuracy:
0.9446 - val_loss: 0.1232 - val_accuracy: 0.9540
Epoch 19/90
66/66 [==============================] - 72s 1s/step - loss: 0.1537 - accuracy:
0.9407 - val_loss: 0.1174 - val_accuracy: 0.9521
Epoch 20/90
66/66 [==============================] - 72s 1s/step - loss: 0.1314 - accuracy:
0.9536 - val_loss: 0.1148 - val_accuracy: 0.9549
Epoch 21/90
66/66 [==============================] - 72s 1s/step - loss: 0.1424 - accuracy:
0.9455 - val_loss: 0.1128 - val_accuracy: 0.9569
Epoch 22/90
66/66 [==============================] - 72s 1s/step - loss: 0.1463 - accuracy:
```

```
0.9469 - val_loss: 0.1278 - val_accuracy: 0.9492
Epoch 23/90
66/66 [==============================] - 72s 1s/step - loss: 0.1477 - accuracy:
0.9492 - val_loss: 0.1137 - val_accuracy: 0.9530
Epoch 24/90
66/66 [==============================] - 72s 1s/step - loss: 0.1337 - accuracy:
0.9490 - val_loss: 0.1028 - val_accuracy: 0.9626
Epoch 25/90
66/66 [==============================] - 72s 1s/step - loss: 0.1551 - accuracy:
0.9425 - val_loss: 0.1168 - val_accuracy: 0.9626
Epoch 26/90
66/66 [==============================] - 72s 1s/step - loss: 0.1334 - accuracy:
0.9515 - val_loss: 0.1012 - val_accuracy: 0.9559
Epoch 27/90
66/66 [==============================] - 71s 1s/step - loss: 0.1312 - accuracy:
0.9503 - val_loss: 0.1065 - val_accuracy: 0.9597
Epoch 28/90
66/66 [==============================] - 71s 1s/step - loss: 0.1147 - accuracy:
0.9585 - val_loss: 0.1158 - val_accuracy: 0.9530
Epoch 29/90
66/66 [==============================] - 72s 1s/step - loss: 0.1310 - accuracy:
0.9509 - val_loss: 0.1078 - val_accuracy: 0.9578
Epoch 30/90
66/66 [==============================] - 71s 1s/step - loss: 0.1199 - accuracy:
0.9517 - val_loss: 0.1149 - val_accuracy: 0.9588
Epoch 31/90
66/66 [==============================] - 72s 1s/step - loss: 0.1134 - accuracy:
0.9566 - val_loss: 0.0987 - val_accuracy: 0.9655
Epoch 32/90
66/66 [==============================] - 72s 1s/step - loss: 0.1218 - accuracy:
0.9517 - val_loss: 0.1011 - val_accuracy: 0.9569
Epoch 33/90
66/66 [==============================] - 72s 1s/step - loss: 0.1179 - accuracy:
0.9592 - val_loss: 0.1017 - val_accuracy: 0.9588
Epoch 34/90
66/66 [==============================] - 71s 1s/step - loss: 0.1263 - accuracy:
0.9532 - val_loss: 0.1006 - val_accuracy: 0.9636
Epoch 35/90
66/66 [==============================] - 72s 1s/step - loss: 0.1247 - accuracy:
0.9517 - val_loss: 0.0957 - val_accuracy: 0.9655
Epoch 36/90
66/66 [==============================] - 72s 1s/step - loss: 0.1234 - accuracy:
0.9533 - val_loss: 0.1059 - val_accuracy: 0.9569
Epoch 37/90
66/66 [==============================] - 72s 1s/step - loss: 0.1159 - accuracy:
0.9572 - val_loss: 0.1182 - val_accuracy: 0.9540
Epoch 38/90
66/66 [==============================] - 72s 1s/step - loss: 0.1087 - accuracy:
```

```
0.9582 - val_loss: 0.1030 - val_accuracy: 0.9597
Epoch 39/90
66/66 [==============================] - 72s 1s/step - loss: 0.1269 - accuracy:
0.9551 - val_loss: 0.1209 - val_accuracy: 0.9521
Epoch 40/90
66/66 [==============================] - 72s 1s/step - loss: 0.1278 - accuracy:
0.9574 - val_loss: 0.1079 - val_accuracy: 0.9588
Epoch 41/90
66/66 [==============================] - 72s 1s/step - loss: 0.1174 - accuracy:
0.9548 - val_loss: 0.1202 - val_accuracy: 0.9492
Epoch 42/90
66/66 [==============================] - 72s 1s/step - loss: 0.1365 - accuracy:
0.9443 - val_loss: 0.0997 - val_accuracy: 0.9607
Epoch 43/90
66/66 [==============================] - 72s 1s/step - loss: 0.1311 - accuracy:
0.9488 - val_loss: 0.1025 - val_accuracy: 0.9664
Epoch 44/90
66/66 [==============================] - 72s 1s/step - loss: 0.1114 - accuracy:
0.9582 - val_loss: 0.1092 - val_accuracy: 0.9549
Epoch 45/90
66/66 [==============================] - 72s 1s/step - loss: 0.1081 - accuracy:
0.9584 - val_loss: 0.1129 - val_accuracy: 0.9645
```

[216]:
```
#saving the model and histories
#tf.keras.models.save_model(filepath=cwd, model=model)
model.save(cwd+'/assets/model_vgg16')
np.save(cwd+'/assets/history_vgg16.npy', history_vgg16.history)
#np.save(cwd+'/assets/history_xception_tuning.npy', history_xception_tuning.
 ↪history)
```

```
INFO:tensorflow:Assets written to:
/home/jovyan/work/fc_project/assets/model_vgg16/assets
```

[217]:
```
#tuning
# Unfreeze the base_model. Note that it keeps running in inference mode
# since we passed `training=False` when calling it. This means that
# the batchnorm layers will not update their batch statistics.
# This prevents the batchnorm layers from undoing all the training
# we've done so far.
base_model.trainable = True
model.summary()

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),  # Low learning rate
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)
```

```
epochs = 10
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 512)               14714688
_________________________________________________________________
flatten_17 (Flatten)         (None, 512)               0
_________________________________________________________________
batch_normalization_33 (Batc (None, 512)               2048
_________________________________________________________________
dense_60 (Dense)             (None, 128)               65664
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
batch_normalization_34 (Batc (None, 128)               512
_________________________________________________________________
dense_61 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_16 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_35 (Batc (None, 64)                256
_________________________________________________________________
dense_62 (Dense)             (None, 2)                 130
=================================================================
Total params: 14,791,554
Trainable params: 14,790,146
Non-trainable params: 1,408
_________________________________________________________________
```

[218]:

```
start=time.time()
history_vgg16_tuning = model.
 ↪fit_generator(train_generator,epochs=10,validation_data=validation_generator)
runtime=time.time()-start+runtime
```

```
Epoch 1/10
66/66 [==============================] - 99s 1s/step - loss: 0.1276 - accuracy:
0.9570 - val_loss: 0.1298 - val_accuracy: 0.9549
Epoch 2/10
66/66 [==============================] - 73s 1s/step - loss: 0.1094 - accuracy:
0.9586 - val_loss: 0.1634 - val_accuracy: 0.9377
Epoch 3/10
66/66 [==============================] - 74s 1s/step - loss: 0.0909 - accuracy:
0.9626 - val_loss: 0.0751 - val_accuracy: 0.9674
Epoch 4/10
```

```
66/66 [==============================] - 74s 1s/step - loss: 0.0801 - accuracy:
0.9740 - val_loss: 0.0944 - val_accuracy: 0.9645
Epoch 5/10
66/66 [==============================] - 74s 1s/step - loss: 0.0833 - accuracy:
0.9712 - val_loss: 0.1683 - val_accuracy: 0.9348
Epoch 6/10
66/66 [==============================] - 74s 1s/step - loss: 0.0749 - accuracy:
0.9761 - val_loss: 0.1139 - val_accuracy: 0.9540
Epoch 7/10
66/66 [==============================] - 74s 1s/step - loss: 0.0656 - accuracy:
0.9803 - val_loss: 0.0748 - val_accuracy: 0.9703
Epoch 8/10
66/66 [==============================] - 74s 1s/step - loss: 0.0749 - accuracy:
0.9766 - val_loss: 0.0698 - val_accuracy: 0.9722
Epoch 9/10
66/66 [==============================] - 74s 1s/step - loss: 0.0689 - accuracy:
0.9761 - val_loss: 0.0656 - val_accuracy: 0.9751
Epoch 10/10
66/66 [==============================] - 75s 1s/step - loss: 0.0635 - accuracy:
0.9767 - val_loss: 0.0763 - val_accuracy: 0.9722
```

[219]:

```
#saving the model and histories
#tf.keras.models.save_model(filepath=cwd, model=model)
model.save(cwd+'/assets/model_vgg16_tuning')
np.save(cwd+'/assets/history_vgg16_tuning.npy', history_vgg16_tuning.history)
#np.save(cwd+'/assets/history_xception_tuning.npy', history_xception_tuning.
 ↪history)
```

```
INFO:tensorflow:Assets written to:
/home/jovyan/work/fc_project/assets/model_vgg16_tuning/assets
```

[220]:

```
score = model.evaluate(test_generator,  verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.17918924987316132
Test accuracy: 0.9294871687889099
```

[221]:

```
results['model'].append('vgg16')
results['total_param'].append(14791146)
results['time'].append(runtime)
results['score'].append(score[1])
np.save(cwd+'/assets/results.npy', results)
```

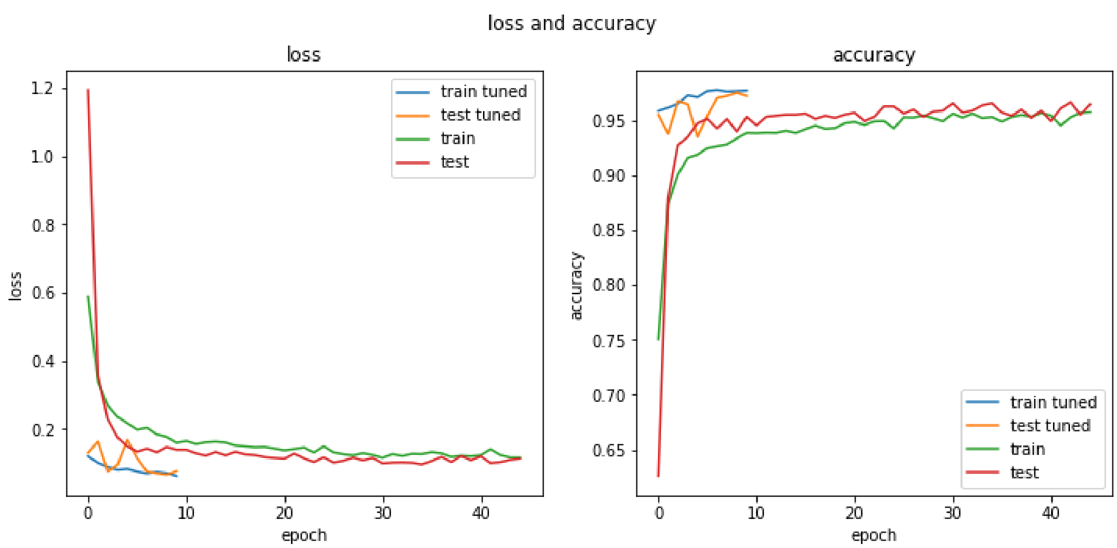
[222]:

```
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
test_generator.reset()
preds = model.predict(test_generator,
steps=STEP_SIZE_TEST,
```

```
verbose=1)
```

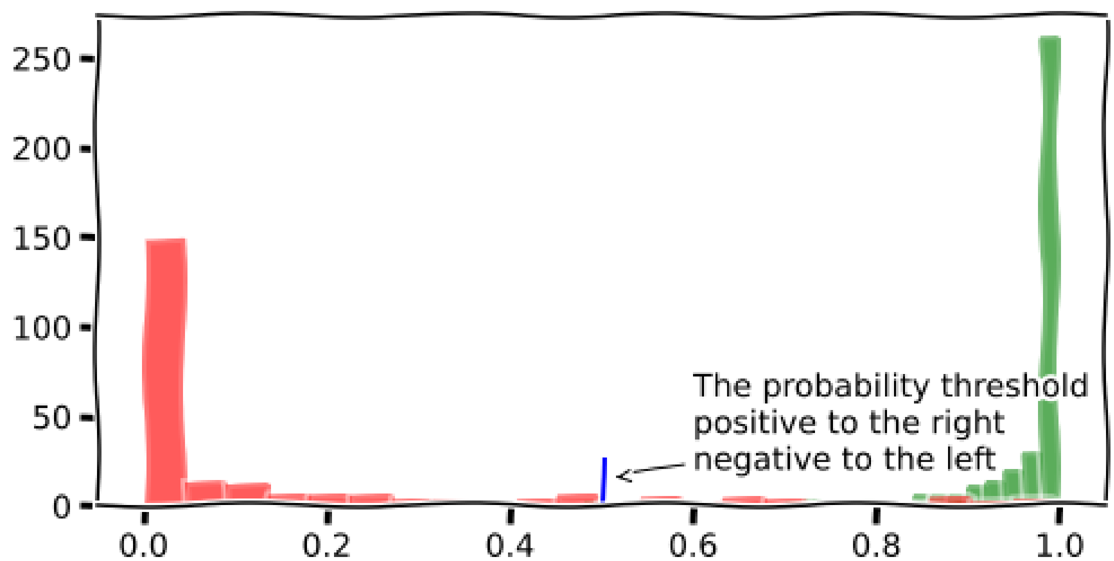
```
624/624 [==============================] - 5s 8ms/step
```

[228]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
ax1.plot(history_vgg16_tuning.history['loss'])
ax1.plot(history_vgg16_tuning.history['val_loss'])
ax1.plot(history_vgg16.history['loss'])
ax1.plot(history_vgg16.history['val_loss'])
ax1.set_title('loss')
ax1.set_ylabel('loss')
ax1.set_xlabel('epoch')
ax1.legend(['train tuned', 'test tuned','train','test'], loc='upper right')
ax2.plot(history_vgg16_tuning.history['accuracy'])
ax2.plot(history_vgg16_tuning.history['val_accuracy'])
ax2.plot(history_vgg16.history['accuracy'])
ax2.plot(history_vgg16.history['val_accuracy'])

ax2.set_title('accuracy')
ax2.set_ylabel('accuracy')
ax2.set_xlabel('epoch')
ax2.legend(['train tuned', 'test tuned','train','test'], loc='lower right')
fig.suptitle('loss and accuracy')
plt.show()
```



[238]:
```
#scale the output of the model to [0,1]
#scaled_preds=(preds[:,1]-min(preds[:,1]))/(max(preds[:,1])-min(preds[:,1]))
pos = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 1]
neg = [i for i, j in zip(preds[:,1], test_generator.classes) if j == 0]
```

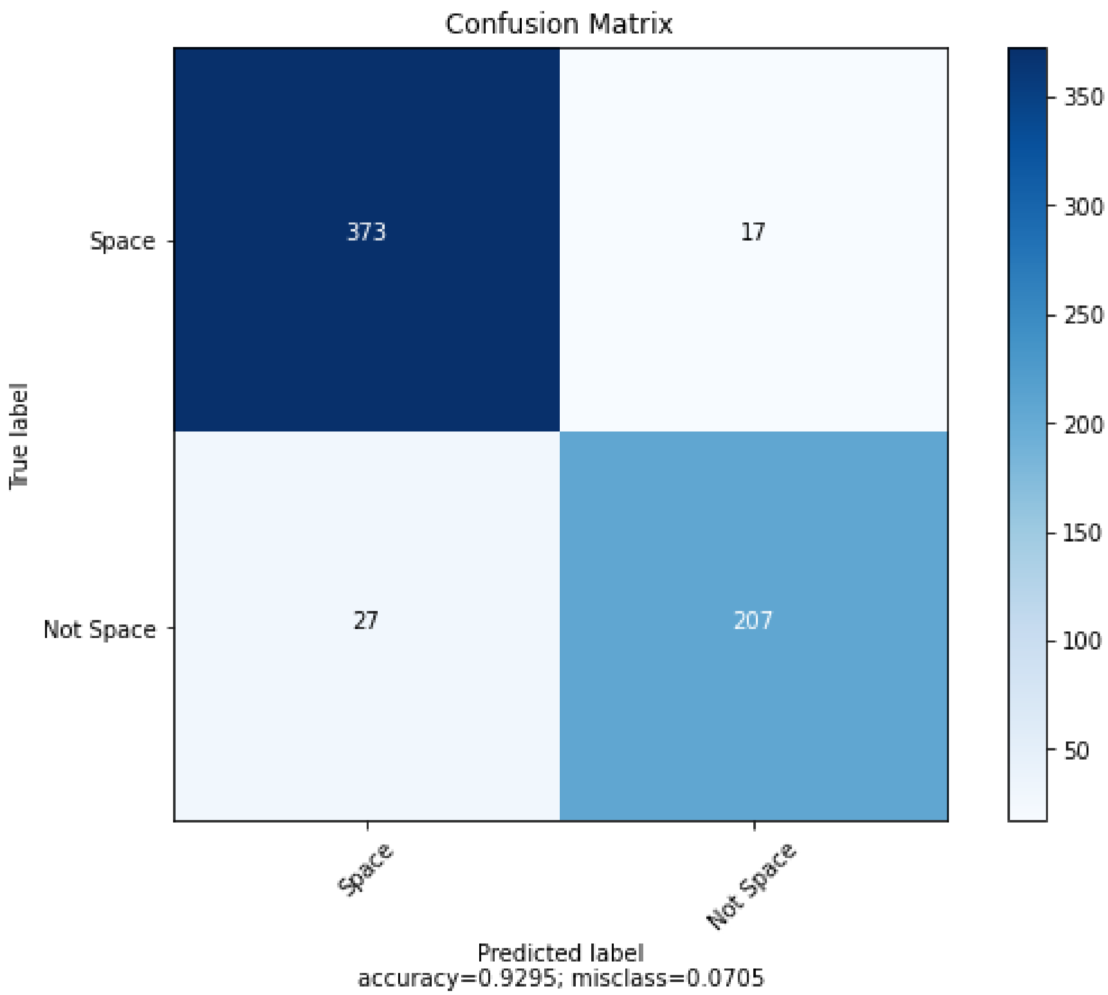
```
with plt.xkcd():
  fig = plt.figure(figsize=(8, 4))

  sns.distplot(pos, hist = True, kde = False, color='g',
               kde_kws = {'shade': True, 'linewidth': 3})

  sns.distplot(neg, hist = True, kde = False, color='r',
               kde_kws = {'shade': True, 'linewidth': 3})

  plt.plot([0.5, 0.5], [0, 25], '-b')
  plt.annotate(
        'The probability threshold\npositive to the right\nnegative to the␣
 ↪left',
        xy=(0.51, 15), arrowprops=dict(arrowstyle='->'), xytext=(0.6, 20))

plt.show()
```



[239]:
```
confusion = confusion_matrix(test_generator.classes, np.where(preds[:,1]>0.
 ↪5,1,0), labels=[1, 0])
print(confusion)
```

```
[[373  17]
 [ 27 207]]
```

[240]:
```
plot_confusion_matrix(cm=confusion, target_names = ['Space', 'Not Space'],␣
 ↪title = 'Confusion Matrix',normalize=False)
```

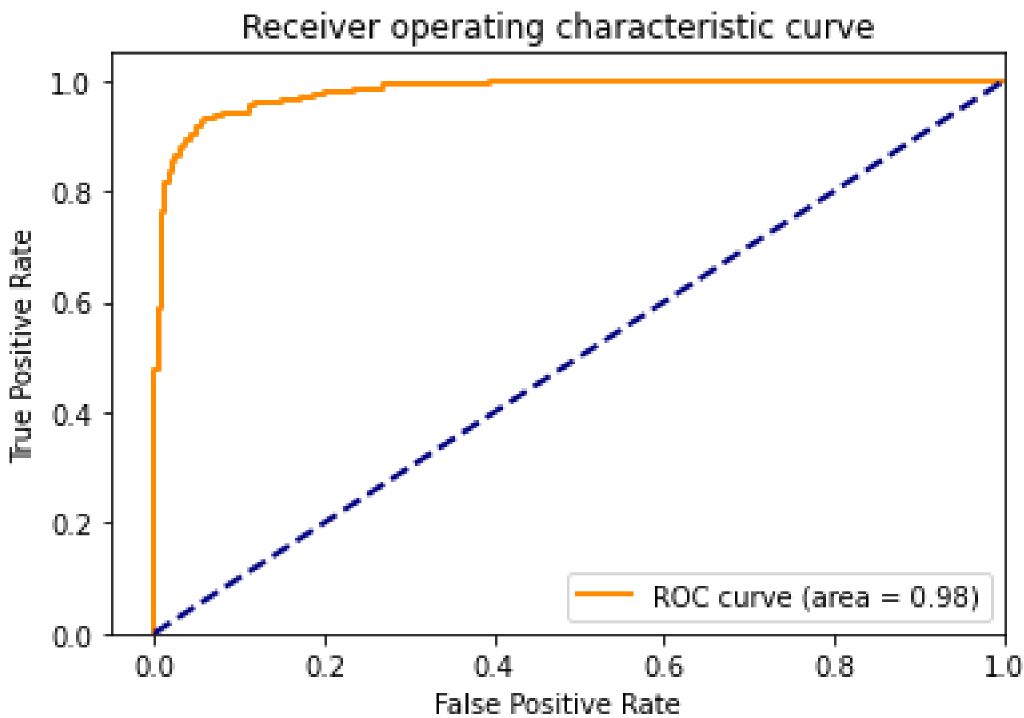


[241]:
```
target_names = ['negative', 'positive']
print(classification_report(test_generator.classes, np.where(preds[:,1]>0.
↪5,1,0) , target_names=target_names))
```

```
              precision    recall  f1-score   support

    negative       0.92      0.88      0.90       234
    positive       0.93      0.96      0.94       390

    accuracy                           0.93       624
   macro avg       0.93      0.92      0.92       624
weighted avg       0.93      0.93      0.93       624
```

[242]:
```
fpr, tpr, _ = roc_curve(test_generator.classes, preds[:,1])
roc_auc = auc(fpr, tpr)
```

```
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```



[ ]:

```

```