geschirr-evaluation-v1

December 31, 2025

1 Evaluation des Neuronalen Netzes zur Geschirr-Erkennung

Das Notebook Giraffenpuzzle-Training produzierte als Ausgabe ein fertig trainiertes neuronales Netz, sowie eine Aufsplittung des Giraffenpuzzleteile-Datensatzes in einen Trainings- und einen Testdatensatz. Hier wollen wir das Modell laden und auf dem Testdatensatz evaluieren.

[92]:
```
import matplotlib.pyplot as plt #Plottingroutinen zur Visualisierung
import numpy as np # Arrays- "The fundamental package for scientific computing␣
 ↪with Python"
import pandas as pd #Umgang mit Tabellen- hier nur zum Laden der Klassenlabels
import tensorflow as tf #Tensorflow, trainieren von neuronale Netzen (inkl.␣
 ↪Keras)
import shutil # Dateisystemmanipulationen, insbesondere für das Kopieren von␣
 ↪Dateien
from pathlib import Path #Pathlib, Umgang mit Dateipfaden
import scikitplot # Visualisierungen von Klassifikationsperformanz
```

[93]:
```
Baumstruktur_schon_vorhanden=True
```

[94]:
```
# Modell laden:
loaded_model = tf.keras.models.load_model('./working/model/model_Geschirr.
 ↪keras')
# Excel-Datei mit Angabe, ob ein Bild ein Test- oder ein Trainingsbild ist:
```

[95]:
```
Baumstruktur_schon_vorhanden
```

[95]:
```
True
```

[96]:
```
if Baumstruktur_schon_vorhanden:
    testBaumstrukturpfad = Path('./output/geschirr/1_object_extraction_output/
 ↪2_Baumstruktur_test') #MUSS ANGEPASST WERDEN
    np.set_printoptions(suppress=True, linewidth=500)
    testBilder_fn = pd.Series([fn for fn in testBaumstrukturpfad.glob('*/*')])
    testKlassenlabel = pd.Series([fn.parent.name for fn in␣
 ↪testBilder_fn],index=[fn.name for fn in testBilder_fn])
    y_test = testKlassenlabel
else:
```

```
    train_or_test = pd.read_excel('/kaggle/input/giraffenpuzzle_beispielmodell/
↪keras/baseline/1/train_test_split.xlsx') #MUSS ANGEPASST WERDEN
    train_or_test.columns = ['Dateiname','TrainOrTest']

    testBilder_fn = train_or_test[train_or_test['TrainOrTest']=='test' ]

    # Extrahiere Klassenlabel aus Dateiname
    testKlassenlabel = train_or_test.Dateiname.map(lambda s:s[:s.find('_')])
    train_or_test['Klasse'] = testKlassenlabel
    testBilder_fn = pd.
↪Series(train_or_test[train_or_test['TrainOrTest']=='test' ])

    # Wähle nur die Testdaten
    y_test = testKlassenlabel[ train_or_test['TrainOrTest']=='test' ]

    testBilder_fn=[]
    # Kopiere die Testbilder in eine Baumstruktur
    for irow,row in testBilder_fn.iterrows():
        Dateiname,TrainOrTest,Klasse = (row.Dateiname,row.TrainOrTest,row.
↪Klasse)
        fullfilename = Path('/kaggle/input/giraffenpuzzleteile')/Dateiname
        testBaumstrukturpfad = Path('/kaggle/working')/'test'
        (testBaumstrukturpfad/Klasse).mkdir(exist_ok=True,parents=True)
        shutil.copyfile(fullfilename,testBaumstrukturpfad/Klasse/Dateiname)
        testBilder_fn.append(testBaumstrukturpfad/Klasse/Dateiname)
    testBilder_fn = pd.Series(testBilder_fn)
y_test = pd.Series(y_test.values,index=testBilder_fn)
```

[97]:
```
y_test
```

[97]:
```
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_0_9687.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9686.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9684.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_0_9690.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_0_9691.jpg    trinkhalm-01
                              …
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/loeffel-
01/loeffel-01_1_9508.jpg          loeffel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/loeffel-
01/loeffel-01_0_9521.jpg          loeffel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/loeffel-
01/loeffel-01_0_9523.jpg          loeffel-01
```

```
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/loeffel-
01/loeffel-01_0_9522.jpg          loeffel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/loeffel-
01/loeffel-01_0_9536.jpg          loeffel-01
Length: 91, dtype: object
```

Überprüfen wir das Resultat:

[98]:
```
!ls {testBaumstrukturpfad}
#!ls /kaggle/working/test/*   #Ausgabe ist etwas lang, aber nützlich für die␣
 ↪Überprüfung der Vollständigkeit
print('-'*70) #nur zur visuellen Strukturierung; 3*"a" ergibt "aaa"
!ls {testBaumstrukturpfad}/loeffel-01
```

```
gabel-01        korkenzieher-01 loeffel-01
messer-01       trinkhalm-01
----------------------------------------------------------------------
loeffel-01_0_9514.jpg loeffel-01_0_9536.jpg loeffel-01_1_9542.jpg
loeffel-01_0_9519.jpg loeffel-01_0_9541.jpg loeffel-01_1_9546.jpg
loeffel-01_0_9521.jpg loeffel-01_1_9505.jpg loeffel-01_2_9509.jpg
loeffel-01_0_9522.jpg loeffel-01_1_9507.jpg loeffel-01_2_9531.jpg
loeffel-01_0_9523.jpg loeffel-01_1_9508.jpg loeffel-01_2_9534.jpg
loeffel-01_0_9525.jpg loeffel-01_1_9510.jpg loeffel-01_2_9542.jpg
loeffel-01_0_9528.jpg loeffel-01_1_9513.jpg loeffel-01_2_9545.jpg
loeffel-01_0_9529.jpg loeffel-01_1_9517.jpg loeffel-01_2_9546.jpg
loeffel-01_0_9531.jpg loeffel-01_1_9528.jpg
loeffel-01_0_9534.jpg loeffel-01_1_9533.jpg
```

test_ds ist eine `tf.Dataset` ist eine Instanz einer Tensorflowklasse, welche effizient Trainingsbilder aus der Baumstruktur lädt. Natürlich ist das eigentlich erst bei grossen Datenmengen relevant, aber wir zeigen hier den Code. Wir könnten auch wieder `tf.keras.utils.image_dataset_from_directory` benutzen, aber dann würden wir den zugehörigen Dateipfad nicht erhalten. Nicht schlimm, aber wir könnten die Bilder nicht nochmals mit ihrem Label und Dateipfad zusammen anzeigen.

[99]:
```
def lade_testdaten_mit_pfad(baumstrukturpfad, image_size=(128, 128),␣
 ↪batch_size=32, suffix='.jpg',shuffle=False):
    """
    Diese Funktion baut ein tf.Dataset basierend auf den Daten in der␣
 ↪Baumstruktur baumstrukturpfad, welches effizient Bilddaten in das neuronale␣
 ↪Netz füttert.

    """
    # Alle Bildpfade erfassen
    all_files = sorted(list(Path(baumstrukturpfad).rglob("*/*"+suffix)))  # ggf.
 ↪ .png anpassen
    class_names = sorted({p.parent.name for p in all_files})
    label_map = {name: idx for idx, name in enumerate(class_names)}
```

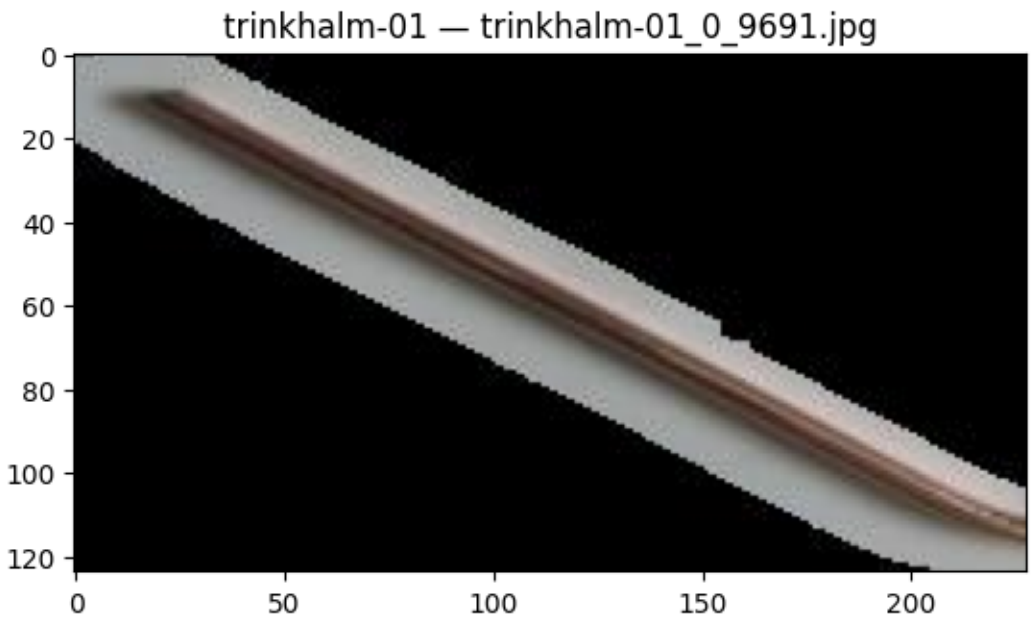
```
    # Pfade und Labels vorbereiten
    file_paths = np.array([str(p) for p in all_files])
    labels = np.array([label_map[Path(p).parent.name] for p in file_paths])

    def lade_bild(pfad, label):
        image = tf.io.read_file(pfad)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, image_size)
        return image, label, pfad

    if shuffle:
        # Gleicher Shuffle-Index für beide
        shuffle_idx = np.random.permutation(len(file_paths))
        # Koordiniert shuffeln
        file_paths_shuffled = file_paths[shuffle_idx]
        labels_shuffled = labels[shuffle_idx]
        # Dataset bauen
        ds_fn = tf.data.Dataset.from_tensor_slices((file_paths_shuffled,␣
 ↪labels_shuffled))
        ds = ds_fn.map(lade_bild).batch(batch_size).prefetch(tf.data.AUTOTUNE)
        return ds,file_paths_shuffled, labels_shuffled, class_names
    else:
        # Dataset bauen
        ds_fn = tf.data.Dataset.from_tensor_slices((file_paths, labels))
        ds = ds_fn.map(lade_bild).batch(batch_size).prefetch(tf.data.AUTOTUNE)
        return ds, file_paths, labels, class_names


test_ds, test_paths, y_test, class_names =␣
 ↪lade_testdaten_mit_pfad(testBaumstrukturpfad, image_size=(128,128),␣
 ↪batch_size=32, suffix='.jpg',shuffle=False)
test_ds,test_paths.shape,y_test.shape
```

[99]:
```
(<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 128, 128, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int64,
name=None), TensorSpec(shape=(None,), dtype=tf.string, name=None))>,
 (91,),
 (91,))
```

Wir können uns ein paar Bilder, ihre Labels und die zugehörigen Dateinamen anschauen:

[100]:
```
i=np.random.randint(test_paths.shape[0])
bild = plt.imread(test_paths[i])
label = class_names[y_test[i]]
plt.imshow(bild)
plt.title(f"{label} - {Path(test_paths[i]).name}")
```

[100]: Text(0.5, 1.0, 'trinkhalm-01 - trinkhalm-01_0_9691.jpg')



[101]:
```
yhat_test = loaded_model.predict(test_ds)
yhatnum_test = yhat_test.argmax(axis=1)

yhat_test.shape,y_test.shape
```

```
3/3 [==============================] - 0s 13ms/step
```

[101]: ((91, 5), (91,))

[102]:
```
y_test,yhatnum_test
```

[102]:
```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]),
 array([0, 0, 0, 0, 3, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3,
2, 2, 2, 2, 2, 2, 3, 3, 2, 3, 3, 2, 2, 2, 3, 3, 2, 3, 2, 2, 2, 2, 2, 1, 3, 3, 2,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 1, 3, 3, 1, 3, 3, 3,
3, 0, 4, 1, 4, 4, 0, 4, 4, 4, 0, 0, 2]))
```

Das ist nun also der Klassenindex- da wir die Bilder nicht “geshuffled” haben, ist zumindest die Sequenz `y_test` monoton. Bei `yhatnum_test` (also unseren Vorhersagen auf diesen Bildern) gibt es einige wenige Abweichungen. Das ist zu erwarten. Wenn wir nur wüssten, welcher Klasse das zugehört?!
Aber weil wir zu neugierig sind, wie gut das Modell sein könnte, rechnen wir ohne dieses Wissen

mal die Genauigkeit aus:

[103]:
```
Genauigkeit = np.mean(np.where(y_test==yhatnum_test,1,0))
# np.where könnte man auch weglassen, so: Genauigkeit = np.
 ↪mean(y_test==yhatnum_test).
# Aber was ist der Mittelwert von True, False, True?
# Klarer ist's wenn wir mit where ersetzen: True->1 und False->0
assert Genauigkeit==np.mean(y_test==yhatnum_test) # Ergibt einen Fehler, sollte␣
 ↪das mal nicht mehr stimmen
print(f'Die Testgenauigkeit ist {100*Genauigkeit:3.1f}%.')
```

```
Die Testgenauigkeit ist 78.0%.
```

Nun würden wir also auch gerne die Label kennen. Wir müssen den Labelindex (die Zahl) mit dem Namen der Klasse in Verbindung bringen. Zum Glück werden die Klassenlabel sortiert durchgezählt, so dass dieser Schritt nicht schwer ist:

[104]:
```
sorted_class_labels = np.array(sorted(testKlassenlabel.unique())) #Labels in␣
 ↪Keras der Funktion image_dataset_from_directory werden immer alphanummerisch␣
 ↪sortiert den Indices zugeordnet
sorted_class_labels
```

[104]:
```
array(['gabel-01', 'korkenzieher-01', 'loeffel-01', 'messer-01',
'trinkhalm-01'], dtype='<U15')
```

[105]:
```
testBilder_fn
yhatnum_test
```

[105]:
```
array([0, 0, 0, 0, 3, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3,
2, 2, 2, 2, 2, 2, 3, 3, 2, 3, 3, 2, 2, 2, 3, 3, 2, 3, 2, 2, 2, 2, 2, 1, 3, 3, 2,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 1, 3, 3, 1, 3, 3, 3,
3, 0, 4, 1, 4, 4, 0, 4, 4, 4, 0, 0, 2])
```

[106]:
```
yhat_label = pd.Series(sorted_class_labels[yhatnum_test],index=test_paths)
y_test_label = pd.Series(sorted_class_labels[y_test],index=test_paths)
# das vorhergesagte Label:
yhat_label
```

[106]:
```
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9639.jpg                gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9640.jpg                gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9641.jpg                gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9644.jpg                gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9647.jpg               messer-01
                             ...
```

```
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9686.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9688.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9694.jpg        gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9697.jpg        gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9699.jpg      loeffel-01
Length: 91, dtype: object
```

[107]:
```
# und das Tatsächliche:
y_test_label
```

[107]:
```
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9639.jpg                 gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9640.jpg                 gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9641.jpg                 gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9644.jpg                 gabel-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/gabel-01/gabel-
01_0_9647.jpg                 gabel-01
                             ...
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9686.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9688.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9694.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9697.jpg    trinkhalm-01
output/geschirr/1_object_extraction_output/2_Baumstruktur_test/trinkhalm-
01/trinkhalm-01_1_9699.jpg    trinkhalm-01
Length: 91, dtype: object
```

[108]:
```
# Natürlich muss die gleiche Genauigkeit rauskommen, wenn wir sie mit den␣
  ↪Labels berechnen, anstatt mit den Labelindices:
Genauigkeit2 = (yhat_label==y_test_label).mean()
print(f'Die Testgenauigkeit ist {100*Genauigkeit2:3.1f}%.')
```

```
Die Testgenauigkeit ist 78.0%.
```

Schliesslich können wir uns noch die Wahrscheinlichkeit merken, mit welcher das Neuronale Netz eine Klasse identifiziert hat. Dies ist also die grösste Zahl unter den $P(y = c_i|X)$ für alle Klassen c_1,...,c_C (C der Anzahl Klassen unseres Problems, `yhat_test.shape[1]`.

[109]:
```
P = np.max(yhat_test,axis=1)
P.shape
```

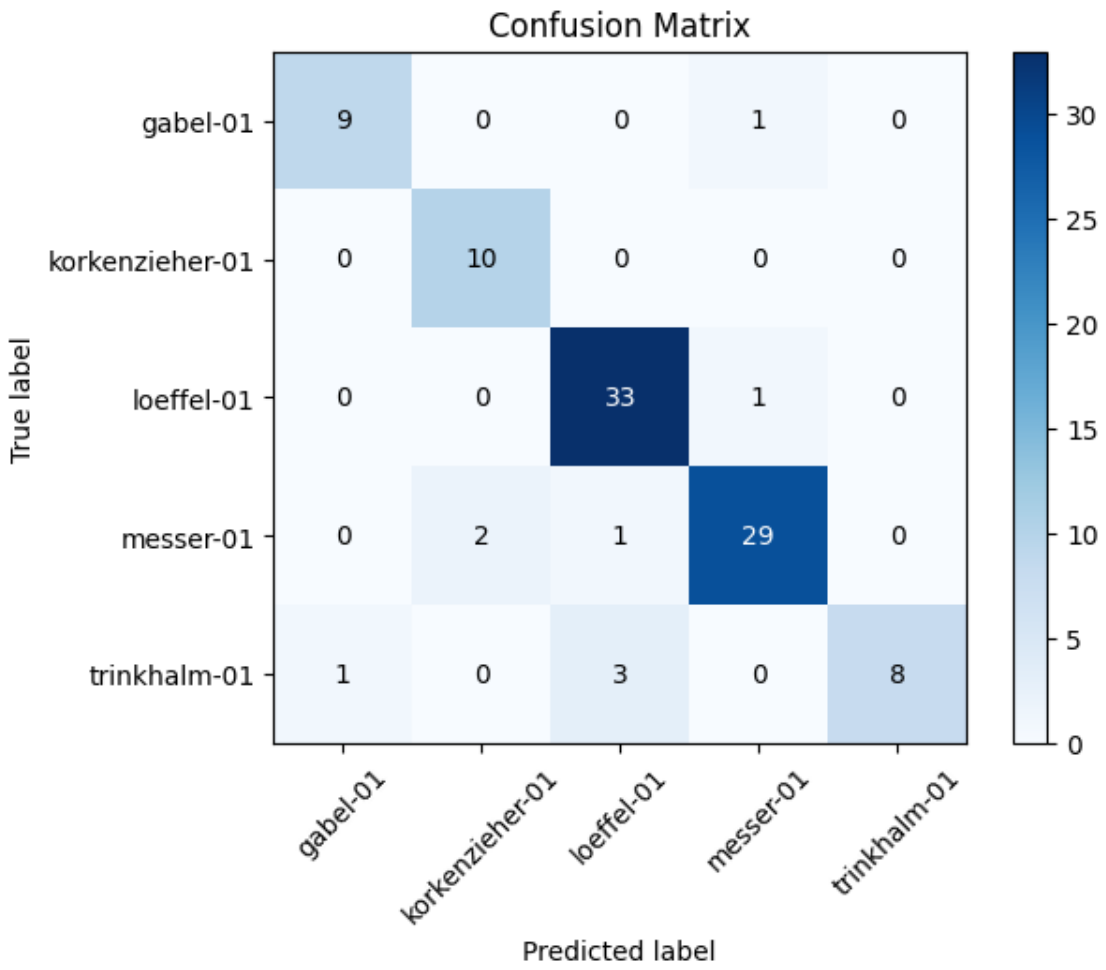
[109]:
```
(91,)
```

2 Scikit-Plot

Wir sind bereit! Wir kennen die Testbilder, deren zugehörige Klassenlabel und Vorhersagen durch das geladene Modell. Wir folgen der Dokumentation hier. Berechnen wir Metriken! Wir beginnen mit der Confusion Matrix:

[89]:
```
from scikitplot.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
print(classification_report(y_test_label,yhat_label,zero_division=0))
plot_confusion_matrix(y_test_label,yhat_label,labels=sorted_class_labels,x_tick_rotation=45);
```

```
                precision    recall  f1-score   support

       gabel-01       0.90      0.90      0.90        10
korkenzieher-01       0.83      1.00      0.91        10
     loeffel-01       0.89      0.97      0.93        34
      messer-01       0.94      0.91      0.92        32
   trinkhalm-01       1.00      0.67      0.80        12

       accuracy                           0.91        98
      macro avg       0.91      0.89      0.89        98
   weighted avg       0.91      0.91      0.91        98
```



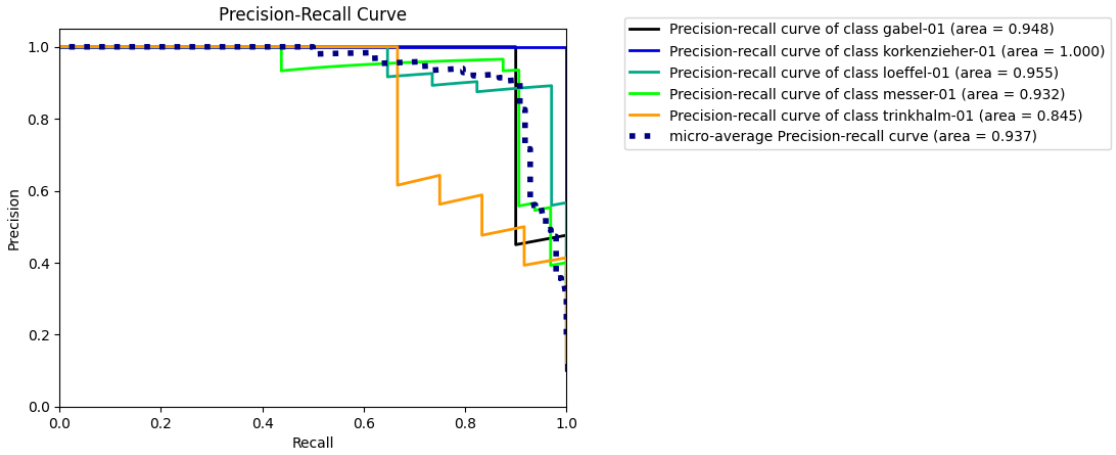
Beachten Sie, wie hübsch blockdiagonal die confusion matrix geworden ist. Dies zeigt, dass das Modell sich meist in der Form, und selten bis nie in der Farbe irrt.
Ein offensichtliches Problem ist natürlich, dass wir den Grossteil der Daten zum Training benutzt haben (das ist gängige Praxis), und nun nur wenige Beispiele (pro Klasse) übrig haben, um die Präzision und den Recall der Klasse zu schätzen. Mehr Daten wären schon praktisch…

[90]:
```
display((y_test_label==yhat_label).value_counts())
print('-'*30)
for curr_class_str in yhat_label.unique():
    ␣
  ↪curr_class_results=(y_test_label[y_test_label==curr_class_str]==yhat_label[y_test_label==cu
    print(f'{curr_class_str:<10}:{100*curr_class_results.mean():3.0f}%')
```

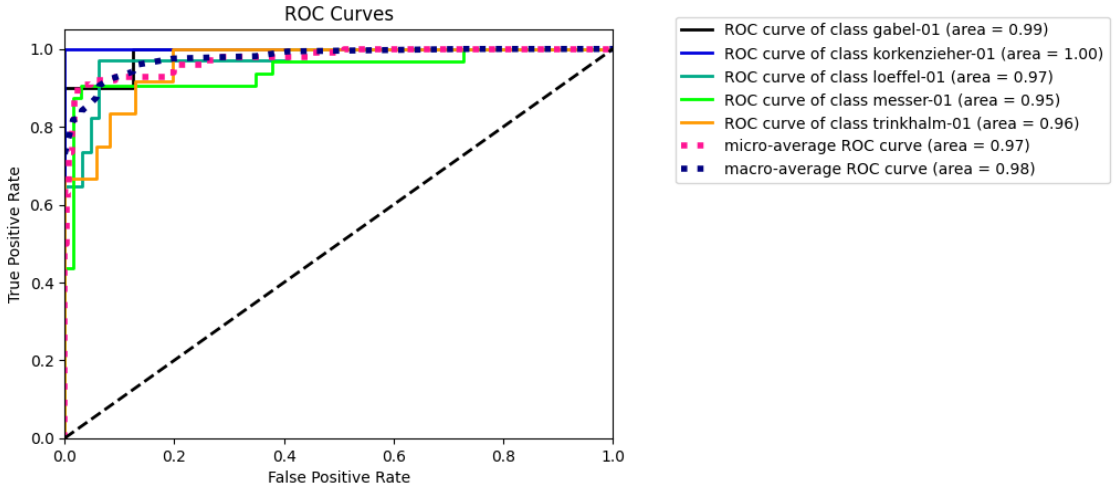
```
True     89
False     9
Name: count, dtype: int64

------------------------------
```

```
gabel-01  : 90%
messer-01 : 91%
korkenzieher-01:100%
loeffel-01: 97%
trinkhalm-01: 67%
```

[65]:
```
ax = plt.subplot(1,1,1)
scikitplot.metrics.plot_precision_recall(y_test_label,yhat_test,ax=ax)
ax.legend(bbox_to_anchor=(1.1, 1.05)); #Legende ausserhalb des Plots- sonst␣
 ↪sieht man nichts
```



[66]:
```
ax = plt.subplot(1,1,1)
scikitplot.metrics.plot_roc(y_test_label,yhat_test,ax=ax)
ax.legend(bbox_to_anchor=(1.1, 1.05)); #Legende ausserhalb des Plots- sonst␣
 ↪sieht man nichts
```



Ein Problem hier ist, dass das neuronale Netz für recht viele Klassen ("area=1.000") eine extreme Precision-Recall-Kurve ausgibt- von (0,1) über (1,1) nach (1,0). Aber wer würde sich schon darüber beklagen!

Zur Erinnerung, dies ist das Modell, das die obigen Resultate erziehlt hat. Wir können es grundsätzlich irgendwo (auf einem Raspberry-Pi?) laden und dort Vorhersagen generieren.

[67]:
```
loaded_model.layers[-1]
```

[67]:
```
<keras.src.layers.core.dense.Dense at 0x37f87b340>
```

[68]:
```
loaded_model.summary()
```

```
Model: "Conv-Model-Standard"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Eingabe (InputLayer)        [(None, 128, 128, 3)]     0

 rescaling_7 (Rescaling)     (None, 128, 128, 3)       0

 random_flip_7 (RandomFlip)  (None, 128, 128, 3)       0

 random_rotation_7 (RandomR  (None, 128, 128, 3)       0
 otation)

 conv2d_21 (Conv2D)          (None, 128, 128, 16)      448

 max_pooling2d_21 (MaxPooli  (None, 64, 64, 16)        0
 ng2D)

 conv2d_22 (Conv2D)          (None, 64, 64, 32)        4640

 max_pooling2d_22 (MaxPooli  (None, 32, 32, 32)        0
 ng2D)

 conv2d_23 (Conv2D)          (None, 32, 32, 64)        18496

 max_pooling2d_23 (MaxPooli  (None, 16, 16, 64)        0
 ng2D)

 flatten_7 (Flatten)         (None, 16384)             0

 dense_7 (Dense)             (None, 128)               2097280

 Ausgabe (Dense)             (None, 5)                 645
```

```
=================================================================
Total params: 2121509 (8.09 MB)
Trainable params: 2121509 (8.09 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Eingabe (InputLayer)        [(None, 128, 128, 3)]     0

 rescaling_7 (Rescaling)     (None, 128, 128, 3)       0

 random_flip_7 (RandomFlip)  (None, 128, 128, 3)       0

 random_rotation_7 (RandomR  (None, 128, 128, 3)       0
 otation)

 conv2d_21 (Conv2D)          (None, 128, 128, 16)      448

 max_pooling2d_21 (MaxPooli  (None, 64, 64, 16)        0
 ng2D)

 conv2d_22 (Conv2D)          (None, 64, 64, 32)        4640

 max_pooling2d_22 (MaxPooli  (None, 32, 32, 32)        0
 ng2D)

 conv2d_23 (Conv2D)          (None, 32, 32, 64)        18496

 max_pooling2d_23 (MaxPooli  (None, 16, 16, 64)        0
 ng2D)

 flatten_7 (Flatten)         (None, 16384)             0

 dense_7 (Dense)             (None, 128)               2097280

 Ausgabe (Dense)             (None, 5)                 645

=================================================================
Total params: 2121509 (8.09 MB)
Trainable params: 2121509 (8.09 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

3 Conv Layer Outputs

In den nächsten paar Zeilen werden wir ("spasseshalber", ist nicht weiter wichtig), den Output der letzten Conv-Schicht ("conv2d_5") ausgeben und visualisieren.

[70]:
```
conv_layer = loaded_model.get_layer("conv2d_22")
```

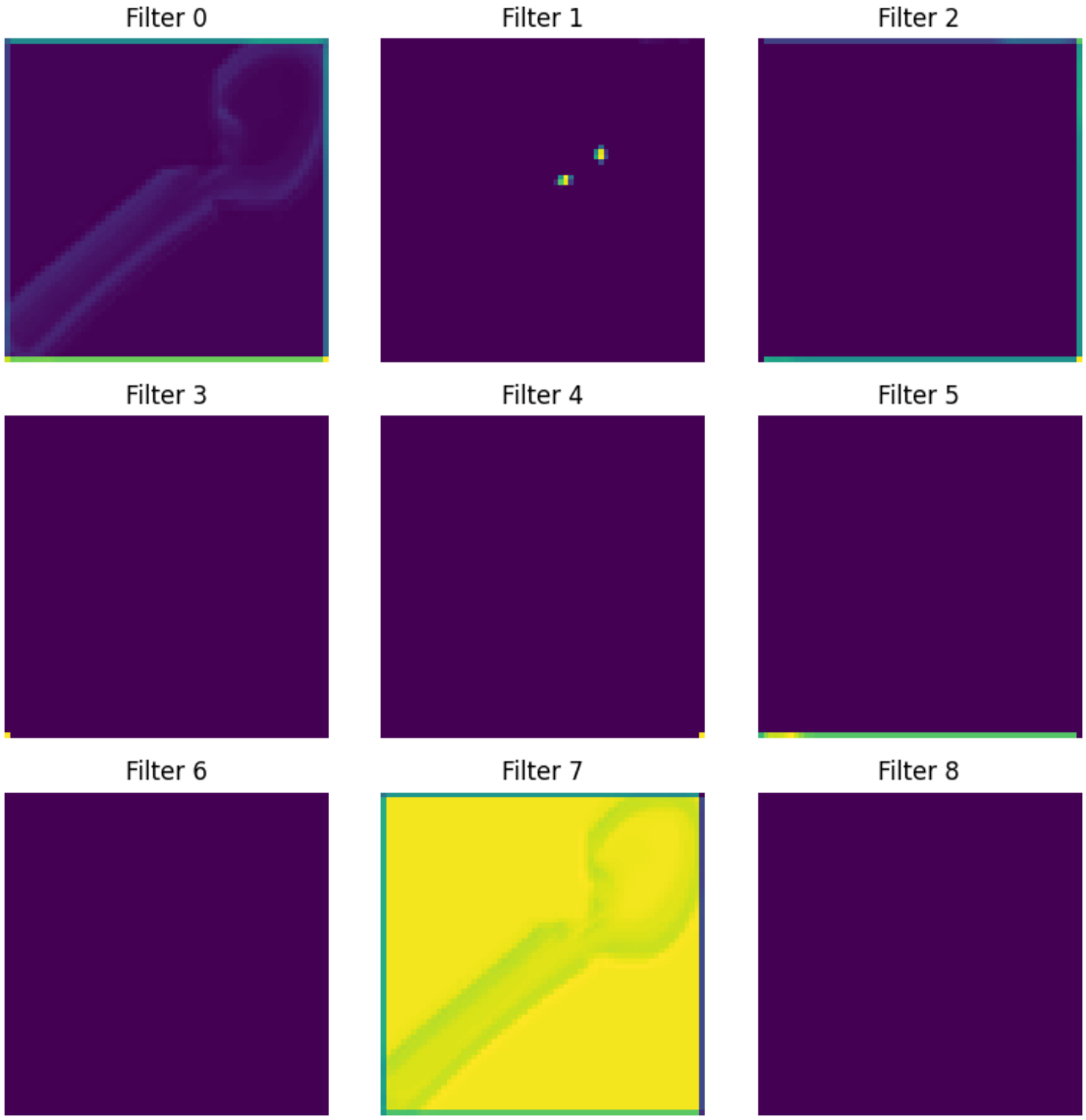
[71]:
```
from tensorflow.keras import Model, Input

feature_model = Model(
    inputs=loaded_model.input,
    outputs=conv_layer.output,
    name="FeatureExtractor"
)
```

[77]:
```
from PIL import Image
import numpy as np

color_image_size = (128, 128)

# Bild laden und skalieren
im = Image.open('./output/geschirr/1_object_extraction_output/Ausschnitte/
 ↪loeffel-01_0_9519.jpg')
im_resized = im.resize(color_image_size, Image.Resampling.LANCZOS)
img = np.array(im_resized) / 255.0  # Normalisierung

# Batch-Dimension hinzufügen
img_batch = img.reshape(1, *color_image_size, 3)
```

[78]:
```
feature_maps = feature_model.predict(img_batch)
print("Feature-Map Shape:", feature_maps.shape)
```

```
1/1 [==============================] - 0s 24ms/step
Feature-Map Shape: (1, 64, 64, 32)
1/1 [==============================] - 0s 24ms/step
Feature-Map Shape: (1, 64, 64, 32)
```

[91]:
```
import matplotlib.pyplot as plt

# Einzeln anzeigen (z. B. die ersten 9 Kanäle)
fig, axes = plt.subplots(3, 3, figsize=(8, 8))
for i, ax in enumerate(axes.flat):
    ax.imshow(feature_maps[0, :, :, i], cmap='viridis')
    ax.set_title(f'Filter {i}')
    ax.axis('off')
plt.tight_layout()
plt.show()
```


Filter 0
Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Filter 6
Filter 7
Filter 8