geschirr-training-v4

December 31, 2025

1 Model training

In diesem Notebook trainieren wir ein Neuronales Netz auf den Trainingsdaten, welche wir im Schritt Objektextraktion erstellt haben. Die Daten sollten in einer sog. Baumstruktur angeordnet sein: Wir brauchen ein Verzeichnis für jede Klasse, und darin alle Bilder dieser Klasse.

Zunächst müssen wir unsere Objektbilder finden. Dies sind Bilder mit einer engen *Bounding Box* um das Objekt herum:

[49]:
```
#Der folgende Befehl ist ein Linux-Befehl
!ls ./data/input/geschirr | head   # gebe nur die ersten 10 Dateien aus
```

```
gabel-01_9638.jpg
gabel-01_9639.jpg
gabel-01_9640.jpg
gabel-01_9641.jpg
gabel-01_9642.jpg
gabel-01_9643.jpg
gabel-01_9644.jpg
gabel-01_9645.jpg
gabel-01_9646.jpg
gabel-01_9647.jpg
```

In der nächsten Zelle extrahieren wir die Klasse aus dem Dateinamen des Bildes

[50]:
```
from pathlib import Path
import numpy as np
import scipy
import scikitplot
print(scipy.__version__, scikitplot.__version__)

files = [f for f in Path('./data/input/geschirr').glob('*')]
s = files[0].name
idx = s.find('_')
Klassenliste = []
filenameliste = [fn for fn in Path('./data/input/geschirr').glob('*')]
for filename in filenameliste:
    idx=filename.name.find('_')
    klasse = filename.name[:idx]
```

```
    Klassenliste.append(klasse)
len(Klassenliste)
Klassennamen = np.unique(Klassenliste).tolist()
```

```
1.11.4 0.3.7
```

[51]:
```
import pandas as pd
Anzahl_Trainingsbeispiele = np.floor(pd.Series(Klassenliste).value_counts()*0.
 ↪6).astype('int')
Anzahl_Trainingsbeispiele
```

[51]:
```
gabel-01           26
loeffel-01         25
messer-01          24
korkenzieher-01    21
trinkhalm-01       10
Name: count, dtype: int64
```

Nun erstellen wir die Baumstruktur, und zwar unter dem folgenden Pfad (nicht in allen Pfaden lassen sich Unterverzeichnisse erstellen!):

[52]:
```
Baumstrukturpfad = './media/geschirr-raw'
Baumstrukturpfad=Path(Baumstrukturpfad)
(Baumstrukturpfad/'train').mkdir(parents=True,exist_ok=True)
(Baumstrukturpfad/'test').mkdir(parents=True,exist_ok=True)
for klasse in Klassennamen:
    pfad_train = (Baumstrukturpfad/'train'/klasse)
    pfad_test = (Baumstrukturpfad/'test'/klasse)

    pfad_train.mkdir(parents=True,exist_ok=True)
    pfad_test.mkdir(parents=True,exist_ok=True)
```

DefaultDict-Dictionaries sind im folgenden nützlich. Dieses Video gibt eine gute Einführung.

[53]:
```
from collections import defaultdict
already_copied=defaultdict(lambda :0) #defaultdict's sind Dictionaries mit␣
 ↪einem Default-Returnwert, hier 0
already_copied['noch nicht existierender Key']
```

[53]:
```
0
```

Wir bauen damit nun eine if/else-Klausel, so dass die in `Anzahl_Trainingsbeispiele` angegebene Anzahl Dateien als Trainingsdaten verwendet (in das Unterverzeichnis 'train' kopiert) werden, und der Rest als Testdaten verwendet werden.

[54]:
```
import shutil
train_or_test=dict()
i=0
for ifile,(filename,klasse) in enumerate(zip(filenameliste,Klassenliste)):
```

```
    if ifile<6: #einfach, um ein paar Erklärungen zu erhalten, aber nicht zu␣
 ↪viele
        print('kopiere',filename,'nach',Baumstrukturpfad/klasse/filename.name)
    elif ifile==6:
        print('...')
    if already_copied[klasse]<Anzahl_Trainingsbeispiele[klasse]:
        shutil.copyfile(filename,Baumstrukturpfad/'train'/klasse/filename.name)
        already_copied[klasse]+=1 # zähle mit, wieviele Beispiele schon in das␣
 ↪Verzeichnis 'train' kopiert wurden.
        train_or_test[filename.name]='train'
    else:
        shutil.copyfile(filename,Baumstrukturpfad/'test'/klasse/filename.name)
        train_or_test[filename.name]='test'

print('Done.')
```

```
kopiere data/input/geschirr/messer-01_9554.jpg nach media/geschirr-
raw/messer-01/messer-01_9554.jpg
kopiere data/input/geschirr/messer-01_9568.jpg nach media/geschirr-
raw/messer-01/messer-01_9568.jpg
kopiere data/input/geschirr/messer-01_9583.jpg nach media/geschirr-
raw/messer-01/messer-01_9583.jpg
kopiere data/input/geschirr/gabel-01_9638.jpg nach media/geschirr-
raw/gabel-01/gabel-01_9638.jpg
kopiere data/input/geschirr/korkenzieher-01_9732.jpg nach media/geschirr-
raw/korkenzieher-01/korkenzieher-01_9732.jpg
kopiere data/input/geschirr/korkenzieher-01_9726.jpg nach media/geschirr-
raw/korkenzieher-01/korkenzieher-01_9726.jpg
…
Done.
```

Überprüfen wir, dass die Bilder an die richtige Stelle kopiert wurden!

[55]:
```
!ls {Baumstrukturpfad}/*/*
```

```
media/geschirr-raw/test/gabel-01:
gabel-01_0_9638.jpg gabel-01_0_9654.jpg gabel-01_0_9671.jpg gabel-01_9643.jpg
gabel-01_0_9639.jpg gabel-01_0_9655.jpg gabel-01_0_9672.jpg gabel-01_9644.jpg
gabel-01_0_9640.jpg gabel-01_0_9657.jpg gabel-01_0_9673.jpg gabel-01_9645.jpg
gabel-01_0_9641.jpg gabel-01_0_9659.jpg gabel-01_0_9674.jpg gabel-01_9650.jpg
gabel-01_0_9643.jpg gabel-01_0_9660.jpg gabel-01_0_9675.jpg gabel-01_9651.jpg
gabel-01_0_9644.jpg gabel-01_0_9661.jpg gabel-01_0_9676.jpg gabel-01_9654.jpg
gabel-01_0_9645.jpg gabel-01_0_9662.jpg gabel-01_0_9679.jpg gabel-01_9655.jpg
gabel-01_0_9646.jpg gabel-01_0_9663.jpg gabel-01_0_9680.jpg gabel-01_9656.jpg
gabel-01_0_9647.jpg gabel-01_0_9664.jpg gabel-01_1_9658.jpg gabel-01_9657.jpg
gabel-01_0_9648.jpg gabel-01_0_9665.jpg gabel-01_1_9667.jpg gabel-01_9668.jpg
gabel-01_0_9649.jpg gabel-01_0_9666.jpg gabel-01_1_9677.jpg gabel-01_9669.jpg
gabel-01_0_9650.jpg gabel-01_0_9667.jpg gabel-01_1_9678.jpg gabel-01_9678.jpg
gabel-01_0_9651.jpg gabel-01_0_9668.jpg gabel-01_9640.jpg   gabel-01_9679.jpg
```

```
gabel-01_0_9652.jpg gabel-01_0_9669.jpg gabel-01_9641.jpg   gabel-01_9680.jpg
gabel-01_0_9653.jpg gabel-01_0_9670.jpg gabel-01_9642.jpg   gabel-01_9681.jpg

media/geschirr-raw/test/korkenzieher-01:
korkenzieher-01_0_9700.jpg korkenzieher-01_0_9726.jpg
korkenzieher-01_0_9701.jpg korkenzieher-01_0_9727.jpg
korkenzieher-01_0_9702.jpg korkenzieher-01_0_9728.jpg
korkenzieher-01_0_9703.jpg korkenzieher-01_0_9729.jpg
korkenzieher-01_0_9704.jpg korkenzieher-01_0_9730.jpg
korkenzieher-01_0_9705.jpg korkenzieher-01_0_9731.jpg
korkenzieher-01_0_9706.jpg korkenzieher-01_0_9732.jpg
korkenzieher-01_0_9707.jpg korkenzieher-01_0_9733.jpg
korkenzieher-01_0_9708.jpg korkenzieher-01_0_9734.jpg
korkenzieher-01_0_9709.jpg korkenzieher-01_1_9716.jpg
korkenzieher-01_0_9710.jpg korkenzieher-01_9700.jpg
korkenzieher-01_0_9711.jpg korkenzieher-01_9701.jpg
korkenzieher-01_0_9712.jpg korkenzieher-01_9702.jpg
korkenzieher-01_0_9713.jpg korkenzieher-01_9703.jpg
korkenzieher-01_0_9714.jpg korkenzieher-01_9704.jpg
korkenzieher-01_0_9715.jpg korkenzieher-01_9705.jpg
korkenzieher-01_0_9717.jpg korkenzieher-01_9710.jpg
korkenzieher-01_0_9718.jpg korkenzieher-01_9711.jpg
korkenzieher-01_0_9719.jpg korkenzieher-01_9714.jpg
korkenzieher-01_0_9720.jpg korkenzieher-01_9715.jpg
korkenzieher-01_0_9721.jpg korkenzieher-01_9716.jpg
korkenzieher-01_0_9722.jpg korkenzieher-01_9717.jpg
korkenzieher-01_0_9723.jpg korkenzieher-01_9728.jpg
korkenzieher-01_0_9724.jpg korkenzieher-01_9729.jpg
korkenzieher-01_0_9725.jpg

media/geschirr-raw/test/loeffel-01:
loeffel-01_0_9511.jpg loeffel-01_1_9511.jpg loeffel-01_9518.jpg
loeffel-01_0_9514.jpg loeffel-01_1_9516.jpg loeffel-01_9519.jpg
loeffel-01_0_9518.jpg loeffel-01_1_9517.jpg loeffel-01_9524.jpg
loeffel-01_0_9519.jpg loeffel-01_1_9527.jpg loeffel-01_9525.jpg
loeffel-01_0_9520.jpg loeffel-01_1_9533.jpg loeffel-01_9527.jpg
loeffel-01_0_9521.jpg loeffel-01_1_9535.jpg loeffel-01_9530.jpg
loeffel-01_0_9522.jpg loeffel-01_1_9537.jpg loeffel-01_9531.jpg
loeffel-01_0_9523.jpg loeffel-01_1_9539.jpg loeffel-01_9532.jpg
loeffel-01_0_9528.jpg loeffel-01_1_9540.jpg loeffel-01_9533.jpg
loeffel-01_0_9529.jpg loeffel-01_1_9542.jpg loeffel-01_9536.jpg
loeffel-01_0_9530.jpg loeffel-01_1_9544.jpg loeffel-01_9540.jpg
loeffel-01_0_9532.jpg loeffel-01_1_9545.jpg loeffel-01_9541.jpg
loeffel-01_0_9534.jpg loeffel-01_2_9509.jpg loeffel-01_9542.jpg
loeffel-01_0_9536.jpg loeffel-01_2_9531.jpg loeffel-01_9543.jpg
loeffel-01_0_9537.jpg loeffel-01_2_9534.jpg loeffel-01_9544.jpg
loeffel-01_0_9540.jpg loeffel-01_2_9539.jpg loeffel-01_9545.jpg
loeffel-01_0_9541.jpg loeffel-01_2_9542.jpg loeffel-01_9546.jpg
```

```
loeffel-01_0_9546.jpg loeffel-01_2_9545.jpg
loeffel-01_1_9508.jpg loeffel-01_3_9509.jpg

media/geschirr-raw/test/messer-01:
messer-01_0_9548.jpg messer-01_0_9572.jpg messer-01_1_9586.jpg
messer-01_0_9549.jpg messer-01_0_9573.jpg messer-01_1_9587.jpg
messer-01_0_9552.jpg messer-01_0_9574.jpg messer-01_9548.jpg
messer-01_0_9554.jpg messer-01_0_9575.jpg messer-01_9558.jpg
messer-01_0_9555.jpg messer-01_0_9577.jpg messer-01_9559.jpg
messer-01_0_9556.jpg messer-01_0_9578.jpg messer-01_9560.jpg
messer-01_0_9557.jpg messer-01_0_9579.jpg messer-01_9562.jpg
messer-01_0_9558.jpg messer-01_0_9580.jpg messer-01_9563.jpg
messer-01_0_9559.jpg messer-01_0_9582.jpg messer-01_9564.jpg
messer-01_0_9560.jpg messer-01_0_9583.jpg messer-01_9565.jpg
messer-01_0_9561.jpg messer-01_1_9547.jpg messer-01_9566.jpg
messer-01_0_9563.jpg messer-01_1_9550.jpg messer-01_9567.jpg
messer-01_0_9564.jpg messer-01_1_9551.jpg messer-01_9570.jpg
messer-01_0_9565.jpg messer-01_1_9553.jpg messer-01_9572.jpg
messer-01_0_9566.jpg messer-01_1_9562.jpg messer-01_9573.jpg
messer-01_0_9567.jpg messer-01_1_9576.jpg messer-01_9574.jpg
messer-01_0_9568.jpg messer-01_1_9581.jpg messer-01_9576.jpg
messer-01_0_9569.jpg messer-01_1_9584.jpg messer-01_9577.jpg
messer-01_0_9570.jpg messer-01_1_9585.jpg

media/geschirr-raw/test/trinkhalm-01:
trinkhalm-01_0_9682.jpg trinkhalm-01_0_9698.jpg trinkhalm-01_9683.jpg
trinkhalm-01_0_9689.jpg trinkhalm-01_0_9699.jpg trinkhalm-01_9684.jpg
trinkhalm-01_0_9690.jpg trinkhalm-01_1_9683.jpg trinkhalm-01_9690.jpg
trinkhalm-01_0_9691.jpg trinkhalm-01_1_9684.jpg trinkhalm-01_9694.jpg
trinkhalm-01_0_9694.jpg trinkhalm-01_1_9685.jpg trinkhalm-01_9695.jpg
trinkhalm-01_0_9695.jpg trinkhalm-01_1_9686.jpg trinkhalm-01_9696.jpg
trinkhalm-01_0_9696.jpg trinkhalm-01_1_9688.jpg trinkhalm-01_9697.jpg
trinkhalm-01_0_9697.jpg trinkhalm-01_9682.jpg

media/geschirr-raw/train/gabel-01:
gabel-01_0_9642.jpg gabel-01_9649.jpg   gabel-01_9663.jpg   gabel-01_9673.jpg
gabel-01_0_9656.jpg gabel-01_9652.jpg   gabel-01_9664.jpg   gabel-01_9674.jpg
gabel-01_1_9681.jpg gabel-01_9653.jpg   gabel-01_9665.jpg   gabel-01_9675.jpg
gabel-01_9638.jpg   gabel-01_9658.jpg   gabel-01_9666.jpg   gabel-01_9676.jpg
gabel-01_9639.jpg   gabel-01_9659.jpg   gabel-01_9667.jpg   gabel-01_9677.jpg
gabel-01_9646.jpg   gabel-01_9660.jpg   gabel-01_9670.jpg
gabel-01_9647.jpg   gabel-01_9661.jpg   gabel-01_9671.jpg
gabel-01_9648.jpg   gabel-01_9662.jpg   gabel-01_9672.jpg

media/geschirr-raw/train/korkenzieher-01:
korkenzieher-01_9706.jpg korkenzieher-01_9719.jpg korkenzieher-01_9726.jpg
korkenzieher-01_9707.jpg korkenzieher-01_9720.jpg korkenzieher-01_9727.jpg
korkenzieher-01_9708.jpg korkenzieher-01_9721.jpg korkenzieher-01_9730.jpg
```

```
korkenzieher-01_9709.jpg korkenzieher-01_9722.jpg korkenzieher-01_9731.jpg
korkenzieher-01_9712.jpg korkenzieher-01_9723.jpg korkenzieher-01_9732.jpg
korkenzieher-01_9713.jpg korkenzieher-01_9724.jpg korkenzieher-01_9733.jpg
korkenzieher-01_9718.jpg korkenzieher-01_9725.jpg korkenzieher-01_9734.jpg

media/geschirr-raw/train/loeffel-01:
loeffel-01_0_9512.jpg loeffel-01_9506.jpg   loeffel-01_9520.jpg
loeffel-01_0_9538.jpg loeffel-01_9507.jpg   loeffel-01_9521.jpg
loeffel-01_1_9504.jpg loeffel-01_9508.jpg   loeffel-01_9522.jpg
loeffel-01_1_9505.jpg loeffel-01_9509.jpg   loeffel-01_9523.jpg
loeffel-01_1_9506.jpg loeffel-01_9510.jpg   loeffel-01_9528.jpg
loeffel-01_1_9507.jpg loeffel-01_9511.jpg   loeffel-01_9529.jpg
loeffel-01_1_9510.jpg loeffel-01_9512.jpg   loeffel-01_9534.jpg
loeffel-01_1_9513.jpg loeffel-01_9513.jpg   loeffel-01_9535.jpg
loeffel-01_2_9543.jpg loeffel-01_9514.jpg   loeffel-01_9537.jpg
loeffel-01_3_9543.jpg loeffel-01_9515.jpg   loeffel-01_9538.jpg
loeffel-01_9504.jpg   loeffel-01_9516.jpg   loeffel-01_9539.jpg
loeffel-01_9505.jpg   loeffel-01_9517.jpg

media/geschirr-raw/train/messer-01:
messer-01_9547.jpg messer-01_9554.jpg messer-01_9569.jpg messer-01_9582.jpg
messer-01_9549.jpg messer-01_9555.jpg messer-01_9575.jpg messer-01_9583.jpg
messer-01_9550.jpg messer-01_9556.jpg messer-01_9578.jpg messer-01_9584.jpg
messer-01_9551.jpg messer-01_9557.jpg messer-01_9579.jpg messer-01_9585.jpg
messer-01_9552.jpg messer-01_9561.jpg messer-01_9580.jpg messer-01_9586.jpg
messer-01_9553.jpg messer-01_9568.jpg messer-01_9581.jpg messer-01_9587.jpg

media/geschirr-raw/train/trinkhalm-01:
trinkhalm-01_0_9692.jpg trinkhalm-01_9687.jpg   trinkhalm-01_9693.jpg
trinkhalm-01_0_9693.jpg trinkhalm-01_9688.jpg   trinkhalm-01_9698.jpg
trinkhalm-01_1_9687.jpg trinkhalm-01_9689.jpg   trinkhalm-01_9699.jpg
trinkhalm-01_9685.jpg   trinkhalm-01_9691.jpg
trinkhalm-01_9686.jpg   trinkhalm-01_9692.jpg
```

Wir sollten im Folgenden *nur noch Trainingsbeispiele zum Training des neuronalen Netzes verwenden.* Die Testbeispiele nutzen wir erst, wenn das Neuronale Netzwerk fertig optimiert ist und kurz vor der Inbetriebnahme steht.

[56]:
```
import os,datetime
os.environ["TF_CPP_MIN_LOG_LEVEL"] = '1'
#os.environ["CUDA_VISIBLE_DEVICES"] = "-1" #dies schaltet die GPU aus.␣
 ↪GPU-Nutzung führt bei mir zu einem Kernel-Crash...
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers
import scikitplot as skplt
import numpy as np
```

```
tf.config.list_physical_devices('GPU') #Wenn keine GPU Verfügbar ist, ist das␣
 ↪nicht weiter schlimm (nur langsamer)
```

[56]: []

Die Unterverzeichnisse von `path` (2_labelling/Beispiel/Bauteile/) stellen die Klassen dar, welche gelernt werden. In jedem Unterverzeichniss sollten sich Bilder dieser Klasse befinden.

[57]:
```
Datensatzname = 'Geschirr-v4'  #Unbedingt anpassen! Das Trainierte Modell wird␣
 ↪später mit diesem String bezeichnet.
```

[58]:
```
Baumstrukturpfad = Path(os.path.expanduser(Baumstrukturpfad))
downloadable_output_path = './working' #Nach Klick auf "Save Version" wird der␣
 ↪Inhalt dieses Verzeichnisses als Output zugänglich.
model_output_path = os.path.join(downloadable_output_path,'model')  # kann␣
 ↪angepasst werden! Gut überprüfen
if not os.path.exists(model_output_path):
    os.mkdir(model_output_path)
print(f'Modell wird nach {model_output_path} geschrieben')
```

```
Modell wird nach ./working/model geschrieben
```

[59]:
```
!ls {Baumstrukturpfad}/'train'
```

```
gabel-01        korkenzieher-01 loeffel-01
messer-01       trinkhalm-01
```

In der letzten Zelle sollten die Klassen der Aufgabe angezeigt werden (nicht etwa die einzelnen Dateien). Ansonsten ist die Zeile mit `path=...` anzupassen!
Wir trainieren auf Bildern, welche 64x64 Pixel gross sein sollten. Wenn sie es im obigen Pfad nicht sind, werden sie entsprechend angepasst.

[60]:
```
image_size=(224,224) # increase to 128 x 128x to capture more details // mine
batch_size=16 # use smaller batch size to allow more learning updates per epoch␣
 ↪// mine
```

[61]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  Baumstrukturpfad/'train',
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=image_size,
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  Baumstrukturpfad/'train',  #es ist wichtig, dass zur Validierung keine␣
 ↪Testdaten verwendet werden!
  validation_split=0.2,
  subset="validation",
```

```
  seed=123,
  image_size=image_size,
  batch_size=batch_size)
```

```
Found 122 files belonging to 5 classes.
Using 98 files for training.
Found 122 files belonging to 5 classes.
Using 24 files for validation.
```

Keras hat nun alle Unterverzeichnisse eingelesen. Die Reihenfolge der Klassennamen können wir so ausgeben:

[62]:
```
class_names = train_ds.class_names
class_names
```

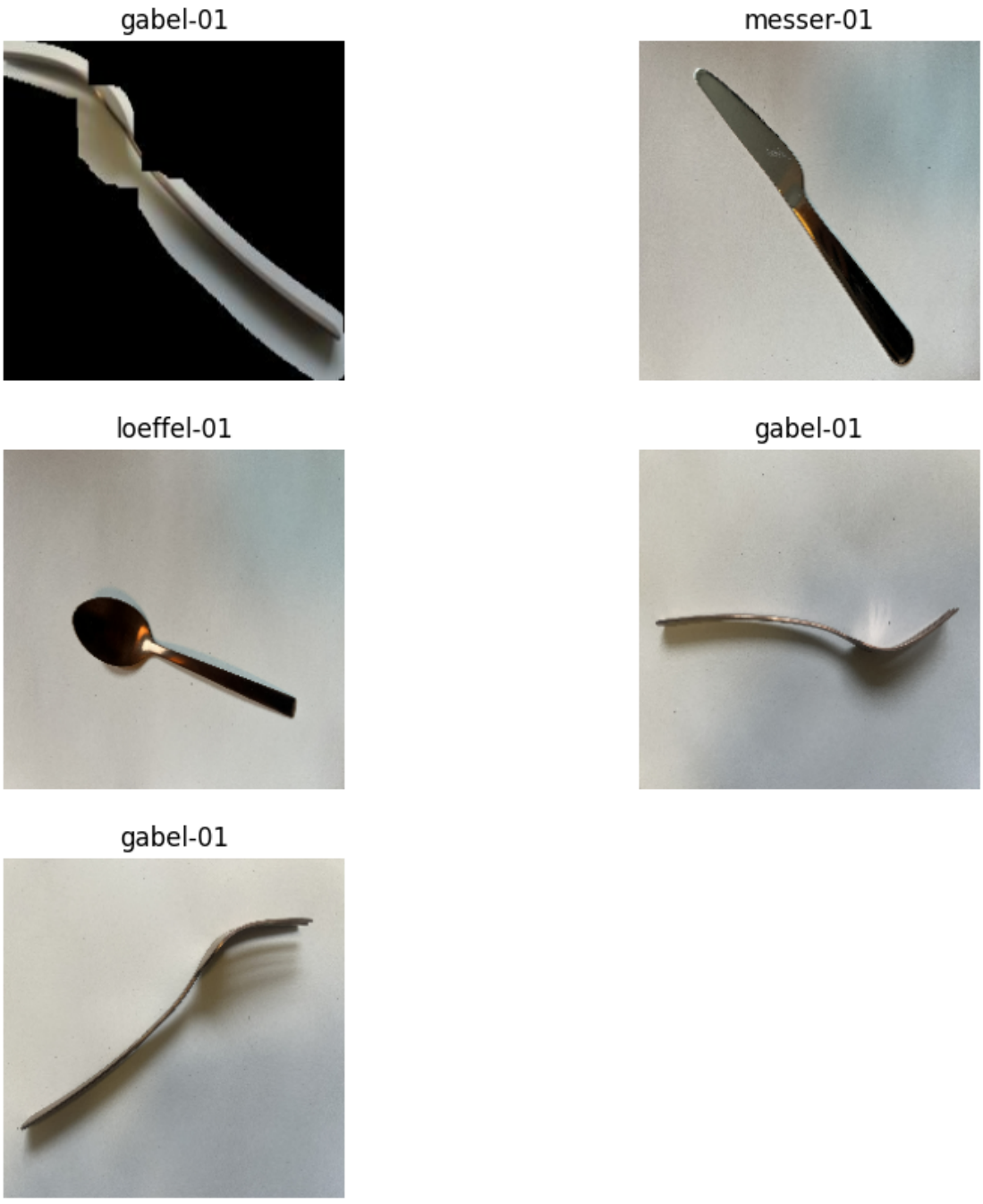
[62]:
```
['gabel-01', 'korkenzieher-01', 'loeffel-01', 'messer-01', 'trinkhalm-01']
```

Unser Modell wird später einfach nur z.B. “Klasse 3” ausspucken. Dies bedeutet dann, dass ein Bild der Klasse vorliegt, welche den namen des 4. Eintrags in obiger Liste hat.

Schauen wir uns ein paar Trainingsbilder an! Wenn Sie denken ,dass diese Bilder für das neuronale Netz eine zu geringe Auflösung haben, als dass etwas erkannt werden könnte: dann kann weiter oben der Wert der Variablen image_size angepasst werden. `image_size=(64,64)` sollte aber meist ausreichen.

[63]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(5):
        ax = plt.subplot(3, 2, i + 1)
        im = images[i].numpy().astype("uint8")
        plt.imshow(im)
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



Die folgende Zelle versucht, die Hardware gut auszulasten. Es gäbe viel dazu zu sagen, aber sagen wir einfach, dass die Daten für einen raschen Zugriff vorbereitet werden:

[64]:
```
train_ds = train_ds.cache().shuffle(100).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
```

Nun definieren wir unser Modell! Hier können Sie Schichten von Neuronalen Netzen hinzufügen,

die Aktivierungsfunktionen verändern und vieles mehr. Wichtig ist, dass `num_classes` stimmt, die Anzahl der Klassen und gleichzeitig die Anzahl Neuronen in der letzten (Ausgabe-)Schicht. Und natürlich die Grösse des Eingabebildes, `image_size`.

[65]:
```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import Model

# Parameter für die Bildgröße (MobileNetV2 erwartet standardmäßig 224x224)
IMG_SIZE = (224, 224)
def make_model():
    # 1. Basis-Modell laden (vortrainiert auf ImageNet)
    base_model = MobileNetV2(input_shape=IMG_SIZE + (3,),
                             include_top=False,
                             weights='imagenet')

    # 2. Basis-Modell "einfrieren" (Gewichte werden nicht verändert)
    base_model.trainable = False

    # 3. Eigene Schichten oben hinzufügen
    inputs = tf.keras.Input(shape=IMG_SIZE + (3,))
    # MobileNetV2 benötigt eine spezielle Vorverarbeitung der Pixelwerte (-1␣
 ↪bis 1)
    x = tf.keras.applications.mobilenet_v2.preprocess_input(inputs)
    x = base_model(x, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.2)(x) # Verhindert Overfitting
    outputs = layers.Dense(len(Klassennamen), activation='softmax')(x)

    model = Model(inputs, outputs)

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model

model = make_model()
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv_2 (TFOpLam  (None, 224, 224, 3)       0
 bda)
```

```
 tf.math.subtract_2 (TFOpLa  (None, 224, 224, 3)       0
 mbda)

 mobilenetv2_1.00_224 (Func  (None, 7, 7, 1280)        2257984
 tional)

 global_average_pooling2d_2  (None, 1280)              0
  (GlobalAveragePooling2D)

 dropout_2 (Dropout)         (None, 1280)              0

 dense_2 (Dense)             (None, 5)                 6405

=================================================================
Total params: 2264389 (8.64 MB)
Trainable params: 6405 (25.02 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv_2 (TFOpLam  (None, 224, 224, 3)       0
 bda)

 tf.math.subtract_2 (TFOpLa  (None, 224, 224, 3)       0
 mbda)

 mobilenetv2_1.00_224 (Func  (None, 7, 7, 1280)        2257984
 tional)

 global_average_pooling2d_2  (None, 1280)              0
  (GlobalAveragePooling2D)

 dropout_2 (Dropout)         (None, 1280)              0

 dense_2 (Dense)             (None, 5)                 6405

=================================================================
Total params: 2264389 (8.64 MB)
Trainable params: 6405 (25.02 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Kompilieren des Modells heisst, dass der Gradientenabstieg und das Lernverfahren spezifiziert werden, und der low-level Code vorbereitet wird, welcher dann später die Hauptarbeit leistet.

[66]:
```
#
```

Es ist immer gut zu verstehen, was während des Trainings passiert- und allenfalls im richtigen Moment zu reagieren. Tensorboard ist eine Web-Applikation, mit der Sie den Trainingsfortschritt verfolgen können. EarlyStopping ist ein Callback, der genau das tut: Er unterbricht das Training, sobald sich die gemonitorte Grösse (hier die Validierungsgenauigkeit) nicht mehr verbessert.

[ ]:
```

```

[67]:
```
early_stopping_callback = tf.keras.callbacks.
↪EarlyStopping(monitor='val_accuracy',patience=100) #patience = 0? patience =␣
↪40? patience=200?
```

Und…**LOS GEHT'S!** Je nach Grösse des Trainingsdatensatzes kann die nächste Zeile länger dauern. Die Anzahl Epochen kann auch limitiert werden. Stellen Sie sie vorerst so ein, dass Sie sich das Warten *leisten* können. Ev. ist es besser, vorerst mal ein sehr schlechtes Modell zu erzeugen- einfach um zu sehen, ob der Rest des Codes auch wirklich funktioniert.

[68]:
```
epochs=100000000
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[early_stopping_callback]
)
```

```
Epoch 1/100000000
7/7 [==============================] - 2s 159ms/step - loss: 1.5842 - accuracy:
0.3980 - val_loss: 1.2577 - val_accuracy: 0.5833
Epoch 2/100000000
7/7 [==============================] - 1s 78ms/step - loss: 1.0108 - accuracy:
0.6327 - val_loss: 0.7565 - val_accuracy: 0.7917
Epoch 3/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.6029 - accuracy:
0.7857 - val_loss: 0.4661 - val_accuracy: 0.9167
Epoch 4/100000000
7/7 [==============================] - 1s 79ms/step - loss: 0.3907 - accuracy:
0.9184 - val_loss: 0.3240 - val_accuracy: 0.9167
Epoch 5/100000000
7/7 [==============================] - 1s 78ms/step - loss: 0.2786 - accuracy:
0.9490 - val_loss: 0.2294 - val_accuracy: 0.9167
Epoch 6/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.2126 - accuracy:
0.9592 - val_loss: 0.1729 - val_accuracy: 1.0000
Epoch 7/100000000
7/7 [==============================] - 1s 78ms/step - loss: 0.1522 - accuracy:
0.9796 - val_loss: 0.1547 - val_accuracy: 1.0000
Epoch 8/100000000
7/7 [==============================] - 1s 89ms/step - loss: 0.1364 - accuracy:
```

```
0.9796 - val_loss: 0.1348 - val_accuracy: 1.0000
Epoch 9/100000000
7/7 [==============================] - 1s 83ms/step - loss: 0.1123 - accuracy:
0.9796 - val_loss: 0.1141 - val_accuracy: 1.0000
Epoch 10/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.1056 - accuracy:
0.9898 - val_loss: 0.1080 - val_accuracy: 1.0000
Epoch 11/100000000
7/7 [==============================] - 1s 82ms/step - loss: 0.0920 - accuracy:
0.9898 - val_loss: 0.1020 - val_accuracy: 1.0000
Epoch 12/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0877 - accuracy:
0.9898 - val_loss: 0.1044 - val_accuracy: 1.0000
Epoch 13/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0683 - accuracy:
0.9898 - val_loss: 0.1044 - val_accuracy: 0.9583
Epoch 14/100000000
7/7 [==============================] - 1s 77ms/step - loss: 0.0724 - accuracy:
1.0000 - val_loss: 0.1092 - val_accuracy: 0.9583
Epoch 15/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0736 - accuracy:
0.9898 - val_loss: 0.0966 - val_accuracy: 1.0000
Epoch 16/100000000
7/7 [==============================] - 1s 82ms/step - loss: 0.0527 - accuracy:
1.0000 - val_loss: 0.0885 - val_accuracy: 1.0000
Epoch 17/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0507 - accuracy:
1.0000 - val_loss: 0.0827 - val_accuracy: 1.0000
Epoch 18/100000000
7/7 [==============================] - 1s 80ms/step - loss: 0.0404 - accuracy:
1.0000 - val_loss: 0.0800 - val_accuracy: 1.0000
Epoch 19/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0545 - accuracy:
0.9898 - val_loss: 0.0816 - val_accuracy: 1.0000
Epoch 20/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0493 - accuracy:
0.9898 - val_loss: 0.0788 - val_accuracy: 1.0000
Epoch 21/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0354 - accuracy:
1.0000 - val_loss: 0.0743 - val_accuracy: 1.0000
Epoch 22/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0365 - accuracy:
1.0000 - val_loss: 0.0750 - val_accuracy: 1.0000
Epoch 23/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0396 - accuracy:
1.0000 - val_loss: 0.0754 - val_accuracy: 1.0000
Epoch 24/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0281 - accuracy:
```

```
1.0000 - val_loss: 0.0769 - val_accuracy: 1.0000
Epoch 25/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0280 - accuracy:
1.0000 - val_loss: 0.0761 - val_accuracy: 1.0000
Epoch 26/100000000
7/7 [==============================] - 0s 73ms/step - loss: 0.0296 - accuracy:
1.0000 - val_loss: 0.0696 - val_accuracy: 1.0000
Epoch 27/100000000
7/7 [==============================] - 1s 81ms/step - loss: 0.0247 - accuracy:
1.0000 - val_loss: 0.0659 - val_accuracy: 1.0000
Epoch 28/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0177 - accuracy:
1.0000 - val_loss: 0.0621 - val_accuracy: 1.0000
Epoch 29/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0191 - accuracy:
1.0000 - val_loss: 0.0571 - val_accuracy: 1.0000
Epoch 30/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0189 - accuracy:
1.0000 - val_loss: 0.0554 - val_accuracy: 1.0000
Epoch 31/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0207 - accuracy:
1.0000 - val_loss: 0.0555 - val_accuracy: 1.0000
Epoch 32/100000000
7/7 [==============================] - 1s 79ms/step - loss: 0.0207 - accuracy:
1.0000 - val_loss: 0.0587 - val_accuracy: 1.0000
Epoch 33/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0186 - accuracy:
1.0000 - val_loss: 0.0617 - val_accuracy: 1.0000
Epoch 34/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0180 - accuracy:
1.0000 - val_loss: 0.0635 - val_accuracy: 1.0000
Epoch 35/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0204 - accuracy:
1.0000 - val_loss: 0.0674 - val_accuracy: 1.0000
Epoch 36/100000000
7/7 [==============================] - 1s 85ms/step - loss: 0.0171 - accuracy:
1.0000 - val_loss: 0.0688 - val_accuracy: 1.0000
Epoch 37/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0169 - accuracy:
1.0000 - val_loss: 0.0681 - val_accuracy: 1.0000
Epoch 38/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0178 - accuracy:
1.0000 - val_loss: 0.0630 - val_accuracy: 1.0000
Epoch 39/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0165 - accuracy:
1.0000 - val_loss: 0.0585 - val_accuracy: 1.0000
Epoch 40/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0148 - accuracy:
```

```
1.0000 - val_loss: 0.0548 - val_accuracy: 1.0000
Epoch 41/100000000
7/7 [==============================] - 0s 71ms/step - loss: 0.0156 - accuracy:
1.0000 - val_loss: 0.0492 - val_accuracy: 1.0000
Epoch 42/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0146 - accuracy:
1.0000 - val_loss: 0.0455 - val_accuracy: 1.0000
Epoch 43/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0131 - accuracy:
1.0000 - val_loss: 0.0427 - val_accuracy: 1.0000
Epoch 44/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0151 - accuracy:
1.0000 - val_loss: 0.0508 - val_accuracy: 1.0000
Epoch 45/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0132 - accuracy:
1.0000 - val_loss: 0.0519 - val_accuracy: 1.0000
Epoch 46/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0133 - accuracy:
1.0000 - val_loss: 0.0539 - val_accuracy: 1.0000
Epoch 47/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0102 - accuracy:
1.0000 - val_loss: 0.0536 - val_accuracy: 1.0000
Epoch 48/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0117 - accuracy:
1.0000 - val_loss: 0.0546 - val_accuracy: 1.0000
Epoch 49/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0120 - accuracy:
1.0000 - val_loss: 0.0491 - val_accuracy: 1.0000
Epoch 50/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0121 - accuracy:
1.0000 - val_loss: 0.0470 - val_accuracy: 1.0000
Epoch 51/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0079 - accuracy:
1.0000 - val_loss: 0.0476 - val_accuracy: 1.0000
Epoch 52/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0099 - accuracy:
1.0000 - val_loss: 0.0471 - val_accuracy: 1.0000
Epoch 53/100000000
7/7 [==============================] - 1s 77ms/step - loss: 0.0107 - accuracy:
1.0000 - val_loss: 0.0470 - val_accuracy: 1.0000
Epoch 54/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0080 - accuracy:
1.0000 - val_loss: 0.0460 - val_accuracy: 1.0000
Epoch 55/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0093 - accuracy:
1.0000 - val_loss: 0.0425 - val_accuracy: 1.0000
Epoch 56/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0068 - accuracy:
```

```
1.0000 - val_loss: 0.0383 - val_accuracy: 1.0000
Epoch 57/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0090 - accuracy:
1.0000 - val_loss: 0.0359 - val_accuracy: 1.0000
Epoch 58/100000000
7/7 [==============================] - 0s 78ms/step - loss: 0.0080 - accuracy:
1.0000 - val_loss: 0.0366 - val_accuracy: 1.0000
Epoch 59/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0088 - accuracy:
1.0000 - val_loss: 0.0357 - val_accuracy: 1.0000
Epoch 60/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0069 - accuracy:
1.0000 - val_loss: 0.0354 - val_accuracy: 1.0000
Epoch 61/100000000
7/7 [==============================] - 0s 73ms/step - loss: 0.0086 - accuracy:
1.0000 - val_loss: 0.0380 - val_accuracy: 1.0000
Epoch 62/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0066 - accuracy:
1.0000 - val_loss: 0.0401 - val_accuracy: 1.0000
Epoch 63/100000000
7/7 [==============================] - 1s 82ms/step - loss: 0.0075 - accuracy:
1.0000 - val_loss: 0.0425 - val_accuracy: 1.0000
Epoch 64/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0083 - accuracy:
1.0000 - val_loss: 0.0430 - val_accuracy: 1.0000
Epoch 65/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0061 - accuracy:
1.0000 - val_loss: 0.0419 - val_accuracy: 1.0000
Epoch 66/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0064 - accuracy:
1.0000 - val_loss: 0.0414 - val_accuracy: 1.0000
Epoch 67/100000000
7/7 [==============================] - 1s 81ms/step - loss: 0.0065 - accuracy:
1.0000 - val_loss: 0.0416 - val_accuracy: 1.0000
Epoch 68/100000000
7/7 [==============================] - 1s 82ms/step - loss: 0.0069 - accuracy:
1.0000 - val_loss: 0.0398 - val_accuracy: 1.0000
Epoch 69/100000000
7/7 [==============================] - 0s 77ms/step - loss: 0.0060 - accuracy:
1.0000 - val_loss: 0.0391 - val_accuracy: 1.0000
Epoch 70/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0053 - accuracy:
1.0000 - val_loss: 0.0379 - val_accuracy: 1.0000
Epoch 71/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0053 - accuracy:
1.0000 - val_loss: 0.0361 - val_accuracy: 1.0000
Epoch 72/100000000
7/7 [==============================] - 0s 71ms/step - loss: 0.0068 - accuracy:
```

```
1.0000 - val_loss: 0.0348 - val_accuracy: 1.0000
Epoch 73/100000000
7/7 [==============================] - 0s 77ms/step - loss: 0.0052 - accuracy:
1.0000 - val_loss: 0.0322 - val_accuracy: 1.0000
Epoch 74/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0052 - accuracy:
1.0000 - val_loss: 0.0314 - val_accuracy: 1.0000
Epoch 75/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0057 - accuracy:
1.0000 - val_loss: 0.0317 - val_accuracy: 1.0000
Epoch 76/100000000
7/7 [==============================] - 1s 81ms/step - loss: 0.0061 - accuracy:
1.0000 - val_loss: 0.0331 - val_accuracy: 1.0000
Epoch 77/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0049 - accuracy:
1.0000 - val_loss: 0.0328 - val_accuracy: 1.0000
Epoch 78/100000000
7/7 [==============================] - 1s 76ms/step - loss: 0.0043 - accuracy:
1.0000 - val_loss: 0.0325 - val_accuracy: 1.0000
Epoch 79/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0049 - accuracy:
1.0000 - val_loss: 0.0318 - val_accuracy: 1.0000
Epoch 80/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0048 - accuracy:
1.0000 - val_loss: 0.0321 - val_accuracy: 1.0000
Epoch 81/100000000
7/7 [==============================] - 0s 71ms/step - loss: 0.0041 - accuracy:
1.0000 - val_loss: 0.0313 - val_accuracy: 1.0000
Epoch 82/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0040 - accuracy:
1.0000 - val_loss: 0.0314 - val_accuracy: 1.0000
Epoch 83/100000000
7/7 [==============================] - 1s 81ms/step - loss: 0.0044 - accuracy:
1.0000 - val_loss: 0.0318 - val_accuracy: 1.0000
Epoch 84/100000000
7/7 [==============================] - 1s 81ms/step - loss: 0.0050 - accuracy:
1.0000 - val_loss: 0.0312 - val_accuracy: 1.0000
Epoch 85/100000000
7/7 [==============================] - 1s 79ms/step - loss: 0.0047 - accuracy:
1.0000 - val_loss: 0.0296 - val_accuracy: 1.0000
Epoch 86/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0054 - accuracy:
1.0000 - val_loss: 0.0278 - val_accuracy: 1.0000
Epoch 87/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0042 - accuracy:
1.0000 - val_loss: 0.0262 - val_accuracy: 1.0000
Epoch 88/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0039 - accuracy:
```

```
1.0000 - val_loss: 0.0249 - val_accuracy: 1.0000
Epoch 89/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0045 - accuracy:
1.0000 - val_loss: 0.0247 - val_accuracy: 1.0000
Epoch 90/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0050 - accuracy:
1.0000 - val_loss: 0.0242 - val_accuracy: 1.0000
Epoch 91/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0045 - accuracy:
1.0000 - val_loss: 0.0236 - val_accuracy: 1.0000
Epoch 92/100000000
7/7 [==============================] - 0s 68ms/step - loss: 0.0053 - accuracy:
1.0000 - val_loss: 0.0237 - val_accuracy: 1.0000
Epoch 93/100000000
7/7 [==============================] - 0s 70ms/step - loss: 0.0042 - accuracy:
1.0000 - val_loss: 0.0255 - val_accuracy: 1.0000
Epoch 94/100000000
7/7 [==============================] - 0s 71ms/step - loss: 0.0042 - accuracy:
1.0000 - val_loss: 0.0257 - val_accuracy: 1.0000
Epoch 95/100000000
7/7 [==============================] - 0s 73ms/step - loss: 0.0047 - accuracy:
1.0000 - val_loss: 0.0246 - val_accuracy: 1.0000
Epoch 96/100000000
7/7 [==============================] - 1s 73ms/step - loss: 0.0046 - accuracy:
1.0000 - val_loss: 0.0242 - val_accuracy: 1.0000
Epoch 97/100000000
7/7 [==============================] - 0s 72ms/step - loss: 0.0034 - accuracy:
1.0000 - val_loss: 0.0240 - val_accuracy: 1.0000
Epoch 98/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 0.0242 - val_accuracy: 1.0000
Epoch 99/100000000
7/7 [==============================] - 0s 71ms/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 0.0249 - val_accuracy: 1.0000
Epoch 100/100000000
7/7 [==============================] - 1s 72ms/step - loss: 0.0026 - accuracy:
1.0000 - val_loss: 0.0252 - val_accuracy: 1.0000
Epoch 101/100000000
7/7 [==============================] - 0s 71ms/step - loss: 0.0046 - accuracy:
1.0000 - val_loss: 0.0253 - val_accuracy: 1.0000
Epoch 102/100000000
7/7 [==============================] - 1s 74ms/step - loss: 0.0043 - accuracy:
1.0000 - val_loss: 0.0244 - val_accuracy: 1.0000
Epoch 103/100000000
7/7 [==============================] - 1s 83ms/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 0.0244 - val_accuracy: 1.0000
Epoch 104/100000000
7/7 [==============================] - 1s 77ms/step - loss: 0.0031 - accuracy:
```

```
1.0000 - val_loss: 0.0251 - val_accuracy: 1.0000
Epoch 105/100000000
7/7 [==============================] - 1s 81ms/step - loss: 0.0037 - accuracy:
1.0000 - val_loss: 0.0253 - val_accuracy: 1.0000
Epoch 106/100000000
7/7 [==============================] - 1s 75ms/step - loss: 0.0033 - accuracy:
1.0000 - val_loss: 0.0249 - val_accuracy: 1.0000
```

Das Modell hat hier (hoffentlich) fertig trainiert, und keinen Fehler produziert (möglich wären z.B. OutofMemory-Fehler o.ä.). Überprüfen Sie das! Denn nur bei Erfolgreichem Training lohnt es sich nun, die Validierungsgenauigkeit angeben zu lassen.

[69]:
```
loss, metrics = model.evaluate(val_ds)
```

```
2/2 [==============================] - 0s 44ms/step - loss: 0.0249 - accuracy:
1.0000
```

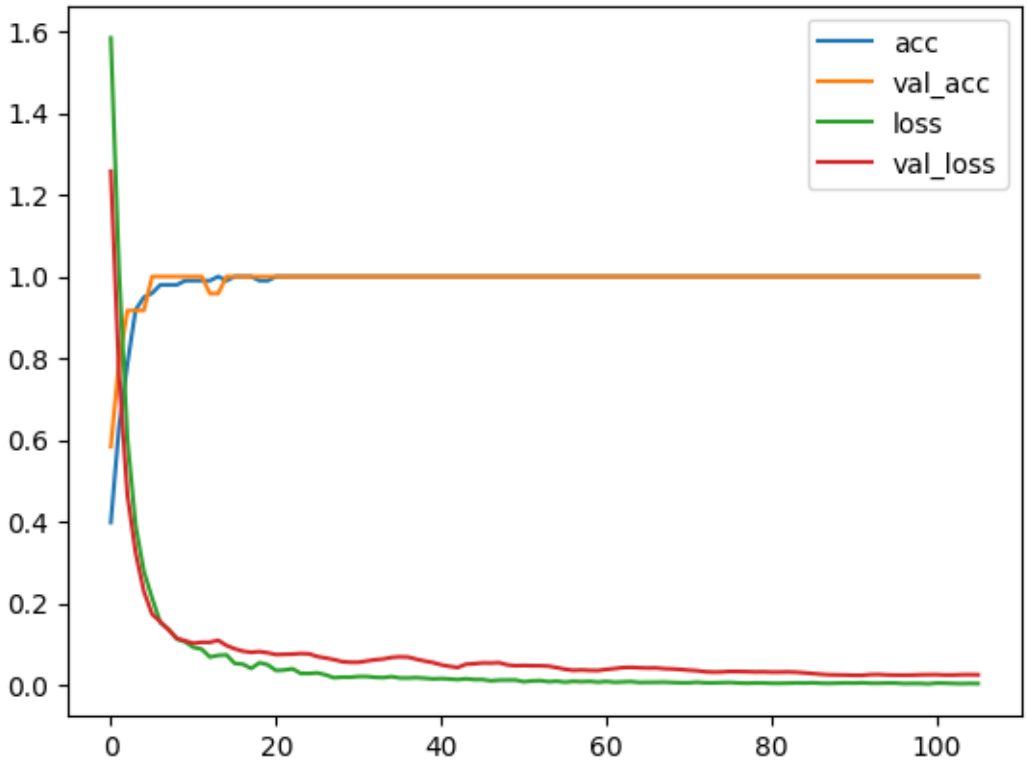
[70]:
```
loss, metrics ,history.history.keys()
```

[70]:
```
(0.024893201887607574,
 1.0,
 dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy']))
```

Das `history`-Objekt ist interessant, um zu sehen, wie der Trainingsverlauf war (wenn Sie das nicht schon auf Tensorboard gesehen haben.

[71]:
```
plt.plot(history.history['accuracy'],label='acc')
plt.plot(history.history['val_accuracy'],label='val_acc')
plt.plot(history.history['loss'],label='loss')
plt.plot(history.history['val_loss'],label='val_loss')
plt.legend()
#plt.savefig('Trainingskurven.png')
```

[71]:
```
<matplotlib.legend.Legend at 0x386845f60>
```



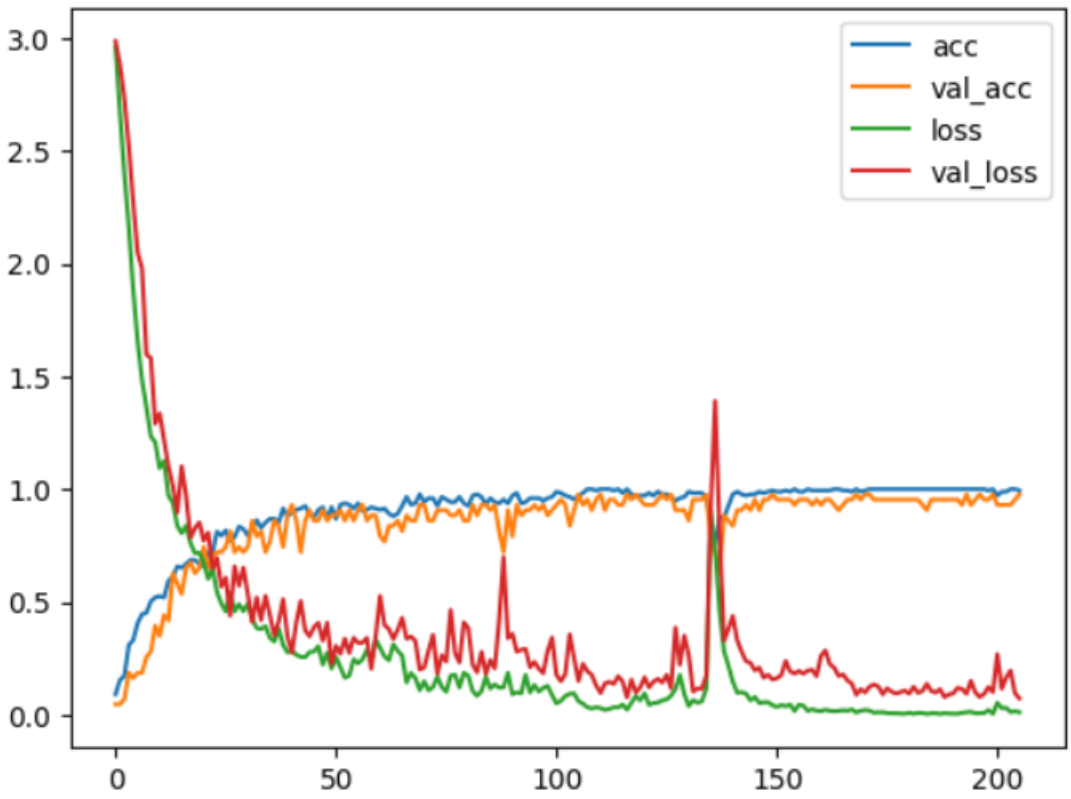
Es lohnt sich, die Zeile

```
early_stopping_callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=20)
```

mal auf

```
early_stopping_callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=200)
```

o.ä. zu setzen, und ab der Zelle welche die Modelldefinition enthält (mit `model = keras.Model(...)`) den Code nochmals auszuführen (das Modell enthält die aktuellen Gewichte und trainiert bei erneutem Ausführen der `.fit`-Methode das bisherige Modell weiter, anstatt wieder neu bei zufällig initialisierten Gewichten zu beginnen!). Das Modell wird dann deutlich länger trainiert.



Beachten Sie, wie die Validierungsgenauigkeit schon lange stagniert, der Validierungsloss, d.h. der Wert der Verlustfunktion ausgewertet auf dem Validierungsdatensatz, seit etwa der 40. Epoche nur noch zunimmt. Der Validierungsdatensatz ist hier zu klein, um mehr sagen zu können, aber auch die Validierungsgenauigkeit scheint nach etwa der 40. Epoche abzunehmen! Es ist daher Wohl sinnvoll, dieses Modell auf diesem Datensatz nur 40 Epochen lang zu trainieren.

Dazu können Sie also nochmals das Modell erstellen, die Anzahl Epochen auf 40 setzen und Ihr finales Modell trainieren.

1.0.1 Modell eine vorgegebene Anzahl Epochen lang trainieren

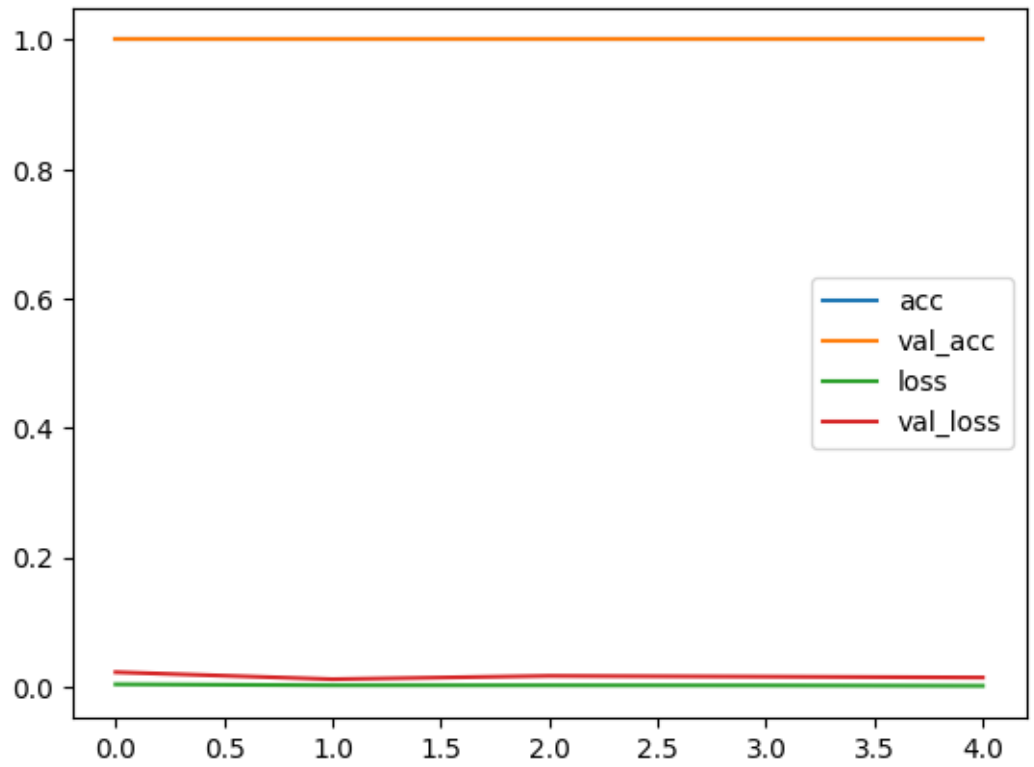
[72]:
```
vorgegebene_Anzahl_Epochen = 5 # Hier den Wert gemäss dem obigen Plot anpassen!
final_model = model #make_model() # Die bisherigen Gewichte sind damit␣
 ↪zurückgesetzt, d.h. es wird nicht weitertrainiert, sondern nochmals ganz von␣
 ↪vorne.
final_model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

final_history = final_model.fit(
  train_ds,
  validation_data=val_ds,
```

```
  epochs=vorgegebene_Anzahl_Epochen
)
```

```
Epoch 1/5
7/7 [==============================] - 1s 119ms/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 0.0223 - val_accuracy: 1.0000
Epoch 2/5
7/7 [==============================] - 1s 73ms/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 0.0115 - val_accuracy: 1.0000
Epoch 3/5
7/7 [==============================] - 1s 74ms/step - loss: 0.0020 - accuracy:
1.0000 - val_loss: 0.0166 - val_accuracy: 1.0000
Epoch 4/5
7/7 [==============================] - 0s 72ms/step - loss: 0.0018 - accuracy:
1.0000 - val_loss: 0.0154 - val_accuracy: 1.0000
Epoch 5/5
7/7 [==============================] - 0s 71ms/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 0.0141 - val_accuracy: 1.0000
```

[73]:
```
plt.plot(final_history.history['accuracy'],label='acc')
plt.plot(final_history.history['val_accuracy'],label='val_acc')
plt.plot(final_history.history['loss'],label='loss')
plt.plot(final_history.history['val_loss'],label='val_loss')
plt.legend();
```



1.1 Evaluation mit Validierungsdatensatz

Eine angemessene Evaluation der Güte des Modells sollte auf dem Validierungsdatensatz passieren. Der Beispiel-Datensatz ist allerdings sehr klein! Entsprechend sind hier keine besonders robusten Resultate zu erwarten.

[74]:
```
#images = np.asarray(list(val_ds2.map(lambda x, y: x)))
y = np.asarray(list(val_ds.unbatch().map(lambda x, y: y)))
y.shape
```

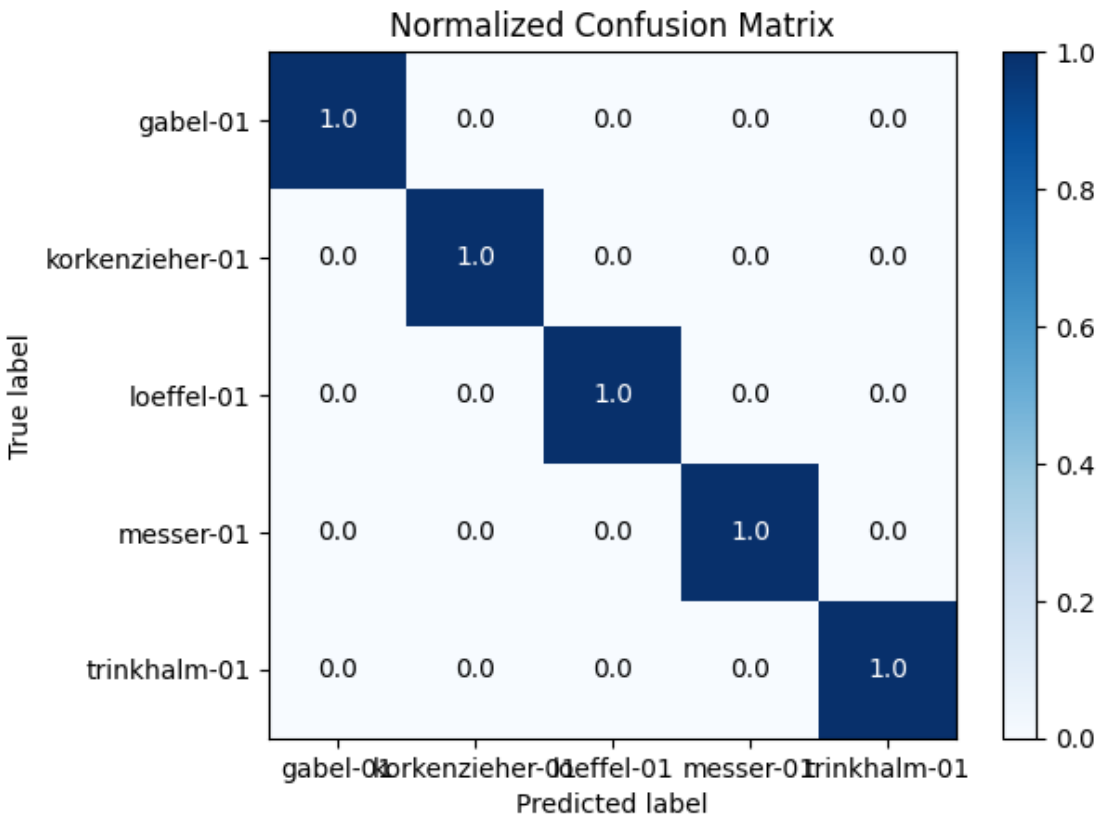
[74]:
```
(24,)
```

[75]:
```
yhat = final_model.predict(val_ds)
yhatnum = yhat.argmax(axis=1)
yhatnum
```

```
2/2 [==============================] - 0s 43ms/step
```

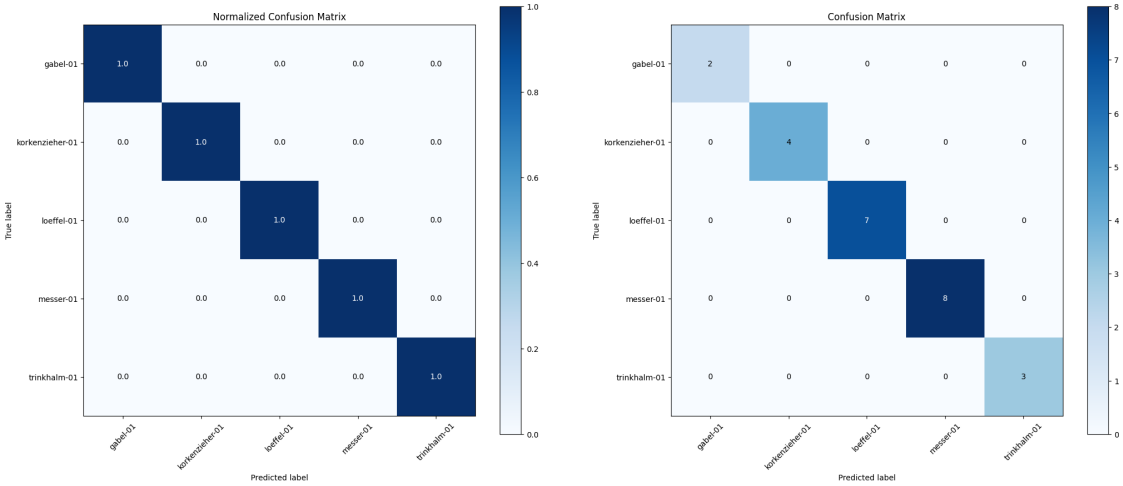
[75]:
```
array([3, 2, 0, 2, 3, 2, 3, 1, 2, 3, 2, 3, 3, 3, 4, 2, 4, 4, 0, 1, 1, 2,
       3, 1])
```

[76]:
```
skplt.metrics.plot_confusion_matrix(y_str, yhat_str, normalize=True)
plt.show()
```



[77]:
```
y_str = np.array(class_names)[y]
yhat_str = np.array(class_names)[yhatnum]
```

[78]:
```
fig,axlist = plt.subplots(1,2,figsize=(25,10));
skplt.metrics.plot_confusion_matrix(y_str, yhat_str,␣
 ↪normalize=True,ax=axlist[0]);axlist[0].tick_params(axis='x', rotation=45)
skplt.metrics.plot_confusion_matrix(y_str, yhat_str,␣
 ↪normalize=False,ax=axlist[1]);axlist[1].tick_params(axis='x', rotation=45)
```



2 Abspeichern des Modells als Keras-Modell

[79]:
```
Keras_Dateiname = f'model_{Datensatzname}.keras'
voller_Keras_Dateiname = str(Path(model_output_path)/Keras_Dateiname)
final_model.save(voller_Keras_Dateiname)
print(f'Keras-Modell nach {voller_Keras_Dateiname} abgespeichert.')
```

```
Keras-Modell nach working/model/model_Geschirr-v4.keras abgespeichert.
```

So kann man das Modell wieder laden:

```
model = tf.keras.models.load_model(voller_Keras_Dateiname)
```

(Es gibt aber verwirrend viele Formate- .h5, .keras, etc. Siehe auch hier. Wir möchten uns an das .h5-Format halten.)
Der folgende Befehl kann zur Vorhersage genutzt werden:

[80]:
```
loaded_model = tf.keras.models.load_model(voller_Keras_Dateiname)
```

[81]:
```
loaded_model.predict(val_ds)
```

```
2/2 [==============================] - 0s 41ms/step
```

[81]:
```
array([[1.10178627e-03, 8.73450244e-06, 3.65997781e-03, 9.95222032e-01,
        7.43523424e-06],
       [1.53820394e-04, 5.56924942e-06, 9.99768436e-01, 5.08489647e-05,
        2.13280746e-05],
       [9.99735653e-01, 1.56699559e-06, 1.66362908e-04, 6.34789176e-05,
        3.29083159e-05],
       [2.06393961e-05, 8.84938459e-07, 9.99971628e-01, 3.72258319e-06,
        3.13653891e-06],
       [2.97860883e-04, 3.77139659e-05, 1.29572616e-03, 9.98364866e-01,
```

```
 3.81200266e-06],
[2.19452013e-05, 8.84999849e-07, 9.99884605e-01, 8.95503836e-05,
 2.99394446e-06],
[5.01675706e-04, 3.19705714e-06, 9.00974148e-04, 9.98593748e-01,
 3.42633030e-07],
[3.88143280e-06, 9.99965191e-01, 3.01558248e-05, 4.50556655e-07,
 2.26185378e-07],
[4.95708309e-06, 5.74689025e-08, 9.99990463e-01, 4.05618812e-06,
 4.22651539e-07],
[1.01906089e-05, 4.39627038e-05, 2.97671704e-05, 9.99915481e-01,
 5.68965163e-07],
[1.42323217e-04, 2.04225489e-06, 9.99786913e-01, 6.75002229e-05,
 1.31800925e-06],
[7.15258284e-05, 2.31344757e-07, 2.35821164e-04, 9.99692202e-01,
 1.86584117e-07],
[7.96032953e-04, 2.30915903e-05, 2.22758809e-03, 9.96929586e-01,
 2.37590230e-05],
[1.40718708e-03, 2.40912068e-05, 2.59217760e-03, 9.95789349e-01,
 1.87210273e-04],
[1.08813614e-01, 6.08636510e-05, 5.83222732e-02, 4.06120787e-04,
 8.32397163e-01],
[1.84801866e-05, 2.17363663e-07, 9.99978900e-01, 1.51182473e-06,
 7.99505472e-07],
[7.99068511e-02, 3.12583579e-05, 4.48630676e-02, 2.87089526e-04,
 8.74911726e-01],
[3.25804809e-04, 1.94628205e-06, 1.91654908e-05, 4.15965405e-06,
 9.99648929e-01],
[9.98016238e-01, 1.13030683e-05, 1.84786879e-03, 9.68942768e-05,
 2.77792406e-05],
[3.87277287e-05, 9.99927402e-01, 1.40009015e-05, 1.90061728e-05,
 8.88265276e-07],
[1.39953281e-05, 9.99984145e-01, 1.82053566e-06, 2.87714919e-08,
 2.04047303e-08],
[1.67627138e-06, 3.81645532e-06, 9.99994397e-01, 8.12707768e-08,
 4.24068531e-08],
[3.10511095e-04, 6.92417234e-05, 1.41859998e-03, 9.98197854e-01,
 3.76267394e-06],
[3.15996112e-05, 9.99954224e-01, 1.39296908e-05, 2.04354038e-07,
 4.07400549e-08]], dtype=float32)
```

Das wär's! Nach dem Speichern ("Save Version" in Kaggle) kann das Modell heruntergeladen werden. Vergessen Sie nicht, welche Bilder Trainingsbilder, und welche Testbilder sind. Ein Modell, das overfittet, wird auf den Trainingsdaten eine viel bessere Performanz zeigen, als auf den Testbildern. Nur jene der Testbilder darf rapportiert werden, denn nur so gut ist die Verallgemeinerungsleistung des neuronalen Netzes. Wir speichern daher noch eine Excel-Datei mit den entsprechenden Zuordnungen ab:

[82]:
```
ser = pd.Series(train_or_test)
ser
```

[82]:
```
messer-01_9554.jpg          train
messer-01_9568.jpg          train
messer-01_9583.jpg          train
gabel-01_9638.jpg           train
korkenzieher-01_9732.jpg    train
                            …
korkenzieher-01_9702.jpg     test
korkenzieher-01_9703.jpg     test
korkenzieher-01_9717.jpg     test
messer-01_9565.jpg           test
messer-01_9559.jpg           test
Length: 179, dtype: object
```

[84]:
```
#ser.to_excel('train_test_split.xlsx')
#ser.value_counts()
```

Weiter geht's mit dem Notebook Giraffenpuzzle-Evaluation. Dort wollen wir die Performanz eines Modells evaluieren.

[ ]:
```

```