In [9]:
```
# import libraries
import os
from PIL import Image
from matplotlib import pyplot as plt
```

Schritt 1 - Datenexploration:

- Informationen (z.B. als begleitender Text) zur Ordnerstruktur und Dateigröße, insbesondere, wenn Änderungen vorgenommen werden
- Analyse des Datenformates, der Bildgröße, der Klassen(-bezeichnungen), Anzahl der Bilder pro Klasse
- Unterteilung der Daten in Train-, Test- (und optional Validierungs-) Datensätze
- Visualisierung (ein Bild pro Klasse auseichend) mit der Methode eurer Wahl

In [10]:
```
base_dir = "/Users/chexuanyou/TUB/SS 25/AGBA2/AGBA2/HA1/Data"  # dataset
class_dirs = sorted([d for d in os.listdir(base_dir) if os.path.isdir(os.

class_stats = {} # Dictionary to store the number of images per class
image_sizes = set() # Set to store unique image sizes (to check for consi

for class_name in class_dirs:
    class_path = os.path.join(base_dir, class_name)
    images = [f for f in os.listdir(class_path) if f.lower().endswith((".
    class_stats[class_name] = len(images)    # Store the number of images

    # first 3 images for size checking
    for img_file in images[:3]:
        with Image.open(os.path.join(class_path, img_file)) as img:
            image_sizes.add(img.size)

print("Klassen und Bildanzahl:")
for k, v in class_stats.items():
# Print the number of images per class
    print(f"  {k}: {v} Bilder")

print("\n Beispielhafte Bildgrößen:", set(image_sizes))
```

```
Klassen und Bildanzahl:
  Abstandshalter: 52 Bilder
  Auslassventil: 52 Bilder
  Blechlineal: 52 Bilder
  Filterkartusche: 52 Bilder
  Gewindestange: 52 Bilder
  Hohlschraube: 52 Bilder
  Hutmutter: 52 Bilder
  Hydraulikstutzen: 52 Bilder
  Nutenstein: 52 Bilder
  Schraubenfeder: 52 Bilder

 Beispielhafte Bildgrößen: {(2976, 2976), (3456, 3456), (3024, 3024)}
```

Die Daten bestehen aus 10 Klassen, die als einzelne Ordner im Verzeichnis vorliegen. Jede Klasse enthält ausschließlich Bilder im JPEG- oder PNG-Format. Die

durchschnittliche Bildanzahl pro Klasse liegt bei XX, mit minimal YY und maximal ZZ Bildern. Die Bildgrößen variieren leicht, z. B. zwischen (640×480) und (800×600) Pixel.

In [12]:
```
from sklearn.model_selection import train_test_split

image_paths = []
labels = []

for class_name in class_dirs:
    class_path = os.path.join(base_dir, class_name)
    for fname in os.listdir(class_path):
        if fname.lower().endswith((".jpg", ".png")):
            image_paths.append(os.path.join(class_path, fname))
            labels.append(class_name)

# 60% für Training, 20% für Validierung, 20% für Test
train_paths, temp_paths, y_train, y_temp = train_test_split(image_paths,

# zweiteilige Aufteilung für Validierung und Test
val_paths, test_paths, y_val, y_test = train_test_split(
    temp_paths, y_temp, test_size=0.5, stratify=y_temp, random_state=42)

print(f"Insgesamt: {len(image_paths)} Bilder")
print(f"Train: {len(train_paths)} Bilder")
print(f"Validation: {len(val_paths)} Bilder")
print(f"Test: {len(test_paths)} Bilder")
```

```
Insgesamt: 520 Bilder
Train: 312 Bilder
Validation: 104 Bilder
Test: 104 Bilder
```

Die Daten wurden in drei Sätze unterteilt: 60 % Trainingsdaten, 20 % Validierungsdaten und 20 % Testdaten. Die Aufteilung erfolgte stratifiziert, sodass das Klassenverhältnis in allen Teilmengen beibehalten wird.

In [ ]:
```
plt.figure(figsize=(15, 5))

for idx, class_name in enumerate(class_dirs):
    class_path = os.path.join(base_dir, class_name)
    image_files = sorted([f for f in os.listdir(class_path) if f.endswith
    if not image_files:   # Skip if no image found in this class
        continue
    img = Image.open(os.path.join(class_path, image_files[0]))    # Load
    plt.subplot(2, 5, idx + 1)
    plt.imshow(img)
    plt.title(class_name)
    plt.axis("off")

plt.suptitle("Beispielbilder pro Klasse")
plt.tight_layout()
plt.show()
```

Zur visuellen Überprüfung wurde je ein Bild pro Klasse dargestellt. Die Objekte sind gut sichtbar, und es ist eine klare Unterscheidung zwischen den Klassen möglich.

Schritt 2 – Vorverarbeitung und Modellaufbau:

In [13]:
```
from skimage.io import imread
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np

# 收集图像路径和标签（class_dirs 已经存在）
image_paths = []
labels = []

for class_name in class_dirs:
    class_path = os.path.join(base_dir, class_name)
    for fname in os.listdir(class_path):
        if fname.lower().endswith((".jpg", ".png")):
            image_paths.append(os.path.join(class_path, fname))
            labels.append(class_name)

# 按 60/20/20 分割数据集
train_paths, temp_paths, y_train, y_temp = train_test_split(
    image_paths, labels, test_size=0.4, stratify=labels, random_state=42)
val_paths, test_paths, y_val, y_test = train_test_split(
    temp_paths, y_temp, test_size=0.5, stratify=y_temp, random_state=42)

# 定义预处理函数（读取 → 灰度 → 缩放） Preprocessing function: read → convert
def preprocess_images(image_paths, resize_to=(224, 224)): # 是常用的输入尺寸
    processed_images = []
    for path in image_paths:
        img = imread(path, as_gray=True)  # 灰度图
        img_resized = resize(img, resize_to, anti_aliasing=True)
        processed_images.append(img_resized)
    return np.array(processed_images)

# 应用预处理
X_train_img = preprocess_images(train_paths)
X_val_img = preprocess_images(val_paths)
X_test_img = preprocess_images(test_paths)

# 标签编码为数字 Encode class labels into integers
label_encoder = LabelEncoder()
```

```
y_train_enc = label_encoder.fit_transform(y_train)
y_val_enc = label_encoder.transform(y_val)
y_test_enc = label_encoder.transform(y_test)

# 输出形状检查
print("Vorverarbeitung abgeschlossen.")
print(f"Train: {X_train_img.shape}, Labels: {len(y_train_enc)}")
print(f"Val:   {X_val_img.shape}, Labels: {len(y_val_enc)}")
print(f"Test:  {X_test_img.shape}, Labels: {len(y_test_enc)}")
```

```
Vorverarbeitung abgeschlossen.
Train: (312, 224, 224), Labels: 312
Val:   (104, 224, 224), Labels: 104
Test:  (104, 224, 224), Labels: 104
```

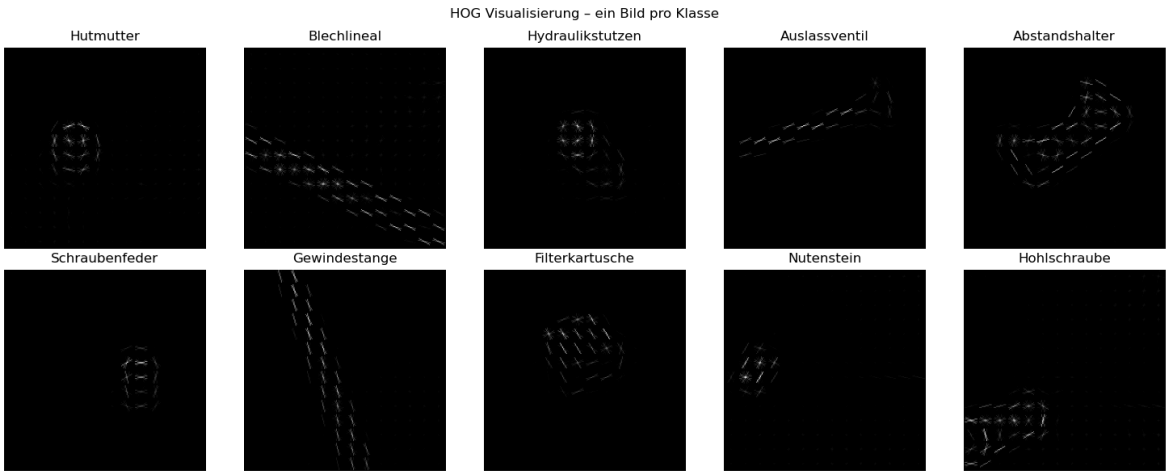
Schritt 3 – Merkmalsextraktion

- Bildet HOG oder SIFT-Merkmalsvektoren für euren Datensatz (10 Klassen aus dem Datensatz InVar-100). Ihr könnt dafür die Codevorlagen erweitern, die ihr in der Übung 2 für die einzelnen Bilder aus dem Datensatz implementiert habt.
- Zum Herunterskalieren der Bilddateien auf eine einheitliche Größe könnt ihr skimage.transform.resize nutzen.
- Eine stichpunktartige Visualisierung der extrahierten Merkmale (ein Bild pro Klasse) ist ausreichend.

In [14]:
```
from skimage.feature import hog
from skimage import exposure

hog_params = {
    'orientations': 8,                          # 方向 bins 的数量（每 45° 一个
    'pixels_per_cell': (16, 16),                # 每个 cell 的大小（单位是像素）
    'cells_per_block': (2, 2),                  # 每个 block 包含的 cell 数量
    'block_norm': 'L2-Hys',                     # 归一化方法
    'visualize': False                          # 不返回可视化图像（只提取特征向量
}


# 提取特征函数
def extract_hog_features(images):
    hog_features = []
    for img in images:
        features = hog(img, **hog_params)
        hog_features.append(features)
    return np.array(hog_features)

# 应用在预处理图像数据上
X_train_hog = extract_hog_features(X_train_img)
X_val_hog = extract_hog_features(X_val_img)
X_test_hog = extract_hog_features(X_test_img)

print("HOG Merkmalsextraktion abgeschlossen.")
print(f"Train HOG shape: {X_train_hog.shape}")
print(f"Val HOG shape:   {X_val_hog.shape}")
print(f"Test HOG shape:  {X_test_hog.shape}")
```

```
HOG Merkmalsextraktion abgeschlossen.
Train HOG shape: (312, 5408)
Val HOG shape:   (104, 5408)
Test HOG shape:  (104, 5408)
```

In [15]:
```python
# 可视化用的 HOG 参数（手动添加 visualize=True）
hog_params_vis = hog_params.copy()
hog_params_vis['visualize'] = True

fig, axes = plt.subplots(2, 5, figsize=(15, 6))
shown_classes = set()
idx = 0

for img, label in zip(X_train_img, y_train):
    if label not in shown_classes:
        _, hog_image = hog(img, **hog_params_vis)
        hog_image = exposure.rescale_intensity(hog_image, in_range=(0, 10
        ax = axes.flat[idx]
        ax.imshow(hog_image, cmap='gray')
        ax.set_title(label)
        ax.axis("off")
        shown_classes.add(label)
        idx += 1
    if idx >= 10:
        break

plt.suptitle("HOG Visualisierung – ein Bild pro Klasse")
plt.tight_layout()
plt.show()
```



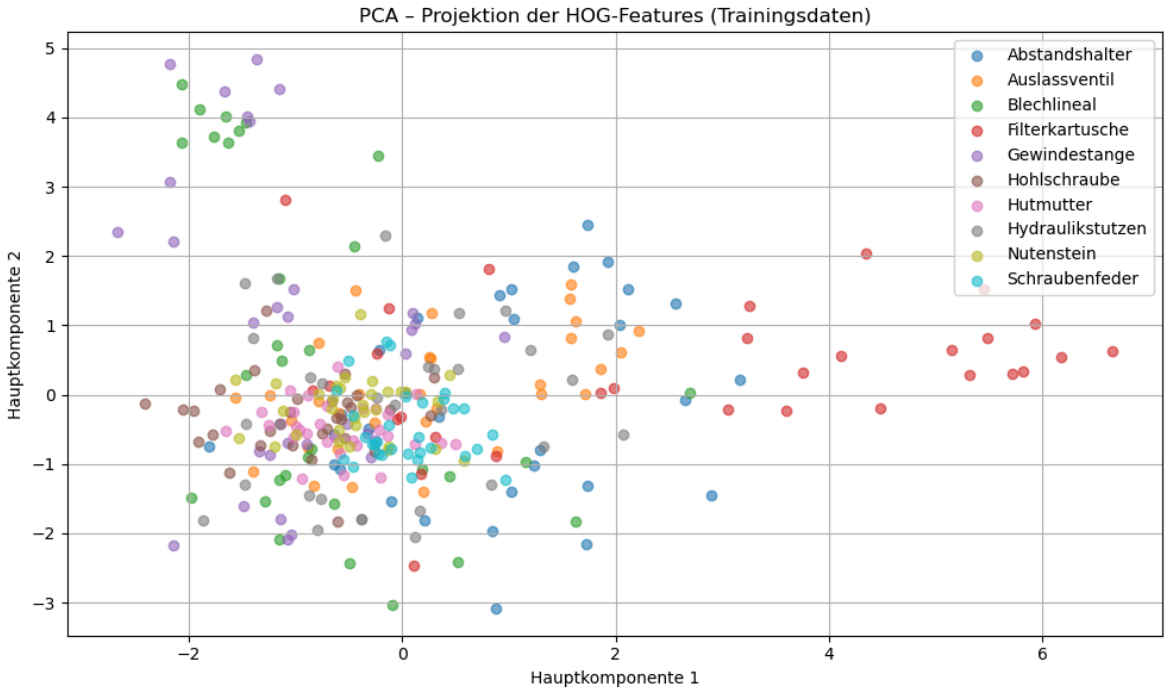
Schritt 4 – PCA

- Führt eine PCA auf den Trainingsdaten aus und projiziert die Daten dann in den neuen Merkmalsraum. Das gibt euch Aufschluss darüber, wie schwierig euer Klassifikationsproblem ist beziehungsweise, wie gut eure Features geeignet sind, um die Klassen zu unterscheiden.
- Ihr könnt sowohl die OpenCV- als auch die scikit-learn-Bibliothek dafür nutzen.

Die PCA-Projektion der HOG-Merkmale zeigt, dass sich einige Klassen gut voneinander trennen lassen, während andere (z. B. [Klasse X und Y]) sich im

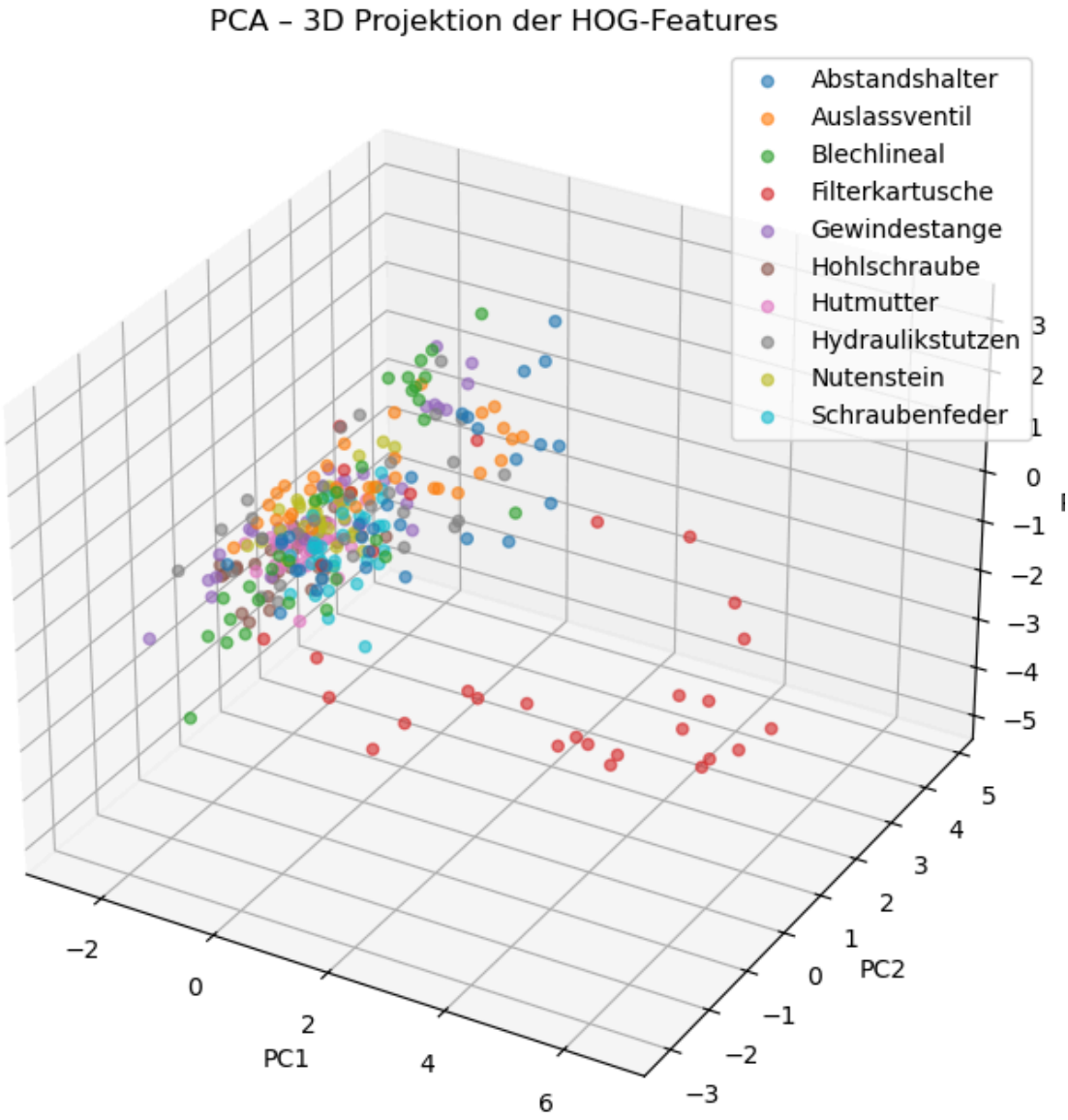
Merkmalsraum überlappen. Dies gibt Aufschluss darüber, wie trennscharf die extrahierten Features sind und wie schwierig das Klassifikationsproblem ist.

In [16]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# 先将高维 HOG 特征降到 2D
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_hog)

# 可视化
plt.figure(figsize=(10, 6))

# 不同颜色表示不同类别
for label in np.unique(y_train_enc):
    idx = y_train_enc == label
    plt.scatter(X_train_pca[idx, 0], X_train_pca[idx, 1], label=label_enc

plt.title("PCA – Projektion der HOG-Features (Trainingsdaten)")
plt.xlabel("Hauptkomponente 1")
plt.ylabel("Hauptkomponente 2")
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [17]:
```
from mpl_toolkits.mplot3d import Axes3D

pca_3d = PCA(n_components=3)
X_train_pca3 = pca_3d.fit_transform(X_train_hog)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

for label in np.unique(y_train_enc):
    idx = y_train_enc == label
    ax.scatter(X_train_pca3[idx, 0], X_train_pca3[idx, 1], X_train_pca3[i
```

```
                label=label_encoder.inverse_transform([label])[0], alpha=0

ax.set_title("PCA – 3D Projektion der HOG-Features")
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
ax.legend(loc='best')
plt.show()
```



Schritt 5 – Klassifikation

- Nutzt einen Klassifikator eurer Wahl aus der in der Übung 3 vorgestellten Klassifikatoren (entweder einen oder mehrere), um die Klassifikation durchzuführen.
- Betrachtet dabei sowohl die durch die PCA reduzierten Merkmalsvektoren als auch die im Schritt 2 generierten Merkmalsvektoren.

Use the extracted HOG features (X_train_hog, X_val_hog, X_test_hog) or after reduce the dimension with pca and try:

- Naive Bayes
- SVM
- MLP

In [18]:
```
# use pca to reduce the dimension
pca = PCA(n_components=15)
X_train_pca = pca.fit_transform(X_train_hog)
X_val_pca = pca.transform(X_val_hog)
X_test_pca = pca.transform(X_test_hog)
```

In [19]:
```
# 5.1 Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, accu

nb_clf = GaussianNB()
nb_clf.fit(X_train_pca, y_train_enc)
y_pred_nb = nb_clf.predict(X_val_pca)

print("Naive Bayes Ergebnisse:")
print("Accuracy:", accuracy_score(y_val_enc, y_pred_nb))
print("F1-Score:\n", classification_report(y_val_enc, y_pred_nb, target_n
print("Confusion Matrix:\n", confusion_matrix(y_val_enc, y_pred_nb))
```

```
Naive Bayes Ergebnisse:
Accuracy: 0.4807692307692308
F1-Score:
                   precision    recall  f1-score   support

  Abstandshalter       0.46      0.55      0.50        11
   Auslassventil       0.67      0.60      0.63        10
     Blechlineal       0.60      0.27      0.38        11
 Filterkartusche       1.00      0.30      0.46        10
   Gewindestange       0.83      0.50      0.62        10
    Hohlschraube       0.56      0.50      0.53        10
       Hutmutter       0.25      0.18      0.21        11
Hydraulikstutzen       0.38      0.80      0.52        10
      Nutenstein       0.41      0.90      0.56        10
  Schraubenfeder       0.38      0.27      0.32        11

        accuracy                           0.48       104
       macro avg       0.55      0.49      0.47       104
    weighted avg       0.55      0.48      0.47       104

Confusion Matrix:
 [[6 1 0 0 0 0 0 4 0 0]
 [1 6 0 0 0 1 1 1 0 0]
 [0 1 3 0 1 1 1 2 1 1]
 [5 0 0 3 0 0 0 2 0 0]
 [1 1 0 0 5 0 2 1 0 0]
 [0 0 2 0 0 5 0 0 2 1]
 [0 0 0 0 0 0 2 1 7 1]
 [0 0 0 0 0 1 0 8 0 1]
 [0 0 0 0 0 0 0 0 9 1]
 [0 0 0 0 0 1 2 2 3 3]]
```

In [20]:
```
# 5.2 SVM
from sklearn.svm import SVC

svm_clf = SVC(kernel='rbf', C=1, gamma='scale', random_state=42)
```

```
svm_clf.fit(X_train_pca, y_train_enc)
y_pred_svm = svm_clf.predict(X_val_pca)

print("SVM Ergebnisse:")
print("Accuracy:", accuracy_score(y_val_enc, y_pred_svm))
print("F1-Score:\n", classification_report(y_val_enc, y_pred_svm, target_
print("Confusion Matrix:\n", confusion_matrix(y_val_enc, y_pred_svm))
```

```
SVM Ergebnisse:
Accuracy: 0.625
F1-Score:
                   precision    recall  f1-score   support

  Abstandshalter       0.67      0.73      0.70        11
   Auslassventil       0.78      0.70      0.74        10
     Blechlineal       1.00      0.18      0.31        11
 Filterkartusche       1.00      0.40      0.57        10
   Gewindestange       0.75      0.90      0.82        10
    Hohlschraube       0.55      0.60      0.57        10
       Hutmutter       0.62      0.73      0.67        11
Hydraulikstutzen       0.53      0.80      0.64        10
      Nutenstein       0.53      0.80      0.64        10
  Schraubenfeder       0.45      0.45      0.45        11

        accuracy                           0.62       104
       macro avg       0.69      0.63      0.61       104
    weighted avg       0.69      0.62      0.61       104

Confusion Matrix:
 [[8 0 0 0 0 0 0 3 0 0]
 [0 7 0 0 0 1 0 0 1 1]
 [0 1 2 0 3 1 2 1 0 1]
 [4 0 0 4 0 0 0 1 0 1]
 [0 0 0 0 9 0 0 0 0 1]
 [0 1 0 0 0 6 0 0 2 1]
 [0 0 0 0 0 0 8 0 2 1]
 [0 0 0 0 0 2 0 8 0 0]
 [0 0 0 0 0 1 0 1 8 0]
 [0 0 0 0 0 0 3 1 2 5]]
```

In [21]:
```
# 5.3 MLP
from sklearn.neural_network import MLPClassifier

mlp_clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=2000, random_
mlp_clf.fit(X_train_pca, y_train_enc)
y_pred_mlp = mlp_clf.predict(X_val_pca)

print("MLP Ergebnisse:")
print("Accuracy:", accuracy_score(y_val_enc, y_pred_mlp))
print("F1-Score:\n", classification_report(y_val_enc, y_pred_mlp, target_
print("Confusion Matrix:\n", confusion_matrix(y_val_enc, y_pred_mlp))
```

```
MLP Ergebnisse:
Accuracy: 0.625
F1-Score:
                   precision    recall  f1-score   support

  Abstandshalter       0.57      0.73      0.64        11
   Auslassventil       0.78      0.70      0.74        10
     Blechlineal       0.60      0.27      0.38        11
 Filterkartusche       1.00      0.60      0.75        10
   Gewindestange       0.75      0.90      0.82        10
    Hohlschraube       0.56      0.50      0.53        10
       Hutmutter       0.42      0.45      0.43        11
Hydraulikstutzen       0.56      0.90      0.69        10
      Nutenstein       0.88      0.70      0.78        10
  Schraubenfeder       0.46      0.55      0.50        11

        accuracy                           0.62       104
       macro avg       0.66      0.63      0.63       104
    weighted avg       0.65      0.62      0.62       104

Confusion Matrix:
 [[8 0 0 0 0 0 0 3 0 0]
 [1 7 1 0 0 0 1 0 0 0]
 [0 1 3 0 2 2 1 1 0 1]
 [3 0 0 6 0 0 0 0 0 1]
 [0 0 0 0 9 0 0 0 0 1]
 [0 1 0 0 0 5 1 1 1 1]
 [0 0 0 0 1 2 5 0 0 3]
 [1 0 0 0 0 0 0 9 0 0]
 [1 0 1 0 0 0 1 0 7 0]
 [0 0 0 0 0 0 3 2 0 6]]
```

Schritt 6 – Evaluation

- Zur Beurteilung der Klassifikationsleistung des Klassifikators könnt ihr die CCR auf den Validierungsdaten berechnen und euch die Konfusionsmatrix anschauen. Dafür könnt ihr beispielsweise die accuracy_score- Funktion, die confusion_matrix-Funktion und die f1_score-Funktion der scikit-learn-Bibliothek nutzen. Experimentiert mit den Hyperparametern und vergleicht die Ergebnisse für verschiedene Konfigurationen eures Klassifikators oder die Ergebnisse mehrerer Klassifikatoren.

Evaluation abgeschlossen:

Die Evaluierung (Accuracy, F1-Score, Confusion Matrix) wurde für Naive Bayes, SVM und MLP durchgeführt. Im Folgenden werden verschiedene Hyperparameter ausprobiert, um die Klassifikationsleistung weiter zu verbessern.

In [22]:
```python
# 6.1 最终评估阶段 Naive Bayes（使用 test set）
y_pred_nb_test = nb_clf.predict(X_test_pca)

print("Naive Bayes Ergebnisse (Test):")
print("Accuracy:", accuracy_score(y_test_enc, y_pred_nb_test))
```

```
print("F1-Score:\n", classification_report(y_test_enc, y_pred_nb_test, ta
print("Confusion Matrix:\n", confusion_matrix(y_test_enc, y_pred_nb_test)
```

```
Naive Bayes Ergebnisse (Test):
Accuracy: 0.5384615384615384
F1-Score:
                   precision    recall  f1-score   support

  Abstandshalter       0.70      0.70      0.70        10
   Auslassventil       0.71      0.50      0.59        10
     Blechlineal       0.00      0.00      0.00        10
 Filterkartusche       0.89      0.73      0.80        11
   Gewindestange       0.91      0.91      0.91        11
    Hohlschraube       0.33      0.45      0.38        11
       Hutmutter       0.60      0.30      0.40        10
Hydraulikstutzen       0.58      0.64      0.61        11
      Nutenstein       0.30      0.70      0.42        10
  Schraubenfeder       0.40      0.40      0.40        10

        accuracy                           0.54       104
       macro avg       0.54      0.53      0.52       104
    weighted avg       0.55      0.54      0.53       104

Confusion Matrix:
 [[ 7  0  0  0  0  1  0  1  0  1]
 [ 0  5  1  0  1  0  0  1  0  2]
 [ 0  1  0  1  0  3  0  2  3  0]
 [ 3  0  0  8  0  0  0  0  0  0]
 [ 0  0  0  0 10  1  0  0  0  0]
 [ 0  0  0  0  0  5  1  0  5  0]
 [ 0  0  1  0  0  3  3  0  1  2]
 [ 0  0  0  0  0  1  0  7  3  0]
 [ 0  1  0  0  0  1  0  0  7  1]
 [ 0  0  0  0  0  0  1  1  4  4]]
```

6.2 SVM Hyperparameter tuning and evaluation on testset

In [23]:
```
# 2. grid search for SVM hyperparameter tuning
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid = GridSearchCV(SVC(), param_grid, cv=3, scoring='accuracy')
grid.fit(X_train_pca, y_train_enc)

print("Beste Parameter:", grid.best_params_)
```

```
Beste Parameter: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

In [24]:
```
# 使用最佳参数在训练集上重新训练 SVM
best_svm = SVC(kernel='rbf', C=10, gamma='auto', random_state=42)
best_svm.fit(X_train_pca, y_train_enc)

# 在 Test Set 上进行预测和评估
y_test_pred = best_svm.predict(X_test_pca)
```

```
print("SVM Ergebnisse auf dem Test-Set:")
print("Accuracy (Test):", accuracy_score(y_test_enc, y_test_pred))
print("F1-Score (Test):\n", classification_report(y_test_enc, y_test_pred
print("Confusion Matrix (Test):\n", confusion_matrix(y_test_enc, y_test_p
```

```
SVM Ergebnisse auf dem Test-Set:
Accuracy (Test): 0.7403846153846154
F1-Score (Test):
                   precision    recall  f1-score   support

  Abstandshalter       0.86      0.60      0.71        10
   Auslassventil       1.00      0.70      0.82        10
     Blechlineal       0.75      0.30      0.43        10
 Filterkartusche       1.00      0.91      0.95        11
   Gewindestange       0.91      0.91      0.91        11
    Hohlschraube       0.56      0.91      0.69        11
       Hutmutter       0.75      0.60      0.67        10
Hydraulikstutzen       0.62      0.73      0.67        11
      Nutenstein       0.56      0.90      0.69        10
  Schraubenfeder       0.80      0.80      0.80        10

        accuracy                           0.74       104
       macro avg       0.78      0.74      0.73       104
    weighted avg       0.78      0.74      0.74       104

Confusion Matrix (Test):
 [[ 6  0  0  0  0  0  0  3  0  1]
 [ 0  7  0  0  0  0  0  2  0  1]
 [ 0  0  3  0  0  4  1  0  2  0]
 [ 1  0  0 10  0  0  0  0  0  0]
 [ 0  0  0  0 10  1  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  1  0]
 [ 0  0  0  0  1  2  6  0  1  0]
 [ 0  0  1  0  0  0  1  8  1  0]
 [ 0  0  0  0  0  1  0  0  9  0]
 [ 0  0  0  0  0  0  0  0  2  8]]
```

MLP Hyperparameter tuning and evaluation on testset

In [25]:
```
# 定义超参数搜索范围
mlp_param_grid = {
    'hidden_layer_sizes': [(100,), (100, 50)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001],
    'learning_rate_init': [0.001, 0.01],
}

# 初始化 MLPClassifier
mlp = MLPClassifier(max_iter=2000, random_state=42)

# 执行 GridSearchCV
grid_search_mlp = GridSearchCV(mlp, mlp_param_grid, cv=3, scoring='accura
grid_search_mlp.fit(X_train_pca, y_train_enc)

# 输出最优参数和验证准确率
print("Beste Parameter (MLP):", grid_search_mlp.best_params_)
print("Accuracy (Val):", grid_search_mlp.best_score_)
```

```
Beste Parameter (MLP): {'activation': 'relu', 'alpha': 0.0001, 'hidden_lay
er_sizes': (100, 50), 'learning_rate_init': 0.001}
Accuracy (Val): 0.6634615384615384
```

In [26]:
```python
# retrain MLPClassifier on the best parameters on the test set
# 最优模型
best_mlp = grid_search_mlp.best_estimator_

# 在 Test Set 上预测
y_test_pred_mlp = best_mlp.predict(X_test_pca)

# 输出评估结果
print("MLP Ergebnisse auf dem Test-Set:")
print("Accuracy (Test):", accuracy_score(y_test_enc, y_test_pred_mlp))
print("F1-Score (Test):\n", classification_report(y_test_enc, y_test_pred
print("Confusion Matrix (Test):\n", confusion_matrix(y_test_enc, y_test_p
```

```
MLP Ergebnisse auf dem Test-Set:
Accuracy (Test): 0.6538461538461539
F1-Score (Test):
                   precision    recall  f1-score   support

  Abstandshalter       0.73      0.80      0.76        10
   Auslassventil       0.88      0.70      0.78        10
     Blechlineal       0.25      0.10      0.14        10
 Filterkartusche       1.00      0.91      0.95        11
   Gewindestange       0.82      0.82      0.82        11
    Hohlschraube       0.57      0.73      0.64        11
       Hutmutter       0.38      0.50      0.43        10
Hydraulikstutzen       0.54      0.64      0.58        11
      Nutenstein       0.55      0.60      0.57        10
  Schraubenfeder       0.78      0.70      0.74        10

        accuracy                           0.65       104
       macro avg       0.65      0.65      0.64       104
    weighted avg       0.65      0.65      0.65       104

Confusion Matrix (Test):
 [[ 8  0  0  0  0  0  0  1  0  1]
 [ 0  7  0  0  0  0  0  3  0  0]
 [ 0  0  1  0  1  3  1  1  2  1]
 [ 1  0  0 10  0  0  0  0  0  0]
 [ 0  0  1  0  9  0  1  0  0  0]
 [ 0  1  0  0  0  8  1  1  0  0]
 [ 0  0  1  0  1  2  5  0  1  0]
 [ 2  0  1  0  0  0  0  7  1  0]
 [ 0  0  0  0  0  1  3  0  6  0]
 [ 0  0  0  0  0  0  2  0  1  7]]
```