PointNet-Segmentierungsnetzwerk

June 18, 2020

Inhalt

1 PointNet-Segmentierungsnetzwerk

Aufbauend auf dem vorherigen Notebook zum PointNet-Klassifizierungsnetzwerk widmet sich dieses Dokument einem praktischen Beispiel für die Segmentierung von 3D-Punktwolken mithilfe von PointNet. Der verwendete Code entstammt einer Implementation von G. Li, zu finden im folgenden Git-Repository: https://github.com/garyli1019/pointnet-keras.

Segmentierung ist eine besondere Form der Klassifizierung, bei der die Elemente eines Objektes spezifischen Teilstücken oder -Bereichen des Gesamtobjekts zugeordnet werden. Ein klassisches Beispiel für Segmente in einem Objekt sind die Beine, Sitzfläche und die Lehne eines Stuhls. Das segmentierte Objekt könnte etwa so aussehen wie in der folgenden Grafik:

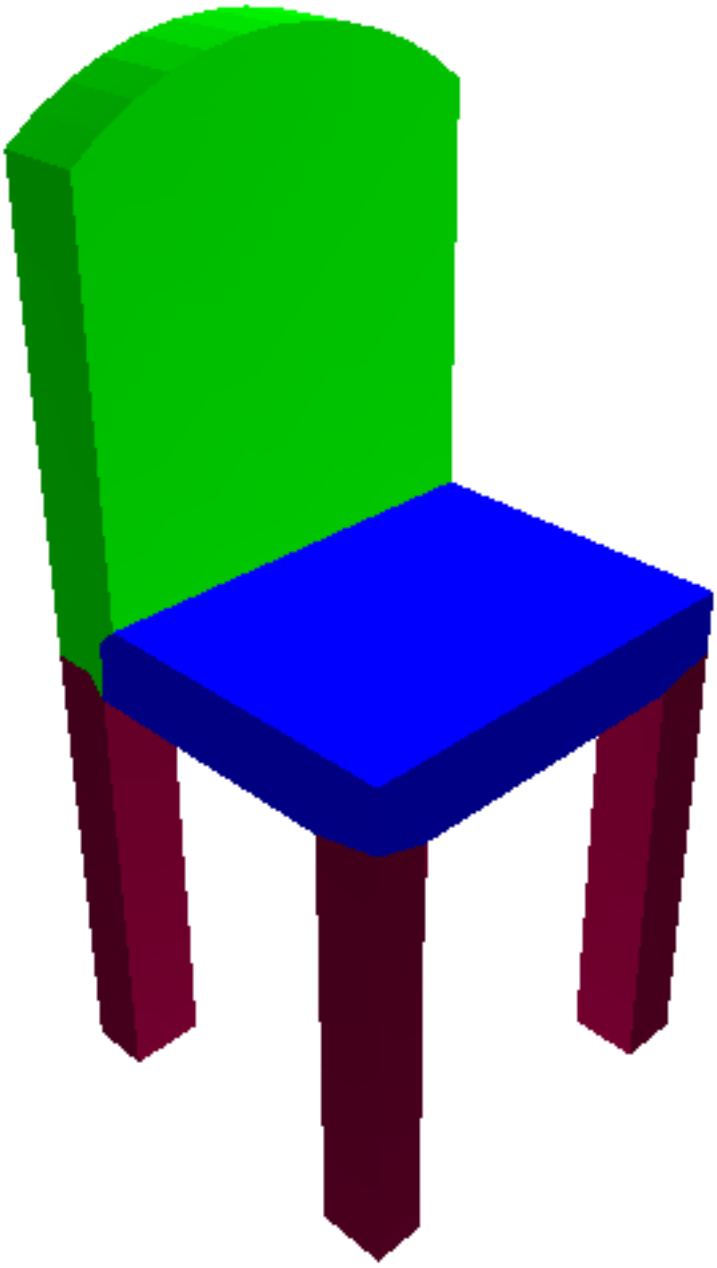


Abbildung 1: Segmente eines Stuhls [1]

Die folgende Grafik von Qi et al. [1] zeigt die Architektur von PointNet. Der markierte Bereich stellt hierbei das eigentliche Segmentierungsnetzwerk dar. Wie unschwer zu erkennen ist, wird die Architektur vom Input bis zur Aggregation der globalen Feature-Signatur gleichermaßen vom Klassifikations- und vom Segmentierungsnetzwerk genutzt.

Diese Darstellung wird wie auch im vorherigen Notebook im weiteren Verlauf wiederholt genutzt, um das Durchlaufen der Pipeline zu visualisieren.

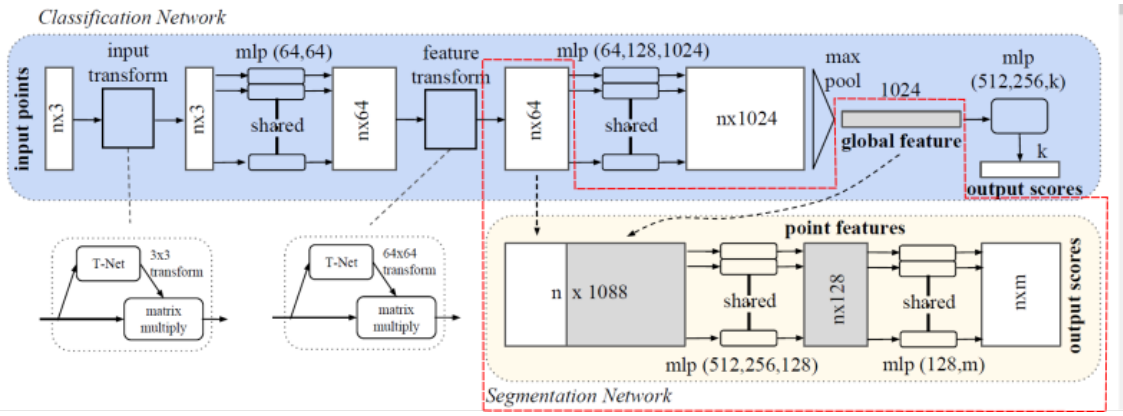


Abbildung 2: PointNet-Segmentierungsnetzwerk [2]

1.1 Anwendungshinweis

Da es sich bei dem Code in diesem Notebook um ein zusammenhängendes Script handelt, empfiehlt sich bei Änderungen stets das Zurücksetzen des Kernels. Hierfür wird unter dem Reiter "Kernel" die Option "Restart & Run all" ausgewählt.

1.2 Imports

Für die hier analyiserte Keras-Implementation der PointNet-Architektur werden drei Pakete benötigt: * NumPy für die Verwaltung von Datenstrukturen sowie der Handhabung von Vektoren und Matrizen * H5Py als Schnittstelle für das HDF5-Datenformat * Tensorflow für die Verwendung der Keras-API, welche Teil des Tensorflow-Cores ist

Darüber hinaus werden die nötigen Keras-Pakete für die Schichttypen und die Erzeugung des Modells importiert.

Für die Darstellung einer segmentierten Punktwolke werden pyplot und Axes3D genutzt.

[1]:
```
import warnings
warnings.simplefilter(action='ignore')

import numpy as np
import os
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tensorflow as tf
from keras import optimizers
from keras.layers import Input
from keras.models import Model
from keras.layers import Dense, Reshape
from keras.layers import Convolution1D, MaxPooling1D, BatchNormalization
from keras.layers import Lambda, concatenate
#from keras.utils import np_utils
import h5py
```

```
Using TensorFlow backend.
```

1.3 Hilfsfunktionen

Nachfolgend werden Hilfsfunktionen definiert. Wie beim Klassifizierungsnetzwerk werden die Funktionen `mat_mul` und `load_5` für die Matrizenmultiplikation und das Laden von HPF5-Files benötigt.

Die Funktion `exp_dim` wird im Modell im Rahmen einer Lambda Layer genutzt, um den globalen Feature-Vektor für die nachfolgende Konkatenation zu erweitern.

[2]:
```
def mat_mul(A, B):
    return tf.matmul(A, B)
```

[3]:
```
def exp_dim(global_feature, num_points):
    return tf.tile(global_feature, [1, num_points, 1])
```

[4]:
```
def load_h5(h5_filename):
    f = h5py.File(h5_filename)
    data = f['data'][:]
    label = f['label'][:]
    return (data, label)
```

Die Funktionen rotate_point_cloud sowie jitter_point_cloud dienen dazu, die Trainingsdaten wiederholt nutzbar zu machen. Durch zufällige Rotationen und Mikro-Translationen der Objekte bzw. Punkte erzeugen die Funktionen aus einem Datensatz eine beliebige Menge unterschiedlicher Trainingsdaten.

rotate_point_cloud erzeugt zunächst eine Nullmatrix mit den Dimensionen des Inputs. Anschließend wird für jedes Objekt eine zufällige Rotationsmatrix erzeugt. Die Produkte der Rotationsmatrix und der Objektpunkte werden in die Nullmatrix eingefügt, welche nach Durchlaufen aller Objekte zurückgegeben wird. Die Rotationsachse ist die Y-Achse.

jitter_point_cloud erzeugt ebenfalls eine Kopie des Inputs. Diese wird jedoch nicht als Nullmatrix initialisiert, sondern mit kleinen Zufallszahlen gefüllt. Anschließend erfolgt eine elementweise Addition mit der Input-Matrix.

[5]:
```
def rotate_point_cloud(batch_data):
    """ Randomly rotate the point clouds to augument the dataset
        rotation is per shape based along up direction
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, rotated batch of point clouds
    """
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        rotation_angle = np.random.uniform() * 2 * np.pi
        cosval = np.cos(rotation_angle)
```

```
        sinval = np.sin(rotation_angle)
        rotation_matrix = np.array([[cosval, 0, sinval],
                                    [0, 1, 0],
                                    [-sinval, 0, cosval]])
        shape_pc = batch_data[k, ...]
        rotated_data[k, ...] = np.matmul(shape_pc.reshape((-1, 3)),␣
 ↪rotation_matrix)
    return rotated_data
```

[6]:
```
def jitter_point_cloud(batch_data, sigma=0.01, clip=0.05):
    """ Randomly jitter points. jittering is per point.
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, jittered batch of point clouds
    """
    B, N, C = batch_data.shape
    assert(clip > 0)
    jittered_data = np.clip(sigma * np.random.randn(B, N, C), -1 * clip, clip)
    jittered_data += batch_data
    return jittered_data
```

Um das Netzwerk erfolgreich trainieren zu können, muss der Input auf einen Objekttypen wie bspw. Stühle festgelegt werden. Hierfür dient die Variable `s`. Alle verfügbaren Typen sind im Dictionary `sList` in der darauffolgenden Codelzelle zu finden.

[7]:
```
# chosen shape
s = 'Guitar'
```

Die folgenden globalen Variablen bestimmen wichtige Randbedingungen der verwendeten Datensätze sowie den verwendenten Optimierungsalgorithmus und sollten daher nicht verändert werden.

[8]:
```
# number of points in each sample
num_points = 1024

# shapes and categories
sList = {'Airplane': 4, 'Chair': 4, 'Guitar': 3, 'Knife': 2, 'Lamp': 3, 'Table':␣
 ↪2}

# number of categories
k = sList[s]
# define optimizer
adam = optimizers.Adam(lr=0.001, decay=0.7)
```

1.4 Keras-Modell

Wie auch die PointNet-Variante zur Klassifizierung wird das Segmentierungsnetzwerk als Keras-Modell implementiert. Der Aufbau ist vom Input bis zur Aggregation der globalen Feature-Signatur identisch, daher wird im Weiteren nur kurz darauf eingegangen.

Das Hauptaugenmerk liegt auf den letzten Teil des Netzwerks, welcher für die eigentliche Segmentierung genutzt wird.

Zu Beginn des Netzwerks findet die bereits bekannte Input-Transformation statt. Hierbei wird eine Transformationsmatrix bestimmt, mit welcher die Elemente einer eingegebenen Punktwolke räumlich ausgerichtet werden.

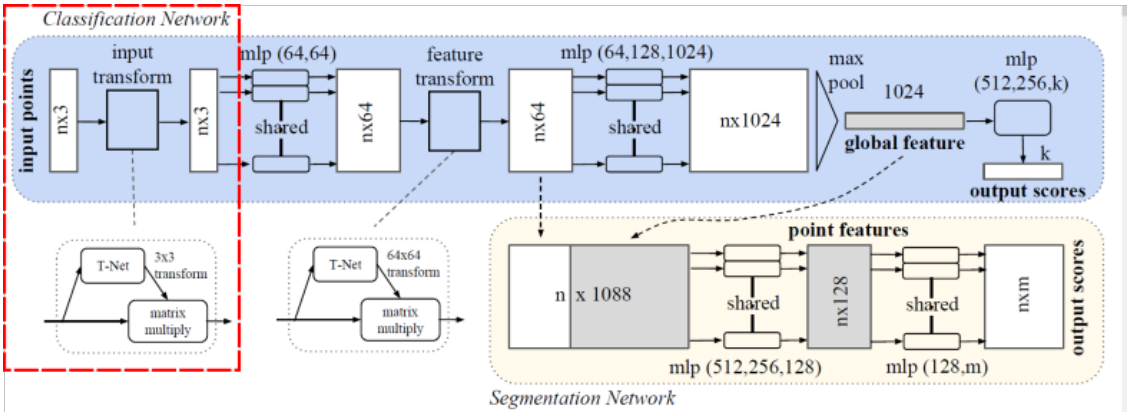


Abbildung 3: Input-Transformation [2]

[9]:
```
# ---------------------------------- Pointnet Architecture
# input_Transformation_net
input_points = Input(shape=(num_points, 3))
x = Convolution1D(64, 1, activation='relu',
                  input_shape=(num_points, 3))(input_points)
x = BatchNormalization()(x)
x = Convolution1D(128, 1, activation='relu')(x)
x = BatchNormalization()(x)
x = Convolution1D(1024, 1, activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling1D(pool_size=num_points)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dense(9, weights=[np.zeros([256, 9]), np.array([1, 0, 0, 0, 1, 0, 0, 0, 1]).
 ↪astype(np.float32)])(x)
input_T = Reshape((3, 3))(x)
```

Im zweiten Schritt findet das bereits bekannte Feature-Building statt. Zur Erinnerung: Hierbei werden die Raumkoordinaten eines jeden Punktes auf einen höherdimensionalen Raum abgebildet, um vom Netzwerk analysierbare Features zu gewinnen. Diese Form der Abstraktion ermöglicht es, beliebige Punktwolken mit PointNet zu verarbeiten, ohne dass diese vorher in eine

spezifische Form konvertiert werden müssen.

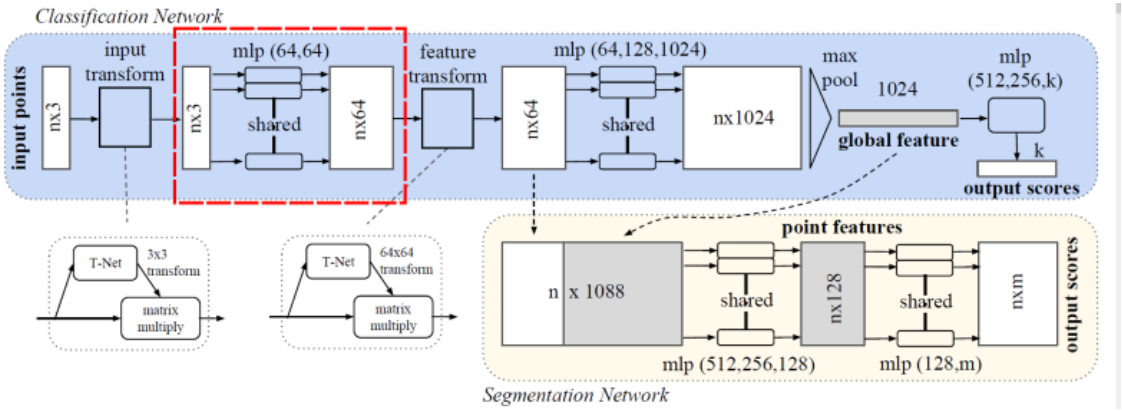


Abbildung 4: Feature Building 1 [2]

[10]:
```
# forward net
g = Lambda(mat_mul, arguments={'B': input_T})(input_points)
g = Convolution1D(64, 1, input_shape=(num_points, 3), activation='relu')(g)
g = BatchNormalization()(g)
g = Convolution1D(64, 1, input_shape=(num_points, 3), activation='relu')(g)
g = BatchNormalization()(g)
```

Auch die Feature-Transformation findet in bereits bekannter Weise statt. Die Ausrichtung der Feature-Vektoren im $\mathbb{R}^{64}$ dient ebenso wie die räumliche Ausrichtung im dreidimensionalen Raum dazu, etwaige Transformationen des Inputs wie bspw. Rotationen zu kompensieren.

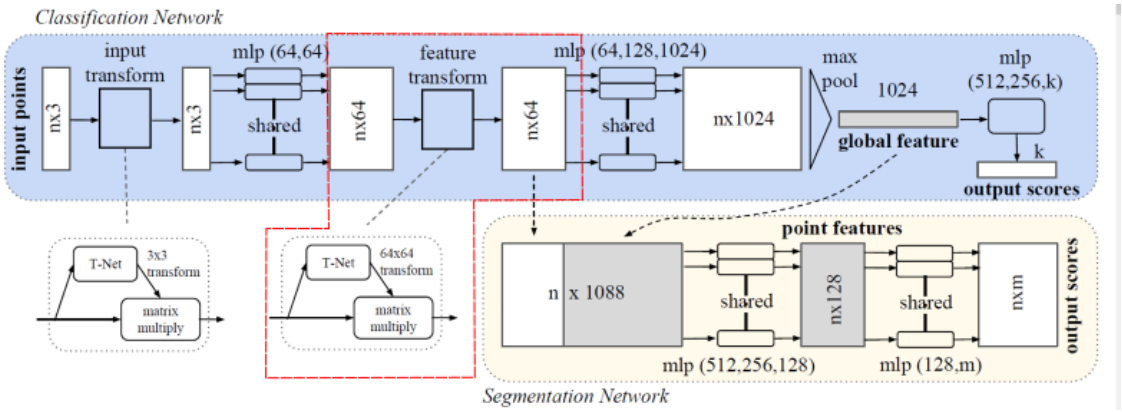


Abbildung 5: Feature-Transformation [2]

[11]:
```
# feature transformation net
f = Convolution1D(64, 1, activation='relu')(g)
f = BatchNormalization()(f)
f = Convolution1D(128, 1, activation='relu')(f)
f = BatchNormalization()(f)
f = Convolution1D(1024, 1, activation='relu')(f)
f = BatchNormalization()(f)
f = MaxPooling1D(pool_size=num_points)(f)
```

```
f = Dense(512, activation='relu')(f)
f = BatchNormalization()(f)
f = Dense(256, activation='relu')(f)
f = BatchNormalization()(f)
f = Dense(64 * 64, weights=[np.zeros([256, 64 * 64]), np.eye(64).flatten().
 ↪astype(np.float32)])(f)
feature_T = Reshape((64, 64))(f)
```

Im nachfolgenden Schritt findet sich der erste Unterschied zwischen Klassifizierung und Segmentierung. Nach Anwendung der Feature-Transformationsmatrix durch eine Lambda Layer wird der Zwischenstand in der Variablen `seg_part1` zwischengespeichert. Anschließend werden die ausgerichteten Feature-Vektoren über Faltungsschichten auf den $\mathbb{R}^{1024}$ abgebildet.

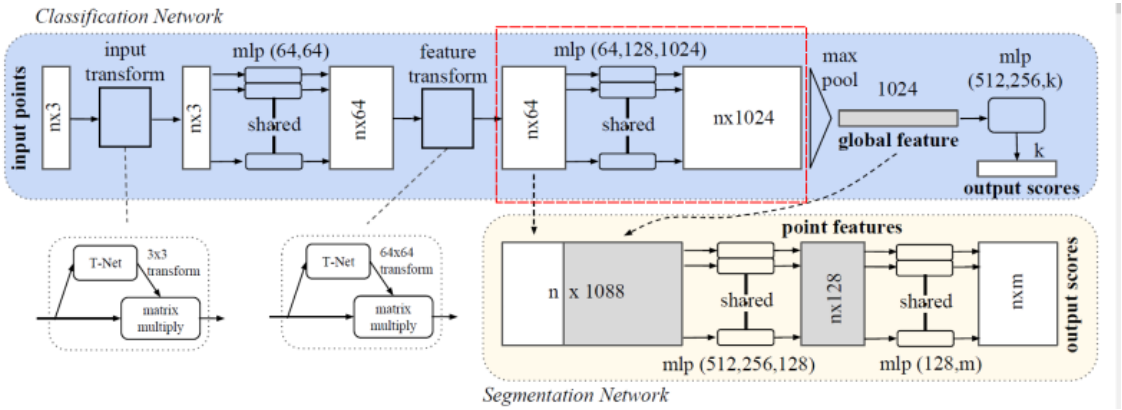


Abbildung 6: Feature Building 2 [2]

[12]:
```
# forward net
g = Lambda(mat_mul, arguments={'B': feature_T})(g)
seg_part1 = g
g = Convolution1D(64, 1, activation='relu')(g)
g = BatchNormalization()(g)
g = Convolution1D(128, 1, activation='relu')(g)
g = BatchNormalization()(g)
g = Convolution1D(1024, 1, activation='relu')(g)
g = BatchNormalization()(g)
```

Der durch Max-Pooling aggregierte Feature-Vektor wird über eine Lambda Layer und der eingangs definierten Hilfsfunktion `exp_dim` erweitert. Hierbei werden die 1024 Werte des Vektors für jedes Element einer Punktwolke dupliziert. Im gegebenen Beispiel wird also der Feature-Vektor von 1x1024 auf 1024x1024 erweitert, wobei jede Zeile eine Kopie der globalen Signatur enthält.

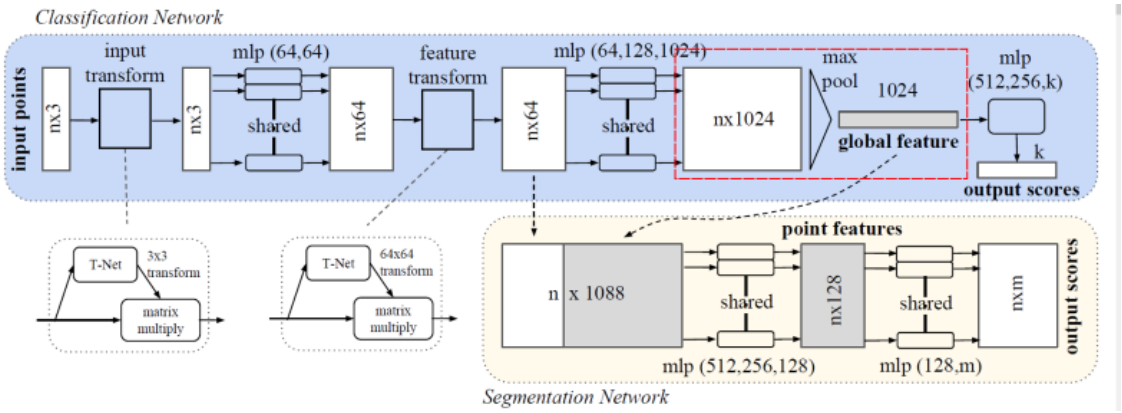


Abbildung 7: Max-Pooling [2]

[13]:
```
# global_feature
global_feature = MaxPooling1D(pool_size=num_points)(g)
global_feature = Lambda(exp_dim, arguments={'num_points':␣
 ↪num_points})(global_feature)
```

Im Anschluss an die Aggregation der globalen Feature-Signatur beginnt das eigentliche Segmentierungsnetzwerk. Im ersten Schritt, welcher in Abbbildung 8 markiert ist, findet eine Konkatenation der zuvor zwischengespeicherten lokalen Features und der globalen Features statt. Durch die nach dem Max-Pooling durchgeführte Erweiterung des globalen Feature-Vektors kann dies ohne Umstände mit der Funktion `concatenate` erreicht werden.

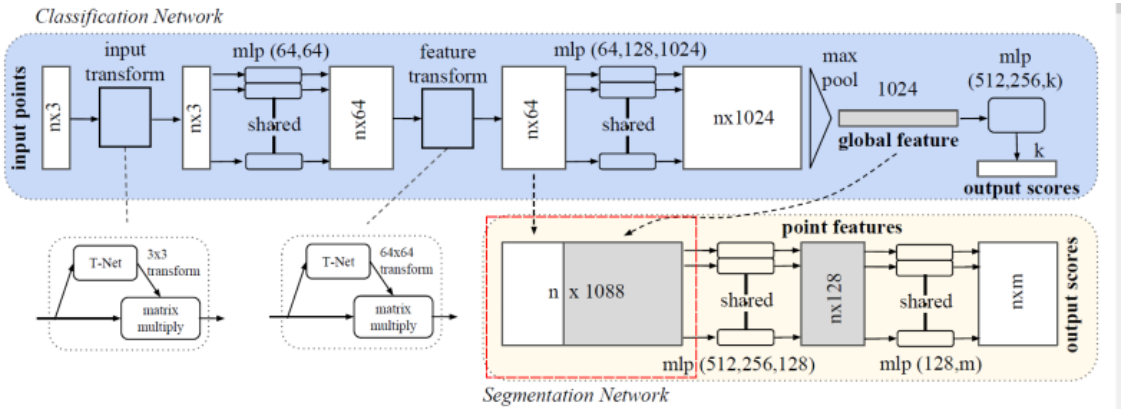


Abbildung 8: Konkatenation lokaler und globaler Features [2]

[14]:
```
# point_net_seg
c = concatenate([seg_part1, global_feature])
```

Mit dem konkatenierten Feature-Vektor wird jedes Element einer Punktwolke durch 1088 Werte beschrieben. Die Kombination von Informationen über lokale Geometrie und globale Semantik ermöglicht dem Netzwerk, mithilfe eines MLP eine akkurate Vorhersage über die Zugehörigkeit der einzelnen Punkte zu den verfügbaren Objektteilen zu treffen. Dieser Schritt wird in Abbildung 9 markiert dargestellt. Der MLP wird mit insgesamt fünf Faltungsschichten mit Filtern der Größe 1 realisiert. Zwischen den Schichten findet wie gewohnt `batch normalization` statt. Die ersten vier Schichten nutzen ReLU als Aktivierungsfunktion, während die letzte mittels Softmax die finalen

Segmentierungswerte für jeden Punkt als Output ermittelt.

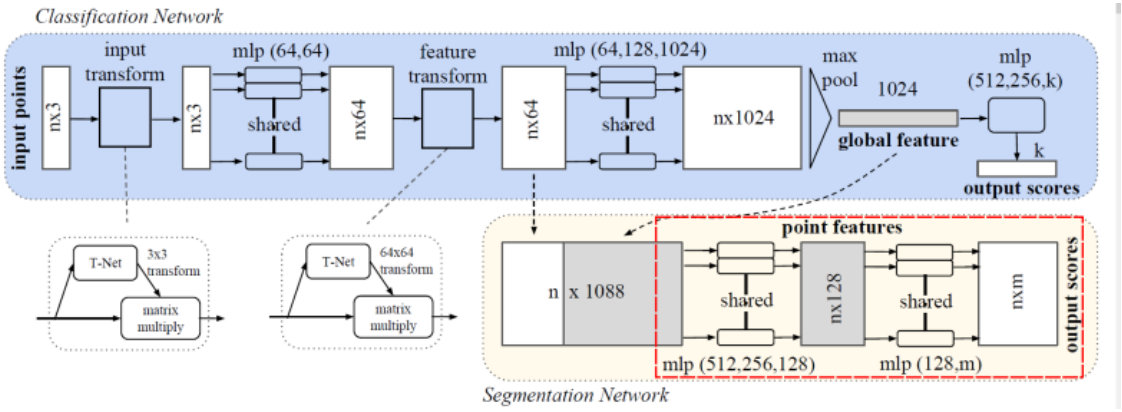


Abbildung 9: Segmentierung [2]

[15]:
```
c = Convolution1D(512, 1, activation='relu')(c)
c = BatchNormalization()(c)
c = Convolution1D(256, 1, activation='relu')(c)
c = BatchNormalization()(c)
c = Convolution1D(128, 1, activation='relu')(c)
c = BatchNormalization()(c)
c = Convolution1D(128, 1, activation='relu')(c)
c = BatchNormalization()(c)
prediction = Convolution1D(k, 1, activation='softmax')(c)
# ------------------------------------------------------end of pointnet
```

Wie im vorherigen Notebook wird nach der Vervollständigung des Netzwerks mittels verknüpftem Input und Output das entsprechende Keras-Modell gebildet und dessen Struktur tabellarisch ausgegeben. Anschließend wird das Modell kompiliert, wobei auch in diesem Beispiel der Optimizer `Adam` und die Verlustfunktion `categorical_crossentropy`genutzt werden.

[16]:
```
# define model
model = Model(inputs=input_points, outputs=prediction)
print(model.summary())
```

```
Model: "model_1"
_________________________________________________________________________________
__________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================
==================
input_1 (InputLayer)         (None, 1024, 3)           0
_________________________________________________________________________________
__________________
lambda_1 (Lambda)            (None, 1024, 3)           0           input_1[0][0]
_________________________________________________________________________________
__________________
```

```
conv1d_4 (Conv1D)               (None, 1024, 64)     256         lambda_1[0][0]
__________________________________________________________________________________
__________________
batch_normalization_6 (BatchNor (None, 1024, 64)     256         conv1d_4[0][0]
__________________________________________________________________________________
__________________
conv1d_5 (Conv1D)               (None, 1024, 64)     4160
batch_normalization_6[0][0]
__________________________________________________________________________________
__________________
batch_normalization_7 (BatchNor (None, 1024, 64)     256         conv1d_5[0][0]
__________________________________________________________________________________
__________________
lambda_2 (Lambda)               (None, 1024, 64)     0
batch_normalization_7[0][0]
__________________________________________________________________________________
__________________
conv1d_9 (Conv1D)               (None, 1024, 64)     4160        lambda_2[0][0]
__________________________________________________________________________________
__________________
batch_normalization_13 (BatchNo (None, 1024, 64)     256         conv1d_9[0][0]
__________________________________________________________________________________
__________________
conv1d_10 (Conv1D)              (None, 1024, 128)    8320
batch_normalization_13[0][0]
__________________________________________________________________________________
__________________
batch_normalization_14 (BatchNo (None, 1024, 128)    512         conv1d_10[0][0]
__________________________________________________________________________________
__________________
conv1d_11 (Conv1D)              (None, 1024, 1024)   132096
batch_normalization_14[0][0]
__________________________________________________________________________________
__________________
batch_normalization_15 (BatchNo (None, 1024, 1024)   4096        conv1d_11[0][0]
__________________________________________________________________________________
__________________
max_pooling1d_3 (MaxPooling1D)  (None, 1, 1024)      0
batch_normalization_15[0][0]
__________________________________________________________________________________
__________________
lambda_3 (Lambda)               (None, 1024, 1024)   0
max_pooling1d_3[0][0]
__________________________________________________________________________________
__________________
concatenate_1 (Concatenate)     (None, 1024, 1088)   0           lambda_2[0][0]
                                                                 lambda_3[0][0]
__________________________________________________________________________________
```

```
------------------
conv1d_12 (Conv1D)              (None, 1024, 512)    557568
concatenate_1[0][0]
__________________________________________________________________________________
------------------
batch_normalization_16 (BatchNo (None, 1024, 512)    2048        conv1d_12[0][0]
__________________________________________________________________________________
------------------
conv1d_13 (Conv1D)              (None, 1024, 256)    131328
batch_normalization_16[0][0]
__________________________________________________________________________________
------------------
batch_normalization_17 (BatchNo (None, 1024, 256)    1024        conv1d_13[0][0]
__________________________________________________________________________________
------------------
conv1d_14 (Conv1D)              (None, 1024, 128)    32896
batch_normalization_17[0][0]
__________________________________________________________________________________
------------------
batch_normalization_18 (BatchNo (None, 1024, 128)    512         conv1d_14[0][0]
__________________________________________________________________________________
------------------
conv1d_15 (Conv1D)              (None, 1024, 128)    16512
batch_normalization_18[0][0]
__________________________________________________________________________________
------------------
batch_normalization_19 (BatchNo (None, 1024, 128)    512         conv1d_15[0][0]
__________________________________________________________________________________
------------------
conv1d_16 (Conv1D)              (None, 1024, 3)      387
batch_normalization_19[0][0]
==================================================================================
==================
Total params: 897,155
Trainable params: 892,419
Non-trainable params: 4,736
__________________________________________________________________________________
------------------
None
```

[17]:
```
# compile classification model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

1.5 Vorbereitung der Daten

Der Ladeprozess der Trainings- und Testdaten gestaltet sich ähnlich wie im Klassifikationsbeispiel. Die in der globalen Variable `s` festgelegte Objektklasse wird genutzt, um die korrekten HPF5-Files aus den Ordnern "Seg_Prep" bzw. "Seg_Prep_test" auszuwählen. Diese werden anschließend über die Hilfsfunktion `load_h5` eingelesen. Die Daten über Punkte und Labels werden in entsprechend benannte Variablen zwischengespeichert und via `reshape` in die Formen `[Objekt, Punkt, Koordinaten]` bzw. `[Objekt, Punkt, Klasse]` gebracht.

[18]:
```
# load TRAIN points and labels
path = os.getcwd()
train_path = os.path.join(path, "Seg_Prep")
filename = s + ".h5"

train_points = None
train_labels = None
cur_points, cur_labels = load_h5(os.path.join(train_path, filename))

train_labels = cur_labels
train_points = cur_points

train_points_r = train_points.reshape(-1, num_points, 3)
train_labels_r = train_labels.reshape(-1, num_points, k)
```

[19]:
```
# load TEST points and labels
test_path = os.path.join(path, "Seg_Prep_test")
filename = s + ".h5"

test_points = None
test_labels = None
cur_points, cur_labels = load_h5(os.path.join(test_path, filename))

test_labels = cur_labels
test_points = cur_points

test_points_r = test_points.reshape(-1, num_points, 3)
test_labels_r = test_labels.reshape(-1, num_points, k)
```

1.6 Trainingsprozess

Das Training des Segmentierungsnetzwerks erfolgt in derselben Weise wie im vorherigen Notebook. Über eine for-Schleife wird eine beliebige Anzahl Iterationen realisiert, in denen die mittels der Funktionen `train_points_rotate` und `train_points_jitter` minimal unterschiedliche Versionen der Trainingsdaten über `model.fit` an das Netzwerk übergeben werden.

Die `batch-size` legt fest, in welchen Chargen die Daten vom Netzwerk verarbeitet bzw. wieviele Elemente pro Update der Gradienten genutzt werden. Darüber hinaus sind bei der `batch normalization` der Bezeichung entsprechend die Elemente einer Charge betroffen.

Der Parameter `epochs` legt die Anzahl der Iterationen fest. Da dies in der übergeordneten Schleife festgelegt ist, um die Trainingsdaten in jeder Iteration zu verändern, liegt der Wert bei 1.

`shuffle` führt dazu, dass die Trainingsdaten vor jeder Epoche in eine zufällige Reihenfolge gebracht werden.

`verbose` legt die Visualisierung fest. Der Wert 1 steht hierbei für die Darstellung eines Fortschrittbalkens.

Wie im vorherigen Notebook findet in jeder fünften Iteration eine Evaluation des Netzwerks mit den Testdaten statt.

[20]:
```
# train model
for i in range(1, 51):
    # rotate and jitter point cloud every epoch
    train_points_rotate = rotate_point_cloud(train_points_r)
    train_points_jitter = jitter_point_cloud(train_points_rotate)
    model.fit(train_points_jitter, train_labels_r, batch_size=32, epochs=1,␣
 ↪shuffle=True, verbose=1)
    e = "Current epoch is:" + str(i)
    print(e)
    # evaluate model
    if i % 5 == 0:
        score = model.evaluate(test_points_r, test_labels_r, verbose=1)
        print('Test loss: ', score[0])
        print('Test accuracy: ', score[1])
```

```
Epoch 1/1
631/631 [==============================] - 6s 10ms/step - loss: 0.4772 -
accuracy: 0.8190
Current epoch is:1
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1835 -
accuracy: 0.9354
Current epoch is:2
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1441 -
accuracy: 0.9476
Current epoch is:3
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1345 -
accuracy: 0.9502
Current epoch is:4
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1322 -
accuracy: 0.9511
Current epoch is:5
158/158 [==============================] - 1s 4ms/step
Test loss:  2.8031916950322406
```

```
Test accuracy:  0.0874270647764206
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1300 -
accuracy: 0.9517
Current epoch is:6
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1295 -
accuracy: 0.9509
Current epoch is:7
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1244 -
accuracy: 0.9522
Current epoch is:8
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1261 -
accuracy: 0.9512
Current epoch is:9
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1234 -
accuracy: 0.9520
Current epoch is:10
158/158 [==============================] - 0s 2ms/step
Test loss:  5.0158094394056105
Test accuracy:  0.0874270647764206
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1187 -
accuracy: 0.9549
Current epoch is:11
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1155 -
accuracy: 0.9552
Current epoch is:12
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1169 -
accuracy: 0.9544
Current epoch is:13
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1154 -
accuracy: 0.9551
Current epoch is:14
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1162 -
accuracy: 0.9548
Current epoch is:15
158/158 [==============================] - 0s 2ms/step
Test loss:  7.235829504230354
Test accuracy:  0.0874270647764206
Epoch 1/1
```

```
631/631 [==============================] - 4s 6ms/step - loss: 0.1146 -
accuracy: 0.9553
Current epoch is:16
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1117 -
accuracy: 0.9562
Current epoch is:17
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1101 -
accuracy: 0.9569
Current epoch is:18
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1077 -
accuracy: 0.9576
Current epoch is:19
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1091 -
accuracy: 0.9571
Current epoch is:20
158/158 [==============================] - 0s 2ms/step
Test loss:  1.4206434337398675
Test accuracy:  0.742502748966217
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1078 -
accuracy: 0.9577
Current epoch is:21
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1092 -
accuracy: 0.9570
Current epoch is:22
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1083 -
accuracy: 0.9574
Current epoch is:23
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1073 -
accuracy: 0.9575
Current epoch is:24
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1063 -
accuracy: 0.9576
Current epoch is:25
158/158 [==============================] - 0s 2ms/step
Test loss:  1.870902877819689
Test accuracy:  0.7081932425498962
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1050 -
accuracy: 0.9583
```

```
Current epoch is:26
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1051 -
accuracy: 0.9583
Current epoch is:27
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1070 -
accuracy: 0.9570
Current epoch is:28
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1050 -
accuracy: 0.9585
Current epoch is:29
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1014 -
accuracy: 0.9601
Current epoch is:30
158/158 [==============================] - 0s 2ms/step
Test loss:  0.8824337901948374
Test accuracy:  0.7382874488830566
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1012 -
accuracy: 0.9599
Current epoch is:31
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1012 -
accuracy: 0.9601
Current epoch is:32
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1023 -
accuracy: 0.9591
Current epoch is:33
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1041 -
accuracy: 0.9579
Current epoch is:34
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1041 -
accuracy: 0.9577
Current epoch is:35
158/158 [==============================] - 0s 2ms/step
Test loss:  0.24156014470359946
Test accuracy:  0.9123071432113647
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0984 -
accuracy: 0.9608
Current epoch is:36
Epoch 1/1
```

```
631/631 [==============================] - 4s 6ms/step - loss: 0.0986 -
accuracy: 0.9607
Current epoch is:37
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0999 -
accuracy: 0.9602
Current epoch is:38
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0949 -
accuracy: 0.9625
Current epoch is:39
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0984 -
accuracy: 0.9605
Current epoch is:40
158/158 [==============================] - 0s 2ms/step
Test loss:  0.1557120973173576
Test accuracy:  0.944508969783783
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.1005 -
accuracy: 0.9592
Current epoch is:41
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0975 -
accuracy: 0.9612
Current epoch is:42
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0952 -
accuracy: 0.9621
Current epoch is:43
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0925 -
accuracy: 0.9630
Current epoch is:44
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0938 -
accuracy: 0.9629
Current epoch is:45
158/158 [==============================] - 0s 2ms/step
Test loss:  0.13900183546769468
Test accuracy:  0.9471296668052673
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0928 -
accuracy: 0.9629
Current epoch is:46
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0952 -
accuracy: 0.9619
```

```
Current epoch is:47
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0943 -
accuracy: 0.9625
Current epoch is:48
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0940 -
accuracy: 0.9625
Current epoch is:49
Epoch 1/1
631/631 [==============================] - 4s 6ms/step - loss: 0.0938 -
accuracy: 0.9622
Current epoch is:50
158/158 [==============================] - 0s 2ms/step
Test loss:  0.13251101866930345
Test accuracy:  0.9516910314559937
```

1.7 Visualisierung

Um die Effektivität des Netzwerks visuell nachvollziehen zu können, wird in den nachfolgenden Code-Zellen ein 3D-Plot eines beliebigen Datensatzes erzeugt.

An dieser Stelle ein wichtiger Hinweis: Um verschiedene Objekte zu visualisieren, muss das Netzwerk nicht jedes Mal von Neuem trainiert werden. Es ist ausreichend, lediglich die nachfolgende Codezelle via "Run"-Befehl auszuführen.

Hierfür wird zunächst ein `figure()`-Objekt des Pakets `matplotlib` erzeugt und mit einem 3D-Subplot gefüllt. Die Achsenbegrenzungen und die Größe der Darstellung können nach Belieben angepasst werden.

Die Arrays `color` und `m` legen die Farbe und Form der einzelnen Segmente in der Darstellung fest und können ebenfalls nach Belieben angepasst werden. Mögliche Werte können in der Dokumentation von `matplotlib` nachgelesen werden.

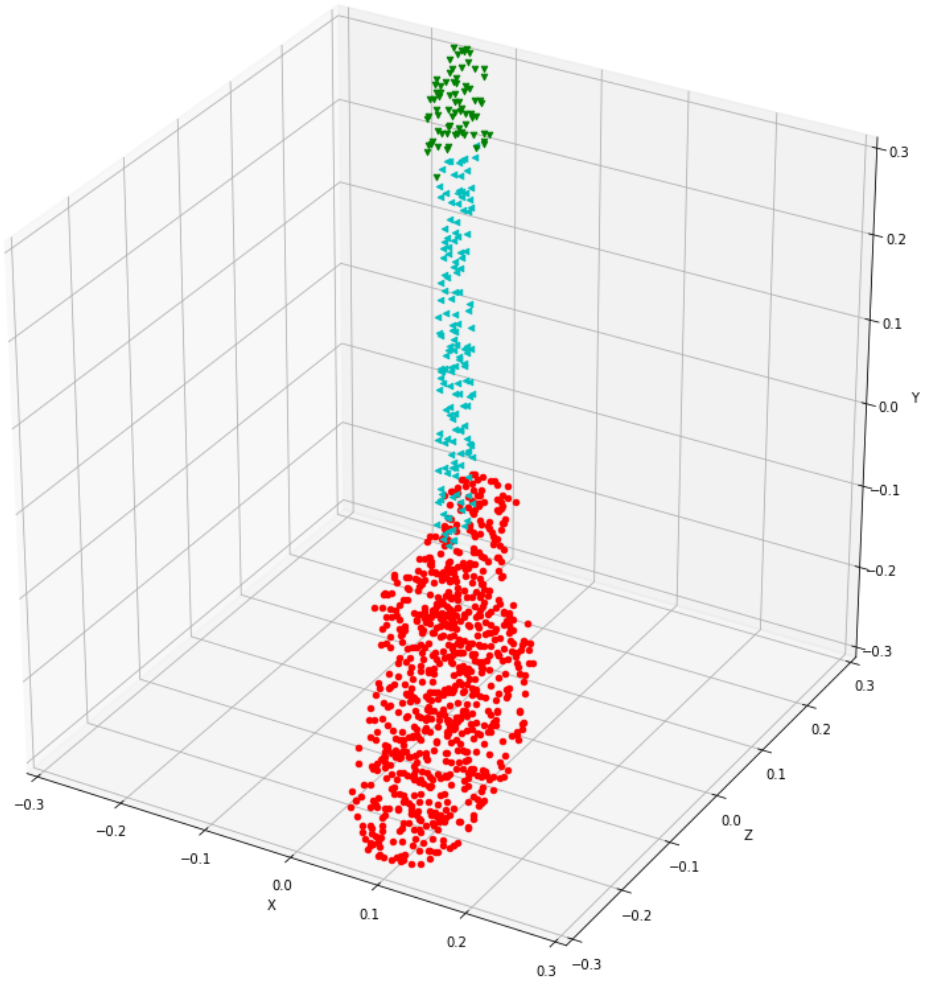
Über die Variable `d_num` kann auf die einzelnen Objekte des Testdatensatzes zugegriffen werden. Die entsprechenden Punkte werden extrahiert und an das Netzwerk übergeben. Der Output des Netzwerks wird in der Variable `pred` gespeichert und in eine Liste konvertiert.

Die `squeeze`-Funktion eliminiert atomare Dimensionen, um das Durchlaufen des Datensatzes zu vereinfachen.

Nachdem die Daten ausgewählt und in eine passende Form gebracht wurden, werden die einzelnen Punkte über eine for-Schleife als Scatter-Plot visualisiert.

[24]:
```python
# initialize plot
fig = plt.figure(figsize=(15,15))
ax = fig.add_subplot(111, projection='3d')

# set marker style and color
color = ['r', 'g', 'c', 'y', 'm']
m= ['o', 'v', '<', '>', 's']
```

```
# select test data to visualize
d_num = 1

v_points = test_points_r[d_num:d_num+1,:,:]
pred = model.predict(v_points)
pred = np.squeeze(pred)
v_points = np.squeeze(v_points)
pred = pred.tolist()

# add data to plot
for i in range(v_points.shape[0]):
    xs = v_points[i,0]
    ys = v_points[i,1]
    zs = v_points[i,2]
    ind = pred[i].index(max(pred[i]))
    ax.scatter(xs, zs, ys, c=color[ind], marker=m[ind])

# set axis labels and limits
ax.set_xlim3d(-0.3,0.3)
ax.set_ylim3d(-0.3,0.3)
ax.set_zlim3d(-0.3,0.3)

ax.set_xlabel('X')
ax.set_ylabel('Z')
ax.set_zlabel('Y')

plt.show()
```



Quellen:

[1] L. Yi et al., "A Scalable Active Framework for Region Annotation in 3D Shape Collections," SIGGRAPH Asia, 2016. [Online]. Available: http://web.stanford.edu/~ericyi/project_page/part_annotation/index.html

[2] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CoRR, vol. abs/1612.00593, 2016, [Online]. Available: http://arxiv.org/abs/1612.00593.