Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Модуль №2
«Изучение библиотеки PointNet»
по курсу: «Языки и методы программирования»

Выполнил:
Студент группы ИУ9-22Б
Гнатенко Т. А.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Знакомство с библиотекой PointNet https://github.com/charlesq34/pointnet.

Задачи

Реализовать пример.

Решение

Тренировка нейросети производилась на языке Python с помощью Google Collab.

Вывод

Рис. 1: Визуализация

```
[ ] model.compile(
        loss="sparse_categorical_crossentropy",
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        metrics=["sparse_categorical_accuracy"],
    )

    model.fit(train_dataset, epochs=20, validation_data=test_dataset)

    Epoch 1/20
    125/125 [==============================] - 411s 3s/step - loss: 3.6340 - sparse_categorical_accuracy: 0.2729 - val_loss: 613779951845376.0000 - val_sparse_categorical_accuracy: 0.2104
    Epoch 2/20
    125/125 [==============================] - 395s 3s/step - loss: 3.2244 - sparse_categorical_accuracy: 0.3385 - val_loss: 75591872.0000 - val_sparse_categorical_accuracy: 0.2037
    Epoch 3/20
    125/125 [==============================] - 405s 3s/step - loss: 2.9516 - sparse_categorical_accuracy: 0.3974 - val_loss: 983314070721527808.0000 - val_sparse_categorical_accuracy: 0.3095
    Epoch 4/20
    125/125 [==============================] - 405s 3s/step - loss: 2.8224 - sparse_categorical_accuracy: 0.4886 - val_loss: 128473587198722048.0000 - val_sparse_categorical_accuracy: 0.4681
    Epoch 5/20
    125/125 [==============================] - 406s 3s/step - loss: 2.5571 - sparse_categorical_accuracy: 0.5455 - val_loss: 130702624.0000 - val_sparse_categorical_accuracy: 0.3205
    Epoch 6/20
    125/125 [==============================] - 404s 3s/step - loss: 2.5185 - sparse_categorical_accuracy: 0.5658 - val_loss: 1024.2030 - val_sparse_categorical_accuracy: 0.3480
    Epoch 7/20
    125/125 [==============================] - 404s 3s/step - loss: 2.3562 - sparse_categorical_accuracy: 0.6151 - val_loss: 658174188737003520.0000 - val_sparse_categorical_accuracy: 0.5540
    Epoch 8/20
    125/125 [==============================] - 406s 3s/step - loss: 2.2483 - sparse_categorical_accuracy: 0.6485 - val_loss: 89558876160000.0000 - val_sparse_categorical_accuracy: 0.6145
    Epoch 9/20
    125/125 [==============================] - 404s 3s/step - loss: 2.0730 - sparse_categorical_accuracy: 0.6966 - val_loss: 912187457536.0000 - val_sparse_categorical_accuracy: 0.7269
    Epoch 10/20
    125/125 [==============================] - 404s 3s/step - loss: 2.0543 - sparse_categorical_accuracy: 0.7048 - val_loss: 2713.7161 - val_sparse_categorical_accuracy: 0.6564
    Epoch 11/20
    125/125 [==============================] - 402s 3s/step - loss: 1.9854 - sparse_categorical_accuracy: 0.7304 - val_loss: 11843.2529 - val_sparse_categorical_accuracy: 0.6630
    Epoch 12/20
    125/125 [==============================] - 404s 3s/step - loss: 1.8662 - sparse_categorical_accuracy: 0.7549 - val_loss: 5025914421248.0000 - val_sparse_categorical_accuracy: 0.7081
    Epoch 13/20
    125/125 [==============================] - 404s 3s/step - loss: 1.8095 - sparse_categorical_accuracy: 0.7622 - val_loss: 7.9643 - val_sparse_categorical_accuracy: 0.7048
    Epoch 14/20
    125/125 [==============================] - 397s 3s/step - loss: 1.8049 - sparse_categorical_accuracy: 0.7690 - val_loss: 2003100928.0000 - val_sparse_categorical_accuracy: 0.6828
    Epoch 15/20
    125/125 [==============================] - 407s 3s/step - loss: 1.8328 - sparse_categorical_accuracy: 0.7615 - val_loss: 1411057814738552907366 4.0000 - val_sparse_categorical_accuracy: 0.6333
    Epoch 16/20
    125/125 [==============================] - 405s 3s/step - loss: 1.8174 - sparse_categorical_accuracy: 0.7690 - val_loss: 156475416758976512.0000 - val_sparse_categorical_accuracy: 0.7478
    Epoch 17/20
    125/125 [==============================] - 394s 3s/step - loss: 1.7784 - sparse_categorical_accuracy: 0.7783 - val_loss: 10508228047667200.0000 - val_sparse_categorical_accuracy: 0.7852
    Epoch 18/20
    125/125 [==============================] - 394s 3s/step - loss: 1.7360 - sparse_categorical_accuracy: 0.7943 - val_loss: 1967144.3750 - val_sparse_categorical_accuracy: 0.7577
    Epoch 19/20
    125/125 [==============================] - 395s 3s/step - loss: 1.6847 - sparse_categorical_accuracy: 0.8046 - val_loss: 117726.3047 - val_sparse_categorical_accuracy: 0.7467
    Epoch 20/20
    125/125 [==============================] - 396s 3s/step - loss: 1.6471 - sparse_categorical_accuracy: 0.8211 - val_loss: 21164581715968.0000 - val_sparse_categorical_accuracy: 0.7863
    <keras.callbacks.History at 0x7f59f14866d0>
```

Рис. 2: Процесс обучения

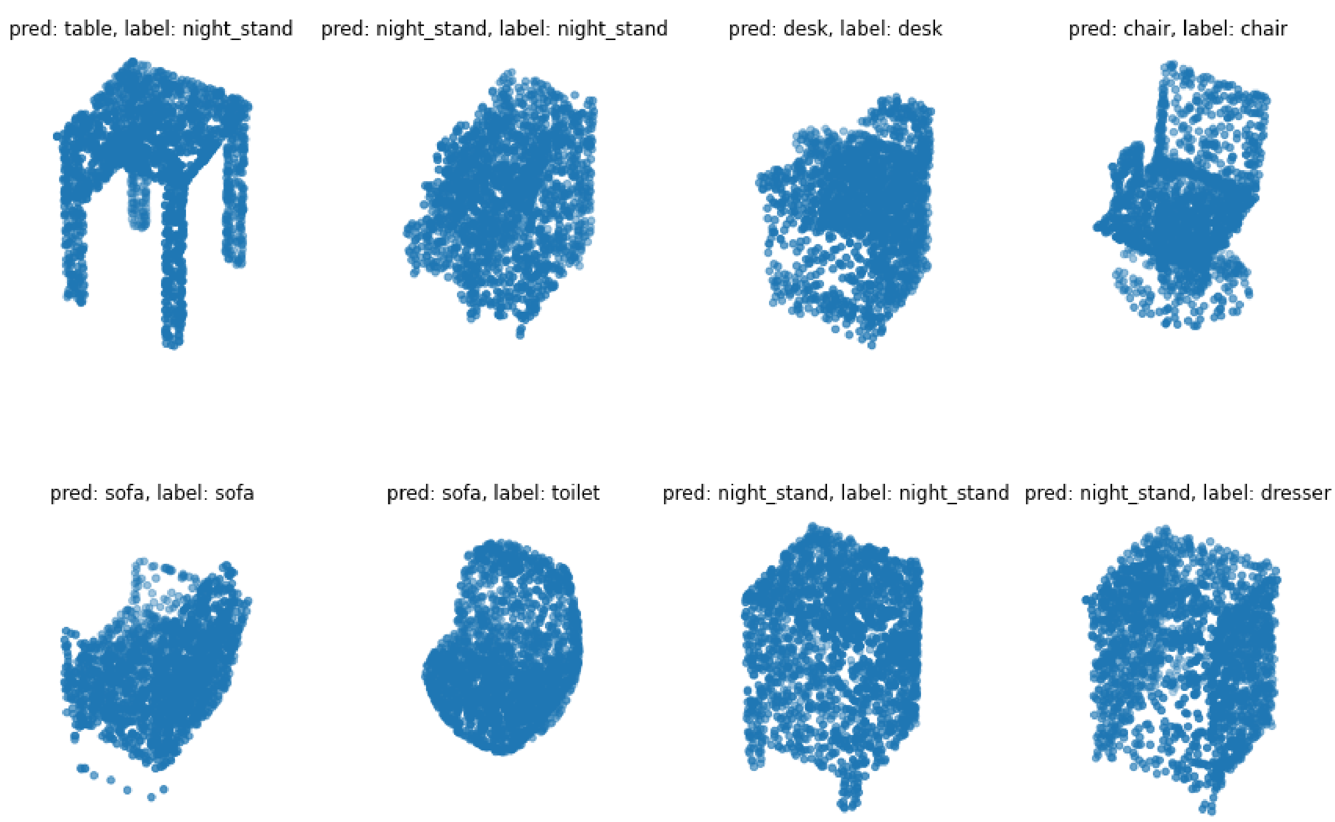

Рис. 3: Вывод натренированной нейросети

Исходный код

example.py

```
"""
Point cloud classification with PointNet
Author: David Griffiths
Date created: 2020/05/25
Last modified: 2020/05/26
Description: Implementation of PointNet for ModelNet10 classification.

Point cloud classification
Introduction
Classification, detection and segmentation of unordered 3D point sets
 ↪  i.e. point clouds is a core problem in computer vision. This example
 ↪  implements the seminal point cloud deep learning paper PointNet (Qi
 ↪  et al., 2017). For a detailed intoduction on PointNet see this blog
 ↪  post.

"""
!pip install trimesh
Setup
If using colab first install trimesh with !pip install trimesh.

"""

import os
import glob
import trimesh
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from matplotlib import pyplot as plt

tf.random.set_seed(1234)

Load dataset
We use the ModelNet10 model dataset, the smaller 10 class version of the
 ↪  ModelNet40 dataset. First download the data:

"""
DATA_DIR = tf.keras.utils.get_file(
    "modelnet.zip",

     ↪  "http://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip",
    extract=True,
)
DATA_DIR = os.path.join(os.path.dirname(DATA_DIR), "ModelNet10")

We can use the trimesh package to read and visualize the .off mesh files.

"""
mesh = trimesh.load(os.path.join(DATA_DIR, "chair/train/chair_0001.off"))
mesh.show()

To convert a mesh file to a point cloud we first need to sample points on
 ↪  the mesh surface. .sample() performs a unifrom random sampling. Here
 ↪  we sample at 2048 locations and visualize in matplotlib.

"""
points = mesh.sample(2048)
```

```
fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(points[:, 0], points[:, 1], points[:, 2])
ax.set_axis_off()
plt.show()

To generate a tf.data.Dataset() we need to first parse through the
↪  ModelNet data folders. Each mesh is loaded and sampled into a point
↪  cloud before being added to a standard python list and converted to a
↪  numpy array. We also store the current enumerate index value as the
↪  object label and use a dictionary to recall this later.

"""

def parse_dataset(num_points=2048):

    train_points = []
    train_labels = []
    test_points = []
    test_labels = []
    class_map = {}
    folders = glob.glob(os.path.join(DATA_DIR, "[!README]*"))

    for i, folder in enumerate(folders):
        print("processing class: {}".format(os.path.basename(folder)))
        # store folder name with ID so we can retrieve later
        class_map[i] = folder.split("/")[-1]
        # gather all files
        train_files = glob.glob(os.path.join(folder, "train/*"))
        test_files = glob.glob(os.path.join(folder, "test/*"))

        for f in train_files:
            train_points.append(trimesh.load(f).sample(num_points))
            train_labels.append(i)

        for f in test_files:
            test_points.append(trimesh.load(f).sample(num_points))
            test_labels.append(i)

    return (
        np.array(train_points),
        np.array(test_points),
        np.array(train_labels),
        np.array(test_labels),
        class_map,
    )


Set the number of points to sample and batch size and parse the dataset.
↪  This can take ~5minutes to complete.

"""
NUM_POINTS = 2048
NUM_CLASSES = 10
BATCH_SIZE = 32

train_points, test_points, train_labels, test_labels, CLASS_MAP =
↪  parse_dataset(
    NUM_POINTS
)
```

```
Our data can now be read into a tf.data.Dataset() object. We set the
 ↪  shuffle buffer size to the entire size of the dataset as prior to
 ↪  this the data is ordered by class. Data augmentation is important
 ↪  when working with point cloud data. We create a augmentation function
 ↪  to jitter and shuffle the train dataset.

"""

def augment(points, label):
    # jitter points
    points += tf.random.uniform(points.shape, -0.005, 0.005,
 ↪  dtype=tf.float64)
    # shuffle points
    points = tf.random.shuffle(points)
    return points, label


train_dataset = tf.data.Dataset.from_tensor_slices((train_points,
 ↪  train_labels))
test_dataset = tf.data.Dataset.from_tensor_slices((test_points,
 ↪  test_labels))

train_dataset =
 ↪  train_dataset.shuffle(len(train_points)).map(augment).batch(BATCH_SIZE)
test_dataset = test_dataset.shuffle(len(test_points)).batch(BATCH_SIZE)

Build a model
Each convolution and fully-connected layer (with exception for end
 ↪  layers) consits of Convolution / Dense -> Batch Normalization -> ReLU
 ↪  Activation.

"""

def conv_bn(x, filters):
    x = layers.Conv1D(filters, kernel_size=1, padding="valid")(x)
    x = layers.BatchNormalization(momentum=0.0)(x)
    return layers.Activation("relu")(x)


def dense_bn(x, filters):
    x = layers.Dense(filters)(x)
    x = layers.BatchNormalization(momentum=0.0)(x)
    return layers.Activation("relu")(x)


PointNet consists of two core components. The primary MLP network, and
 ↪  the transformer net (T-net). The T-net aims to learn an affine
 ↪  transformation matrix by its own mini network. The T-net is used
 ↪  twice. The first time to transform the input features (n, 3) into a
 ↪  canonical representation. The second is an affine transformation for
 ↪  alignment in feature space (n, 3). As per the original paper we
 ↪  constrain the transformation to be close to an orthogonal matrix
 ↪  (i.e. ||X*X^T - I|| = 0).

"""

class OrthogonalRegularizer(keras.regularizers.Regularizer):
    def __init__(self, num_features, l2reg=0.001):
        self.num_features = num_features
        self.l2reg = l2reg
        self.eye = tf.eye(num_features)
```

```
    def __call__(self, x):
        x = tf.reshape(x, (-1, self.num_features, self.num_features))
        xxt = tf.tensordot(x, x, axes=(2, 2))
        xxt = tf.reshape(xxt, (-1, self.num_features, self.num_features))
        return tf.reduce_sum(self.l2reg * tf.square(xxt - self.eye))


We can then define a general function to build T-net layers.

"""

def tnet(inputs, num_features):

    # Initalise bias as the indentity matrix
    bias = keras.initializers.Constant(np.eye(num_features).flatten())
    reg = OrthogonalRegularizer(num_features)

    x = conv_bn(inputs, 32)
    x = conv_bn(x, 64)
    x = conv_bn(x, 512)
    x = layers.GlobalMaxPooling1D()(x)
    x = dense_bn(x, 256)
    x = dense_bn(x, 128)
    x = layers.Dense(
        num_features * num_features,
        kernel_initializer="zeros",
        bias_initializer=bias,
        activity_regularizer=reg,
    )(x)
    feat_T = layers.Reshape((num_features, num_features))(x)
    # Apply affine transformation to input features
    return layers.Dot(axes=(2, 1))([inputs, feat_T])


The main network can be then implemented in the same manner where the
↪   t-net mini models can be dropped in a layers in the graph. Here we
↪   replicate the network architecture published in the original paper
↪   but with half the number of weights at each layer as we are using the
↪   smaller 10 class ModelNet dataset.

"""
inputs = keras.Input(shape=(NUM_POINTS, 3))

x = tnet(inputs, 3)
x = conv_bn(x, 32)
x = conv_bn(x, 32)
x = tnet(x, 32)
x = conv_bn(x, 32)
x = conv_bn(x, 64)
x = conv_bn(x, 512)
x = layers.GlobalMaxPooling1D()(x)
x = dense_bn(x, 256)
x = layers.Dropout(0.3)(x)
x = dense_bn(x, 128)
x = layers.Dropout(0.3)(x)

outputs = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs, name="pointnet")
model.summary()

Train model
```

```
Once the model is defined it can be trained like any other standard
 ↪  classification model using .compile() and .fit().

"""
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["sparse_categorical_accuracy"],
)

model.fit(train_dataset, epochs=20, validation_data=test_dataset)

Visualize predictions
We can use matplotlib to visualize our trained model performance.

"""
data = test_dataset.take(1)

points, labels = list(data)[0]
points = points[:8, ...]
labels = labels[:8, ...]

# run test data through model
preds = model.predict(points)
preds = tf.math.argmax(preds, -1)
```