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

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

Выполнил:
Студент группы ИУ9-21Б
Старовойтов А. И.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Знакомство с библиотекой `PointNet`.

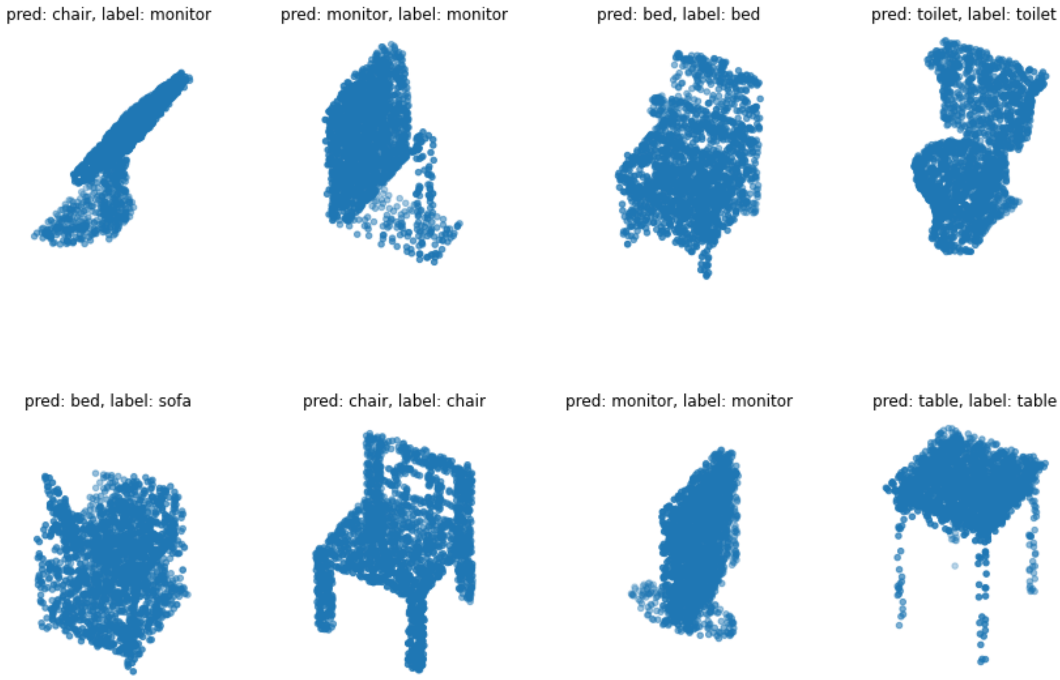
Задачи

Реализовать пример https://keras.io/examples/vision/pointnet/

Решение

Проект был запущен в `Google Colab`.

Пример работы модели:



Вывод при обучении:

```
Epoch 1/20
125/125 [==============================] - 492s 4s/step -
 ↪  loss: 3.5283 - sparse_categorical_accuracy: 0.2902 -
 ↪  val_loss: 210954740446527488.0000 -
 ↪  val_sparse_categorical_accuracy: 0.2313
```

```
Epoch 2/20
125/125 [==============================] - 484s 4s/step -
 ↪  loss: 3.0721 - sparse_categorical_accuracy: 0.3698 -
 ↪  val_loss: 334798047957483520000 -
 ↪  val_sparse_categorical_accuracy: 0.2985
Epoch 3/20
125/125 [==============================] - 491s 4s/step -
 ↪  loss: 2.9749 - sparse_categorical_accuracy: 0.4174 -
 ↪  val_loss: 875655449280512.0000 -
 ↪  val_sparse_categorical_accuracy: 0.4306
Epoch 4/20
125/125 [==============================] - 486s 4s/step -
 ↪  loss: 2.6991 - sparse_categorical_accuracy: 0.5009 -
 ↪  val_loss: 180347.5469 - val_sparse_categorical_accuracy:
 ↪  0.3282
Epoch 5/20
125/125 [==============================] - 486s 4s/step -
 ↪  loss: 2.5644 - sparse_categorical_accuracy: 0.5440 -
 ↪  val_loss: 1061512935416791040000 -
 ↪  val_sparse_categorical_accuracy: 0.4251
Epoch 6/20
125/125 [==============================] - 486s 4s/step -
 ↪  loss: 2.4421 - sparse_categorical_accuracy: 0.5755 -
 ↪  val_loss: 424143.7500 - val_sparse_categorical_accuracy:
 ↪  0.4857
Epoch 7/20
125/125 [==============================] - 485s 4s/step -
 ↪  loss: 2.3482 - sparse_categorical_accuracy: 0.6021 -
 ↪  val_loss: 9424011788288.0000 -
 ↪  val_sparse_categorical_accuracy: 0.5980
Epoch 8/20
125/125 [==============================] - 487s 4s/step -
 ↪  loss: 2.2929 - sparse_categorical_accuracy: 0.6221 -
 ↪  val_loss: 523065917440.0000 -
 ↪  val_sparse_categorical_accuracy: 0.5991
Epoch 9/20
125/125 [==============================] - 489s 4s/step -
 ↪  loss: 2.1982 - sparse_categorical_accuracy: 0.6595 -
 ↪  val_loss: 79335302103040.0000 -
 ↪  val_sparse_categorical_accuracy: 0.7236
```

```
Epoch 10/20
125/125 [==============================] - 492s 4s/step -
 ↪  loss: 2.0412 - sparse_categorical_accuracy: 0.7114 -
 ↪  val_loss: 32762660864.0000 -
 ↪  val_sparse_categorical_accuracy: 0.5859
Epoch 11/20
125/125 [==============================] - 494s 4s/step -
 ↪  loss: 2.0109 - sparse_categorical_accuracy: 0.7098 -
 ↪  val_loss: 57038135296.0000 -
 ↪  val_sparse_categorical_accuracy: 0.6597
Epoch 12/20
125/125 [==============================] - 490s 4s/step -
 ↪  loss: 1.9475 - sparse_categorical_accuracy: 0.7417 -
 ↪  val_loss: 6202808329727337562112.0000 -
 ↪  val_sparse_categorical_accuracy: 0.5154
Epoch 13/20
125/125 [==============================] - 493s 4s/step -
 ↪  loss: 2.0223 - sparse_categorical_accuracy: 0.7234 -
 ↪  val_loss: 5557061689540608.0000 -
 ↪  val_sparse_categorical_accuracy: 0.7852
Epoch 14/20
125/125 [==============================] - 494s 4s/step -
 ↪  loss: 1.9383 - sparse_categorical_accuracy: 0.7374 -
 ↪  val_loss: 149.8369 - val_sparse_categorical_accuracy:
 ↪  0.5892
Epoch 15/20
125/125 [==============================] - 492s 4s/step -
 ↪  loss: 1.8217 - sparse_categorical_accuracy: 0.7675 -
 ↪  val_loss: 2.4758 - val_sparse_categorical_accuracy:
 ↪  0.5088
Epoch 16/20
125/125 [==============================] - 506s 4s/step -
 ↪  loss: 1.7787 - sparse_categorical_accuracy: 0.7760 -
 ↪  val_loss: 2338560.7500 -
 ↪  val_sparse_categorical_accuracy: 0.7192
Epoch 17/20
125/125 [==============================] - 510s 4s/step -
 ↪  loss: 1.7970 - sparse_categorical_accuracy: 0.7853 -
 ↪  val_loss: 146297877168128.0000 -
 ↪  val_sparse_categorical_accuracy: 0.6278
```

```
Epoch 18/20
125/125 [==============================] - 518s 4s/step -
 ↪  loss: 1.7527 - sparse_categorical_accuracy: 0.7870 -
 ↪  val_loss: 6409579659264.0000 -
 ↪  val_sparse_categorical_accuracy: 0.6718
Epoch 19/20
125/125 [==============================] - 509s 4s/step -
 ↪  loss: 1.6873 - sparse_categorical_accuracy: 0.8066 -
 ↪  val_loss: 62779597095174144.0000 -
 ↪  val_sparse_categorical_accuracy: 0.8128
Epoch 20/20
125/125 [==============================] - 514s 4s/step -
 ↪  loss: 1.6595 - sparse_categorical_accuracy: 0.8083 -
 ↪  val_loss: 1481432192.0000 -
 ↪  val_sparse_categorical_accuracy: 0.6894
```

Исходный код:

```
"""
# Point cloud classification with PointNet

**Author:** [David
 ↪  Griffiths](https://dgriffiths3.github.io)<br>
**Date created:** 2020/05/25<br>
**Last modified:** 2020/05/26<br>
**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)](https://arxiv.org/abs/1612.00593). For a
detailed intoduction on PointNet see [this blog
post](https://medium.com/@luis_gonzales/an-in-depth-look-at-
 ↪  pointnet-111d7efdaa1a).
```

```
## 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/ModelNet1
    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)

points = points.numpy()

# plot points with predicted class and label
fig = plt.figure(figsize=(15, 10))
for i in range(8):
    ax = fig.add_subplot(2, 4, i + 1, projection="3d")
    ax.scatter(points[i, :, 0], points[i, :, 1], points[i,
 ↪  :, 2])
    ax.set_title(
        "pred: {:}, label: {:}".format(
            CLASS_MAP[preds[i].numpy()],
 ↪  CLASS_MAP[labels.numpy()[i]]
        )
```