limhpone / computervision-final-prep

Code Issues Pull requests Actions Projects Wiki Security In

computervision-final-prep / lab / Lab 12 (3D Vision)-20251128 / lab12.3_PointNetClass.ipynb

limhpone 3D Vision- lab 12 daadd1a · 2 hours ago

2.1 MB

PointNet

This is an implementation of [PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation] using PyTorch.

Getting started

Don't forget to turn on GPU if you want to start training directly.

Runtime -> **Change runtime type**-> **Hardware accelerator**

In [1]:

```
import numpy as np
import math
import random
import os
import torch
import scipy.spatial.distance
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils

import plotly.graph_objects as go
import plotly.express as px
```

In [2]:

```
!pip install path.py;
from path import Path
```

```
Collecting path.py
  Downloading path.py-12.5.0-py3-none-any.whl.metadata (1.3 kB)
Collecting path (from path.py)
  Downloading path-17.1.1-py3-none-any.whl.metadata (6.5 kB)
Downloading path.py-12.5.0-py3-none-any.whl (2.3 kB)
Downloading path-17.1.1-py3-none-any.whl (23 kB)
Installing collected packages: path, path.py
Successfully installed path-17.1.1 path.py-12.5.0
```

In [3]:

```
random.seed = 42
```

Download the [dataset] directly to the Google Colab Runtime. It comprises 10 categories, 3,991 models for training and 908 for testing.

In [4]:

```
!wget http://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zi
```

```
--2025-11-14 11:20:35--  http://3dvision.princeton.edu/projects/2014/3DShapeNe
ts/ModelNet10.zip
Resolving 3dvision.princeton.edu (3dvision.princeton.edu)... 128.112.136.67
Connecting to 3dvision.princeton.edu (3dvision.princeton.edu)|128.112.136.67|:
80... connected.
```

```
HTTP request sent, awaiting response... 302 Found
Location: https://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.
zip [following]
--2025-11-14 11:20:36--  https://3dvision.princeton.edu/projects/2014/3DShapeN
ets/ModelNet10.zip
Connecting to 3dvision.princeton.edu (3dvision.princeton.edu)|128.112.136.67|:
443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 473402300 (451M) [application/zip]
Saving to: ‘ModelNet10.zip’

ModelNet10.zip      100%[===================>] 451.47M  5.52MB/s    in 2m 20s

2025-11-14 11:22:57 (3.22 MB/s) - ‘ModelNet10.zip’ saved [473402300/473402300]
```

In [5]:

```
!unzip -q ModelNet10.zip;
```

In [6]:

```
path = Path("ModelNet10")
```

In [7]:

```
folders = [dir for dir in sorted(os.listdir(path)) if os.path.isdir(path/di
classes = {folder: i for i, folder in enumerate(folders)};
classes
```

Out[7]:

```
{'bathtub': 0,
 'bed': 1,
 'chair': 2,
 'desk': 3,
 'dresser': 4,
 'monitor': 5,
 'night_stand': 6,
 'sofa': 7,
 'table': 8,
 'toilet': 9}
```

This dataset consists of **.off** files that contain meshes represented by *vertices* and *triangular faces*.

We will need a function to read this type of files:

In [8]:

```
def read_off(file):
    if 'OFF' != file.readline().strip():
        raise('Not a valid OFF header')
    n_verts, n_faces, __ = tuple([int(s) for s in file.readline().strip().s
    verts = [[float(s) for s in file.readline().strip().split(' ')] for i_v
    faces = [[int(s) for s in file.readline().strip().split(' ')][1:] for i
    return verts, faces
```

In [9]:

```
with open(path/"bed/train/bed_0001.off", 'r') as f:
  verts, faces = read_off(f)
```

In [10]:
```
i,j,k = np.array(faces).T
x,y,z = np.array(verts).T
```

In [11]:
```
len(x)
```

Out[11]: 2095

Don't be scared of this function. It's just to display animated rotation of meshes and point clouds.

In [12]:
```
def visualize_rotate(data):
    x_eye, y_eye, z_eye = 1.25, 1.25, 0.8
    frames=[]

    def rotate_z(x, y, z, theta):
        w = x+1j*y
        return np.real(np.exp(1j*theta)*w), np.imag(np.exp(1j*theta)*w), z

    for t in np.arange(0, 10.26, 0.1):
        xe, ye, ze = rotate_z(x_eye, y_eye, z_eye, -t)
        frames.append(dict(layout=dict(scene=dict(camera=dict(eye=dict(x=xe
    fig = go.Figure(data=data,
                    layout=go.Layout(
                        updatemenus=[dict(type='buttons',
                                    showactive=False,
                                    y=1,
                                    x=0.8,
                                    xanchor='left',
                                    yanchor='bottom',
                                    pad=dict(t=45, r=10),
                                    buttons=[dict(label='Play',
                                                    method='animate',
                                                    args=[None, dict(frame=
                                                                    transit
                                                                    fromcur
                                                                    mode='i
                                                                    )]
                                                    )
                                            ]
                                    )
                                ]
                    ),
                    frames=frames
            )

    return fig
```

In [13]:
```
visualize_rotate([go.Mesh3d(x=x, y=y, z=z, color='lightpink', opacity=0.50,
```

This mesh definitely looks like a bed.

In [14]:
```
visualize_rotate([go.Scatter3d(x=x, y=y, z=z,
                                   mode='markers')]).show()
```

Unfortunately, that's not the case for its vertices. It would be difficult for PointNet to classify point clouds like this one.

First things first, let's write a function to accurately visualize point clouds so we could see vertices better.

In [15]:
```
def pcshow(xs,ys,zs):
    data=[go.Scatter3d(x=xs, y=ys, z=zs,
                                   mode='markers')]
    fig = visualize_rotate(data)
    fig.update_traces(marker=dict(size=2,
                      line=dict(width=2,
                      color='DarkSlateGrey')),
                      selector=dict(mode='markers'))
    fig.show()
```

In [16]:
```
pcshow(x,y,z)
```

Transforms

As we want it to look more like a real bed, let's write a function to sample points on the surface uniformly.

Sample points

In [17]:
```
class PointSampler(object):
    def __init__(self, output_size):
        assert isinstance(output_size, int)
        self.output_size = output_size

    def triangle_area(self, pt1, pt2, pt3):
        side_a = np.linalg.norm(pt1 - pt2)
        side_b = np.linalg.norm(pt2 - pt3)
        side_c = np.linalg.norm(pt3 - pt1)
        s = 0.5 * ( side_a + side_b + side_c)
        return max(s * (s - side_a) * (s - side_b) * (s - side_c), 0)**0.5

    def sample_point(self, pt1, pt2, pt3):
        # barycentric coordinates on a triangle
        # https://mathworld.wolfram.com/BarycentricCoordinates.html
        s, t = sorted([random.random(), random.random()])
        f = lambda i: s * pt1[i] + (t-s)*pt2[i] + (1-t)*pt3[i]
```

```
        return (f(0), f(1), f(2))


    def __call__(self, mesh):
        verts, faces = mesh
        verts = np.array(verts)
        areas = np.zeros((len(faces)))

        for i in range(len(areas)):
            areas[i] = (self.triangle_area(verts[faces[i][0]],
                                           verts[faces[i][1]],
                                           verts[faces[i][2]]))

        sampled_faces = (random.choices(faces,
                                      weights=areas,
                                      cum_weights=None,
                                      k=self.output_size))

        sampled_points = np.zeros((self.output_size, 3))

        for i in range(len(sampled_faces)):
            sampled_points[i] = (self.sample_point(verts[sampled_faces[i][0
                                                  verts[sampled_faces[i][1
                                                  verts[sampled_faces[i][2

        return sampled_points
```

In [18]:

```
pointcloud = PointSampler(10000)((verts, faces))
```

In [19]:

```
pcshow(*pointcloud.T)
```

This pointcloud looks much more like a bed!

Normalize

Unit sphere

In [20]:

```
class Normalize(object):
    def __call__(self, pointcloud):
        assert len(pointcloud.shape)==2

        norm_pointcloud = pointcloud - np.mean(pointcloud, axis=0)
        norm_pointcloud /= np.max(np.linalg.norm(norm_pointcloud, axis=1))

        return  norm_pointcloud
```

In [21]:

```
norm_pointcloud = Normalize()(pointcloud)
```

In [22]:
```
pcshow(*norm_pointcloud.T)
```

Notice that axis limits have changed.

Augmentations

Let's add *random rotation* of the whole pointcloud and random noise to its points.

In [23]:
```
class RandRotation_z(object):
    def __call__(self, pointcloud):
        assert len(pointcloud.shape)==2

        theta = random.random() * 2. * math.pi
        rot_matrix = np.array([[ math.cos(theta), -math.sin(theta),    0],
                               [ math.sin(theta),  math.cos(theta),    0],
                               [0,                             0,      1]])

        rot_pointcloud = rot_matrix.dot(pointcloud.T).T
        return  rot_pointcloud

class RandomNoise(object):
    def __call__(self, pointcloud):
        assert len(pointcloud.shape)==2

        noise = np.random.normal(0, 0.02, (pointcloud.shape))

        noisy_pointcloud = pointcloud + noise
        return  noisy_pointcloud
```

In [24]:
```
rot_pointcloud = RandRotation_z()(norm_pointcloud)
noisy_rot_pointcloud = RandomNoise()(rot_pointcloud)
```

In [25]:
```
pcshow(*noisy_rot_pointcloud.T)
```

ToTensor

In [26]:
```
class ToTensor(object):
    def __call__(self, pointcloud):
        assert len(pointcloud.shape)==2

        return torch.from_numpy(pointcloud)
```

In [27]:
```
ToTensor()(noisy_rot_pointcloud)
```

Out[27]:
```
tensor([[ 0.9297, -0.1702, -0.0944],
        [-0.5751, -0.0647,  0.0784],
        [ 0.3049,  0.3057,  0.0485],
        ...,
        [ 0.4874,  0.1920, -0.1126],
```

```
        [ 0.1793, -0.2474, -0.1077],
        [ 0.0504,  0.6125, -0.0805]], dtype=torch.float64)
```

In [28]:

```
def default_transforms():
    return transforms.Compose([
                                PointSampler(1024),
                                Normalize(),
                                ToTensor()
                              ])
```

Dataset

Now we can create a custom PyTorch Dataset

In [29]:

```
class PointCloudData(Dataset):
    def __init__(self, root_dir, valid=False, folder="train", transform=def
        self.root_dir = root_dir
        folders = [dir for dir in sorted(os.listdir(root_dir)) if os.path.i
        self.classes = {folder: i for i, folder in enumerate(folders)}
        self.transforms = transform if not valid else default_transforms()
        self.valid = valid
        self.files = []
        for category in self.classes.keys():
            new_dir = root_dir/Path(category)/folder
            for file in os.listdir(new_dir):
                if file.endswith('.off'):
                    sample = {}
                    sample['pcd_path'] = new_dir/file
                    sample['category'] = category
                    self.files.append(sample)

    def __len__(self):
        return len(self.files)

    def __preproc__(self, file):
        verts, faces = read_off(file)
        if self.transforms:
            pointcloud = self.transforms((verts, faces))
        return pointcloud

    def __getitem__(self, idx):
        pcd_path = self.files[idx]['pcd_path']
        category = self.files[idx]['category']
        with open(pcd_path, 'r') as f:
            pointcloud = self.__preproc__(f)
        return {'pointcloud': pointcloud,
                'category': self.classes[category]}
```

Transforms for training. 1024 points per cloud as in the paper!

In [30]:

```
train_transforms = transforms.Compose([
                    PointSampler(1024),
```

```
                    Normalize(),
                    RandRotation_z(),
                    RandomNoise(),
                    ToTensor()
                    ])
```

In [31]:

```
train_ds = PointCloudData(path, transform=train_transforms)
valid_ds = PointCloudData(path, valid=True, folder='test', transform=train_
```

In [32]:

```
inv_classes = {i: cat for cat, i in train_ds.classes.items()};
inv_classes
```

Out[32]:

```
{0: 'bathtub',
 1: 'bed',
 2: 'chair',
 3: 'desk',
 4: 'dresser',
 5: 'monitor',
 6: 'night_stand',
 7: 'sofa',
 8: 'table',
 9: 'toilet'}
```

In [33]:

```
print('Train dataset size: ', len(train_ds))
print('Valid dataset size: ', len(valid_ds))
print('Number of classes: ', len(train_ds.classes))
print('Sample pointcloud shape: ', train_ds[0]['pointcloud'].size())
print('Class: ', inv_classes[train_ds[0]['category']])
```

```
Train dataset size:  3991
Valid dataset size:  908
Number of classes:  10
Sample pointcloud shape:  torch.Size([1024, 3])
Class:  bathtub
```

In [34]:

```
train_loader = DataLoader(dataset=train_ds, batch_size=32, shuffle=True)
valid_loader = DataLoader(dataset=valid_ds, batch_size=64)
```

Model

In [35]:

```
import torch
import torch.nn as nn
import numpy as np
import torch.nn.functional as F

class Tnet(nn.Module):
   def __init__(self, k=3):
      super().__init__()
      self.k=k
      self.conv1 = nn.Conv1d(k,64,1)
      self.conv2 = nn.Conv1d(64,128,1)
```

```
        self.conv3 = nn.Conv1d(128,1024,1)
        self.fc1 = nn.Linear(1024,512)
        self.fc2 = nn.Linear(512,256)
        self.fc3 = nn.Linear(256,k*k)

        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)
        self.bn4 = nn.BatchNorm1d(512)
        self.bn5 = nn.BatchNorm1d(256)


    def forward(self, input):
        # input.shape == (bs,n,3)
        bs = input.size(0)
        xb = F.relu(self.bn1(self.conv1(input)))
        xb = F.relu(self.bn2(self.conv2(xb)))
        xb = F.relu(self.bn3(self.conv3(xb)))
        pool = nn.MaxPool1d(xb.size(-1))(xb)
        flat = nn.Flatten(1)(pool)
        xb = F.relu(self.bn4(self.fc1(flat)))
        xb = F.relu(self.bn5(self.fc2(xb)))

        #initialize as identity
        init = torch.eye(self.k, requires_grad=True).repeat(bs,1,1)
        if xb.is_cuda:
          init=init.cuda()
        matrix = self.fc3(xb).view(-1,self.k,self.k) + init
        return matrix


class Transform(nn.Module):
    def __init__(self):
          super().__init__()
          self.input_transform = Tnet(k=3)
          self.feature_transform = Tnet(k=64)
          self.conv1 = nn.Conv1d(3,64,1)

          self.conv2 = nn.Conv1d(64,128,1)
          self.conv3 = nn.Conv1d(128,1024,1)


          self.bn1 = nn.BatchNorm1d(64)
          self.bn2 = nn.BatchNorm1d(128)
          self.bn3 = nn.BatchNorm1d(1024)

    def forward(self, input):
          matrix3x3 = self.input_transform(input)
          # batch matrix multiplication
          xb = torch.bmm(torch.transpose(input,1,2), matrix3x3).transpose(1,2

          xb = F.relu(self.bn1(self.conv1(xb)))

          matrix64x64 = self.feature_transform(xb)
          xb = torch.bmm(torch.transpose(xb,1,2), matrix64x64).transpose(1,2)

          xb = F.relu(self.bn2(self.conv2(xb)))
          xb = self.bn3(self.conv3(xb))
```

```
        xb = nn.MaxPool1d(xb.size(-1))(xb)
        output = nn.Flatten(1)(xb)
        return output, matrix3x3, matrix64x64

class PointNet(nn.Module):
    def __init__(self, classes = 10):
        super().__init__()
        self.transform = Transform()
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, classes)


        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.dropout = nn.Dropout(p=0.3)
        self.logsoftmax = nn.LogSoftmax(dim=1)

    def forward(self, input):
        xb, matrix3x3, matrix64x64 = self.transform(input)
        xb = F.relu(self.bn1(self.fc1(xb)))
        xb = F.relu(self.bn2(self.dropout(self.fc2(xb))))
        output = self.fc3(xb)
        return self.logsoftmax(output), matrix3x3, matrix64x64
```

In [36]:

```
def pointnetloss(outputs, labels, m3x3, m64x64, alpha = 0.0001):
    criterion = torch.nn.NLLLoss()
    bs=outputs.size(0)
    id3x3 = torch.eye(3, requires_grad=True).repeat(bs,1,1)
    id64x64 = torch.eye(64, requires_grad=True).repeat(bs,1,1)
    if outputs.is_cuda:
        id3x3=id3x3.cuda()
        id64x64=id64x64.cuda()
    diff3x3 = id3x3-torch.bmm(m3x3,m3x3.transpose(1,2))
    diff64x64 = id64x64-torch.bmm(m64x64,m64x64.transpose(1,2))
    return criterion(outputs, labels) + alpha * (torch.norm(diff3x3)+torch.
```

Training loop

You can find a pretrained model [here]()

In [37]:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

In [38]:

```
pointnet = PointNet()
pointnet.to(device);
```

In [39]:

```
optimizer = torch.optim.Adam(pointnet.parameters(), lr=0.001)
```

In [40]:

```
def train(model, train_loader, val_loader=None,  epochs=3, save=True):
    for epoch in range(epochs):
        pointnet.train()
        running_loss = 0.0
        for i, data in enumerate(train_loader, 0):
            inputs, labels = data['pointcloud'].to(device).float(), data['c
            optimizer.zero_grad()
            outputs, m3x3, m64x64 = pointnet(inputs.transpose(1,2))

            loss = pointnetloss(outputs, labels, m3x3, m64x64)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 10 == 9:    # print every 10 mini-batches
                    print('[Epoch: %d, Batch: %4d / %4d], loss: %.3f' %
                        (epoch + 1, i + 1, len(train_loader), running_loss
                    running_loss = 0.0

        pointnet.eval()
        correct = total = 0

        # validation
        if val_loader:
            with torch.no_grad():
                for data in val_loader:
                    inputs, labels = data['pointcloud'].to(device).float(),
                    outputs, __, __ = pointnet(inputs.transpose(1,2))
                    _, predicted = torch.max(outputs.data, 1)
                    total += labels.size(0)
                    correct += (predicted == labels).sum().item()
            val_acc = 100. * correct / total
            print('Valid accuracy: %d %%' % val_acc)

        # save the model
        if save:
            torch.save(pointnet.state_dict(), "save_"+str(epoch)+".pth")
```

In [41]:

```
train(pointnet, train_loader, valid_loader,  save=False)
```

```
[Epoch: 1, Batch:   10 /  125], loss: 2.087
[Epoch: 1, Batch:   20 /  125], loss: 1.502
[Epoch: 1, Batch:   30 /  125], loss: 1.363
[Epoch: 1, Batch:   40 /  125], loss: 1.283
[Epoch: 1, Batch:   50 /  125], loss: 1.131
[Epoch: 1, Batch:   60 /  125], loss: 1.289
[Epoch: 1, Batch:   70 /  125], loss: 1.182
[Epoch: 1, Batch:   80 /  125], loss: 1.111
[Epoch: 1, Batch:   90 /  125], loss: 1.070
[Epoch: 1, Batch:  100 /  125], loss: 1.096
[Epoch: 1, Batch:  110 /  125], loss: 0.943
[Epoch: 1, Batch:  120 /  125], loss: 1.011
```

```
Valid accuracy: 70 %
[Epoch: 2, Batch:   10 /  125], loss: 0.895
[Epoch: 2, Batch:   20 /  125], loss: 0.958
[Epoch: 2, Batch:   30 /  125], loss: 0.792
[Epoch: 2, Batch:   40 /  125], loss: 0.764
[Epoch: 2, Batch:   50 /  125], loss: 0.820
[Epoch: 2, Batch:   60 /  125], loss: 0.847
[Epoch: 2, Batch:   70 /  125], loss: 0.864
[Epoch: 2, Batch:   80 /  125], loss: 0.728
[Epoch: 2, Batch:   90 /  125], loss: 0.691
[Epoch: 2, Batch:  100 /  125], loss: 0.707
[Epoch: 2, Batch:  110 /  125], loss: 0.661
[Epoch: 2, Batch:  120 /  125], loss: 0.642
Valid accuracy: 67 %
[Epoch: 3, Batch:   10 /  125], loss: 0.643
[Epoch: 3, Batch:   20 /  125], loss: 0.674
[Epoch: 3, Batch:   30 /  125], loss: 0.606
[Epoch: 3, Batch:   40 /  125], loss: 0.541
[Epoch: 3, Batch:   50 /  125], loss: 0.696
[Epoch: 3, Batch:   60 /  125], loss: 0.788
[Epoch: 3, Batch:   70 /  125], loss: 0.636
[Epoch: 3, Batch:   80 /  125], loss: 0.650
[Epoch: 3, Batch:   90 /  125], loss: 0.625
[Epoch: 3, Batch:  100 /  125], loss: 0.583
[Epoch: 3, Batch:  110 /  125], loss: 0.698
[Epoch: 3, Batch:  120 /  125], loss: 0.639
Valid accuracy: 78 %
```

In [45]:

```python
torch.save(pointnet.state_dict(), "save_" +".pth")
```

Test

In [42]:

```python
from sklearn.metrics import confusion_matrix
```

In [46]:

```python
pointnet = PointNet()
pointnet.load_state_dict(torch.load('save_.pth'))
pointnet.eval();
```

In [47]:

```python
all_preds = []
all_labels = []
with torch.no_grad():
    for i, data in enumerate(valid_loader):
        print('Batch [%4d / %4d]' % (i+1, len(valid_loader)))

        inputs, labels = data['pointcloud'].float(), data['category']
        outputs, __, __ = pointnet(inputs.transpose(1,2))
        _, preds = torch.max(outputs.data, 1)
        all_preds += list(preds.numpy())
        all_labels += list(labels.numpy())
```

```
Batch [   1 /   15]
Batch [   2 /   15]
Batch [   3 /   15]
```

```
Batch [   3 /   15]
Batch [   4 /   15]
Batch [   5 /   15]
Batch [   6 /   15]
Batch [   7 /   15]
Batch [   8 /   15]
Batch [   9 /   15]
Batch [  10 /   15]
Batch [  11 /   15]
Batch [  12 /   15]
Batch [  13 /   15]
Batch [  14 /   15]
Batch [  15 /   15]
```

In [48]:

```
cm = confusion_matrix(all_labels, all_preds);
cm
```

Out[48]:

```
array([[ 50,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [100,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [100,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [ 86,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [ 86,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [100,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [ 86,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [100,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [100,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [100,   0,   0,   0,   0,   0,   0,   0,   0,   0]])
```

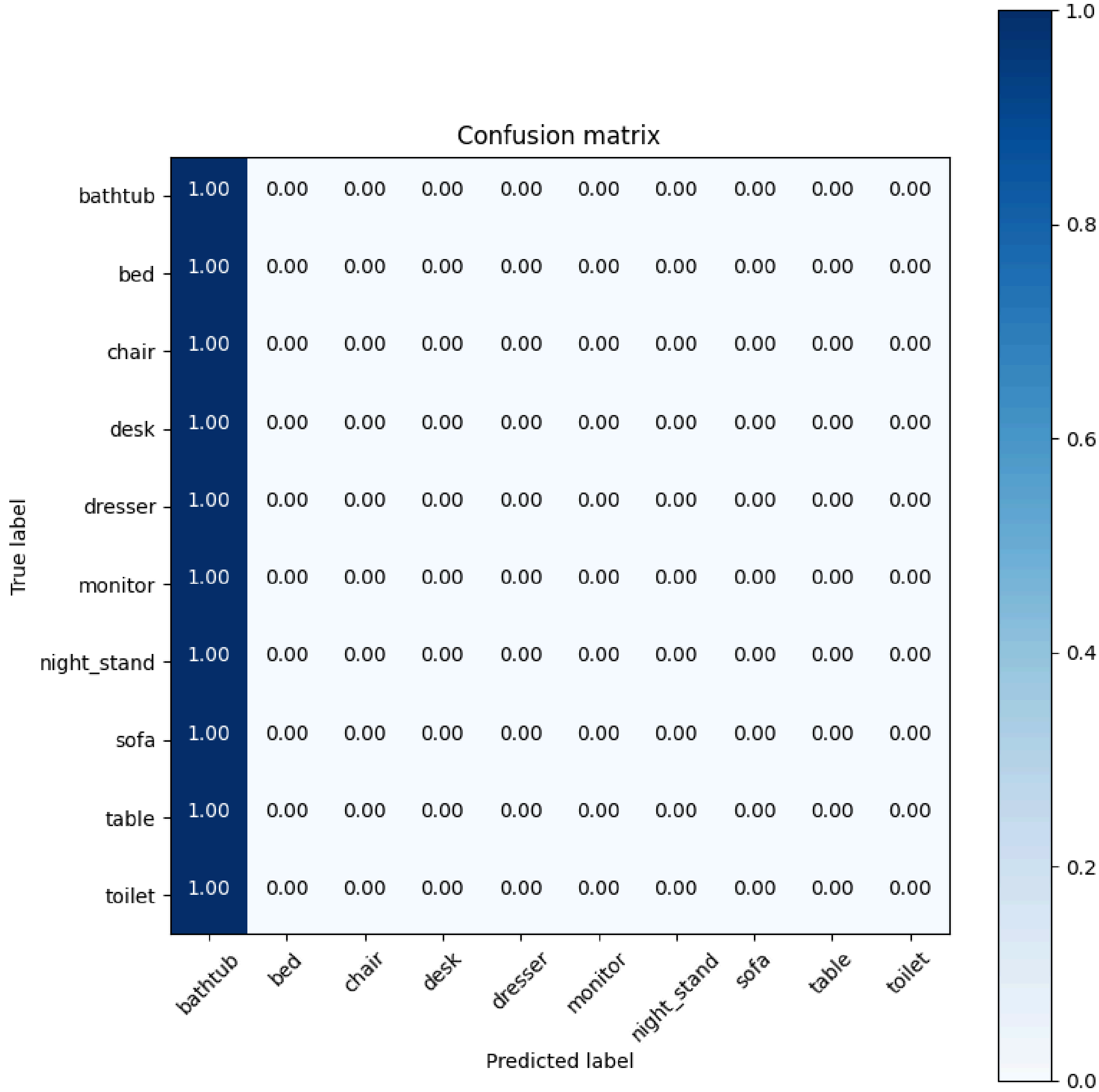
In [49]:

```
import itertools
import numpy as np
import matplotlib.pyplot as plt

# function from https://deeplizard.com/learn/video/0LhiS6yu2qQ
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion ma
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center",

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [50]:

```
plt.figure(figsize=(8,8))
plot_confusion_matrix(cm, list(classes.keys()), normalize=True)
```

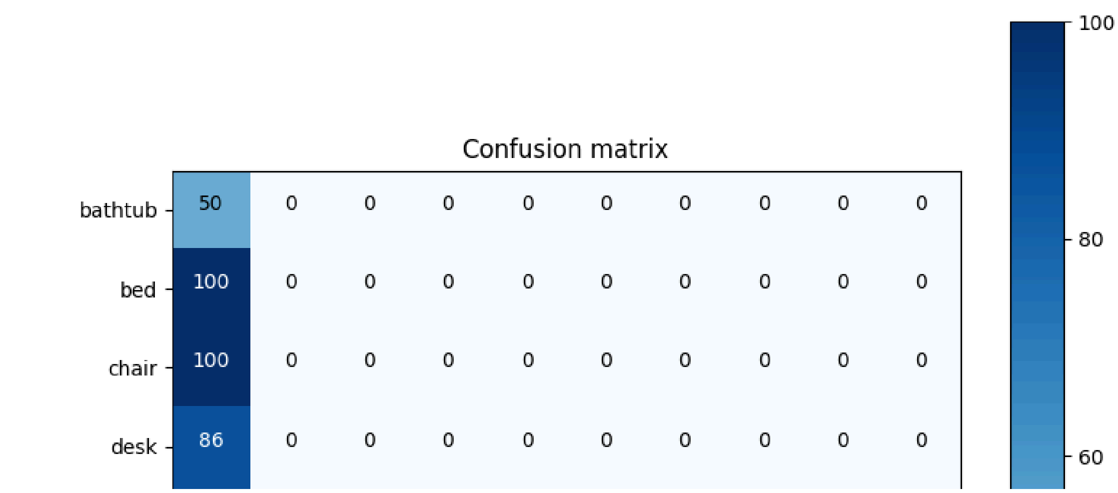
```
Normalized confusion matrix
```

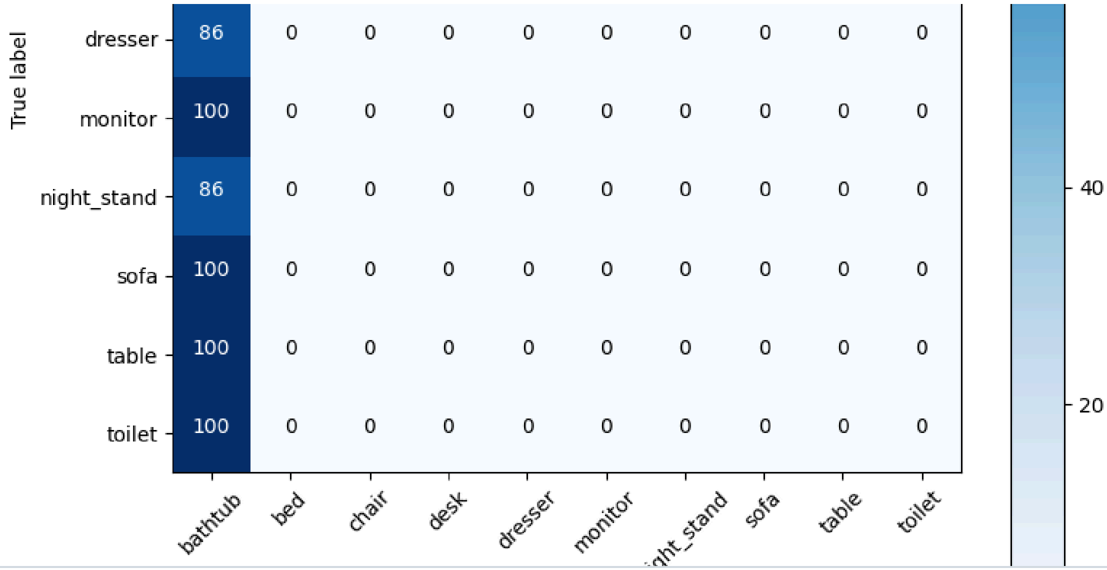


In [51]:

```
plt.figure(figsize=(8,8))
plot_confusion_matrix(cm, list(classes.keys()), normalize=False)
```

```
Confusion matrix, without normalization
```




True label
dresser
monitor
night_stand
sofa
table
toilet
86 0 0 0 0 0 0 0 0 0
100 0 0 0 0 0 0 0 0 0
86 0 0 0 0 0 0 0 0 0
100 0 0 0 0 0 0 0 0 0
100 0 0 0 0 0 0 0 0 0
100 0 0 0 0 0 0 0 0 0
bathtub
bed
chair
desk
dresser
monitor
sofa
table
toilet
40
20