Deep Learning with Point Clouds

Romain Thalineau

qwertee.io

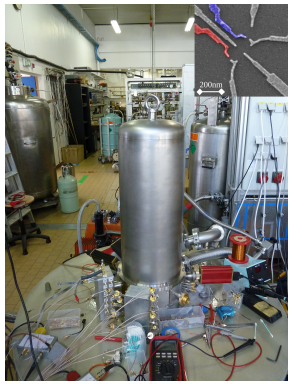
romain.thalineau@qwertee.io

 @romaintha

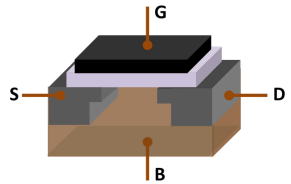
PyData Cluj-Napoca, February 20, 2019

About

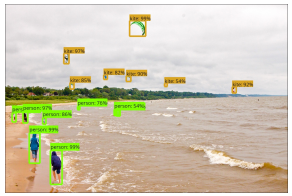
- Ph.D. in Physics (Grenoble, France), working on single electron spin qubits



- Worked at Infineon (Munich, Germany) as a data scientist.

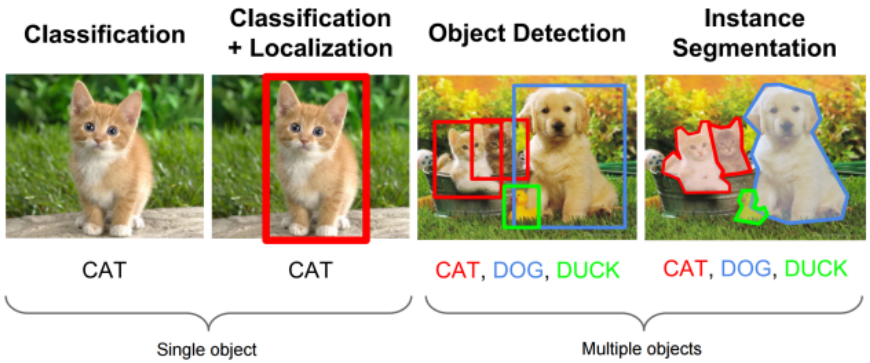


- Living in Cluj since 2016 and now working as a machine learning engineer & data scientist at Qwertee.



From 2D to 3D perception

- Impressive developments over the last decade in the field of 2D vision.



Credit: Kdnuggets

- Advanced robotic systems require visual perception capabilities beyond 2D images.

Overview

Overview

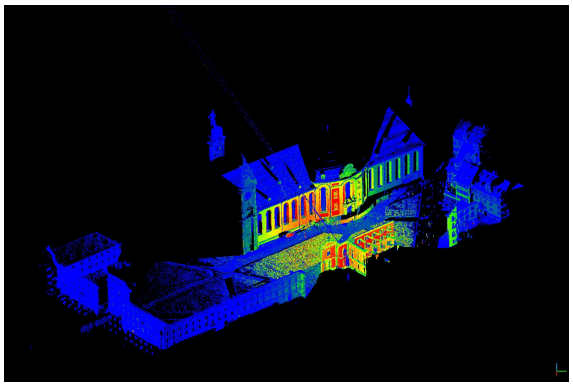
Point Clouds

- A point cloud is just a set of points. E.g.:

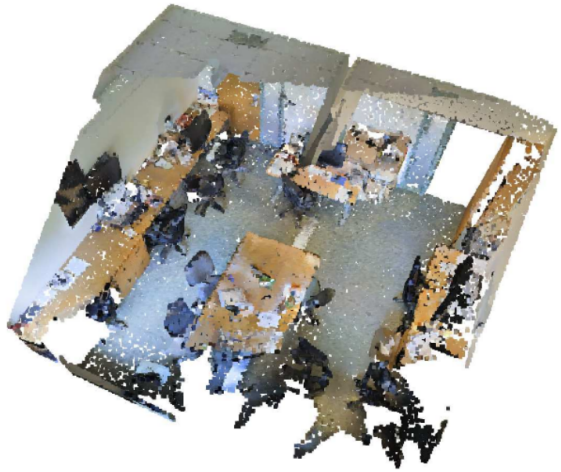
x	y	z
0.07388	0.16975	-0.19326
0.15286	0.15050	0.24355
0.20948	0.15050	0.29081
...	...	...

- By nature, point clouds are **irregular** (density) and **unordered** $\Rightarrow$ invariant to permutations
- A point cloud may also hold additional features such as RGB or normals.

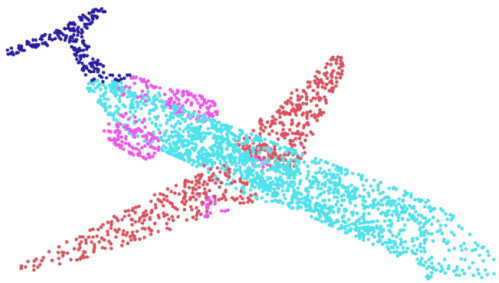
Example of point clouds



Semantic3D dataset



S3DIS dataset



ShapeNet dataset

Visualizing point clouds

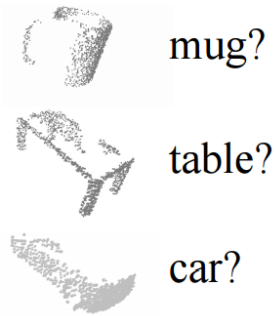
- Visualize point clouds with Python

```
1 import open3d
2
3 point_cloud = open3d.read_point_cloud('pc_file.txt')
4 open3d.draw_geometries([point_cloud])
```

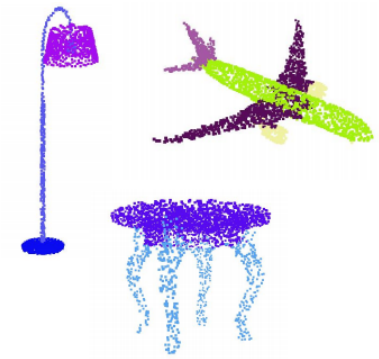
- Use an OSS such as CloudCompare

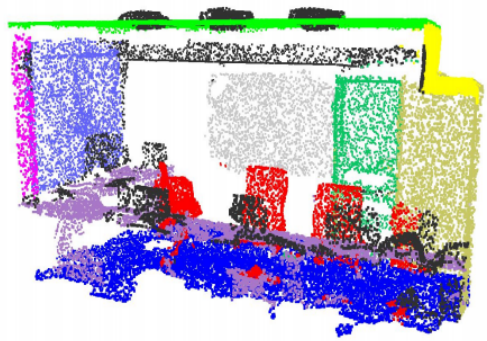
Learning tasks

Classification[1]



Scene/object semantic segmentation[1]

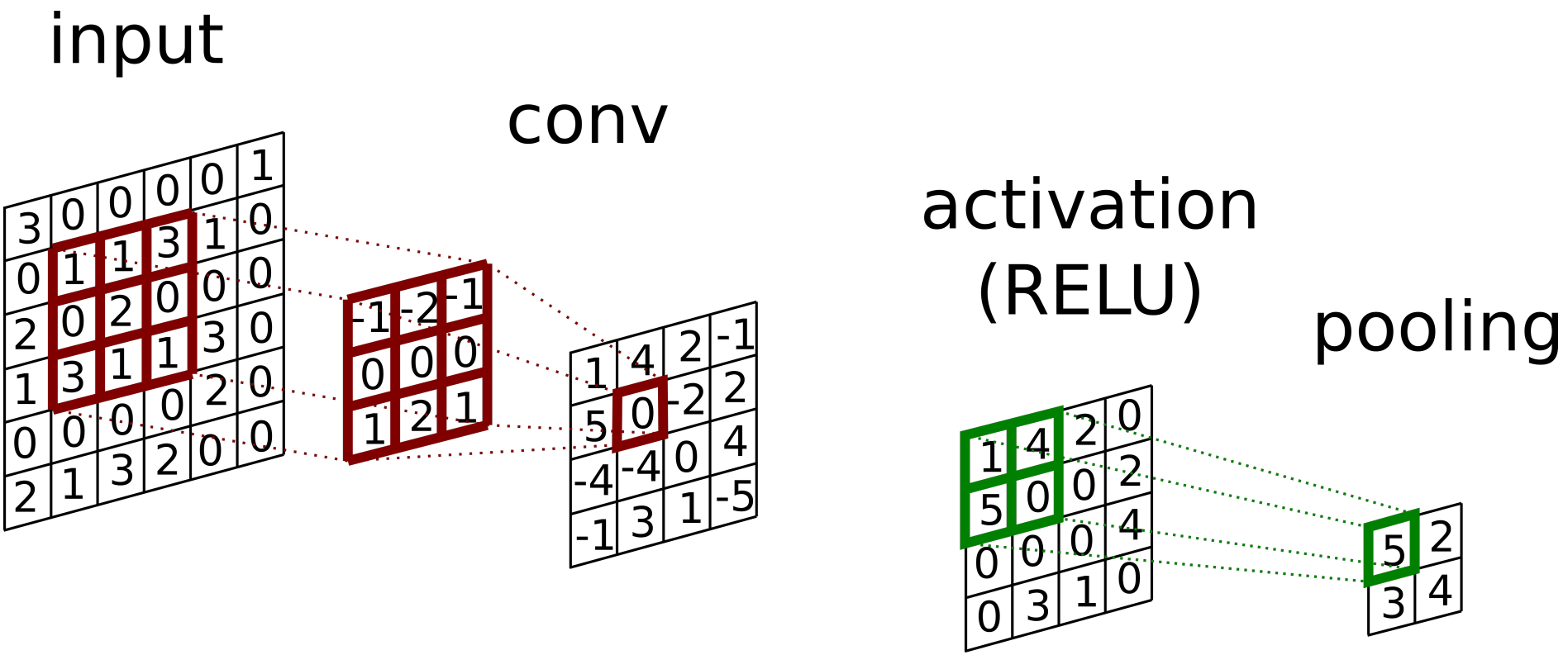




[1] Source: PointNet

Overview

Learning features from regular data



- Use convolution to leverage 3 important idea: **sparse connectivity**, **parameter sharing**, **equivariance to translation**
- Use max pooling for $\sim$ **invariance to translation**.

Learning features from regular data

- Stacking simple blocks (CONV→RELU→POOL) allows for learning higher level features [1]

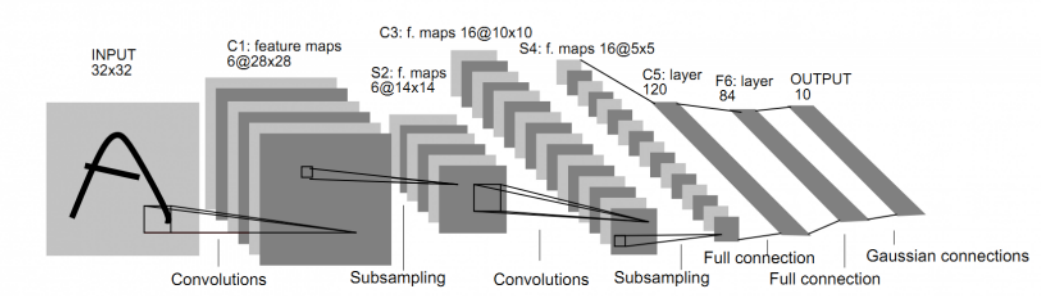
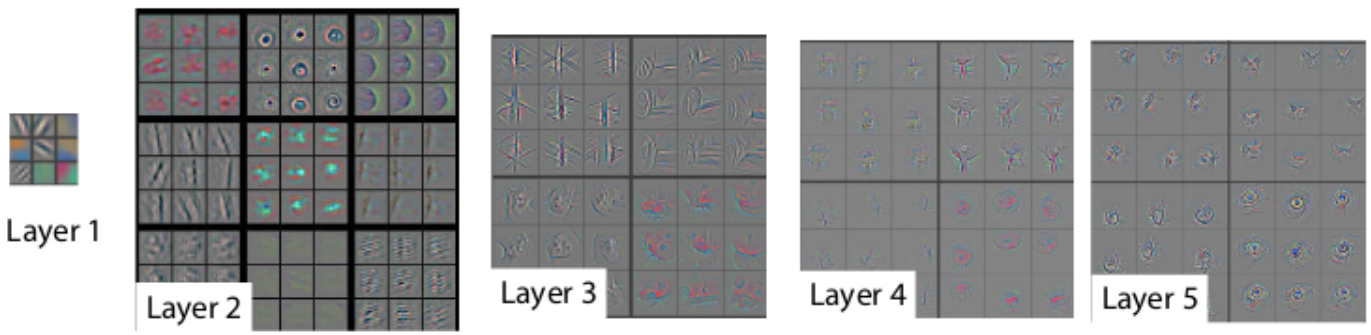


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

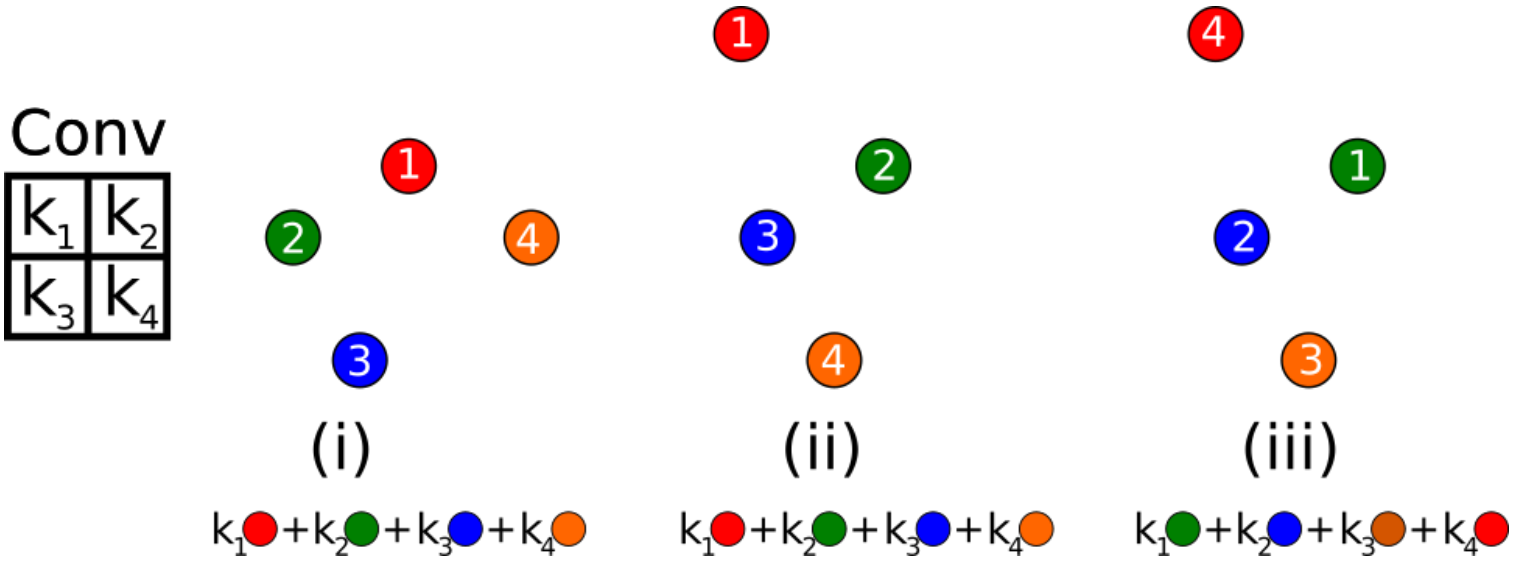
- Visualizing convolutional networks [2]



[1] Gradient based learning applied to document recognition, Lecun et al

[2] Visualizing and Understanding Convolutional Networks, Zeiler & Fergus

Point cloud problem statement



- Applying convolution kernel to point cloud:
  - desertion of shape $conv(i) = conv(ii)$
  - variance to ordering $conv(ii) \neq conv(iii)$
- We need functions f such that :
  - $f(i) \neq f(ii) \rightarrow$ Capturing local structures
  - $f(ii) = f(iii) \rightarrow$ Invariant to permutations
  - Invariant to geometric transofrmations such as rotations or translations of the whole point set .

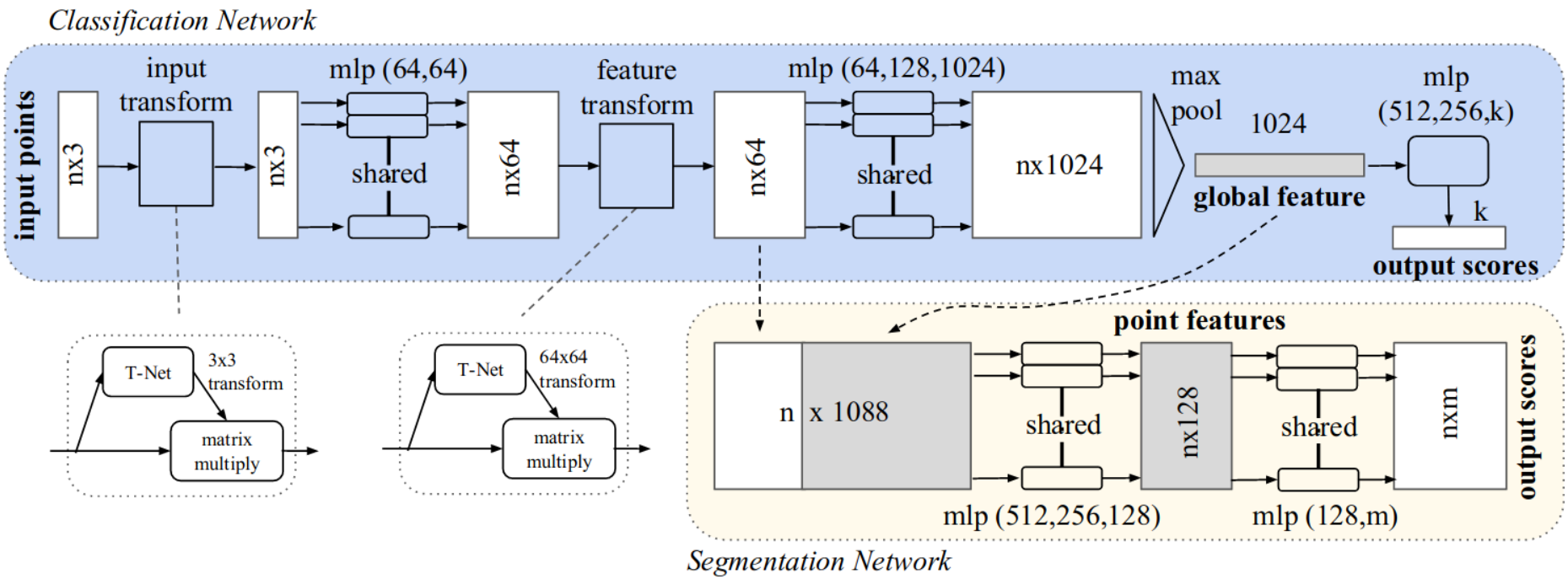
Overview

PointNet proposal

- Permutation invariance: by using a max pooling function. $f(x_1, ..., x_n) = g(h(x_1), ..., h(x_n))$ where:
  - h is a MLP network.
  - g is the max pooling function.
- Invariance to geometric transformations (rotations, translations...): by learning a tranformation network[1]
- By its design, PointNet does not capture well the local structure induced by the distance metric.

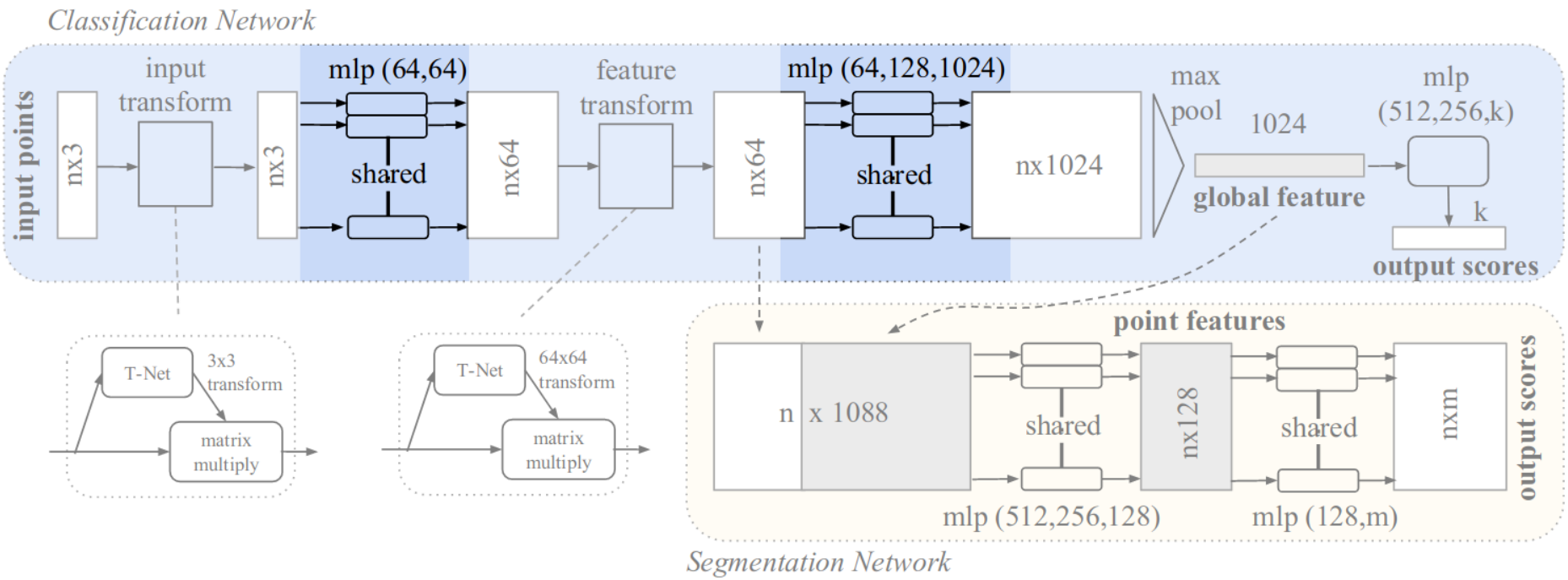
[1] Slightly similar to "Spatial transformer network" by Jaderberg et al

PointNet architecture



Reference: "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", Qi et al

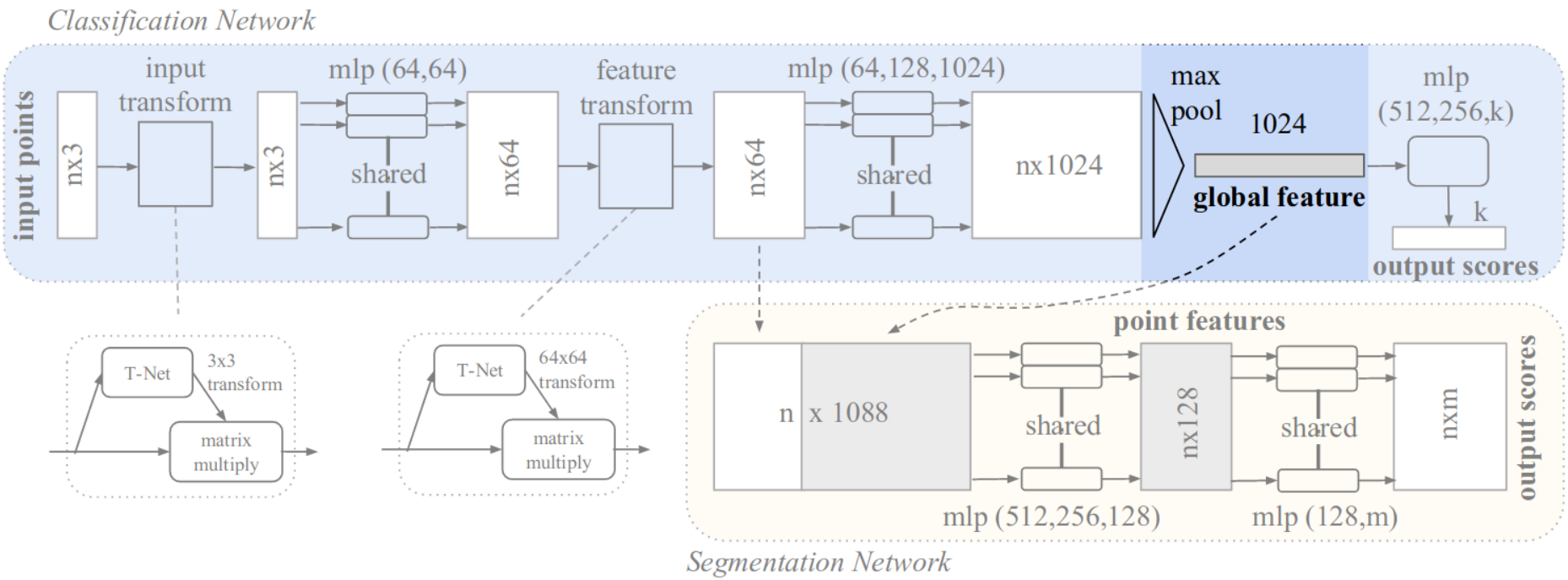
PointNet architecture



Shared MLP to learn spatial encoding of each point.

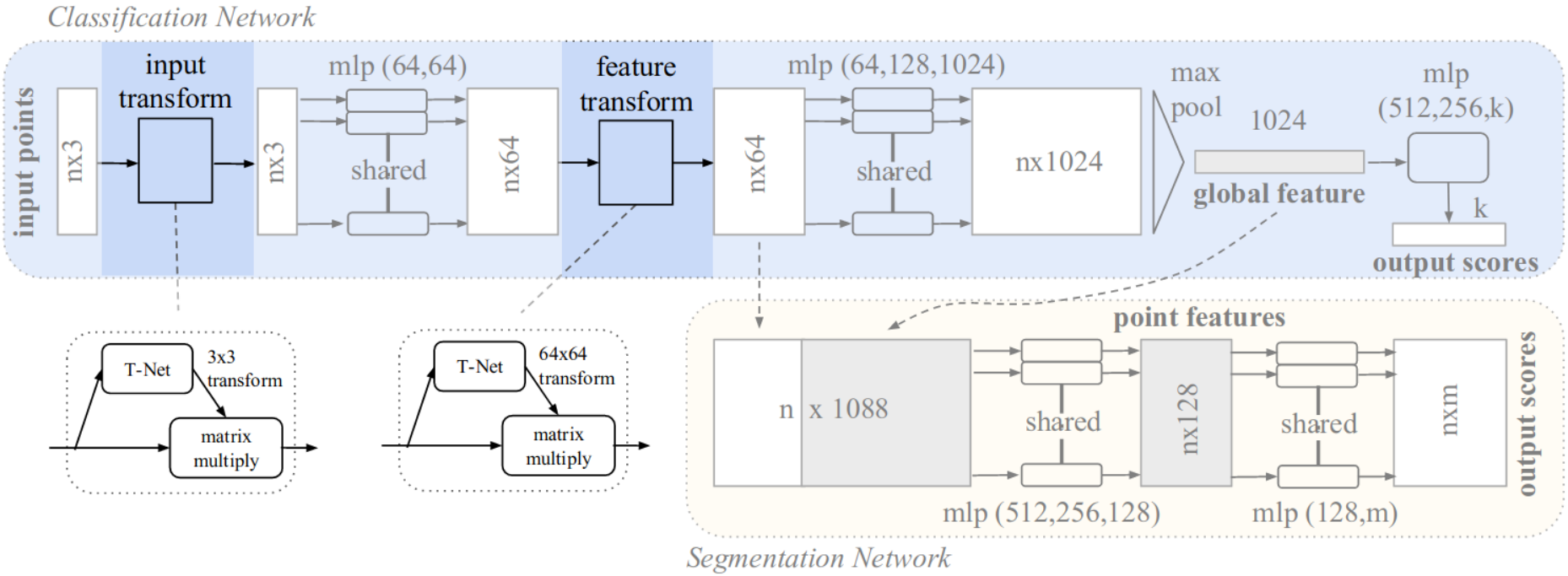
Note: Shared MLP is equivalent to a 1d conv with kernel size of 1

PointNet architecture



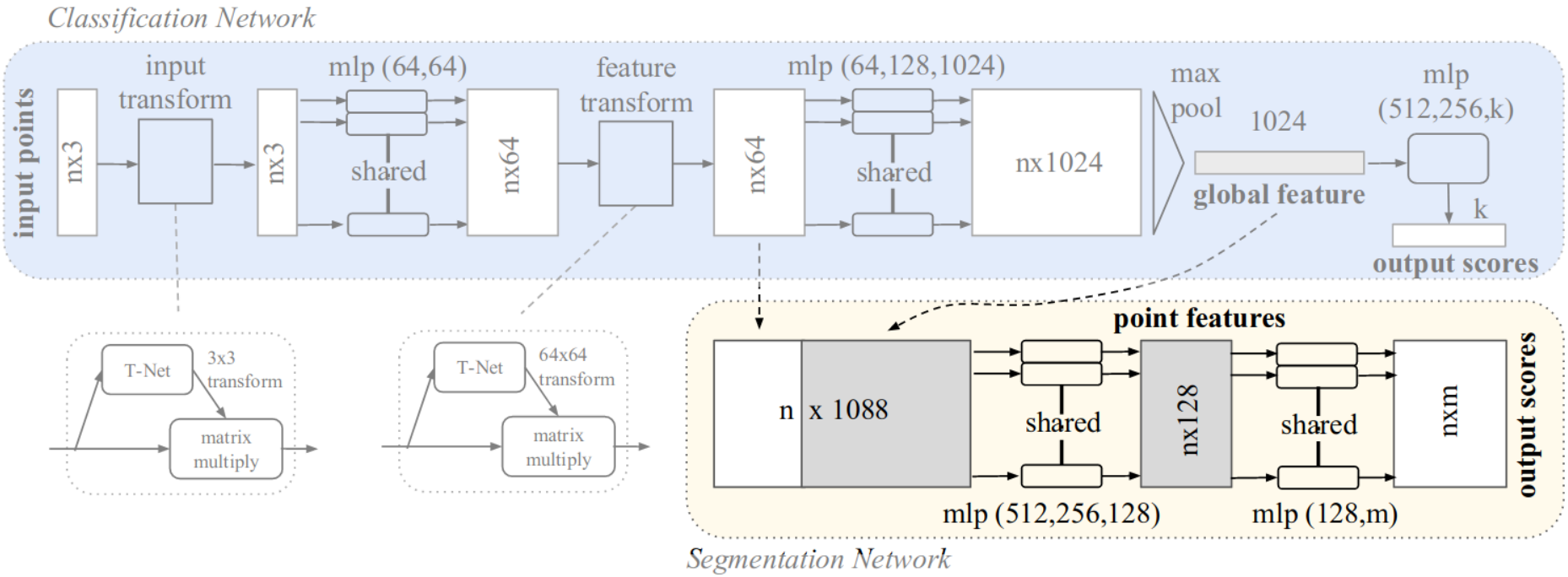
Symmetry function (max pool) in order to make the model invariant to input permutations.

PointNet architecture



Joint alignment network to learn invariance to some geometric transformations. Similar to "Spatial Network Transformer", Jaderberg et al

PointNet architecture



A local and global information combination structure to enable point segmentation learning.

PyTorch 101

- Pytorch tensors similar ($\sim$ API) to numpy ND-arrays, but can be used on GPU.

```
import torch

x = torch.tensor([1, 2])
x = torch.rand(2, 2)
x.shape
y = x + 2
```

- Automatic differentiation

```
x = torch.rand(2, 2)
y = x + 2
z = y.mean()

z.backward()
x.grad
```

PyTorch 101

- Define a network

```
1  import torch
2  import torch.nn as nn
3  import torch.nn.functional as F
4
5  class SimpleCNN(nn.Module):
6
7      def __init__(self):
8          super(SimpleCNN, self).__init__()
9          self.conv = nn.Conv2D(3, 64, 3)
10         self.fc = nn.Linear(64*32*32, 10)
11
12     def forward(self, x):
13         x = self.conv(x)
14         x = F.relu(x)
15         x = F.max_pool2d(x)
16         x = x.view(x.shape[0], -1) # flatten
17         x = self.fc(x)
18         return x
```

PyTorch 101

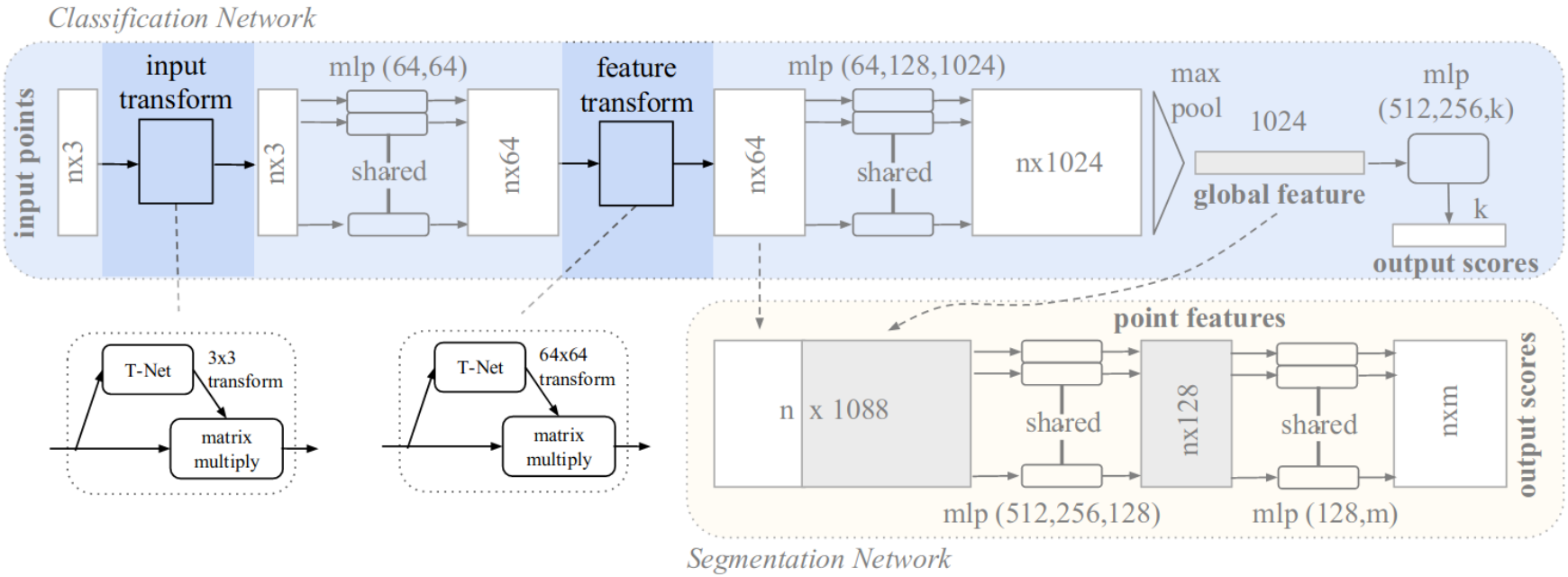
- Train a network

```
import torch.nn as nn
import torch.optim as optim

model = SimpleCNN()
criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=0.01)

for data, target in train_data_generator:
    optimizer.zero_grad()
    model = model.train()
    output = model(data)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()    # Does the update
```

PointNet PyTorch: Transformer

Full implementation on my GitHub account
Warning: The implementation code is simplified for the presentation sake.

- Transformer Network

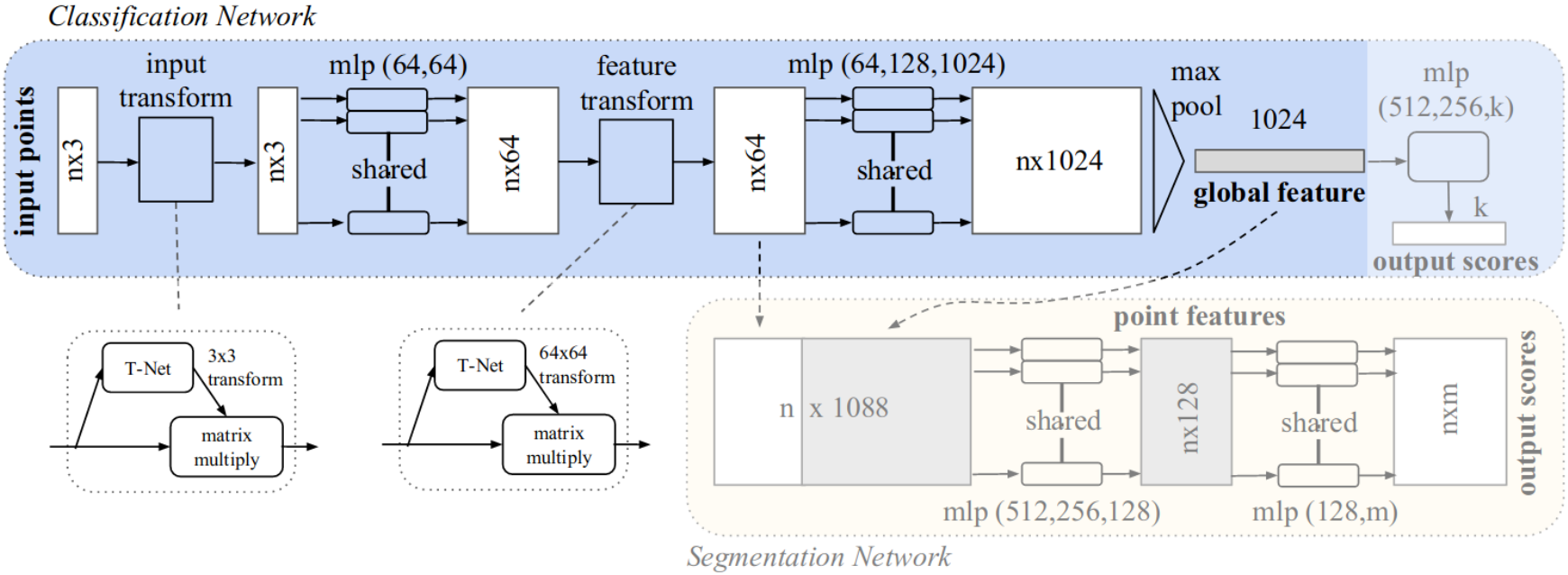


PointNet PyTorch: Transformer

```
class TransformationNet(nn.Module):

    def forward(self, x): # x.shape = (B, N, C)
        num_points = x.shape[1]
        x = x.transpose(2, 1)                          # (B, C, N)
        x = F.relu(nn.Conv1d(C, 64, 1)(x))             # (B, 64, N)
        x = F.relu(nn.Conv1d(64, 128, 1)(x))           # (B, 128, N)
        x = F.relu(nn.Conv1d(128, 1024, 1)(x))         # (B, 1024, N)

        x = nn.MaxPool1d(num_points)(x)                # (B, 1024, 1)
        x = x.view(-1, 1024)                           # (B, 1024)

        x = F.relu(nn.Linear(1024, 512)(x))            # (B, 512)
        x = F.relu(nn.Linear(512, 256)(x))             # (B, 256)
        x = nn.Linear(256, output_dim^2)(x)            # (B, output_dim^2)

        identity_matrix = torch.eye(output_dim)
        x = x.view(-1, output_dim, output_dim)
        return x + identity_matrix # (B, output_dim, output_dim)
```

PointNet PyTorch: Base network

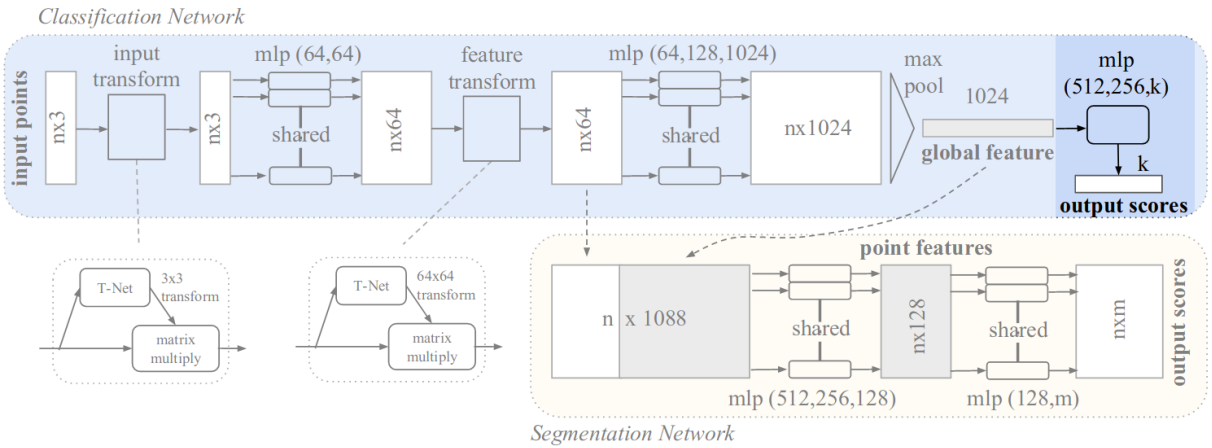
- Base Network



PointNet PyTorch: Base

```python
class BasePointNet(nn.Module):

    def forward(self, x): # x.shape = (B, N, C)
        num_points = x.shape[1]
        input_transform = TransformationNet(C)(x)      # (B, C, C)
        x = torch.bmm(x, input_transform)              # (B, N, C)

        x = x.transpose(2, 1)                          # (B, C, N)
        x = F.relu(nn.Conv1d(C, 64, 1)(x))             # (B, 64, N)
        x = F.relu(nn.Conv1d(64, 64, 1)(x)))           # (B, 64, N)
        x = x.transpose(2, 1))                         # (B, N, 64)

        feature_transform = TransformationNet(64)(x) # (B, 64, 64)
        x = torch.bmm(x, feature_transform)            # (B, N, 64)

        x = x.transpose(2, 1)                          # (B, 64, N)
        x = F.relu(nn.Conv1d(64, 64, 1)(x))            # (B, 64, N)
        x = F.relu(nn.Conv1d(64, 128, 1)(x))           # (B, 128, N)
        x = F.relu(nn.Conv1d(128, 1024, 1)(x))         # (B, 1024, N)
        x = nn.MaxPool1d(num_points)(x)                # (B, 1024, 1)
        x = x.view(-1, 1024)                           # (B, 1024)
        return x                                       # (B, 1024)
```

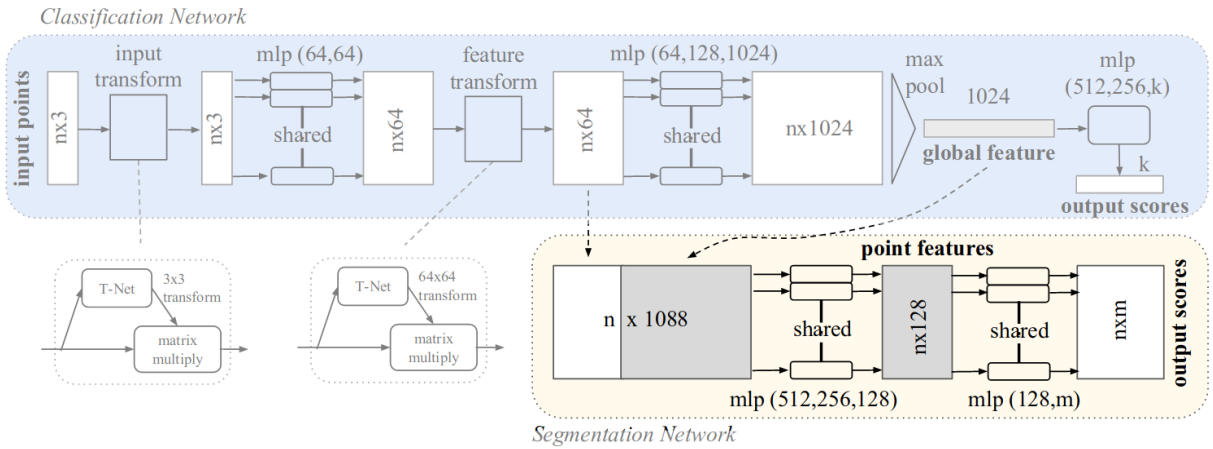
PointNet PyTorch: Classification



Classification Network

```python
class ClassificationPointNet(nn.Module):

    def forward(self, x): # x.shape = (B, N, C)
        x = BasePointNet()(x)                          # (B, 1024)

        x = F.relu(nn.Linear(1024, 512)(x))            # (B, 512)
        x = F.relu(nn.Linear(512, 256)(x))             # (B, 256)
        x = nn.Dropout(0.3)(x)                         # (B, 256)
        x = nn.Linear(256, num_classes)(x)             # (B, num_classes)
        x = F.log_softmax(x, dim=1)                    # (B, num_classes)
        return x                                       # (B, num_classes)
```

PointNet PyTorch: Segmentation

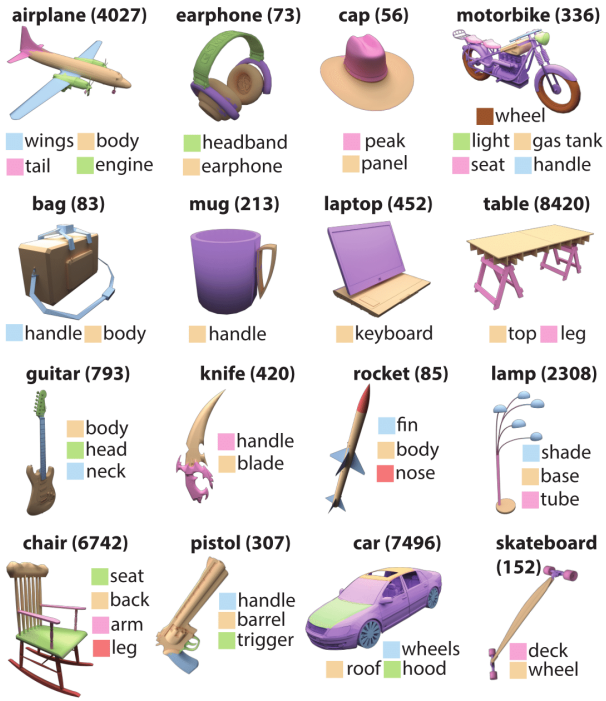


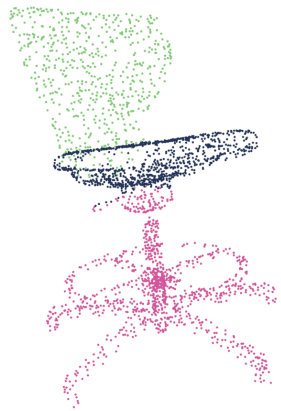
Segmentation Network

```python
class SegmentationPointNet(nn.Module):

    def forward(self, x): # x.shape = (B, N, C)
        x = BasePointNet()(x)                          # (B, N, 1024 + 64)

        x = x.transpose(2, 1)                          # (B, 1024 + 64, N)
        x = F.relu(nn.Conv1d(1088, 512, 1)(x))# (B, 512, N)
        x = F.relu(nn.Conv1d(512, 256, 1)(x)) # (B, 256, N)
        x = F.relu(nn.Conv1d(256, 128, 1)(x)) # (B, 128, N)
        x = nn.Conv1d(128, num_classes, 1)(x) # (B, num_classes, N)
        x = x.transpose(2, 1)                          # (B, N, num_classes)
        x = F.log_softmax(x, dim=1)                    # (B, N, n_classes)
        return x                                       # (B, N, n_classes)
```

Experiment on ShapeNet

ShapeNet dataset[1]: 16 shapes, segmented in parts (up to 6/shape)

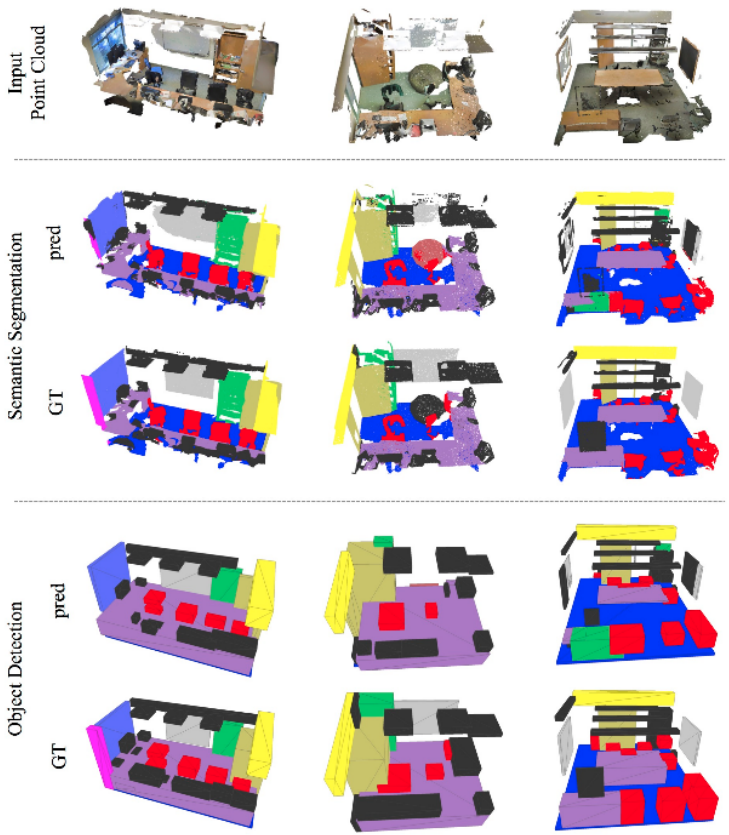




[1] Source: A Scalable Active Framework for Region Annotation in 3D Shape Collections

Others

Semantic scene segmentation



Credit[1]

MNIST classification

	input	error (%)
Multi-layer perceptron [22]	vector	1.60
LeNet5 [12]	image	0.80
Ours PointNet	point set	0.78

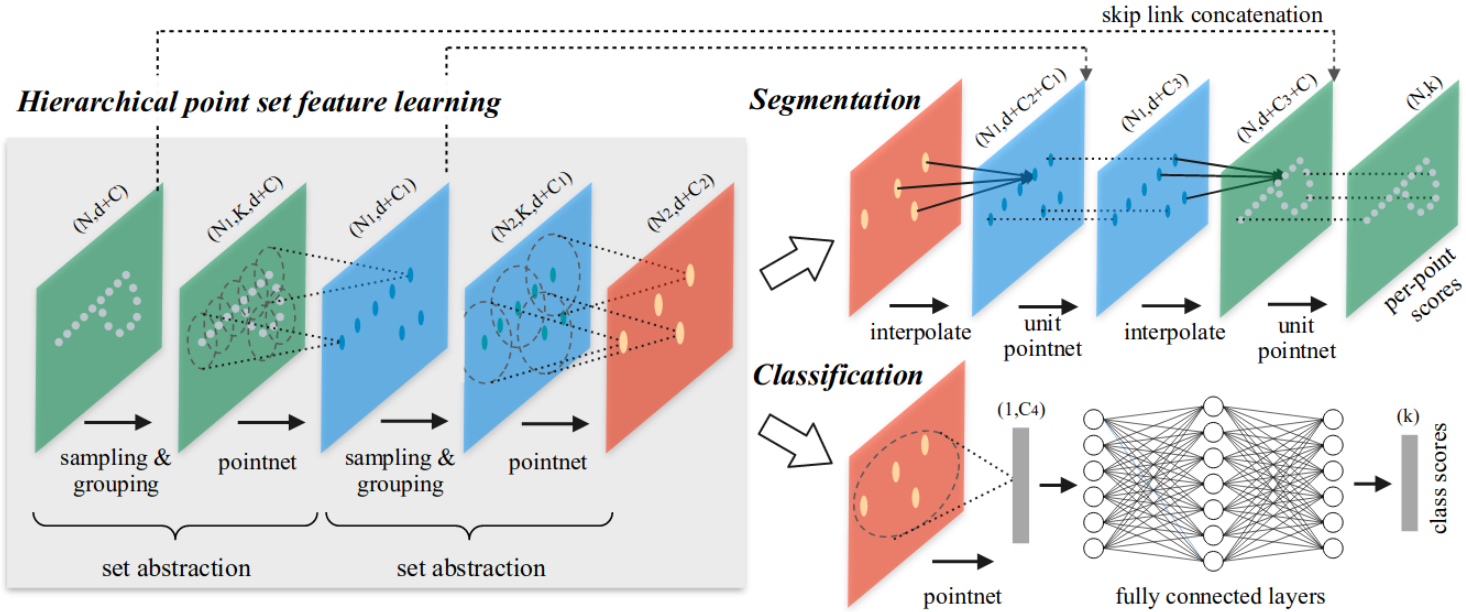
Credit[1]

[1] "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", Qi et al

Overview

PointNet++

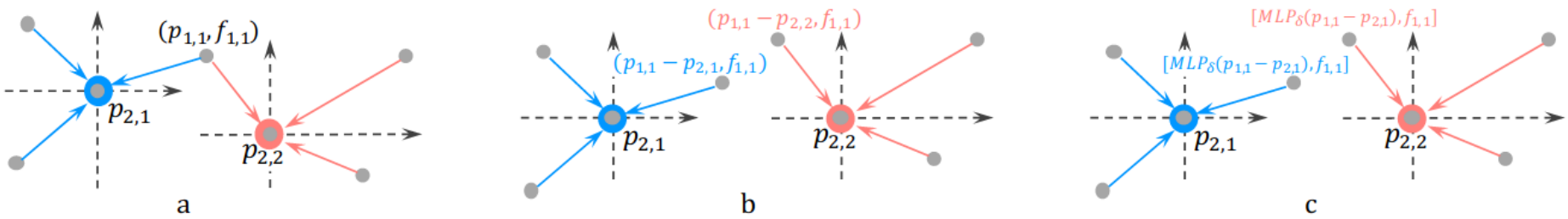
"PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", Qi et al



Main idea: applying recursively PointNet on a nested partitioning of the input point set ($\sim$ stacked CNN layers for 2d images) to capture local structure induced by the metric space.

PointCNN

"PointCNN: Convolution On χ-Transformed Points", Li et al



Main idea: Learn a χ-transformation (local) that transforms features associated with points into a canonical space ($\sim$ invariant to permutation) and then apply a standard convolution.

Thank you for your attention

PyTorch vs TensorFlow

- Dynamic graph (PyTorch) vs static graph (Tensorflow)
- PyTorch is more "pythonic"
- PyTorch is cleaner: better docs, no redundant API...
- Tensorflow has a better ecosystem: e.g. Tensorboard
- Tensorflow is more "production ready"

3D representation

Credit: "Deep Learning Advances on Different 3D DataRepresentations: A Survey", Ahmed et al