CS5182 Computer Graphics Introduction to Tensorflow & 3D Object Detection Tutorial

2024/25 Semester A

City University of Hong Kong (DG)

Deep Learning Framework

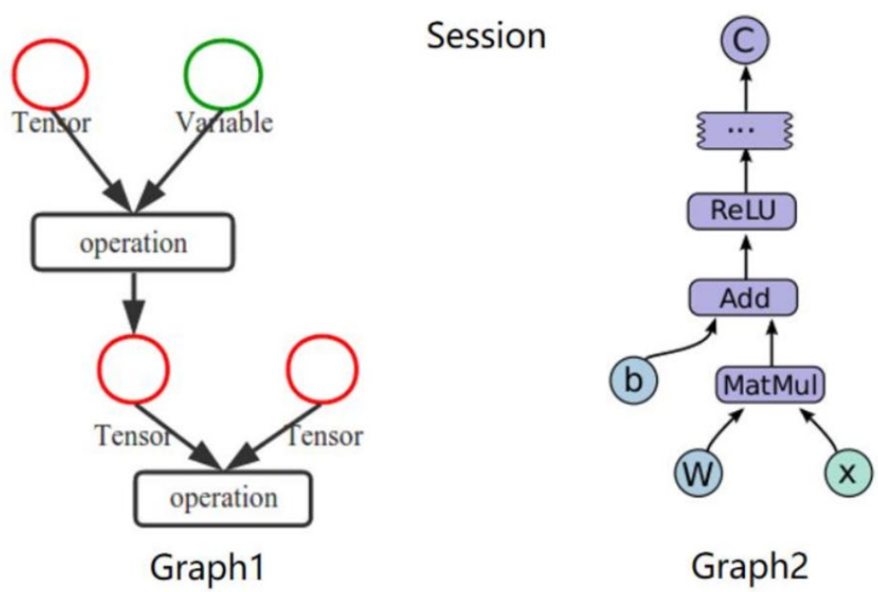
- Why we need framework for coding deep learning?
  - Easy implementation of neural network, e.g. layers, loss functions …
  - Automatically calculate gradients to update models
  - Fast integration, especially using GPU
  - Community are using them, lot of resources, e.g. GitHub
- TensorFlow 1.0 v.s. PyTorch

	Developed by	Graphs	Difficulty	Speed
TensorFlow	Google	Static graphs	Hard	Fast
PYTORCH	Facebook	Dynamic graphs	Easy	Fast enough

  - Keras, TFLearn: high-level API built upon TensorFlow
  - TensorFlow 2.0: using dynamic graphs, but totally different (less resource)

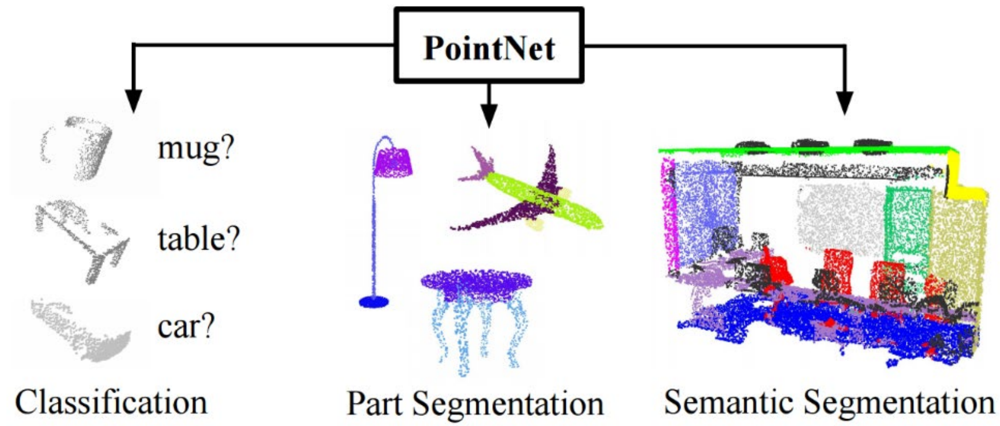
Tensor + Flow

- Main components of TensorFlow
  - Tensor: Define variables, including input/output, model weights.
  - Graph: Define calculation. Build static graph to connect those tensors.
  - Session: Execute calculation with data feeding. (Tensors flow in graph!)



Tensor + Flow

- Two examples
  - MNIST digit recognition using 2D CNN
  - 3D point cloud classification using PointNet



First step: Install TensorFlow

- Preliminary: Python, basic Linux, GPU machine
- Step1: Create a virtual environment (recommend Anaconda)

```
(base) qianyue@ss4028b:~$ conda create --name tf_env
...
(base) qianyue@ss4028b:~$ conda activate tf_env
(tf_env) qianyue@ss4028b:~$
```

- Step2: Install TensorFlow in new environment (recommend tf1)

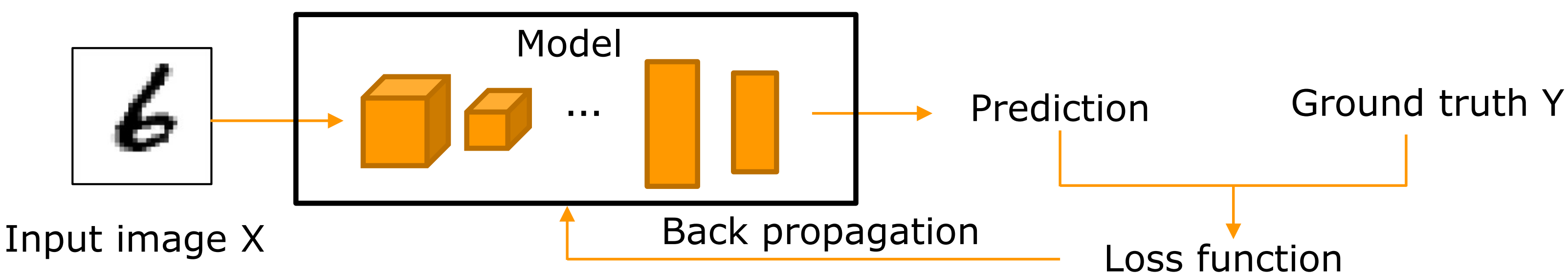
```
(tf_env) qianyue@ss4028b:~$ conda install tensorflow-gpu
(tf_env) qianyue@ss4028b:~$ conda install tensorflow-gpu==1.9.0
```

- Step3: Check whether success installed

```
>>> import tensorflow as tf
>>> tf.__version__
'1.9.0'
>>>
```

MNIST digit recognition using 2D CNN

- Example 1: MNIST digit recognition using 2D CNN
  - Input: 28x28 BW images
  - Output: which 0~9 digit?
  - Dataset: MNIST, 60k training, 10k testing with ground-truth label provided.
- Develop a simple CNN model for MNIST
  - Load dataset
  - Build neural network
  - Define loss function and learning strategies
  - Training iteratively
  - Evaluate performance and save model
  - Use TensorBoard to monitor training process



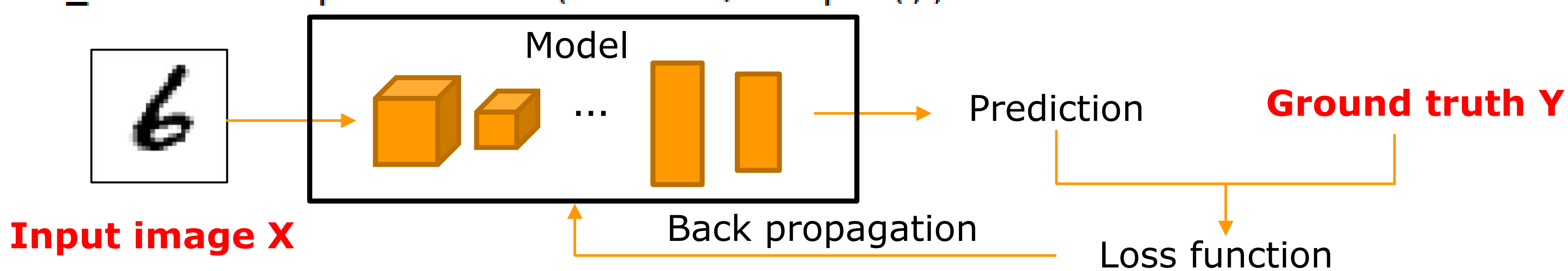
1. Load dataset

- 1.1 Load MNIST

```
import tensorflow as tf

# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
# batch_x, batch_y = mnist.train.next_batch(batch_size)
# test_x, test_y = mnist.test.images[:20], mnist.test.labels[:20]
```

- 1.2 Define placeholder for input and ground truth label

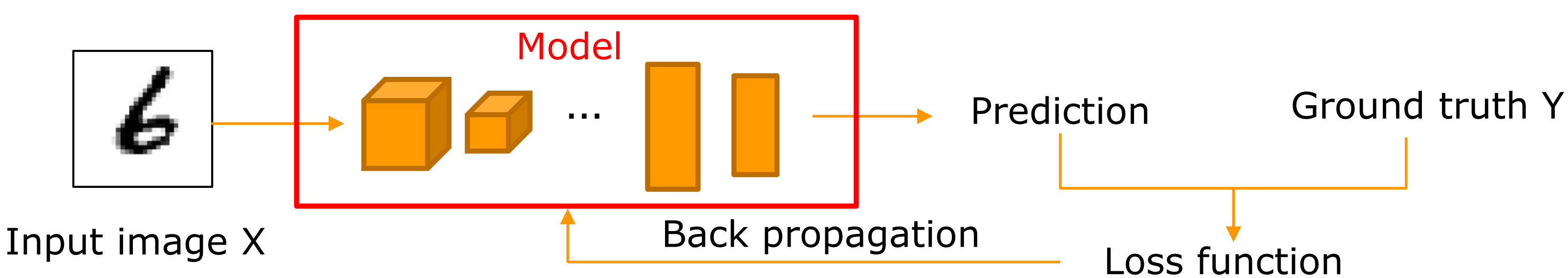
```
num_input = 784 # MNIST data input (img shape: 28*28)
num_classes = 10 # MNIST total classes (0-9 digits)
# tf Graph input
X = tf.placeholder(tf.float32, [None, num_input])
Y = tf.placeholder(tf.float32, [None, num_classes])
keep_prob = tf.placeholder(tf.float32) # dropout (keep probability)
is_reuse = tf.placeholder(tf.bool, shape=())
```



2. Build neural network

2.1 Define network

```python
# Create the neural network
def conv_net(x, dropout, reuse):
    n_classes = 10
    # Define a scope for reusing the variables
    with tf.variable_scope('ConvNet', reuse=reuse):
        # MNIST data input is a 1-D vector of 784 features (28*28 pixels)
        # Reshape to match picture format [Height x Width x Channel]
        # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
        x = tf.reshape(x, shape=[-1, 28, 28, 1])
        # Convolution Layer with 32 filters and a kernel size of 5
        conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
        # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
        conv1 = tf.layers.max_pooling2d(conv1, 2, 2)
        # Convolution Layer with 64 filters and a kernel size of 3
        conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
        # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
        conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

        # Flatten the data to a 1-D vector for the fully connected layer
        fc1 = tf.contrib.layers.flatten(conv2)
        # Fully connected layer (in tf contrib folder for now)
        fc1 = tf.layers.dense(fc1, 1024)
        # Apply Dropout (if is_training is False, dropout is not applied)
        fc1 = tf.nn.dropout(fc1, dropout)

        # Output layer, class prediction
        out = tf.layers.dense(fc1, n_classes)
    return out
```



Build neural network & Loss function

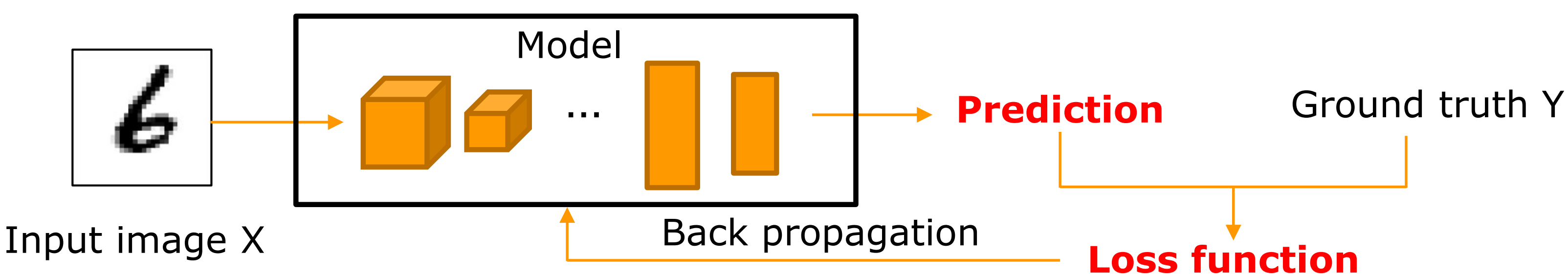
- 2.2 From placeholder X to prediction

```
# tf Graph input
X = tf.placeholder(tf.float32, [None, num_input])
Y = tf.placeholder(tf.float32, [None, num_classes])
keep_prob = tf.placeholder(tf.float32) # dropout (keep probability)
is_reuse = tf.placeholder(tf.bool, shape=())
```

```
logits = conv_net(X, keep_prob, is_reuse)
prediction = tf.nn.softmax(logits)
```

- 3. Define loss function

```
# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
```



4. Model Training

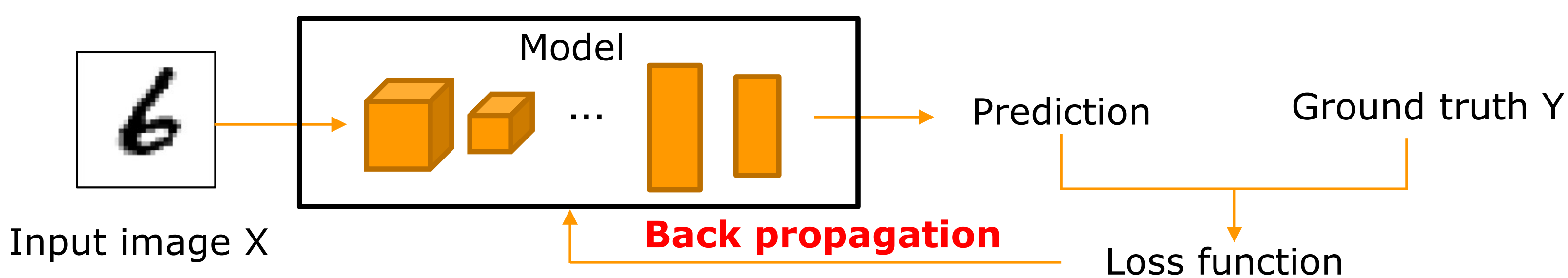
4.1 Define training hyper parameters

```python
# Training Parameters
learning_rate = 0.001
num_steps = 200
batch_size = 128
```

```python
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
```

4.2 Train iteratively

```python
# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
# Start training
with tf.Session() as sess:
    # Run the initializer
    sess.run(init)
    for step in range(1, num_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: 0.8, is_reuse: True})
        # Calculate batch loss and accuracy
        loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                             Y: batch_y,
                                                             keep_prob: 1.0,
                                                             is_reuse: False})

        print("Step " + str(step) + ", Minibatch Loss= " + \
              "{:.4f}".format(loss) + ", Training Accuracy= " + \
              "{:.3f}".format(acc))

    print("Optimization Finished!")
```



5. Evaluate performance and save model

5.1 Save trained model in path

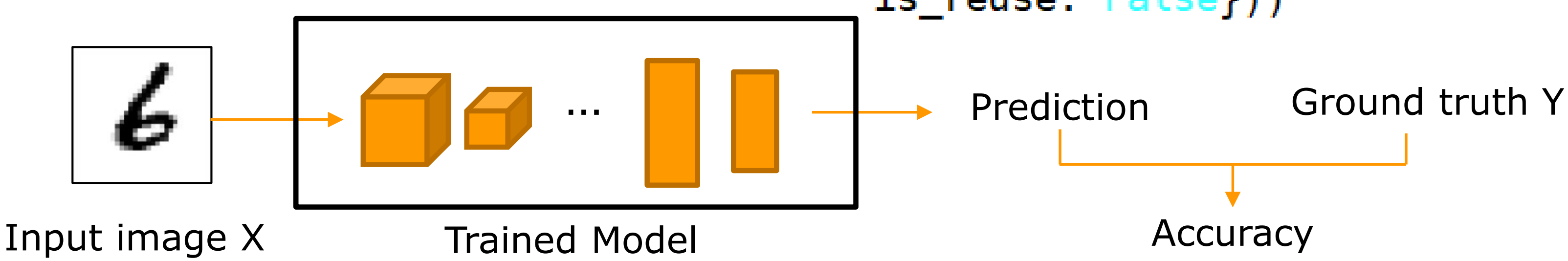
```
saver = tf.train.Saver()
with tf.Session() as sess:
    saver.save(sess, path, global_step=step)
```

5.2 Define accuracy

```
# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

5.3 Evaluate on testing dataset

```
with tf.Session() as sess:
    # Calculate accuracy for 256 MNIST test images
    print("Testing Accuracy:", \
        sess.run(accuracy, feed_dict={X: mnist.test.images[:256],
                                      Y: mnist.test.labels[:256],
                                      keep_prob: 1.0,
                                      is_reuse: False}))
```



6. Use TensorBoard to monitor training process

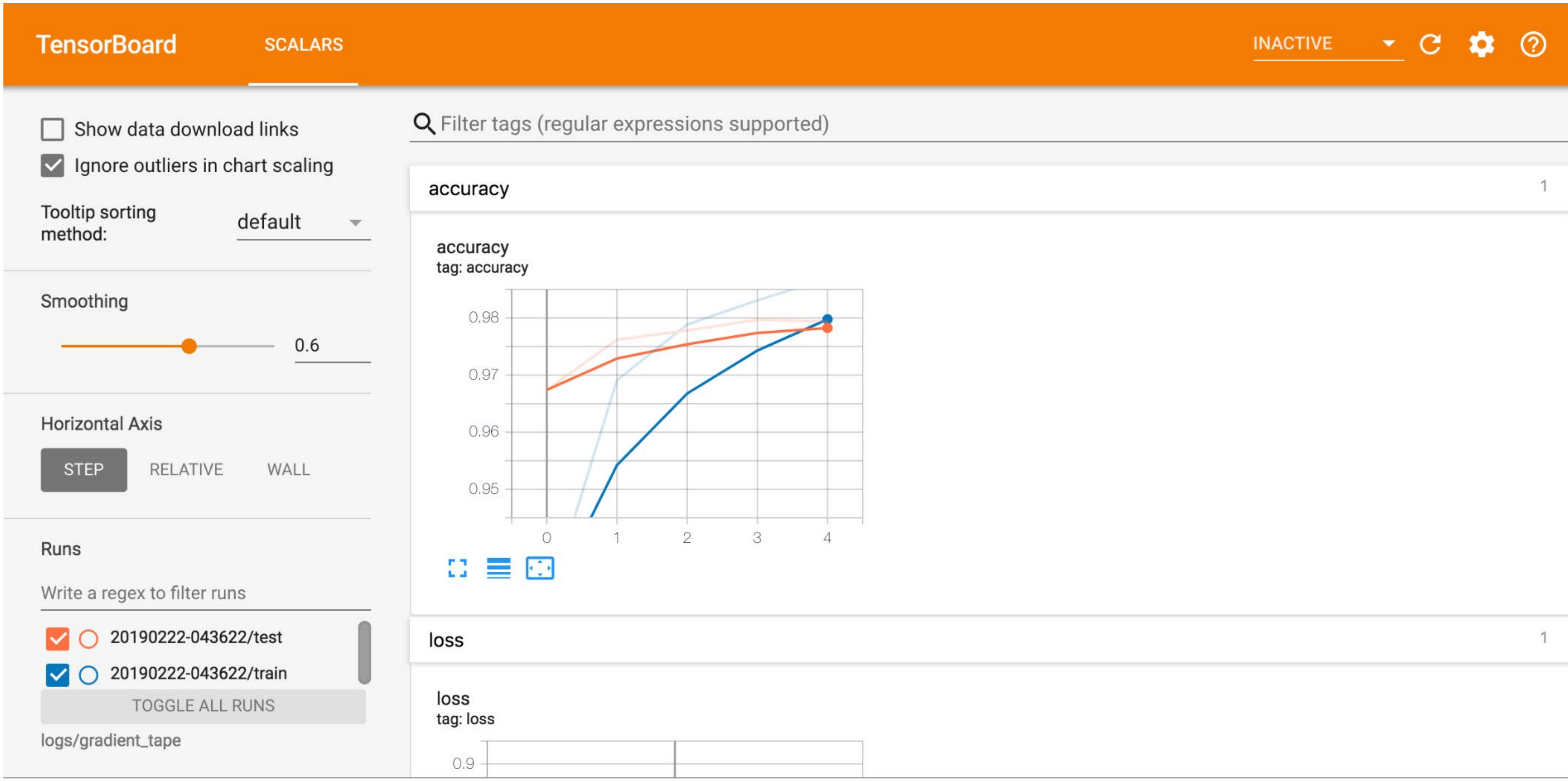
- 3.1 Record information

```
# Create a summary to monitor cost tensor
tf.summary.scalar("loss", cost)
# Create a summary to monitor accuracy tensor
tf.summary.scalar("accuracy", acc)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()

logs_path = '/tmp/tensorflow_logs/example/'
# Start training
with tf.Session() as sess:
    sess.run(init)
    # op to write logs to Tensorboard
    summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())
    for step in range(1, num_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop), cost op (to get loss value)
        # and summary nodes
        _, summary = sess.run([train_op, merged_summary_op],
                feed_dict={x: batch_x, y: batch_y, keep_prob: 0.8})
        # Write logs at every iteration
        summary_writer.add_summary(summary, step)
```

- 3.2 Visualize recorded results

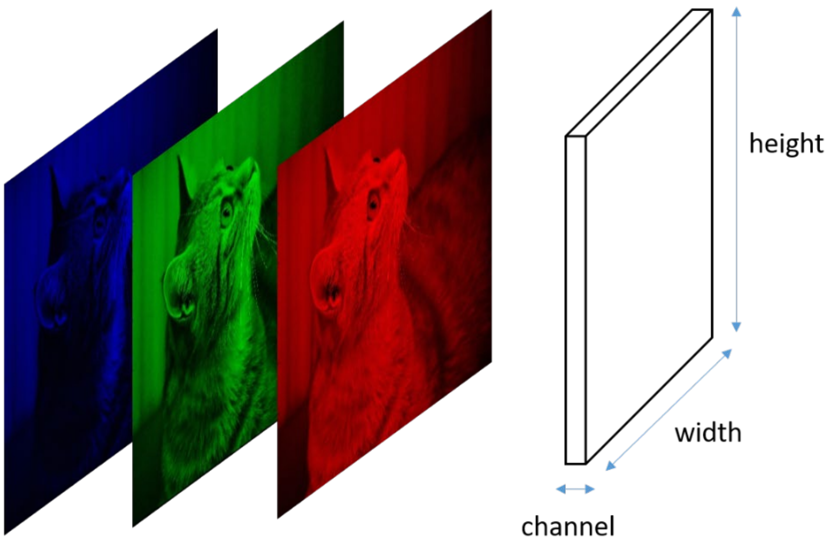
```
(tf_env) qianyue@ss4028b:~$ tensorboard --logdir='/tmp/tensorflow_logs/example/'
```

Tensorboard demo

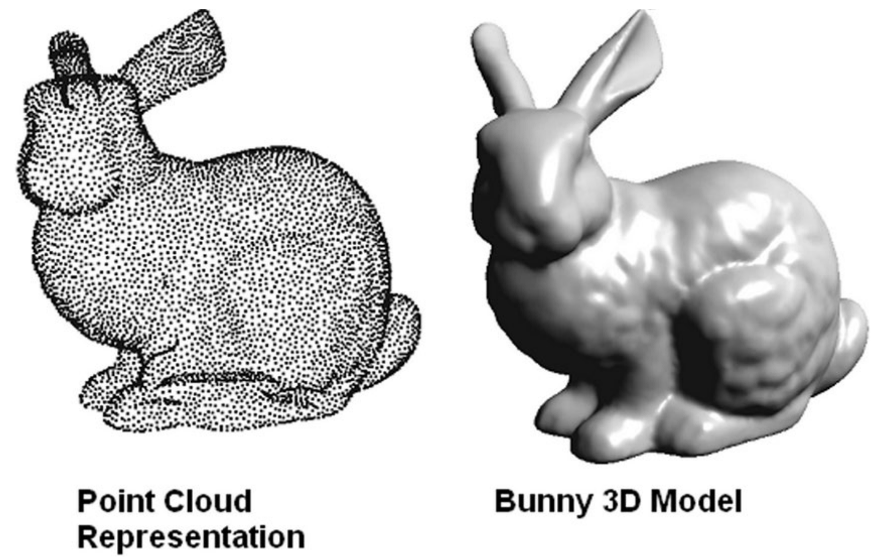


3D point cloud classification using PointNet

- Point cloud v.s. image



v.s.

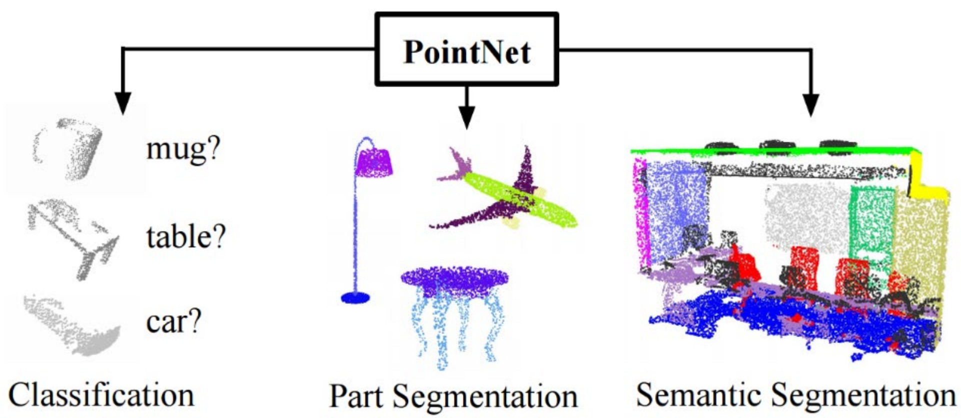


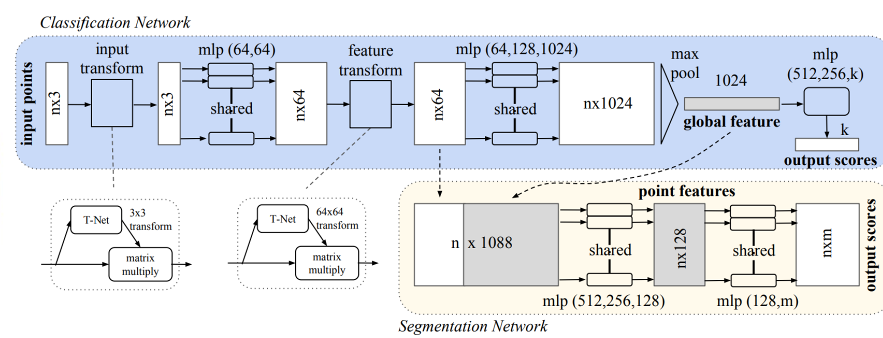
- Size: [H, W, 3], where feature is (r,g,b)
- 2D regular structure, encoded as matrix

- Size: [N, 3]
- Feature is (x,y,z)
- 3D irregular structure, discrete representation for 3D shape

3D point cloud classification using PointNet

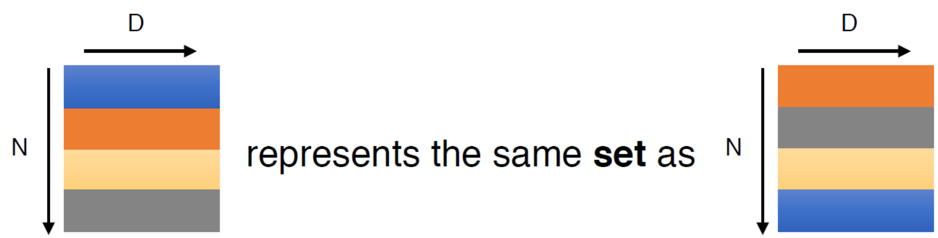
- PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation (paper)
- Input: [N, 3] point clouds
- Output:
  - Classification: [K,1] as scores for K classification classes
  - Segmentation: [N, M] as point-wise scores for M segmentation classes





3D point cloud classification using PointNet

- Challenge1: Permutation Invariance: Point cloud is a set of unordered points

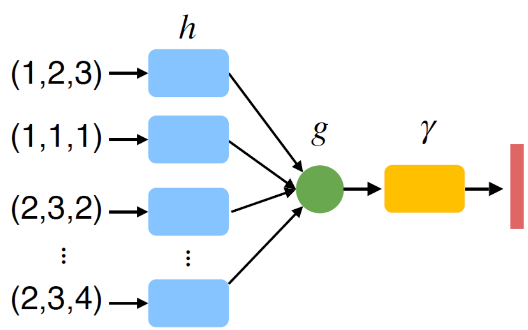


Model needs to be invariant to N! permutations

- Solution: Symmetric Function
  - Examples:

$$f(x_1, x_2, \ldots, x_n) = \max\{x_1, x_2, \ldots, x_n\}$$

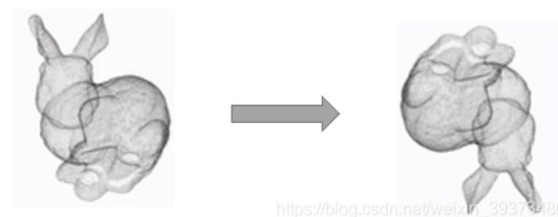
$$f(x_1, x_2, \ldots, x_n) = x_1 + x_2 + \ldots + x_n$$



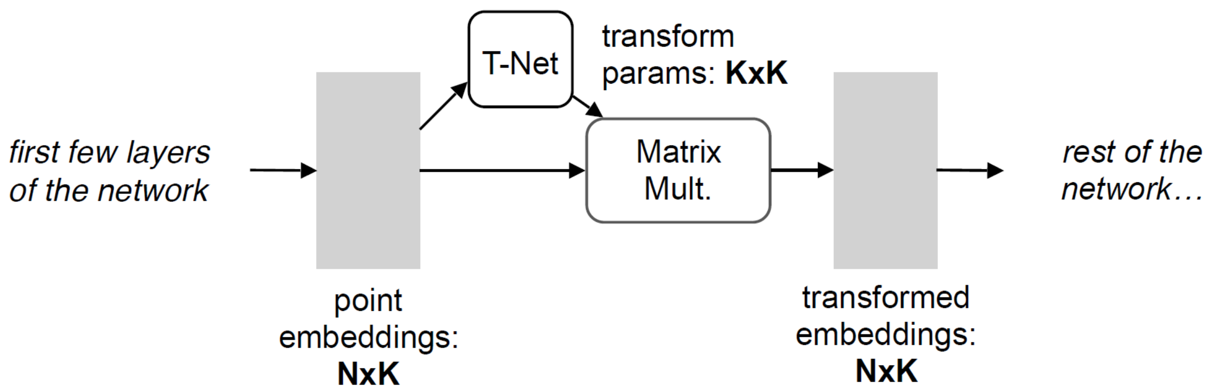
Construction Symmetric Functions by Neural Networks

3D point cloud classification using PointNet

- Challenge2: Invariance under geometric transformations: Point cloud rotation should not affect classification results.



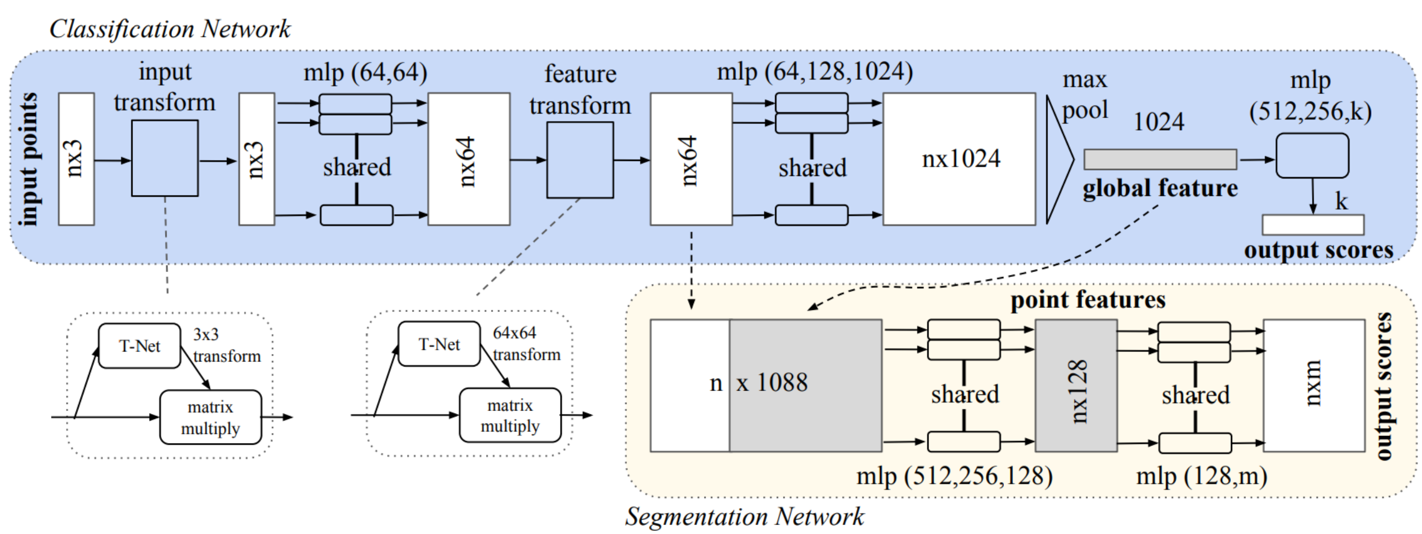
- Solution: Data dependent transformation for automatic alignment
  - Matrix multiplication



Embedding Space Alignment

3D point cloud classification using PointNet

- Input is a set of point cloud data, represented as an nx3 2D tensor, where n represents the number of point clouds, and 3 corresponds to the xyz coordinates.
- The input data is first aligned by multiplying it with a transformation matrix learned by T-Net, which ensures the invariance of the model to specific spatial transformations.
- After extracting the features of each point cloud data through multiple MLPs, use a T-Net to align the features.
- Perform a max-pooling operation on each dimension of the feature to obtain the final global feature.
- Output
  - For the classification task, the global feature is used to predict the final classification score through MLPs;
  - For the segmentation task, the global feature and the local features of each point cloud learned before are concatenated, and then the classification result of each data point is obtained through MLPs .



PointNet code (main - 1/3)

```
import argparse
import math
import h5py
import numpy as np
import tensorflow as tf
import socket
import importlib
import os
import sys
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
sys.path.append(BASE_DIR)
sys.path.append(os.path.join(BASE_DIR, 'models'))
sys.path.append(os.path.join(BASE_DIR, 'utils'))
import provider
import tf_util

parser = argparse.ArgumentParser()
parser.add_argument('--gpu', type=int, default=0, help='GPU to use [default: GPU 0]')
parser.add_argument('--model', default='pointnet_cls', help='Model name: pointnet_cls or pointnet_cls_basic [default: pointnet_cls]')
parser.add_argument('--log_dir', default='log', help='Log dir [default: log]')
parser.add_argument('--num_point', type=int, default=1024, help='Point Number [256/512/1024/2048] [default: 1024]')
parser.add_argument('--max_epoch', type=int, default=250, help='Epoch to run [default: 250]')
parser.add_argument('--batch_size', type=int, default=32, help='Batch Size during training [default: 32]')
parser.add_argument('--learning_rate', type=float, default=0.001, help='Initial learning rate [default: 0.001]')
parser.add_argument('--momentum', type=float, default=0.9, help='Initial learning rate [default: 0.9]')
parser.add_argument('--optimizer', default='adam', help='adam or momentum [default: adam]')
parser.add_argument('--decay_step', type=int, default=200000, help='Decay step for lr decay [default: 200000]')
parser.add_argument('--decay_rate', type=float, default=0.7, help='Decay rate for lr decay [default: 0.8]')
FLAGS = parser.parse_args()

BATCH_SIZE = FLAGS.batch_size
NUM_POINT = FLAGS.num_point
MAX_EPOCH = FLAGS.max_epoch
BASE_LEARNING_RATE = FLAGS.learning_rate
GPU_INDEX = FLAGS.gpu
MOMENTUM = FLAGS.momentum
OPTIMIZER = FLAGS.optimizer
DECAY_STEP = FLAGS.decay_step
DECAY_RATE = FLAGS.decay_rate

MODEL = importlib.import_module(FLAGS.model) # import network module
MODEL_FILE = os.path.join(BASE_DIR, 'models', FLAGS.model+'.py')
LOG_DIR = FLAGS.log_dir
```

Install TensorFlow, h5py
Import all the dependencies

provider: Load data and perform data processing
tf_util: Define layers (convolution, FC, dropout) for point clouds

classification model

Training hyper-parameters

PointNet code (main - 2/3)

```python
def get_learning_rate(batch):
    learning_rate = tf.train.exponential_decay(
                        BASE_LEARNING_RATE,  # Base learning rate.
                        batch * BATCH_SIZE,  # Current index into the dataset.
                        DECAY_STEP,          # Decay step.
                        DECAY_RATE,          # Decay rate.
                        staircase=True)
    learning_rate = tf.maximum(learning_rate, 0.00001) # CLIP THE LEARNING RATE!
    return learning_rate

def get_bn_decay(batch):
    bn_momentum = tf.train.exponential_decay(
                      BN_INIT_DECAY,
                      batch*BATCH_SIZE,
                      BN_DECAY_DECAY_STEP,
                      BN_DECAY_DECAY_RATE,
                      staircase=True)
    bn_decay = tf.minimum(BN_DECAY_CLIP, 1 - bn_momentum)
    return bn_decay
```

Training hyper-parameters

```python
def placeholder_inputs(batch_size, num_point):
    pointclouds_pl = tf.placeholder(tf.float32, shape=(batch_size, num_point, 3))
    labels_pl = tf.placeholder(tf.int32, shape=(batch_size))
    return pointclouds_pl, labels_pl
```

Run on specific GPU

```python
def train():
    with tf.Graph().as_default():
        with tf.device('/gpu:'+str(GPU_INDEX)):
            pointclouds_pl, labels_pl = MODEL.placeholder_inputs(BATCH_SIZE, NUM_POINT)
            is_training_pl = tf.placeholder(tf.bool, shape=())
            print(is_training_pl)

            # Note the global_step=batch parameter to minimize.
            # That tells the optimizer to helpfully increment the 'batch' parameter for you every time it trains.
            batch = tf.Variable(0)
            bn_decay = get_bn_decay(batch)
            tf.summary.scalar('bn_decay', bn_decay)

            # Get model and loss
            pred, end_points = MODEL.get_model(pointclouds_pl, is_training_pl, bn_decay=bn_decay)
            loss = MODEL.get_loss(pred, labels_pl, end_points)
            tf.summary.scalar('loss', loss)

            correct = tf.equal(tf.argmax(pred, 1), tf.to_int64(labels_pl))
            accuracy = tf.reduce_sum(tf.cast(correct, tf.float32)) / float(BATCH_SIZE)
            tf.summary.scalar('accuracy', accuracy)
```

```python
            # Add ops to save and restore all the variables.
            saver = tf.train.Saver()
```

Call PointNet network
pred: [B, K]
classification scores

Define loss

Define accuracy

In order to store trained model

PointNet code (main - 3/3)

```python
        # Create a session
        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        config.allow_soft_placement = True
        config.log_device_placement = False
        sess = tf.Session(config=config)

        # Add summary writers
        #merged = tf.merge_all_summaries()
        merged = tf.summary.merge_all()
        train_writer = tf.summary.FileWriter(os.path.join(LOG_DIR, 'train'),
                                  sess.graph)
        test_writer = tf.summary.FileWriter(os.path.join(LOG_DIR, 'test'))

        # Init variables
        init = tf.global_variables_initializer()
        sess.run(init, {is_training_pl: True})

        for epoch in range(MAX_EPOCH):
            log_string('**** EPOCH %03d ****' % (epoch))
            sys.stdout.flush()

            train_one_epoch(sess, ops, train_writer)
            eval_one_epoch(sess, ops, test_writer)

            # Save the variables to disk.
            if epoch % 10 == 0:
                save_path = saver.save(sess, os.path.join(LOG_DIR, "model.ckpt"))
                log_string("Model saved in file: %s" % save_path)
```

Create session

For TensorBoard visualization

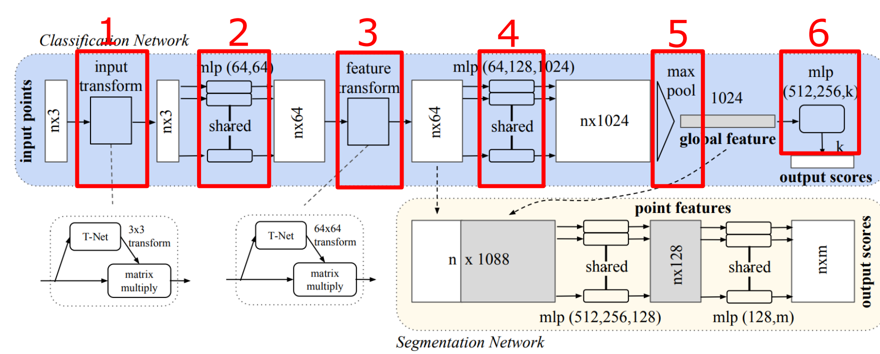
Initialize variables

Train iteratively
Epoch: train whole dataset once

Store model

Network code (pointnet_cls.get_model)

```
import tf_util
from transform_nets import input_transform_net, feature_transform_net

def get_model(point_cloud, is_training, bn_decay=None):
    """ Classification PointNet, input is BxNx3, output Bx40 """
    batch_size = point_cloud.get_shape()[0].value
    num_point = point_cloud.get_shape()[1].value
    end_points = {}

    with tf.variable_scope('transform_net1') as sc:
        transform = input_transform_net(point_cloud, is_training, bn_decay, K=3)
    point_cloud_transformed = tf.matmul(point_cloud, transform)
    input_image = tf.expand_dims(point_cloud_transformed, -1)

    net = tf_util.conv2d(input_image, 64, [1,3],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='conv1', bn_decay=bn_decay)
    net = tf_util.conv2d(net, 64, [1,1],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='conv2', bn_decay=bn_decay)

    with tf.variable_scope('transform_net2') as sc:
        transform = feature_transform_net(net, is_training, bn_decay, K=64)
    end_points['transform'] = transform
    net_transformed = tf.matmul(tf.squeeze(net, axis=[2]), transform)
    net_transformed = tf.expand_dims(net_transformed, [2])

    net = tf_util.conv2d(net_transformed, 64, [1,1],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='conv3', bn_decay=bn_decay)
    net = tf_util.conv2d(net, 128, [1,1],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='conv4', bn_decay=bn_decay)
    net = tf_util.conv2d(net, 1024, [1,1],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='conv5', bn_decay=bn_decay)

    # Symmetric function: max pooling
    net = tf_util.max_pool2d(net, [num_point,1],
                             padding='VALID', scope='maxpool')

    net = tf.reshape(net, [batch_size, -1])
    net = tf_util.fully_connected(net, 512, bn=True, is_training=is_training,
                                  scope='fc1', bn_decay=bn_decay)
    net = tf_util.dropout(net, keep_prob=0.7, is_training=is_training,
                          scope='dp1')
    net = tf_util.fully_connected(net, 256, bn=True, is_training=is_training,
                                  scope='fc2', bn_decay=bn_decay)
    net = tf_util.dropout(net, keep_prob=0.7, is_training=is_training,
                          scope='dp2')
    net = tf_util.fully_connected(net, 40, activation_fn=None, scope='fc3')

    return net, end_points
```



Training code (train_one_epoch)

```
def train_one_epoch(sess, ops, train_writer):
    """ ops: dict mapping from string to tf ops """
    is_training = True

    # Shuffle train files
    train_file_idxs = np.arange(0, len(TRAIN_FILES))
    np.random.shuffle(train_file_idxs)

    for fn in range(len(TRAIN_FILES)):
        log_string('----' + str(fn) + '-----')
        current_data, current_label = provider.loadDataFile(TRAIN_FILES[train_file_idxs[fn]])
        current_data = current_data[:,0:NUM_POINT,:]
        current_data, current_label, _ = provider.shuffle_data(current_data, np.squeeze(current_label))
        current_label = np.squeeze(current_label)

        file_size = current_data.shape[0]
        num_batches = file_size // BATCH_SIZE

        total_correct = 0
        total_seen = 0
        loss_sum = 0

        for batch_idx in range(num_batches):
            start_idx = batch_idx * BATCH_SIZE
            end_idx = (batch_idx+1) * BATCH_SIZE

            # Augment batched point clouds by rotation and jittering
            rotated_data = provider.rotate_point_cloud(current_data[start_idx:end_idx, :, :])
            jittered_data = provider.jitter_point_cloud(rotated_data)
            feed_dict = {ops['pointclouds_pl']: jittered_data,
                         ops['labels_pl']: current_label[start_idx:end_idx],
                         ops['is_training_pl']: is_training,}
            summary, step, _, loss_val, pred_val = sess.run([ops['merged'], ops['step'],
                ops['train_op'], ops['loss'], ops['pred']], feed_dict=feed_dict)
            train_writer.add_summary(summary, step)
            pred_val = np.argmax(pred_val, 1)
            correct = np.sum(pred_val == current_label[start_idx:end_idx])
            total_correct += correct
            total_seen += BATCH_SIZE
            loss_sum += loss_val

        log_string('mean loss: %f' % (loss_sum / float(num_batches)))
        log_string('accuracy: %f' % (total_correct / float(total_seen)))
```

Allow training of weights

Load training data

Train iteratively for training data

Data augmentation

Feed data and run session

Evaluation code (eval_one_epoch)

```
def eval_one_epoch(sess, ops, test_writer):
    """ ops: dict mapping from string to tf ops """
    is_training = False
    total_correct = 0
    total_seen = 0
    loss_sum = 0
    total_seen_class = [0 for _ in range(NUM_CLASSES)]
    total_correct_class = [0 for _ in range(NUM_CLASSES)]

    for fn in range(len(TEST_FILES)):
        log_string('----' + str(fn) + '-----')
        current_data, current_label = provider.loadDataFile(TEST_FILES[fn])
        current_data = current_data[:,0:NUM_POINT,:]
        current_label = np.squeeze(current_label)

        file_size = current_data.shape[0]
        num_batches = file_size // BATCH_SIZE

        for batch_idx in range(num_batches):
            start_idx = batch_idx * BATCH_SIZE
            end_idx = (batch_idx+1) * BATCH_SIZE

            feed_dict = {ops['pointclouds_pl']: current_data[start_idx:end_idx, :, :],
                         ops['labels_pl']: current_label[start_idx:end_idx],
                         ops['is_training_pl']: is_training}
            summary, step, loss_val, pred_val = sess.run([ops['merged'], ops['step'],
                ops['loss'], ops['pred']], feed_dict=feed_dict)
            pred_val = np.argmax(pred_val, 1)
            correct = np.sum(pred_val == current_label[start_idx:end_idx])
            total_correct += correct
            total_seen += BATCH_SIZE
            loss_sum += (loss_val*BATCH_SIZE)
            for i in range(start_idx, end_idx):
                l = current_label[i]
                total_seen_class[l] += 1
                total_correct_class[l] += (pred_val[i-start_idx] == l)

    log_string('eval mean loss: %f' % (loss_sum / float(total_seen)))
    log_string('eval accuracy: %f'% (total_correct / float(total_seen)))
    log_string('eval avg class acc: %f' % (np.mean(np.array(total_correct_class)/np.array(total_seen_class,dtype=np.float))))
```

Freeze model weights

Loop the testing files

Load testing data

Feed data and run session

Some useful links

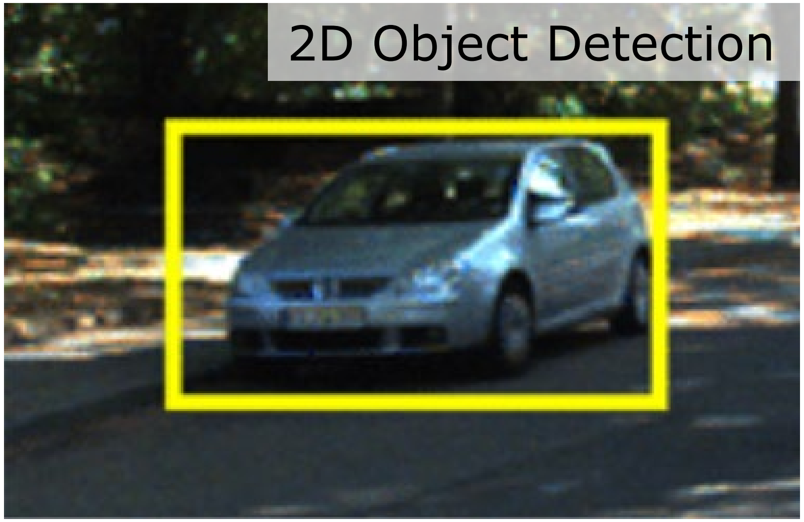
- MNIST example: https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/convolutional_network_raw.py
- PointNet example: https://github.com/charlesq34/pointnet

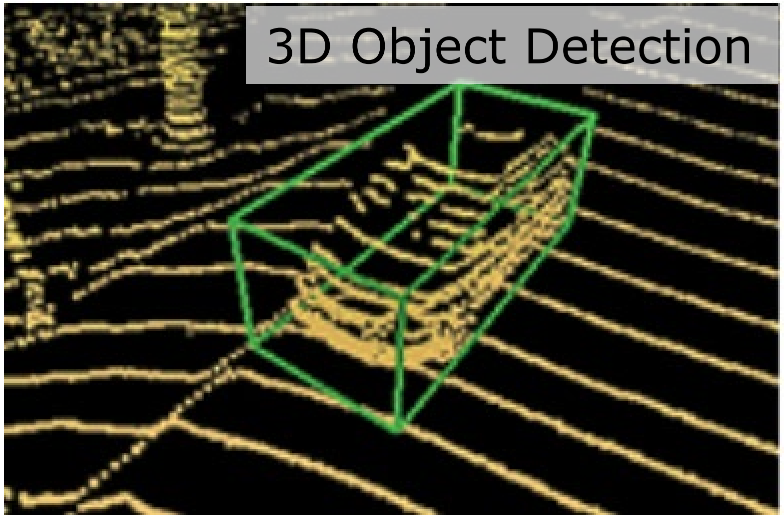
- More examples: https://github.com/aymericdamien/TensorFlow-Examples
- Point clouds projects: https://github.com/Yochengliu/awesome-point-cloud-analysis

- TensorFlow API: https://www.tensorflow.org/versions/r1.15/api_docs/python/tf
- Linux: http://www.ee.surrey.ac.uk/Teaching/Unix/
- Anaconda: https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html
- Stackoverflow: https://stackoverflow.com/questions/tagged/tensorflow

3D Object Detection

- Definition: It extends the concept of 2D object detection to the 3D physical space, and predicts the location, size, and orientation in the 3D space.





- Detection the target object in the image.
- Predicting the location and size of each object in the image.

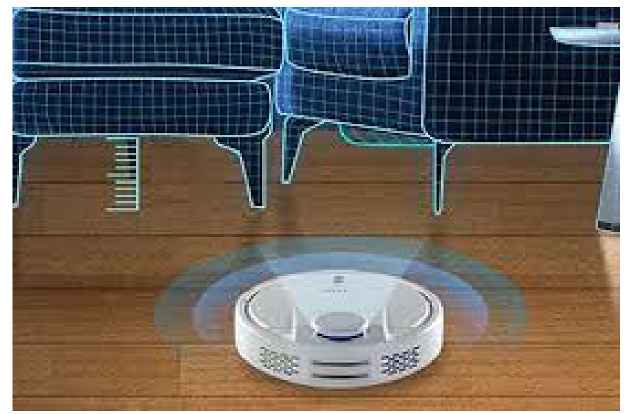
- Detection the target object in the 3D physical space.
- Predicting the location, size, and orientation of each object.

Picture source: CLOCs: Camera-LiDAR Object Candidates Fusion for 3D Object Detection.

3D Object Detection

- The 3D object detection is an important technique that can be widely applied in many realistic scenes for 3D perception.

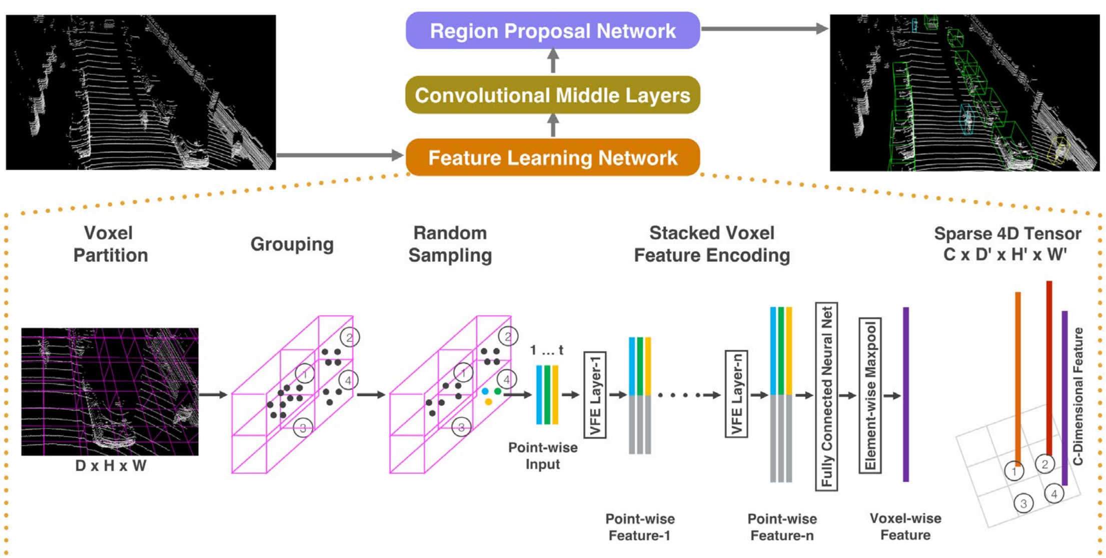
Autonomous Car



Indoor robots

VoxelNet (2017)

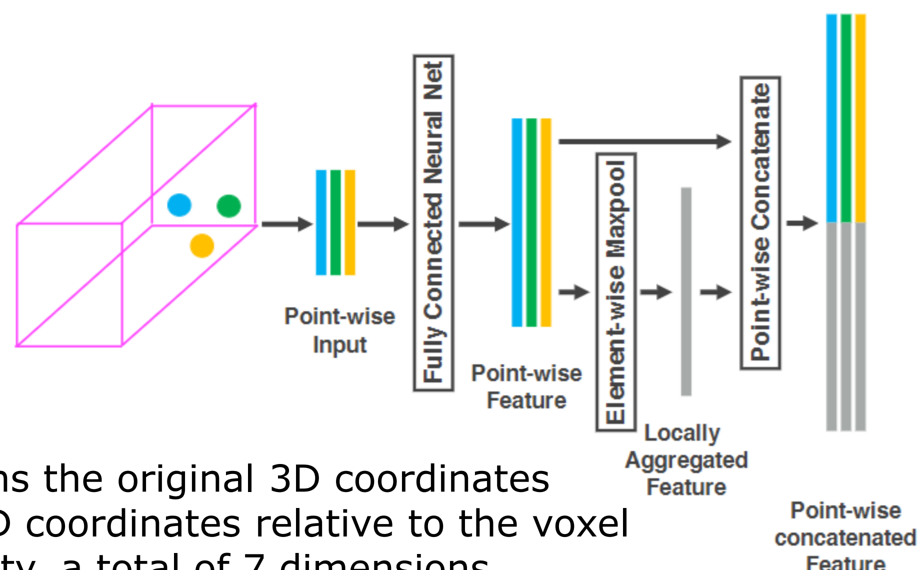
- The first work to apply PointNet to voxels, the overall process:
  - 1. Divide the space into voxels, use the VFE network in the voxel to extract local point cloud features, and obtain a three-dimensional feature volume.
  - 2. Send the features to the 3D convolutional network to further improve the expression ability, and compress the final output in the vertical direction to obtain a 2D feature map.
  - 3. Send the 2D feature map to the RPN network to generate a 3D box prediction.



Zhou et al. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. CVPR2018

VFE network in VoxelNet

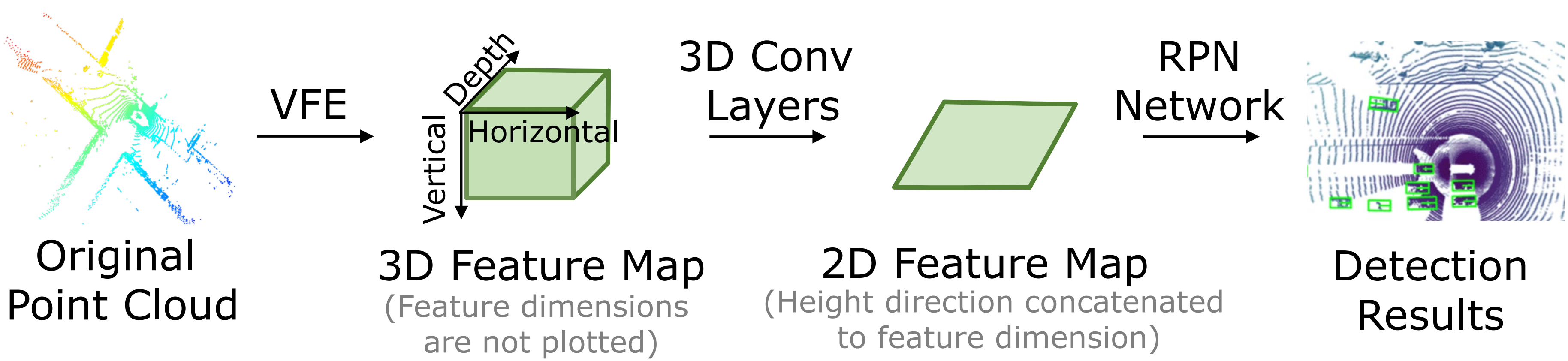
- The VFE (Voxel-Feature-Encoding) structure is used to extract point cloud features within voxels:
  - The structure is similar to **PointNet**: a fully connected layer (including BN and ReLU) is used to transform the original coordinates of each point, and then the global feature is obtained by maximum pooling. Finally, the global feature is spliced onto the point-by-point feature as output.
  - **Stackable**: The output of VFE can be used as the input of the next VFE module, and multi-layer stacking improves the feature expression capability



The original input contains the original 3D coordinates of the point, the local 3D coordinates relative to the voxel center, and the reflectivity, a total of 7 dimensions.

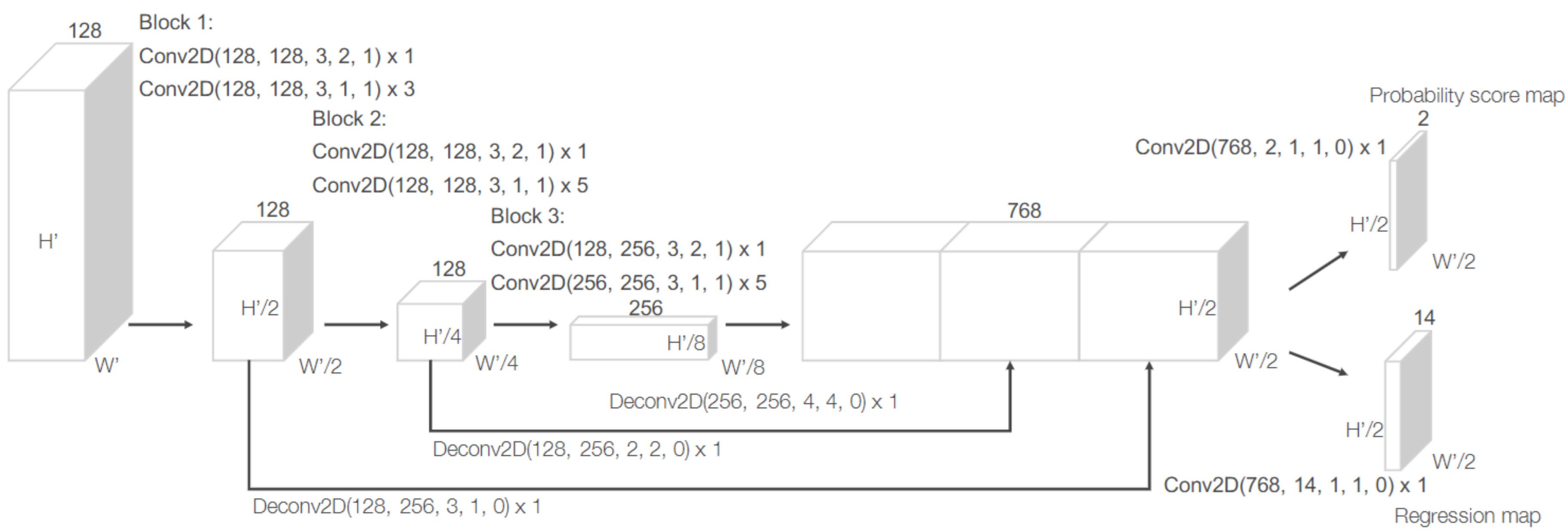
Middle layers in VoxelNet

- 1. The feature volume output by the VFE layer has a size of 128×10×400×352 (feature dimension×vertical×horizontal×depth)
- 2. It is fed into several 3D convolutional networks, and the vertical direction is downsampled by step size, and the output feature size is 64×2×400×352
- 3. The features of the two voxels in the vertical direction are concatenated to obtain a 2D feature map with a size of 128×400×352 (feature dimension×horizontal×depth)



Region proposal network architecture

- VoxelNet uses a custom multi-scale detection network to give the final 3D bounding box prediction based on the 2D feature map
  - 1: The features output by the intermediate layer are fed into a U-Net-like structure to generate feature maps
  - 2. Based on the detection head of the wrong frame, two chain frames are set, so there are 2 category predictions and 14 bounding box regressors



Loss function

- The classification uses the two-class cross entropy BCE loss function, and the regression uses the SmoothL1 loss function.
  - For the regression loss, it follows the 2D detection R-CNN method to encode the residual between the true value box/prediction box and the anchor box.
  - The angle residual = the difference between the true value box and the anchor box angle.

$$L = \underbrace{\alpha \frac{1}{N_{\text{pos}}} \sum_i L_{\text{cls}}(p_i^{\text{pos}}, 1)}_{\text{Classification loss of positive samples}} + \underbrace{\beta \frac{1}{N_{\text{neg}}} \sum_j L_{\text{cls}}(p_j^{\text{neg}}, 0)}_{\text{Classification loss of negative samples}} + \underbrace{\frac{1}{N_{\text{pos}}} \sum_i L_{\text{reg}}(\mathbf{u}_i, \mathbf{u}_i^*)}_{\text{Regression loss of positive samples}}$$

$$\Delta x = \frac{x_c^g - x_c^a}{d^a}, \Delta y = \frac{y_c^g - y_c^a}{d^a}, \Delta z = \frac{z_c^g - z_c^a}{h^a},$$
$$\Delta l = \log(\frac{l^g}{l^a}), \Delta w = \log(\frac{w^g}{w^a}), \Delta h = \log(\frac{h^g}{h^a}),$$
$$\Delta\theta = \theta^g - \theta^a$$