基于Capsule的迁移学习

目录

/capsule-transfer
├─── /data | 数据目录
│ ├─── feature.csv 一些用于迁移学习的features
│ ├─── /source_data | 基础数据集
│ │ └─── /mnist
│ └─── /preprocessed_data | 预处理后数据
├─── /notebooks | jupyter notebook 用于实验和可视化
│ ├─── test.ipynb
│ └─── vew_data.ipynb
└─── /src | 源代码根目录
├─── /models | 包含各类模型函数和工具函数，用于构建模型
│ ├─── __init__.py | 头文件
│ ├─── activation_functions.py | capsule的激活函数
│ ├─── capsule_layers.py | 各类capsule模型
│ ├─── get_transfer_learning_codes.py | 直接获得迁移学习模型的bottleneck features
│ ├─── layers.py | 主要的模型架构层
│ ├─── loss_funcitons.py | 损失函数
│ ├─── model.py | 网络架构
│ ├─── utils.py | 工具函数集合
│ └─── variables.py | 获取变量
├─── download_data.py | 下载数据集，包括MNIST和CIFAR
├─── generate_MNIST_N.py | 生成组合MNIST数据集
├─── preprocess.py | 数据预处理（包括获取迁移学习的bottleneck features）
├─── main.py | 运行主文件，用于训练模型
├─── test.py | 测试主文件，用于训练测试和评估
├─── model_arch.py | 模型结构设置
├─── config.py | 配置文件，包含各类参数和超参数的设置
├─── config_fine_tune.py | 迁移（继承）学习的配置文件
├─── pipeline.py | pipeline训练，一次性跑多个配置的多个模型
├─── /experiments | 各类实验的参数和架构
│ ├─── __init__.py
│ ├─── baseline_arch.py

```
│   ├──── baseline_config.py
│   ├──── baseline_config_fine_tune.py
│   ├──── hinton_arch.py
│   ├──── hinton_config.py
│   ├──── hinton_config_fine_tune.py
│   ├──── pipeline_arch.py
│   ├──── pipeline_config.py
│   └──── pipeline_config_fine_tune.py
├──── clear_all_logs.sh | bash命令文件，清除所有训练记录
├──── remove_all_data.sh | bash命令文件，清除所有预处理的数据
├──── archive_logs.sh | bash命令文件，将所有log归档
└──── zip_logs.sh | bash命令文件，将所有训练log打包
```

数据准备

下载MNIST或CIFAR数据集

```
python download_data.py
```

然后，输入指令选择下载数据集:

```
=========================================================
Input [ 1 ] to download the MNIST database.
Input [ 2 ] to download the CIFAR-10 database.
Input [ 3 ] to download the MNIST and CIFAR-10 database.
---------------------------------------------------------
Input:
```

数据预处理

```
python preprocess.py
```

> *运行时可选参数:*
> *-h, --help show this help message and exit*
> *-b, --baseline Use baseline configurations.*

> *-m, --mnist Preprocess the MNIST database.*
> *-c, --cifar Preprocess the CIFAR-10 database.*
> *-t1, --tl1 Save transfer learning cache data.*
> *-t2, --tl2 Get transfer learning bottleneck features.*
> *-si, --show_img Get transfer learning bottleneck features.*
> *-ft, --fine_tune Fine-tuning.*

参数设置在 `config.py` 中：

```
# Setting test set as validation when preprocessing data
__C.DPP_TEST_AS_VALID = False

# Rate of train-test split
__C.TEST_SIZE = 0.2

# Rate of train-validation split
__C.VALID_SIZE = 5000

# Resize inputs
__C.RESIZE_INPUTS = True
# Input size
__C.INPUT_SIZE = (28, 28)

# Resize images
__C.RESIZE_IMAGES = True
# Image size
__C.IMAGE_SIZE = (28, 28)

# Using data augment
__C.USE_DATA_AUG = False
# Parameters for data augment
__C.DATA_AUG_PARAM = dict(
    rotation_range=40,
    width_shift_range=0.4,
    height_shift_range=0.4,
    # shear_range=0.1,
    zoom_range=[1.0, 2.0],
    horizontal_flip=True,
    fill_mode='nearest'
)
# Keep original images if use data augment
```

```
__C.DATA_AUG_KEEP_SOURCE = True
# The max number of images of a class if use data augment
__C.MAX_IMAGE_NUM = 10000

# Preprocessing images of superpositions of multi-objects
# If None, do not pipeline multi-objects images.
# If n, one image includes a superposition of n objects, the positions of
# those objects are random.
__C.NUM_MULTI_OBJECT = 2
# The number of multi-objects images
__C.NUM_MULTI_IMG = 10000
# If overlap, the multi-objects will be overlapped in a image.
__C.OVERLAP = False
# If Repeat, repetitive labels will appear in a image.
__C.REPEAT = False
```

配置模型架构

在 `model_arch.py` 中配置模型架构，方法和Keras类似：

需要重点注意几点：

1. 请注意通道格式是'NCHW‘还是'NHWC'，如果使用'NCHW‘，请注意添加 `NHWC2NCHW()` 。
2. 当迁移学习继承参数时，请在 `model.add()` 中填写 `weights` 和 `biases` 参数，如果只有一个，则只填一个，获取方式见示例代码。
3. loss和prediction的计算方法使用 `model.get_loss()` ，当然也可以自行添加，本质上输出都是tensor。
4. 注意相同的模块之间，idx需要不一样，否则无法计算。

示例：

```
  if cfg.CLF_LOSS == 'margin':
    clf_loss_fn = margin_loss
    clf_loss_params = cfg.MARGIN_LOSS_PARAMS
  elif cfg.CLF_LOSS == 'margin_h':
    clf_loss_fn = margin_loss_h
    clf_loss_params = cfg.MARGIN_LOSS_H_PARAMS
  else:
    raise ValueError('Wrong CLF_LOSS Name!')

  model = Sequential(inputs, verbose=True)
```

```
  # Pre-training Model
  # ================================================================================
=
  if restore_vars_dict is None:

    with tf.variable_scope('classifier'):

      if cfg.DATA_FORMAT == 'NCHW':
        model.add(NHWC2NCHW())

      model.add(Conv(
          cfg,
          kernel_size=9,
          stride=1,
          n_kernel=256,
          padding='VALID',
          act_fn='relu',
          idx=0
      ))
      model.add(ConvSlimCapsule(
          cfg,
          output_dim=32,
          output_atoms=8,
          kernel_size=9,
          stride=2,
          padding='VALID',
          conv_act_fn='relu',
          caps_act_fn='squash',
          idx=0
      ))
      model.add(Capsule(
          cfg,
          output_dim=num_class,
          output_atoms=16,
          num_routing=3,
          routing_method='v1',
          act_fn='squash',
          use_bias=False,
          share_weights=False,
          add_grads_stop=True,
          idx=1
      ))
```

```
      model.add_name('clf_logits')

      clf_loss, clf_preds = model.get_loss(
          clf_loss_fn, labels, **clf_loss_params)
      clf_loss = tf.identity(clf_loss, name='clf_loss')
      clf_preds = tf.identity(clf_preds, name='clf_preds')

    with tf.variable_scope('reconstruction'):

      model.add(Mask(labels))
      model.add(Dense(
          cfg,
          output_dim=512,
          act_fn='relu',
          idx=0))
      model.add(Dense(
          cfg,
          output_dim=1024,
          act_fn='relu',
          idx=1))
      model.add(Dense(
          cfg,
          output_dim=cfg.IMAGE_SIZE[0] * cfg.IMAGE_SIZE[1],
          act_fn=None,
          idx=2))
      model.add_name('rec_logits')

      rec_loss_params = {
        'decoder_type': 'fc',
        'rec_loss_type': 'mse'
      }
      rec_loss, rec_imgs = model.get_loss(
          reconstruction_loss, input_imgs, **rec_loss_params)
      rec_loss = tf.identity(rec_loss, name='rec_loss')
      rec_imgs = tf.identity(rec_imgs, name='rec_imgs')

    loss = clf_loss + cfg.REC_LOSS_SCALE * rec_loss
    loss = tf.identity(loss, name='loss')

  # Fine-tuning Model
  # ============================================================================
=
  else:
```

```
97
98      w_conv_0 = restore_vars_dict['w_conv_0']
99      b_conv_0 = restore_vars_dict['b_conv_0']
100     w_caps_0 = restore_vars_dict['w_caps_0']
101     b_caps_0 = restore_vars_dict['b_caps_0']
102     w_caps_1 = restore_vars_dict['w_caps_1']
103     b_caps_1 = restore_vars_dict['b_caps_1']
104
105     with tf.variable_scope('classifier'):
106
107       if cfg.DATA_FORMAT == 'NCHW':
108         model.add(NHWC2NCHW())
109       model.add(Conv(
110           cfg,
111           kernel_size=9,
112           stride=1,
113           n_kernel=256,
114           padding='VALID',
115           act_fn='relu',
116           idx=0
117       ), weights=w_conv_0, biases=b_conv_0)
118       model.add(ConvSlimCapsule(
119           cfg,
120           output_dim=32,
121           output_atoms=8,
122           kernel_size=9,
123           stride=2,
124           padding='VALID',
125           conv_act_fn='relu',
126           caps_act_fn='squash',
127           idx=0
128       ), weights=w_caps_0, biases=b_caps_0)
129       model.add(Capsule(
130           cfg,
131           output_dim=10,
132           output_atoms=16,
133           num_routing=3,
134           routing_method='v1',
135           act_fn='squash',
136           use_bias=False,
137           share_weights=False,
138           add_grads_stop=True,
139           idx=1
```

```
        ), weights=w_caps_1, biases=b_caps_1)
        model.add(Capsule(
            cfg,
            output_dim=num_class,
            output_atoms=16,
            num_routing=3,
            routing_method='v1',
            act_fn='squash',
            use_bias=False,
            share_weights=False,
            add_grads_stop=True,
            idx=2
        ))
        model.add_name('clf_logits')

        clf_loss, clf_preds = model.get_loss(
            clf_loss_fn, labels, **clf_loss_params)
        clf_loss = tf.identity(clf_loss, name='clf_loss')
        clf_preds = tf.identity(clf_preds, name='clf_preds')

    with tf.variable_scope('reconstruction'):

        model.add(Mask(labels))
        model.add(Dense(
            cfg,
            output_dim=512,
            act_fn='relu',
            idx=0))
        model.add(Dense(
            cfg,
            output_dim=1024,
            act_fn='relu',
            idx=1))
        model.add(Dense(
            cfg,
            output_dim=cfg.IMAGE_SIZE[0] * cfg.IMAGE_SIZE[1],
            act_fn=None,
            idx=2))
        model.add_name('rec_logits')

        rec_loss_params = {
          'decoder_type': 'fc',
          'rec_loss_type': 'mse'
```

```
        }
        rec_loss, rec_imgs = model.get_loss(
            reconstruction_loss, input_imgs, **rec_loss_params)
        rec_loss = tf.identity(rec_loss, name='rec_loss')
        rec_imgs = tf.identity(rec_imgs, name='rec_imgs')

    loss = clf_loss + cfg.REC_LOSS_SCALE * rec_loss
    loss = tf.identity(loss, name='loss')
```

模型架构可以使用 `capsule_layers.py` 和 `layers.py` 中预先写好的一些模型，包括:

> *Dense // Single full-connected layer*
> *Conv // Single convolution layer*
> *ConvT // Single transpose convolution layer*
> *MaxPool // Max Pooling layer*
> *AveragePool // Average Pooling layer*
> *GlobalAveragePool // Global Average Pooling layer*
> *BatchNorm // Batch normalization layer*
> *Reshape // Reshape a tenso*
> *NHWC2NCHW // Convert a NHWC tensor to NCHW tensor*
> *NCHW2NHWC // Convert a NCHW tensor to NHWC tensor*
> *Capsule // Capsule Layer with dynamic routing*
> *ConvSlimCapsule // Generate a Capsule layer using convolution kernel*
> *CapsuleV2 // Capsule layer version 2.0*
> *ConvSlimCapsuleV2 // A slim convolutional capsule layer*
> *Mask // Get masked tensor*
> *Capsule5Dto3D // Convert a capsule output tensor to 3D tens*
> *Capsule4Dto5D // Convert a conv2d output tensor to 5D tensor*

此外，模型架构中的一些参数可以在 `config.py` 中设置:

```
# ------------------------------------------
# Classification

# Classification loss
# 'margin': margin loss
# 'margin_h': margin loss in Hinton's paper
__C.CLF_LOSS = 'margin'
```

```
8
9  # Parameters of margin loss
10 # default: {'m_plus': 0.9, 'm_minus': 0.1, 'lambda_': 0.5}
11 __C.MARGIN_LOSS_PARAMS = {'m_plus': 0.9,
12                           'm_minus': 0.1,
13                           'lambda_': 0.5}
14 # default: {'margin': 0.4, 'down_weight': 0.5}
15 __C.MARGIN_LOSS_H_PARAMS = {'margin': 0.4,
16                            'down_weight': 0.5}
17
18 # ----------------------------------------
19 # Optimizer and learning rate decay
20
21 # Optimizer
22 # 'gd': GradientDescentOptimizer()
23 # 'adam': AdamOptimizer()
24 # 'momentum': MomentumOptimizer()
25 __C.OPTIMIZER = 'adam'
26
27 # Momentum Optimizer
28 # Boundaries of learning rate
29 __C.LR_BOUNDARIES = [82, 123, 300]
30 # Stage of learning rate
31 __C.LR_STAGE = [1, 0.1, 0.01, 0.002]
32 # Momentum parameter of momentum optimizer
33 __C.MOMENTUM = 0.9
34
35 # ----------------------------------------
36 # Reconstruction
37
38 # Training with reconstruction
39 __C.WITH_REC = True
40
41 # Type of decoder of reconstruction:
42 # 'fc': full_connected layers
43 # 'conv': convolution layers
44 # 'conv_t': transpose convolution layers
45 __C.DECODER_TYPE = 'fc'
46
47 # Reconstruction loss
48 # 'mse': Mean Square Error
49 # 'ce' : sigmoid_cross_entropy_with_logits
50 __C.REC_LOSS = 'mse'
```

```
51
52  # Scaling for reconstruction loss
53  __C.REC_LOSS_SCALE = 0.392  # 0.0005*32*32=0.512  # 0.0005*784=0.392
54
55  # ----------------------------------------
56  # Transfer Learning
57
58  # Transfer learning mode
59  __C.TRANSFER_LEARNING = None  # 'encode'  # None
60
61  # Transfer learning model
62  # 'vgg16', 'vgg19', 'resnet50', 'inceptionv3', 'xception'
63  __C.TL_MODEL = 'xception'
64
65  # Pooling method: 'avg', None
66  __C.BF_POOLING = None
```

模型训练

```
1  python main.py
```

> *运行时可选参数:*
> *-h, --help show this help message and exit*
> *-g , --gpu Run single-gpu version.Choose the GPU from: [0, 1]*
> *-bs , --batch_size Set batch size.*
> *-tn , --task_number Set task number.*
> *-m, --mgpu Run multi-gpu version.*
> *-t, --mtask Run multi-tasks version.*
> *-b, --baseline Use baseline architecture and configurations.*
> *-ht, --hinton Use architecture and configurations of Hinton.*
> *-ft, --fine_tune Fine-tuning.*
> *-p, --pipeline Pipeline of training and fine-tuning.*

训练流程分为两步：

1. 通过下列语句预训练模型，此时使用 `pipeline_config.py` ：

```
python main.py -m
```

2. 通过一下语句进行fine-tuning训练，此时使用 `pipeline_config_fine_tune.py` ：

```
python main.py -ft -m
```

训练时注意调整batch_size大小，否则会内存溢出。
训练时的参数在 `config.py` 中配置：

模型训练超参数

```
# Database name
# 'mnist': MNIST
# 'cifar10' CIFAR-10
__C.DATABASE_NAME = 'mnist'
# __C.DATABASE_MODE = 'small_no_pool_56_56'
# __C.DATABASE_MODE = 'small'
__C.DATABASE_MODE = None

# Training version
# Set None to auto pipeline version
__C.VERSION = None

# Learning rate
__C.LEARNING_RATE = 0.001

# Learning rate with exponential decay
# Use learning rate decay
__C.LR_DECAY = False
# Decay steps
__C.LR_DECAY_STEPS = 2000
# Exponential decay rate
__C.LR_DECAY_RATE = 0.96

# Epochs
__C.EPOCHS = 20

# Batch size
__C.BATCH_SIZE = 512
```

```
29
30  # Data format
31  # 'NCHW': (batch, channel, height, width)
32  # 'NHWC': (batch, height, width, channel)
33  __C.DATA_FORMAT = 'NHWC'
```

训练过程流程和显示信息设置

```
1   # Display step
2   # Set None to not display details
3   __C.DISPLAY_STEP = None  # batches
4
5   # Save summary step
6   # Set None to not save summaries
7   __C.SAVE_LOG_STEP = 100  # batches
8
9   # Save reconstructed images
10  # Set None to not save images
11  __C.SAVE_IMAGE_STEP = 100  # batches
12
13  # Maximum images number in a col
14  __C.MAX_IMAGE_IN_COL = 10
15
16  # Calculate train loss and valid loss using full data set
17  # 'per_epoch': evaluate on full set when n epochs finished
18  # 'per_batch': evaluate on full set when n batches finished
19  __C.FULL_SET_EVAL_MODE = 'per_epoch'
20  # None: not evaluate
21  __C.FULL_SET_EVAL_STEP = 1
22
23  # Save models
24  # 'per_epoch': save models when n epochs finished
25  # 'per_batch': save models when n batches finished
26  # __C.SAVE_MODEL_MODE = None
27  __C.SAVE_MODEL_MODE = 'per_epoch'
28  # None: not save models
29  __C.SAVE_MODEL_STEP = 5
30  # Maximum number of recent checkpoints to keep.
31  __C.MAX_TO_KEEP_CKP = 3
32
33  # Calculate the train loss of full data set, which may take lots of time.
```

```
__C.EVAL_WITH_FULL_TRAIN_SET = False

# ----------------------------------------
# Test
# 'after_training': evaluate after all training finished
# 'per_epoch': evaluate when a epoch finished
# None: Do not test

# Evaluate on single-object test set
__C.TEST_SO_MODE = 'per_epoch'  # 'after_training'

# Evaluate on multi-objects test set
__C.TEST_MO_MODE = None  # 'per_epoch'
```

多显卡分布式计算相关设置

```
# Save trainable variables on CPU
__C.VAR_ON_CPU = True

# Number of GPUs
__C.GPU_NUMBER = 2

# Number of multi-tasks
__C.TASK_NUMBER = 16

# The decay to use for the moving average.
# If None, not use
__C.MOVING_AVERAGE_DECAY = 0.9999
```

模型测试和评估

实际上，模型在训练结束后会根据设置自动进行计算和评估，但是也可以通过 `test.py` 自行测试，但是要注意读取的模型位置和模型编号。

```
python test.py
```

模型测试相关参数在 `config.py` 中设置，这些设置也会影响训练过程后的评估。

```
# Testing version name
__C.TEST_VERSION = __C.VERSION

# Testing checkpoint index
# If None, load the latest checkpoint.
__C.TEST_CKP_IDX = None

# Testing with reconstruction
__C.TEST_WITH_REC = True

# Saving testing reconstruction images
# If None, do not save images.
__C.TEST_SAVE_IMAGE_STEP = 5  # batches

# Batch size of testing
# should be same as training batch_size
__C.TEST_BATCH_SIZE = __C.BATCH_SIZE

# Top_N precision and accuracy
# If None, do not calculate Top_N.
__C.TOP_N_LIST = [1, 2, 5]

# ----------------------------------------
# Multi-objects detection

# Label for generating reconstruction images
# 'pred': Use predicted y
# 'real': Use real labels y
__C.LABEL_FOR_TEST = 'pred'  # 'real'

# Mode of prediction for multi-objects detection
# 'top_n': sort vectors, select longest n classes as y
# 'length_rate': using length rate of the longest vector class as threshold
__C.MOD_PRED_MODE = 'top_n'  # 'length_rate'

# Max number of prediction y
__C.MOD_PRED_MAX_NUM = 2

# Threshold for 'length_rate' mode
__C.MOD_PRED_THRESHOLD = 0.5

# Save test prediction vectors
```

```
__C.SAVE_TEST_PRED = True
```

Pipeline训练

Pipeline训练主要是用于长时间的放置训练，可以一次性跑多个模型，方法也很简单。

```
python pipeline.py
```

> *运行时可选参数:*
> *-h, --help show this help message and exit*
> *-g , --gpu Run single-gpu version.Choose the GPU from: [0, 1]*
> *-m, --mgpu Run multi-gpu version.*

将要修改的参数放在 `pipeline_config.py` 和 `pipeline_config_fine_tune.py` 的结尾就可以了。

```
__C.CAPS_USE_BIAS = False
__C.CAPS_SHARE_WEIGHTS = False
__C.CAPS_GRADS_STOP = True

cfg_0 = copy(__C)

cfg_1_ = copy(__C)
cfg_1_.DATA_FORMAT = 'NCHW'
cfg_1_.VERSION = 'nchw'
cfg_1 = cfg_1_

cfg_2_ = copy(__C)
cfg_2_.LR_DECAY = True
cfg_2_.VERSION = 'lr_decay'
cfg_2 = cfg_2_

cfg_3_ = copy(__C)
cfg_3_.CLF_LOSS = 'margin_h'
cfg_3_.VERSION = 'margin_h'
cfg_3 = cfg_3_

cfg_4_ = copy(__C)
cfg_4_.REC_LOSS_SCALE = 0.0005
cfg_4_.VERSION = 'scale_00005'
```

```
cfg_4 = cfg_4_

cfg_5_ = copy(__C)
cfg_5_.CAPS_USE_BIAS = True
cfg_5_.VERSION = 'caps_use_bias'
cfg_5 = cfg_5_

cfg_6_ = copy(__C)
cfg_6_.CAPS_SHARE_WEIGHTS = True
cfg_6_.VERSION = 'caps_share_weights'
cfg_6 = cfg_6_

cfg_7_ = copy(__C)
cfg_7_.CAPS_GRADS_STOP = False
cfg_7_.VERSION = 'caps_add_grads_stop'
cfg_7 = cfg_7_

cfg_8_ = copy(__C)
cfg_8_.REC_LOSS_SCALE = 0.25
cfg_8_.VERSION = 'scale_025'
cfg_8 = cfg_8_
```

其他设置

一些目录设置在 `config.py` 的结尾处。

```
# Source data directory path
__C.SOURCE_DATA_PATH = '../data/source_data'

# Preprocessed data path
__C.DPP_DATA_PATH = '../data/preprocessed_data'

# Path for saving logs
__C.TRAIN_LOG_PATH = '../train_logs'

# Path for saving summaries
__C.SUMMARY_PATH = '../tf_logs'

# Path for saving models
__C.CHECKPOINT_PATH = '../checkpoints'

```

```
# Path for saving testing logs
__C.TEST_LOG_PATH = '../test_logs'
```