Classifying Food Images by Small-Sample Learning

About the Contest

With Artificial Intelligence now widely applied across a growing range of scenarios, one common finding is that samples prove insufficient for training an artificial intelligence model that can recognize objects in images. In response, training small-sample learning models has become an important research area. In this contest, you are expected to train a small-sample learning model from a large number of images with known characteristics, effectively extract characteristics from new images, and classify these new images into their correct categories.

Datasets

The dataset consists of food images in various categories, such as Chinese food, western food, dessert, porridge, and pastries. The food occupies 3/4 of the total size of each image, and each image falls into one food category.

Sample images:

We will provide two datasets for you: a training dataset containing 75 categories with 1000 images per category and a small-sample dataset containing 25 categories with 5 images per category. You can use only the provided datasets for training and algorithm adjustment and are not permitted to use other datasets.

After decompressing the dataset file aifood.zip (2.1GB), you will obtain the following structure:

```
aifood (root directory)
    |- images (storing images)
        |- sandwich (a food category, with each sub-directory having the same name
as the category label)
            |- 43550.jpg (image file)
            |- 43550.jpg (image file)
            |- … (more images)
        |- Mung bean cake (small-sample category with five images)
            |- 3402.jpg (image file)
            … (four more image files)
        … (more categories)
    |- meta (storing meta)
        |- large_samples_75c.txt (large-sample images of 75 categories with each
image file occupying one line)
```

```
        |- small_samples_25c.txt (small-sample images of 25 categories with each
image file occupying one line)
        |- large_labels_75c.txt (labels of large-sample images with each label
occupying one line)
        |- small_labels_25c.txt (labels of small-sample images with each label
occupying one line)
```

Evaluation

This contest uses the recognition accuracy as the evaluation standard. For each image, the model should calculate the confidence scores for each category listed in the small_labels_25c.txt. The category with the highest confidence score is taken as the prediction result.

The contest auto scoring system will use an evaluation dataset (which is not visible to contesters) to calculate the average recognition accuracy as below:
Average recognition accuracy = (total number of correctly predicted images) / (total number of images in the evaluation dataset). The higher the score, the higher the ranking.

Model Specifications

Please follow the guidance below to ensure your models can be scored correctly by the contest auto scoring system.

1. Use TensorFlow, PyTorch, or MxNet to build your models.
2. The trained models must comply with the following specifications:
    - Input parameters (values returned by _preprocess: dict {'images': ndarray whose shape is [1, ?, ?, 3] indicates an image}
    - Output parameters (values returned by _postprocess): dict {'logits': {'Mung bean cake': 0.1, 'French Fries': -0.12, 'Onion rings': 0.72, ...}}
3. Prepare your model files as packages and import them into ModelArts. Please refer to the detail guidance below “Guidance: How to prepare and import Models to ModelArts”
4. Submit you models via ModelArts Model sharing functionality. Please refer to the detail guidance below “Model Submission”.

Guidance: How to prepare and import Models to ModelArts

This section will provide the detail guidance of how to prepare and import models to ModelArts, including the sample codes. For more details about the import process, please check the product document https://support.huaweicloud.com/en-us/clientogw-obs/obs_03_0064.html .

1. **TensorFlow models**

To import models, you need to first upload your model files into Huawei Cloud Storage Service:

OBS, and follow the directory structure below to prepare your model package.

```
/bucket/dir (your OBS bucket and your directory to store model files)
    |- model (must follow this directory name)
        |- variables (trained TF model file)
            |- variables.data-00000-of-00001 (trained TF model file)
            |- variables.index (trained model file)
        |- saved_model.pb (trained TF model file)
        |- config.json (configuration file required by the scoring system. Must keep the same file name. A sample file will be provided for your convenience.)
        |- customize_service.py (inference code required by the scoring system. Must keep the same file name. A sample file will be provided for your convenience.)
        |- small_labels_25c.txt (label file)
```

***Config.json Reference Code*:**

```
{
  //Model algorithm is image classification
  "model_algorithm": "image_classification",
  //Model type is TensorFlow
  "model_type": "TensorFlow",
  //Model reference API, including the protocol, url, method, request API, and response API
  "apis": [
    {
      "procotol": "http",
      "url": "/",
      "method": "post",
      //Content type is multipart/form-data and key value is 'images'
      "request": {
        "Content-type": "multipart/form-data",
        "data": {
          "type": "object",
          "properties": {
            "images": {
              "type": "file"
            }
          }
        }
      },
      //Response format is json {'logits':{'label_a': 0.1, 'label_b': -0.12, 'label_c': 0.72}}
      "response": {
```

```
        "Content-type": "applicaton/json",
        "data": {
          "type": "object",
          "properties": {
            "logits": {
              "type": "object",
              "properties": {
                "label_a": {
                  "type": "number"
                },
                "label_b": {
                  "type": "number"
                },
                "label_c": {
                  "type": "number"
                }
              }
            }
          }
        }
      }
    }
  ],
  //Model accuracy. It is for your own track of your different version of models. Not used for contest scoring system.
  "metrics": {
    "f1": 0.102058,
    "recall": 0.9975,
    "precision": 0.05378,
    "accuracy": 1
  }
}
```

customize_service.py reference Code:

```
# Copyright 2019 ModelArts Service of Huawei Cloud. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
```

```
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import numpy as np
from PIL import Image
from model_service.tfserving_model_service import TfServingBaseService



IMAGES_KEY = 'images'
MODEL_INPUT_KEY = 'images'
MODEL_OUTPUT_KEY = 'logits'
LABELS_FILE_NAME = 'small_labels_25c.txt'

def decode_image(file_content):
  """
  Decode bytes to a single image
  :param file_content: bytes
  :return: ndarray with rank=3
  """
  image = Image.open(file_content)
  image = image.convert('RGB')
  image = np.asarray(image, dtype=np.float32)
  return image

def read_label_list(path):
  """
  read label list from path
  :param path: a path
  :return: a list of label names like: ['label_a', 'label_b', ...]
  """
  with open(path, 'r') as f:
    label_list = f.read().split(os.linesep)
  label_list = [x.strip() for x in label_list if x.strip()]
  return label_list

class FoodPredictService(TfServingBaseService):
```

```
  def _preprocess(self, data):
    """
    `data` is provided by Upredict service according to the input data. Which is
like:
      {'images': {'image_a.jpg': b'xxx'}}

    For now, predict a single image at a time.

    """
    images = []
    for file_name, file_content in data[IMAGES_KEY].items():
      print('\tAppending image: %s' % file_name)
      images.append(decode_image(file_content))

    preprocessed_data = {MODEL_INPUT_KEY: np.asarray(images)}
    return preprocessed_data

  def _postprocess(self, data):
    """
    `data` is the result of your model. Which is like:
      {'logits': [[0.1, -0.12, 0.72, ...]]}

    value of logits is a single list of list because one image is predicted at a
time for now.
    """

    # label_list = ['label_a', 'label_b', 'label_c', ...]
    label_list = read_label_list(os.path.join(self.model_path, LABELS_FILE_NAME))

    # logits_list = [0.1, -0.12, 0.72, ...]
    logits_list = data[MODEL_OUTPUT_KEY][0]

    # labels_to_logits = {'label_a': 0.1, 'label_b': -0.12, 'label_c': 0.72, ...}
    labels_to_logits = {label_list[i]: s for i, s in enumerate(logits_list)}

    predict_result = {MODEL_OUTPUT_KEY: labels_to_logits}
    return predict_result
```

2. **MxNet models**

To import models, you need to first upload your model files into Huawei Cloud Storage Service: OBS, and follow the directory structure below to prepare your model package.

```
/bucket/dir (your OBS bucket and your directory to store model files)
    |- model (must follow this directory name)
        |- resnet-50-0000.params (MxNet trained model file)
        |- resnet-50-symbol.json (MxNet trained model file)
        |- config.json (configuration file required by the scoring system. Must keep the same file name. A sample file will be provided for your convenience.)
        |- customize_service.py (inference code required by the scoring system. Must keep the same file name. A sample file will be provided for your convenience.)
        |- small_labels_25c.txt (label file)
```

config.json Reference Code:

```
{
  //Model algorithm is image classification
  "model_algorihm": "image_classification",
  //Model type is MXNet
  "model_type": "MXNet",
  //Model reference API, including the protocol, url, method, request interface, and response interface
  "apis": [
    {
      "procotol": "http",
      "url": "/",
      "method": "post",
      //Request is uploading images and key value is 'images'
      "request": {
        "Content-type": "multipart/form-data",
        "data": {
          "type": "object",
          "properties": {
            "images": {
              "type": "file"
            }
          }
        }
      },
      //Response format is json {'logits':{'label_a': 0.1, 'label_b': -0.12, 'label_c': 0.72}}
      "response": {
        "Content-type": "applicaton/json",
        "data": {
          "type": "object",
          "properties": {
```

```
            "logits": {
              "type": "object",
              "properties": {
                "label_a": {
                  "type": "number"
                },
                "label_b": {
                  "type": "number"
                },
                "label_c": {
                  "type": "number"
                }
              }
            }
          }
        }
      }
    }
  ],
  //Model accuracy. It is for your own track of your different version of models. Not used for contest scoring system.
  "metrics": {
    "f1": 0.102058,
    "recall": 0.9975,
    "precision": 0.05378,
    "accuracy": 1
  }
}
```

customize_service.py Reference Code:

```
# See the License for the specific language governing permissions and
# limitations under the License.

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import numpy as np
from PIL import Image
from mms.model_service.mxnet_model_service import MXNetBaseService

IMAGES_KEY = 'images'
```

```
MODEL_INPUT_KEY = 'images'
MODEL_OUTPUT_KEY = 'logits'
LABELS_FILE_NAME = 'small_labels_25c.txt'

def decode_image(file_content):
    """
    Decode bytes to a single image
    :param file_content: bytes
    :return: ndarray with rank=3
    """
    image = Image.open(file_content)
    image = image.convert('RGB')
    image = np.asarray(image, dtype=np.float32)
    return image

def read_label_list(path):
    """
    read label list from path
    :param path: a path
    :return: a list of label names like: ['label_a', 'label_b', ...]
    """
    with open(path, 'r') as f:
        label_list = f.read().split(os.linesep)
    label_list = [x.strip() for x in label_list if x.strip()]
    return label_list

class FoodPredictService(MXNetBaseService):
    def _preprocess(self, data):
        """
        `data` is provided by Upredict service according to the input data. Which is
like:
          {'images': {'image_a.jpg': b'xxx'}}

        For now, predict a single image at a time.

        """
        images = []
        for file_name, file in enumerate(data):
            print('\tAppending image: %s' % file_name)
            images.append(decode_image(file))

        return images
```

```
    def _postprocess(self, data):
        """
        `data` is the result of your model. Which is like:
          {'logits': [[0.1, -0.12, 0.72, ...]]}

        value of logits is a single list of list because one image is predicted at a
time for now.
        """

        # label_list = ['label_a', 'label_b', 'label_c', ...]
        label_list = read_label_list(os.path.join(self.model_path,
LABELS_FILE_NAME))

        # logits_list = [0.1, -0.12, 0.72, ...]
        logits_list = data[0]

        # labels_to_logits = {'label_a': 0.1, 'label_b': -0.12, 'label_c':
0.72, ...}
        labels_to_logits = {label_list[i]: s for i, s in enumerate(logits_list)}

        predict_result = {MODEL_OUTPUT_KEY: labels_to_logits}
        return predict_result
```

3. PyTorch models

To import models, you need to first upload your model files into Huawei Cloud Storage Service: OBS, and follow the directory structure below to prepare your model package.

```
/bucket/dir (your OBS bucket and your directory to store model files)
    |- model (must follow this directory name)
        |- resnet-50.pth (PyTorch trained model file)
        |- config.json (configuration file required by the scoring system. Must keep
the same file name. A sample file will be provided for your convenience.)
        |- customize_service.py (inference code required by the scoring system. Must
keep the same file name. A sample file will be provided for your convenience.)
        |- small_labels_25c.txt (label file)
```

config.json Reference Code:

```
{
  //Model algorithm is image classification
  "model_algorithm": "image_classification",
  //Model type is PyTorch
```

```
  "model_type": "PyTorch",
  //Model reference API, including the protocol, url, method, request interface, and
response interface
  "apis": [
    {
      "procotol": "http",
      "url": "/",
      "method": "post",
      //Request is uploading images, key value is 'images', and curl example is curl
-X POST http://{{endpoint}} -F images=@test.jpg
      "request": {
        "Content-type": "multipart/form-data",
        "data": {
          "type": "object",
          "properties": {
            "images": {
              "type": "file"
            }
          }
        }
      },
      //Response format is json {'logits':{'label_a': 0.1, 'label_b': -0.12,
'label_c': 0.72}}
      "response": {
        "Content-type": "applicaton/json",
        "data": {
          "type": "object",
          "properties": {
            "logits": {
              "type": "object",
              "properties": {
                "label_a": {
                  "type": "number"
                },
                "label_b": {
                  "type": "number"
                },
                "label_c": {
                  "type": "number"
                }
              }
            }
          }
```

```
        }
      }
    }
  ],
  //Model accuracy. It is for your own track of your different version of models.
Not used for contest scoring system.
  "metrics": {
    "f1": 0.102058,
    "recall": 0.9975,
    "precision": 0.05378,
    "accuracy": 1
  }
}
```

customize_service.py Reference Code:

```
# See the License for the specific language governing permissions and
# limitations under the License.

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import numpy as np
from PIL import Image
from model_service.pytorch_model_service import PTServingBaseService

import torch.nn as nn
import torch

import torchvision.transforms as transforms

infer_transformation = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

IMAGES_KEY = 'images'
MODEL_INPUT_KEY = 'images'
MODEL_OUTPUT_KEY = 'logits'
LABELS_FILE_NAME = 'small_labels_25c.txt'
```

```
def decode_image(file_content):
    """
    Decode bytes to a single image
    :param file_content: bytes
    :return: ndarray with rank=3
    """
    image = Image.open(file_content)
    image = image.convert('RGB')
    image = np.asarray(image, dtype=np.float32)
    return image


def read_label_list(path):
    """
    read label list from path
    :param path: a path
    :return: a list of label names like: ['label_a', 'label_b', ...]
    """
    with open(path, 'r') as f:
        label_list = f.read().split(os.linesep)
    label_list = [x.strip() for x in label_list if x.strip()]
    return label_list


class FoodPredictService(PTServingBaseService):
    def __init__(self, model_name, model_path):
        global CLASS_INDEX
        super(FoodPredictService, self).__init__(model_name, model_path)
        self.model = resnet50(model_path)

    def _preprocess(self, data):
        """
        `data` is provided by Upredict service according to the input data. Which is
like:
          {'images': {'image_a.jpg': b'xxx'}}

        For now, predict a single image at a time.

        """
        images = []
        for file_name, file_content in data[IMAGES_KEY].items():
            print('\tAppending image: %s' % file_name)
            images.append(decode_image(file_content))
```

```
        preprocessed_data = {MODEL_INPUT_KEY: np.asarray(images)}
        return preprocessed_data

    def _postprocess(self, data):
        """
        `data` is the result of your model. Which is like:
          {'logits': [[0.1, -0.12, 0.72, ...]]}

        value of logits is a single list of list because one image is predicted at a
time for now.
        """

        # label_list = ['label_a', 'label_b', 'label_c', ...]
        label_list = read_label_list(os.path.join(self.model_path,
LABELS_FILE_NAME))

        # logits_list = [0.1, -0.12, 0.72, ...]
        logits_list = data[MODEL_OUTPUT_KEY][0]

        # labels_to_logits = {'label_a': 0.1, 'label_b': -0.12, 'label_c':
0.72, ...}
        labels_to_logits = {label_list[i]: s for i, s in enumerate(logits_list)}

        predict_result = {MODEL_OUTPUT_KEY: labels_to_logits}
        return predict_result


def conv3x3(in_planes, out_planes, stride=1):
    """3x3 convolution with padding"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                     padding=1, bias=False)


def conv1x1(in_planes, out_planes, stride=1):
    """1x1 convolution"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride,
bias=False)


class BasicBlock(nn.Module):
    expansion = 1
```

```
    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out


class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(Bottleneck, self).__init__()
        self.conv1 = conv1x1(inplanes, planes)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = conv3x3(planes, planes, stride)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = conv1x1(planes, planes * self.expansion)
        self.bn3 = nn.BatchNorm2d(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride
```

```
    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out

//Define the network for your model
class ResNet(nn.Module):

    def __init__(self, block, layers, num_classes=1000, zero_init_residual=False):
        super(ResNet, self).__init__()
        self.inplanes = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,
                               bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
```

```
                nn.init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

        # Zero-initialize the last BN in each residual branch,
        # so that the residual branch starts with zeros, and each residual block
behaves like an identity.
        # This improves the model by 0.2~0.3% according to
https://arxiv.org/abs/1706.02677
        if zero_init_residual:
            for m in self.modules():
                if isinstance(m, Bottleneck):
                    nn.init.constant_(m.bn3.weight, 0)
                elif isinstance(m, BasicBlock):
                    nn.init.constant_(m.bn2.weight, 0)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.Sequential(
                conv1x1(self.inplanes, planes * block.expansion, stride),
                nn.BatchNorm2d(planes * block.expansion),
            )

        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes * block.expansion
        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
```

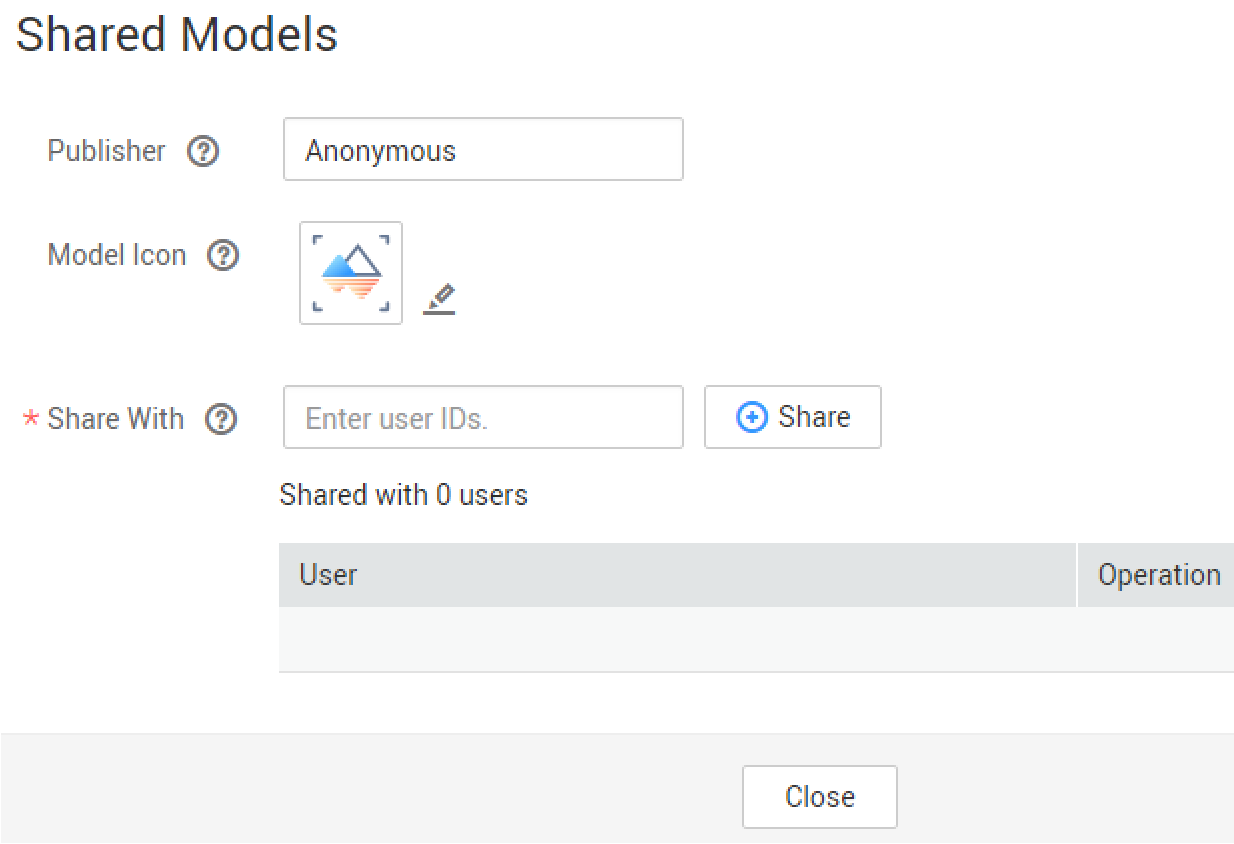
```
        x = self.layer4(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x

def resnet50(model_path, **kwargs):
    """Constructs a ResNet-50 model.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = ResNet(Bottleneck, [3, 4, 6, 3], **kwargs)

    model.load_state_dict(torch.load(model_path))

    model.eval()

    return model
```

Model Submission

After your models are imported to ModelArts, you could submit it to the contest platform to get the score. The submission is via ModelArts model sharing functionality. The submission could be accepted from 25th February, contest Admin will keep update the rankings and send to your group email in every 3 days from 26th February to 10th March.

1. On the Model Management page, click Share in the Operation column.
2. In the dialog box that is displayed, set
    - Publisher: It is optional. This info will not be used by the scoring system.
    - Model Icon: It is optional. This info will not be used by the scoring system.
    - **Share with**: **d8126a20db13499c82e060007d1e8348** . This is the contest evaluation account ID. Please ensure you type or copy it correct.

For more information about model sharing, visit https://support.huaweicloud.com/en-us/usermanual-modelarts/modelarts_02_0036.html.


Shared Models
Publisher
Anonymous
Model Icon
* Share With
Enter user IDs.
Share
Shared with 0 users
User
Operation
Close