Homework 7 - Berkeley STAT 157

Your name: XX, SID YY, teammates A,B,C (Please add your name, SID and teammates to ease Ryan and Rachel to grade.)

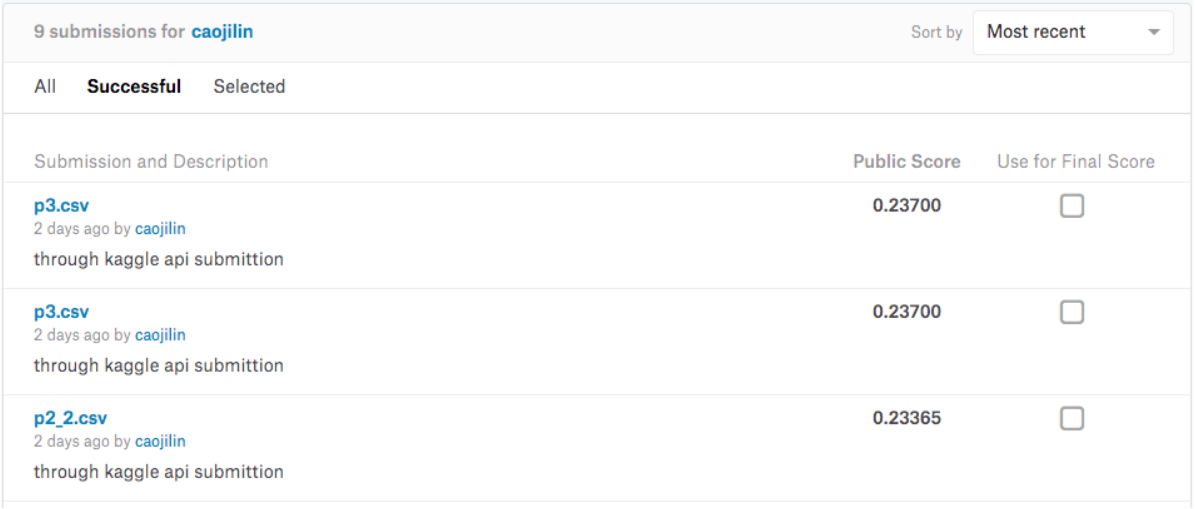
Please submit your homework through gradescope (http://gradescope.com/)

Handout 4/2/2019, due 4/9/2019 by 4pm.

This homework deals with fine-tuning for computer vision. In this task, we attempt to identify 120 different breeds of dogs. The data set used in this competition is actually a subset of the ImageNet data set. Different from the images in the CIFAR-10 data set used in the previous homework, the images in the ImageNet data set are higher and wider and their dimensions are inconsistent. Again, you need to use GPU.

The dataset is available at Kaggle (https://www.kaggle.com/c/dog-breed-identification). The rule is similar to homework 6:

- work as a team
- submit your results into Kaggle
- take a screen shot of your best score and insert it below
- the top 3 teams/individuals will be awarded with 500 dollar AWS credits



First, import the packages or modules required for the competition.

In [0]:
```
!pip install d2l mxnet-cu100
```

```
Collecting d2l
  Downloading https://files.pythonhosted.org/packages/40/2b/618811a6331dc
0cbb5d9731959f0c2b1b63bc1297c24401a3d7076e05624/d2l-0.8.7.tar.gz (http
s://files.pythonhosted.org/packages/40/2b/618811a6331dc0cbb5d9731959f0c2b
1b63bc1297c24401a3d7076e05624/d2l-0.8.7.tar.gz)
Collecting mxnet-cu100
  Downloading https://files.pythonhosted.org/packages/ae/36/40b6d201b4649
5513f7a7fa25fe8b7d85b3602a22efba119e8146d5f1601/mxnet_cu100-1.4.0.post0-p
y2.py3-none-manylinux1_x86_64.whl (https://files.pythonhosted.org/package
s/ae/36/40b6d201b46495513f7a7fa25fe8b7d85b3602a22efba119e8146d5f1601/mxne
t_cu100-1.4.0.post0-py2.py3-none-manylinux1_x86_64.whl) (487.9MB)
    100% |████████████████████████████████| 487.9MB 33kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-pac
kages (from d2l) (1.14.6)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dis
t-packages (from d2l) (3.0.3)
Requirement already satisfied: jupyter in /usr/local/lib/python3.6/dist-p
ackages (from d2l) (1.0.0)
Collecting graphviz<0.9.0,>=0.8.1 (from mxnet-cu100)
```

In [0]:
```
import collections
import d2l
import math
from mxnet import autograd, gluon, init, nd
from mxnet.gluon import model_zoo, nn
from mxnet.gluon import data as gdata, loss as gloss, utils as gutils
import os
import shutil
import time
import zipfile
import mxnet as mx
```

In [0]:
```
!rm -r kaggle_dog
```

In [0]:
```
!mkdir kaggle_dog
```

In [0]:
```
!cp ../gdrive/My\ Drive/labels.csv.zip ./kaggle_dog
!cp ../gdrive/My\ Drive/test.zip ./kaggle_dog
!cp ../gdrive/My\ Drive/train.zip ./kaggle_dog
```

In [0]:
```
# !cp ../gdrive/My\ Drive/stanford_lables.csv ./kaggle_dog
```

```
cp: cannot stat '../gdrive/My Drive/stanford_lables.csv': No such file or
directory
```

In [0]:
```
from google.colab import drive
drive.mount('/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?cl
ient_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercont
ent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20ht
tps%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.goo
gleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdr
ive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleap
i.readonly&response_type=code (https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuserco
ntent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20
https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.g
oogleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2F
drive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeoplea
pi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /gdrive
```

In [0]:
```
!mkdir standford
```

In [0]:
```
!cp ../gdrive/My\ Drive/Images.zip ./standford
```

Obtain and Organize the Data Sets

The competition data is divided into a training set and testing set. The training set contains 10,222 images and the testing set contains 10,357 images. The images in both sets are in JPEG format. These images contain three RGB channels (color) and they have different heights and widths. There are 120 breeds of dogs in the training set, including Labradors, Poodles, Dachshunds, Samoyeds, Huskies, Chihuahuas, and Yorkshire Terriers.

Download the Data Set

After logging in to Kaggle, we can click on the "Data" tab on the dog breed identification competition webpage shown in Figure 9.17 and download the training data set "train.zip", the testing data set "test.zip", and the training data set labels "label.csv.zip". After downloading the files, place them in the three paths below:

- kaggle_dog/train.zip
- kaggle_dog/test.zip
- kaggle_dog/labels.csv.zip

To make it easier to get started, we provide a small-scale sample of the data set mentioned above, "train_valid_test_tiny.zip". If you are going to use the full data set for the Kaggle competition, you will also need to change the `demo` variable below to `False`.

In [0]:
```
# If you use the full data set downloaded for the Kaggle competition,
# change the variable below to False.
demo = False
data_dir = './kaggle_dog'
if demo:
    if not os.path.exists(data_dir):
        os.mkdir(data_dir)
    gutils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kaggl
                    data_dir)
    zipfiles = ['train_valid_test_tiny.zip']
else:
    zipfiles = ['train.zip', 'test.zip', 'labels.csv.zip']
#     zipfiles = ['Images.zip']
for f in zipfiles:
    with zipfile.ZipFile(data_dir + '/' + f, 'r') as z:
        z.extractall(data_dir)
```

In [0]:
```
# If you use the full data set downloaded for the Kaggle competition,
# change the variable below to False.
demo = False
data_dir = './standford'
if demo:
    if not os.path.exists(data_dir):
        os.mkdir(data_dir)
    gutils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kaggl
                    data_dir)
    zipfiles = ['train_valid_test_tiny.zip']
else:
#     zipfiles = ['train.zip', 'test.zip', 'labels.csv.zip']
    zipfiles = ['Images.zip']
for f in zipfiles:
    with zipfile.ZipFile(data_dir + '/' + f, 'r') as z:
        z.extractall(data_dir)
```

In [0]:
```
def reorg_train_valid(data_dir, train_dir, input_dir, valid_ratio, idx_labe
    # The number of examples of the least represented breed in the training
    min_n_train_per_label = (
        collections.Counter(idx_label.values()).most_common()[:-2:-1][0][1]
    # The number of examples of each breed in the validation set.
    n_valid_per_label = math.floor(min_n_train_per_label * valid_ratio)
    label_count = {}
    for train_file in os.listdir(os.path.join(data_dir, train_dir)):
        idx = train_file.split('.')[0]
        label = idx_label[idx]
        d2l.mkdir_if_not_exist([data_dir, input_dir, 'train_valid', label])
        shutil.copy(os.path.join(data_dir, train_dir, train_file),
                    os.path.join(data_dir, input_dir, 'train_valid', label)
        if label not in label_count or label_count[label] < n_valid_per_lab
            d2l.mkdir_if_not_exist([data_dir, input_dir, 'valid', label])
            shutil.copy(os.path.join(data_dir, train_dir, train_file),
                        os.path.join(data_dir, input_dir, 'valid', label))
            label_count[label] = label_count.get(label, 0) + 1
        else:
            d2l.mkdir_if_not_exist([data_dir, input_dir, 'train', label])
            shutil.copy(os.path.join(data_dir, train_dir, train_file),
                        os.path.join(data_dir, input_dir, 'train', label))
```

The `reorg_dog_data` function below is used to read the training data labels, segment the validation set, and organize the training set.

In [0]:
```
def reorg_dog_data(data_dir, label_file, train_dir, test_dir, input_dir,
                   valid_ratio):
    # Read the training data labels.
    with open(os.path.join(data_dir, label_file), 'r') as f:
        # Skip the file header line (column name).
        lines = f.readlines()[1:]
        tokens = [l.rstrip().split(',') for l in lines]
        idx_label = dict(((idx, label) for idx, label in tokens))
    reorg_train_valid(data_dir, train_dir, input_dir, valid_ratio, idx_labe
    # Organize the training set.
    d2l.mkdir_if_not_exist([data_dir, input_dir, 'test', 'unknown'])
    for test_file in os.listdir(os.path.join(data_dir, test_dir)):
        shutil.copy(os.path.join(data_dir, test_dir, test_file),
                    os.path.join(data_dir, input_dir, 'test', 'unknown'))
```

Because we are using a small data set, we set the batch size to 1. During actual training and testing, we would use the entire Kaggle Competition data set and call the `reorg_dog_data` function to organize the data set. Likewise, we would need to set the `batch_size` to a larger integer, such as 128.

In [0]:
```
demo = False
if demo:
    # Note: Here, we use a small data set and the batch size should be set
    # smaller. When using the complete data set for the Kaggle competition,
    # we can set the batch size to a larger integer.
    input_dir, batch_size = 'train_valid_test_tiny', 1
else:
    label_file, train_dir, test_dir = 'labels.csv', 'train', 'test'
    input_dir, batch_size, valid_ratio = 'train_valid_test', 128, 0.1
    reorg_dog_data(data_dir, label_file, train_dir, test_dir, input_dir,
                   valid_ratio)
```

Image Augmentation

The size of the images in this section are larger than the images in the previous section. Here are some more image augmentation operations that might be useful.

In [0]:
```
transform_train = gdata.vision.transforms.Compose([
    # Randomly crop the image to obtain an image with an area of 0.08 to 1
    # of the original area and height to width ratio between 3/4 and 4/3.
    # Then, scale the image to create a new image with a height and width
    # of 224 pixels each.
    gdata.vision.transforms.RandomResizedCrop(224, scale=(0.08, 1.0),
                                              ratio=(3.0/4.0, 4.0/3.0)),
    gdata.vision.transforms.RandomFlipLeftRight(),
    # Randomly change the brightness, contrast, and saturation.
    gdata.vision.transforms.RandomColorJitter(brightness=0.4, contrast=0.4,
                                              saturation=0.4),
    # Add random noise.
    gdata.vision.transforms.RandomLighting(0.1),
    gdata.vision.transforms.ToTensor(),
    # Standardize each channel of the image.
    gdata.vision.transforms.Normalize([0.485, 0.456, 0.406],
                                      [0.229, 0.224, 0.225])])
```

During testing, we only use definite image preprocessing operations.

In [0]:
```
transform_test = gdata.vision.transforms.Compose([
    gdata.vision.transforms.Resize(256),
    # Crop a square of 224 by 224 from the center of the image.
    gdata.vision.transforms.CenterCrop(224),
    gdata.vision.transforms.ToTensor(),
    gdata.vision.transforms.Normalize([0.485, 0.456, 0.406],
                                      [0.229, 0.224, 0.225])])
```

Read the Data Set

As in the previous section, we can create an `ImageFolderDataset` instance to read the data set containing the original image files.

In [0]:
```
#demo
train_ds = gdata.vision.ImageFolderDataset(
    os.path.join(data_dir, input_dir, 'train'), flag=1)
valid_ds = gdata.vision.ImageFolderDataset(
    os.path.join(data_dir, input_dir, 'valid'), flag=1)
train_valid_ds = gdata.vision.ImageFolderDataset(
    os.path.join(data_dir, input_dir, 'train_valid'), flag=1)
train_iter = gdata.DataLoader(train_ds.transform_first(transform_train),
                              batch_size, shuffle=True, last_batch='keep')
```

In [0]:
```
# train_ds = gdata.vision.ImageFolderDataset(
#     os.path.join(data_dir, input_dir, 'train'), flag=1)
# valid_ds = gdata.vision.ImageFolderDataset(
#     os.path.join(data_dir, input_dir, 'valid'), flag=1)
# train_valid_ds = gdata.vision.ImageFolderDataset(
#     os.path.join(data_dir, input_dir, 'train_valid'), flag=1)
test_ds = gdata.vision.ImageFolderDataset(
    os.path.join('./kaggle_dog', input_dir, 'test'), flag=1)
```

Here, we create a `DataLoader` instance, just like in the previous section.

In [0]:
```
batch_size = 128
train_ds = gdata.vision.ImageFolderDataset(
    os.path.join('./standford', 'Images'), flag=1)

train_iter = gdata.DataLoader(train_ds.transform_first(transform_train),
                              batch_size, shuffle=True, last_batch='keep')
# valid_iter = gdata.DataLoader(valid_ds.transform_first(transform_test),
#                               batch_size, shuffle=True, last_batch='keep'
# train_valid_iter = gdata.DataLoader(train_valid_ds.transform_first(
#     transform_train), batch_size, shuffle=True, last_batch='keep')
# test_iter = gdata.DataLoader(test_ds.transform_first(transform_test),
#                              batch_size, shuffle=False, last_batch='keep'
```

In [0]:
```
# train_iter = gdata.DataLoader(train_ds.transform_first(transform_train),
#                               batch_size, shuffle=True, last_batch='keep'
# valid_iter = gdata.DataLoader(valid_ds.transform_first(transform_test),
#                               batch_size, shuffle=True, last_batch='keep'
# train_valid_iter = gdata.DataLoader(train_valid_ds.transform_first(
#     transform_train), batch_size, shuffle=True, last_batch='keep')
test_iter = gdata.DataLoader(test_ds.transform_first(transform_test),
                             batch_size, shuffle=False, last_batch='keep')
```

Define the Model

The data set for this competition is a subset of the ImageNet data set. Therefore, we can use the approach discussed in the "Fine Tuning" (fine-tuning.md) section to select a model pre-trained on the entire ImageNet data set and use it to extract image features to be input in the custom small-scale output network. Gluon provides a wide range of pre-trained models. Here, we will use the pre-trained ResNet-34 model. Because the competition data set is a subset of the pre-training data set, we simply reuse the input of the pre-trained model's output layer, i.e. the extracted features. Then, we can replace the original output layer with a small custom output network that can be

trained, such as two fully connected layers in a series. Different from the experiment in the "Fine Tuning" (fine-tuning.md) section, here, we do not retrain the pre-trained model used for feature extraction. This reduces the training time and the memory required to store model parameter gradients.

You must note that, during image augmentation, we use the mean values and standard deviations of the three RGB channels for the entire ImageNet data set for normalization. This is consistent with the normalization of the pre-trained model.

In [0]:
```
def get_net(ctx):
    finetune_net = model_zoo.vision.resnet152_v2(pretrained=True)
#     finetune_net = model_zoo.vision.resnet34_v2(pretrained=True)
    # Define a new output network.
    finetune_net.output_new = nn.HybridSequential(prefix='')
    finetune_net.output_new.add(nn.Dense(256, activation='relu'))
    # There are 120 output categories.
    finetune_net.output_new.add(nn.Dropout(0.5))
    finetune_net.output_new.add(nn.Dense(120))
    # Initialize the output network.
    finetune_net.output_new.initialize(init.Xavier(), ctx=ctx)
    # Distribute the model parameters to the CPUs or GPUs used for computat
    finetune_net.collect_params().reset_ctx(ctx)
    return finetune_net
```

When calculating the loss, we first use the member variable `features` to obtain the input of the pre-trained model's output layer, i.e. the extracted feature. Then, we use this feature as the input for our small custom output network and compute the output.

In [0]:
```
loss = gloss.SoftmaxCrossEntropyLoss()

def evaluate_loss(data_iter, net, ctx):
    l_sum, n = 0.0, 0
    for X, y in data_iter:
        y = y.as_in_context(ctx)
        output_features = net.features(X.as_in_context(ctx))
        outputs = net.output_new(output_features)
        l_sum += loss(outputs, y).sum().asscalar()
        n += y.size
    return l_sum / n
```

Define the Training Functions

We will select the model and tune hyper-parameters according to the model's performance on the validation set. The model training function `train` only trains the small custom output network.

In [0]:
```
def train(net, train_iter, valid_iter, num_epochs, lr, wd, ctx, lr_period,
          lr_decay):
    # Only train the small custom output network.
    trainer = gluon.Trainer(net.output_new.collect_params(), 'sgd',
                            {'learning_rate': lr, 'momentum': 0.9, 'wd': wd
    for epoch in range(num_epochs):
        train_l_sum, n, start = 0.0, 0, time.time()
        acc_top1_val = mx.metric.Accuracy()

        for X, y in train_iter:
            y = y.as_in_context(ctx)
            output_features = net.features(X.as_in_context(ctx))
            with autograd.record():
                outputs = net.output_new(output_features)
                l = loss(outputs, y).sum()
                acc_top1_val.update(y, outputs)
            l.backward()
            trainer.step(batch_size)
            train_l_sum += l.asscalar()
            n += y.size
        time_s = "time %.2f sec" % (time.time() - start)
        if valid_iter is not None:
            valid_loss = evaluate_loss(valid_iter, net, ctx)
            epoch_s = ("epoch %d, train loss %f, valid loss %f, "
                       % (epoch + 1, train_l_sum / n, valid_loss))
        else:
            epoch_s = ("epoch %d, train loss %f, "
                       % (epoch + 1, train_l_sum / n))
        acc = acc_top1_val.get()[1]
        if train_l_sum / n <=0.6:
            trainer.set_learning_rate(trainer.learning_rate * lr_decay)
        print(epoch_s + time_s + ', lr ' + str(trainer.learning_rate)+" acc
```

Train and Validate the Model

Now, we can train and validate the model. The following hyper-parameters can be tuned. For example, we can increase the number of epochs. Because `lr_period` and `lr_decay` are set to 10 and 0.1 respectively, the learning rate of the optimization algorithm will be multiplied by 0.1 after every 10 epochs.

In [0]:
```
ctx, num_epochs, lr, wd = d2l.try_gpu(), 20, 0.01, 1e-4
lr_period, lr_decay, net = 20, 0.1, get_net(ctx)
net.hybridize()
# train(net, train_iter, None, num_epochs, lr, wd, ctx, lr_period,
#       lr_decay)
```

```
Downloading /root/.mxnet/models/resnet152_v2-f2695542.zip from https://ap
ache-mxnet.s3-accelerate.dualstack.amazonaws.com/gluon/models/resnet152_v
2-f2695542.zip... (https://apache-mxnet.s3-accelerate.dualstack.amazonaw
s.com/gluon/models/resnet152_v2-f2695542.zip...)
```

Classify the Testing Set and Submit Results on Kaggle

After obtaining a satisfactory model design and hyper-parameters, we use all training data sets (including validation sets) to retrain the model and then classify the testing set. Note that predictions are made by the output network we just trained.

In [0]:
```
net.save_parameters("net.params")
# Install the PyDrive wrapper & import libraries.
# This only needs to be done once in a notebook.
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once in a notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Create & upload a text file.
uploaded = drive.CreateFile({'title': 'net.params'})
uploaded.SetContentString('Sample upload file content')
uploaded.Upload()
print('Uploaded file with ID {}'.format(uploaded.get('id')))
```

In [0]:
```
net = get_net(ctx)
# net.save_parameters("net.params")
# net.load_parameters("net.params", ctx)
net.load_parameters("resnet152v2.params", ctx)
```

In [0]:
```
# net = get_net(ctx)
# net.hybridize()
# train(net, train_valid_iter, None, num_epochs, lr, wd, ctx, lr_period,
#       lr_decay)

preds = []
for data, label in test_iter:
    output_features = net.features(data.as_in_context(ctx))
    output = nd.softmax(net.output_new(output_features))
    preds.extend(output.asnumpy())
ids = sorted(os.listdir(os.path.join(data_dir, input_dir, 'test/unknown')))
with open('submission.csv', 'w') as f:
    f.write('id,' + ','.join(train_valid_ds.synsets) + '\n')
    for i, output in zip(ids, preds):
        f.write(i.split('.')[0] + ',' + ','.join(
            [str(num) for num in output]) + '\n')
```

In [0]:
```
# !pip install kaggle
```

After executing the above code, we will generate a "submission.csv" file. The format of this file is consistent with the Kaggle competition requirements.

Hints to Improve Your Results

- You should download the whole data set from Kaggle and switch to `demo=False`.
- Try to increase the `batch_size` (batch size) and `num_epochs` (number of epochs).
- Try a deeper pre-trained model, you may find models from gluoncv (https://gluon-cv.mxnet.io/model_zoo/classification.html).

In [0]:
```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-pa
ckages (1.5.3)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/py
thon3.6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist
-packages (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-p
ackages (from kaggle) (2019.3.9)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.
6/dist-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-
packages (from kaggle) (2.21.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-pack
ages (from kaggle) (4.28.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.
6/dist-packages (from kaggle) (3.0.2)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/py
thon3.6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.
6/dist-packages (from requests->kaggle) (2.6)
Requirement already satisfied: text-unidecode==1.2 in /usr/local/lib/pyth
on3.6/dist-packages (from python-slugify->kaggle) (1.2)
```

In [0]:
```
# Create a JSON file containing user-specific metadata.
# This step is required if you want to access the Kaggle API.
# For more info see: https://github.com/Kaggle/kaggle-api#api-credentials
USER_ID = 'caojilin' # REPLACE WITH YOUR OWN USER NAME
USER_SECRET = "95413dff633e902356ea9fc5b87e682d" # REPLACE WITH YOUR OWN PR
import os, json, nbformat, pandas as pd
KAGGLE_CONFIG_DIR = os.path.join(os.path.expandvars('$HOME'), '.kaggle')
os.makedirs(KAGGLE_CONFIG_DIR, exist_ok = True)
with open(os.path.join(KAGGLE_CONFIG_DIR, 'kaggle.json'), 'w') as f:
    json.dump({'username': USER_ID, 'key': USER_SECRET}, f)
!chmod 600 {KAGGLE_CONFIG_DIR}/kaggle.json
```

In [0]:
```
!kaggle competitions submit -c dog-breed-identification -f submission.csv -
```

```
100% 16.4M/16.4M [00:05<00:00, 3.33MB/s]
Successfully submitted to Dog Breed Identification
```

In [0]:
```
!kaggle competitions submit -c dog-breed-identification -f p3.csv -m "throu
```

```
100% 16.4M/16.4M [00:05<00:00, 3.18MB/s]
Successfully submitted to Dog Breed Identification
```

In [0]:
```
from mxnet import autograd
from mxnet import gluon
from mxnet import image
from mxnet import init
from mxnet import nd
from mxnet.gluon.data import vision
import numpy as np

label_file = 'labels.csv'
train_dir = 'train'
test_dir = 'test'
input_dir = 'train_valid_test'
batch_size = 128
valid_ratio = 0.1

def transform_train(data, label):
    im = image.imresize(data.astype('float32') / 255, 363, 363)
    #im = image.imresize(data.astype('float32') / 255, 400, 400)
    auglist = image.CreateAugmenter(data_shape=(3, 363, 363), resize=0,
                        rand_crop=False, rand_resize=False, rand_mirror=Tru
                        mean=np.array([0.485, 0.456, 0.406]), std=np.array(
                        brightness=0, contrast=0,
                        saturation=0, hue=0,
                        pca_noise=0, rand_gray=0, inter_method=2)
    for aug in auglist:
        im = aug(im)

    # transform height x weight x channels to channels x height x weight
    im = nd.transpose(im, (2,0,1))
    return (im, nd.array([label]).asscalar().astype('float32'))

def transform_test(data, label):
    im = image.imresize(data.astype('float32') / 255, 363, 363)
    auglist = image.CreateAugmenter(data_shape=(3, 363, 363),
                        mean=np.array([0.485, 0.456, 0.406]),
                        std=np.array([0.229, 0.224, 0.225]))
    for aug in auglist:
        im = aug(im)
    im = nd.transpose(im, (2,0,1))
    return (im, nd.array([label]).asscalar().astype('float32'))
```

In [0]:
```
input_str = data_dir + '/' + input_dir + '/'

train_ds = vision.ImageFolderDataset(input_str + 'train', flag=1,
                                     transform=transform_train)
valid_ds = vision.ImageFolderDataset(input_str + 'valid', flag=1,
                                     transform=transform_test)
train_valid_ds = vision.ImageFolderDataset(input_str + 'train_valid',
                                           flag=1, transform=transform_trai
test_ds = vision.ImageFolderDataset(input_str + 'test', flag=1,
                                    transform=transform_test)

loader = gluon.data.DataLoader
train_data = loader(train_ds, batch_size, shuffle=True, last_batch='keep')
valid_data = loader(valid_ds, batch_size, shuffle=True, last_batch='keep')
train_valid_data = loader(train_valid_ds, batch_size, shuffle=True,
                          last_batch='keep')
test_data = loader(test_ds, batch_size, shuffle=False, last_batch='keep')
```

In [0]:
```
#hererererrererere
train_ds = gdata.vision.ImageFolderDataset(
    os.path.join(data_dir, input_dir, 'train'), flag=1)
valid_ds = gdata.vision.ImageFolderDataset(
    os.path.join(data_dir, input_dir, 'valid'), flag=1)
train_valid_ds = gdata.vision.ImageFolderDataset(
    os.path.join(data_dir, input_dir, 'train_valid'), flag=1)
test_ds = gdata.vision.ImageFolderDataset(
    os.path.join(data_dir, input_dir, 'test'), flag=1)

train_data = gdata.DataLoader(train_ds.transform_first(transform_train),
                              batch_size, shuffle=True, last_batch='keep')
valid_data = gdata.DataLoader(valid_ds.transform_first(transform_test),
                              batch_size, shuffle=True, last_batch='keep')
train_valid_data = gdata.DataLoader(train_valid_ds.transform_first(
    transform_train), batch_size, shuffle=True, last_batch='keep')
test_data = gdata.DataLoader(test_ds.transform_first(transform_test),
                             batch_size, shuffle=False, last_batch='keep')
```

In [0]:
```
from mxnet import gluon
from mxnet import init
from mxnet.gluon import nn

def get_features(ctx):
    inception = model_zoo.vision.inception_v3(pretrained=True,ctx=ctx)
    return inception.features


def get_output(ctx,ParamsName=None):
    net = nn.HybridSequential()
    with net.name_scope():
        net.add(nn.Dropout(.2))
        net.add(nn.Dense(256, activation="relu"))
        net.add(nn.Dropout(.6))
        net.add(nn.Dense(120))
    if ParamsName is not None:
        #net.collect_params().load(ParamsName,ctx)
        net.load_parameters(ParamsName,ctx)
    else:
        net.initialize(init = init.Xavier(),ctx=ctx)
    return net

def get_net(ParamsName,ctx):
    output = get_output(ctx,ParamsName)
    features = get_features(ctx)
    net = nn.HybridSequential()
    with net.name_scope():
        net.add(features)
        net.add(output)
    return net
```

In [0]:
```
import mxnet as mx
net2=get_features(mx.gpu())
# print(net2)
```

In [0]:
```
from mxnet import nd
import numpy as np
import mxnet as mx
import pandas as pd
import pickle
from tqdm import tqdm

net = get_features(mx.gpu())
net.hybridize()

def SaveNd(data,net,name):
    x =[]
    y =[]
    print('extract features %s' % name)
    for fear,label in tqdm(data):
        x.append(net(fear.as_in_context(mx.gpu())).as_in_context(mx.cpu()))
        y.append(label)
    x = nd.concat(*x,dim=0)
    y = nd.concat(*y,dim=0)
    print('save features %s' % name)
    nd.save(name,[x,y])


SaveNd(train_data,net,'train_inception_v3.nd')
SaveNd(valid_data,net,'valid_inception_v3.nd')
SaveNd(train_valid_data,net,'input_inception_v3.nd')
# SaveNd(test_data,net,'test_resnet152_v1.nd')
ids = ids = sorted(os.listdir(os.path.join(data_dir, input_dir, 'test/unknc
synsets = train_valid_ds.synsets
f = open('ids_synsets','wb')
pickle.dump([ids,synsets],f)
f.close()
```

```
 69%|██████▉   | 52/75 [02:43<01:12,  3.15s/it]

 71%|███████   | 53/75 [02:46<01:09,  3.17s/it]

 72%|███████▏  | 54/75 [02:49<01:05,  3.13s/it]

 73%|███████▎  | 55/75 [02:52<01:02,  3.15s/it]

 75%|███████▍  | 56/75 [02:55<00:59,  3.12s/it]


 76%|███████▌  | 57/75 [02:58<00:56,  3.11s/it]

 77%|███████▋  | 58/75 [03:02<00:53,  3.12s/it]

 79%|███████▊  | 59/75 [03:05<00:50,  3.14s/it]

 80%|████████  | 60/75 [03:08<00:46,  3.12s/it]
```

In [0]:
```
import datetime
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
from mxnet import autograd
from mxnet import gluon
from mxnet import image
from mxnet import init
from mxnet import nd
from mxnet.gluon.data import vision
from mxnet.gluon import nn
from mxnet import nd
import pandas as pd
import mxnet as mx
import pickle

train_nd = nd.load('train_inception_v3.nd')

valid_nd = nd.load('valid_inception_v3.nd')

input_nd = nd.load('input_inception_v3.nd')

f = open('ids_synsets','rb')
ids_synsets = pickle.load(f)
f.close()

num_epochs = 150
batch_size = 128
learning_rate = 1e-4
weight_decay = 1e-5
lr_period = 100
lr_decay = 0.1
pngname='1'
modelparams='1'

train_data_d = gluon.data.DataLoader(gluon.data.ArrayDataset(train_nd[0],tr
valid_data_d = gluon.data.DataLoader(gluon.data.ArrayDataset(valid_nd[0],va
input_data_d = gluon.data.DataLoader(gluon.data.ArrayDataset(input_nd[0],in


def get_loss(data, net, ctx):
    loss = 0.0
    for feas, label in data:
        label = label.as_in_context(ctx)
        output = net(feas.as_in_context(ctx))
        cross_entropy = softmax_cross_entropy(output, label)
        loss += nd.mean(cross_entropy).asscalar()
    return loss / len(data)

softmax_cross_entropy = gluon.loss.SoftmaxCrossEntropyLoss()

def train(net, train_data, valid_data, num_epochs, lr, wd, ctx, lr_period,
          lr_decay):
    trainer = gluon.Trainer(
        net.collect_params(), 'adam', {'learning_rate': lr, 'wd': wd})
    #trainer = gluon.Trainer(
```

```
     #   net.collect_params(), 'sgd', {'learning_rate': lr, 'momentum': 0.9
      #                                'wd': wd})
    train_loss = []
    if valid_data is not None:
        test_loss = []

    prev_time = datetime.datetime.now()
    for epoch in range(num_epochs):
        _loss = 0.
        if epoch > 0 and epoch % lr_period == 0:
           trainer.set_learning_rate(trainer.learning_rate * lr_decay)
        for data, label in train_data:
            label = label.as_in_context(ctx)
            with autograd.record():
                output = net(data.as_in_context(ctx))
                loss = softmax_cross_entropy(output, label)
            loss.backward()
            trainer.step(batch_size)
            _loss += nd.mean(loss).asscalar()
        cur_time = datetime.datetime.now()
        h, remainder = divmod((cur_time - prev_time).seconds, 3600)
        m, s = divmod(remainder, 60)
        time_str = "Time %02d:%02d:%02d" % (h, m, s)
        __loss = _loss/len(train_data)
        train_loss.append(__loss)

        if valid_data is not None:
            valid_loss = get_loss(valid_data, net, ctx)
            epoch_str = ("Epoch %d. Train loss: %f, Valid loss %f, "
                         % (epoch,__loss , valid_loss))
            test_loss.append(valid_loss)
        else:
            epoch_str = ("Epoch %d. Train loss: %f, "
                         % (epoch, __loss))

        prev_time = cur_time
        print(epoch_str + time_str + ', lr ' + str(trainer.learning_rate))


    plt.plot(train_loss, 'r')
    if valid_data is not None:
        plt.plot(test_loss, 'g')
    plt.legend(['Train_Loss', 'Test_Loss'], loc=2)


    plt.savefig(pngname, dpi=1000)
    #net.collect_params().save(modelparams)
    savefilename = "./inception_v3.params"
    net.save_parameters(savefilename)

ctx = mx.gpu()
net = get_output(ctx)
net.hybridize()

#train(net, input_data_d,None, num_epochs, learning_rate, weight_decay,
 #     ctx, lr_period, lr_decay)
train(net, train_data_d,valid_data_d, num_epochs, learning_rate, weight_dec
```

```
    ctx, lr_period, lr_decay)
```

```
Epoch 0. Train loss: 4.597775, Valid loss 4.101457, Time 00:00:00, lr 0.0
001
Epoch 1. Train loss: 3.824505, Valid loss 2.981406, Time 00:00:00, lr 0.0
001
Epoch 2. Train loss: 2.860231, Valid loss 1.957403, Time 00:00:00, lr 0.0
001
Epoch 3. Train loss: 2.109669, Valid loss 1.282627, Time 00:00:00, lr 0.0
001
Epoch 4. Train loss: 1.615409, Valid loss 0.903502, Time 00:00:00, lr 0.0
001
Epoch 5. Train loss: 1.287589, Valid loss 0.698177, Time 00:00:00, lr 0.0
001
Epoch 6. Train loss: 1.100800, Valid loss 0.568821, Time 00:00:00, lr 0.0
001
Epoch 7. Train loss: 0.949393, Valid loss 0.496424, Time 00:00:00, lr 0.0
001
Epoch 8. Train loss: 0.833068, Valid loss 0.443741, Time 00:00:00, lr 0.0
001
Epoch 9. Train loss: 0.777226, Valid loss 0.417439, Time 00:00:00, lr 0.0
001
```

In [0]:
```
import datetime
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
from mxnet import autograd
from mxnet import gluon
from mxnet import image
from mxnet import init
from mxnet import nd
#from mxnet.gluon.data import vision
from mxnet.gluon.model_zoo import vision
from mxnet.gluon import nn
from mxnet import nd
import pandas as pd
import mxnet as mx
import pickle
import numpy as np
from tqdm import tqdm
#from model import get_net

data_dir = './'
test_dir = 'test'
input_dir = 'train_valid_test'
valid_dir = 'valid'
input_str = data_dir + '/' + input_dir + '/'

netparams ="./inception_v3.params"
csvname = 'p3.csv'
ids_synsets_name = 'ids_synsets'

f = open(ids_synsets_name,'rb')
ids_synsets = pickle.load(f)
f.close()

def SaveTest(test_data,net,ctx,name,ids,synsets):
    outputs = []
    for data, label in tqdm(test_data):
        output = nd.softmax(net(data.as_in_context(ctx)))
        outputs.extend(output.asnumpy())
    with open(name, 'w') as f:
        f.write('id,' + ','.join(synsets) + '\n')
        for i, output in zip(ids, outputs):
            f.write(i.split('.')[0] + ',' + ','.join(
                [str(num) for num in output]) + '\n')

net = get_net(netparams,mx.gpu())
net.hybridize()

softmax_cross_entropy = gluon.loss.SoftmaxCrossEntropyLoss()
print(get_loss(valid_data,net,mx.gpu()))

SaveTest(test_data,net,mx.gpu(),csvname,ids_synsets[0],ids_synsets[1])
```

```
/usr/local/lib/python3.6/dist-packages/mxnet/gluon/block.py:420: UserWarn
ing: load_params is deprecated. Please use load_parameters.
  warnings.warn("load_params is deprecated. Please use load_parameters.")
  0%|          | 0/81 [00:00<?, ?it/s]
```

```
0.2565094828605652

100%|██████████| 81/81 [04:35<00:00,  3.37s/it]
```

class ConcatNet(nn.HybridBlock): def **init**(self,net1,net2,**kwargs): **super(ConcatNet,self).init(**kwargs) self.net1 = nn.HybridSequential() self.net1.add(net1) self.net1.add(nn.GlobalAvgPool2D()) self.net2 = nn.HybridSequential() self.net2.add(net2) self.net2.add(nn.GlobalAvgPool2D()) def hybrid_forward(self,F,x1,x2): return F.concat(* [self.net1(x1),self.net2(x2)])

class OneNet(nn.HybridBlock): def **init**(self,features,output,**kwargs): **super(OneNet,self).init(**kwargs) self.features = features self.output = output def hybrid_forward(self,F,x1,x2): return self.output(self.features(x1,x2))

class Net(): def **init**(self,ctx,nameparams=None): inception = vision.inception_v3(pretrained=True,ctx=ctx).features resnet = vision.resnet152_v1(pretrained=True,ctx=ctx).features self.features = ConcatNet(resnet,inception) self.output = self._*get*output(ctx,nameparams) self.net = OneNet(self.features,self.output) def _*get*output(self,ctx,ParamsName=None): net = nn.HybridSequential("output") with net.name_scope(): net.add(nn.Dense(256,activation='relu')) net.add(nn.Dropout(.5)) net.add(nn.Dense(120)) if ParamsName is not None: net.collect_params().load(ParamsName,ctx) else: net.initialize(init = init.Xavier(),ctx=ctx) return net

In [0]: