In [3]:

```
demo = False # You need to change demo to False for this homework.
if demo:
    import zipfile
    for f in ['train_tiny.zip', 'test_tiny.zip', 'trainLabels.csv.zip']:
        with zipfile.ZipFile('../data/kaggle_cifar10/' + f, 'r') as z:
            z.extractall('../data/kaggle_cifar10/')
```

Organize the Data Set

In [7]:

```
def read_label_file(data_dir, label_file, train_dir, valid_ratio):
    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(((int(idx), label) for idx, label in tokens))
    labels = set(idx_label.values())
    n_train_valid = len(os.listdir(os.path.join(data_dir, train_dir)))
    n_train = int(n_train_valid * (1 - valid_ratio))
    assert 0 < n_train < n_train_valid
    return n_train // len(labels), idx_label
```

Below we define a helper function to create a path only if the path does not already exist.

In [8]:

```
def mkdir_if_not_exist(path):
    if not os.path.exists(os.path.join(*path)):
        os.makedirs(os.path.join(*path))
```

Next, we define the `reorg_train_valid` function to segment the validation set from the original training set. Here, we use `valid_ratio=0.1` as an example. Since the original training set has 50,000 images, there will be 45,000 images used for training and stored in the path “ `input_dir/train` ” when tuning hyper-parameters, while the other 5,000 images will be stored as validation set in the path “ `input_dir/valid` ”. After organizing the data, images of the same type will be placed under the same folder so that we can read them later.

In [9]:

```python
def reorg_train_valid(data_dir, train_dir, input_dir, n_train_per_label,
                      idx_label):
    label_count = {}
    for train_file in os.listdir(os.path.join(data_dir, train_dir)):
        idx = int(train_file.split('.')[0])
        label = idx_label[idx]
        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_train_per_label:
            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))
            label_count[label] = label_count.get(label, 0) + 1
        else:
            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))
```

The `reorg_test` function below is used to organize the testing set to facilitate the reading during prediction.

In [10]:

```python
def reorg_test(data_dir, test_dir, input_dir):
    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'))
```

Finally, we use a function to call the previously defined `reorg_test` , `reorg_train_valid` , and `reorg_test` functions.

In [11]:

```python
def reorg_cifar10_data(data_dir, label_file, train_dir, test_dir, input_dir,
                       valid_ratio):
    n_train_per_label, idx_label = read_label_file(data_dir, label_file,
                                                   train_dir, valid_ratio)
    reorg_train_valid(data_dir, train_dir, input_dir, n_train_per_label,
                      idx_label)
    reorg_test(data_dir, test_dir, input_dir)
```

After creat the corresponding folders and files, we only need to run this part of the code.

In [8]:

```
#if demo:
    # Note: Here, we use small training sets and small testing sets and the
    # batch size should be set smaller. When using the complete data set for
    # the Kaggle competition, the batch size can be set to a large integer
#    train_dir, test_dir, batch_size = 'train_tiny', 'test_tiny', 1
#else:
train_dir, test_dir, batch_size = 'train', 'test', 128
data_dir, label_file = 'C:/Users/Seebarsh/Documents/Stat_157/Stat_157/hw6/data/kaggle_cifar10', 'trainLabels.csv'
input_dir, valid_ratio = 'train_valid_test', 0.1
#reorg_cifar10_data(data_dir, label_file, train_dir, test_dir, input_dir,
#                   valid_ratio)
```

Image Augmentation

We did not change this part of the homework

In [3]:

```
transform_train = gdata.vision.transforms.Compose([
    # Magnify the image to a square of 40 pixels in both height and width
    gdata.vision.transforms.Resize(40),
    # Randomly crop a square image of 40 pixels in both height and width to
    # produce a small square of 0.64 to 1 times the area of the original
    # image, and then shrink it to a square of 32 pixels in both height and
    # width
    gdata.vision.transforms.RandomResizedCrop(32, scale=(0.64, 1.0),
                                              ratio=(1.0, 1.0)),
    gdata.vision.transforms.RandomFlipLeftRight(),
    gdata.vision.transforms.ToTensor(),
    # Normalize each channel of the image
    gdata.vision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                      [0.2023, 0.1994, 0.2010])])
```

In order to ensure the certainty of the output during testing, we only perform normalization on the image.

In [4]:

```
transform_test = gdata.vision.transforms.Compose([
    gdata.vision.transforms.ToTensor(),
    gdata.vision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                      [0.2023, 0.1994, 0.2010])])
```

Read the Data Set

Next, we can create the `ImageFolderDataset` instance to read the organized data set containing the original image files, where each data instance includes the image and label.

In [9]:

```
# Read the original image file. Flag=1 indicates that the input image has
# three channels (color)
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)
```

We specify the defined image augmentation operation in `DataLoader` . During training, we only use the validation set to evaluate the model, so we need to ensure the certainty of the output. During prediction, we will train the model on the combined training set and validation set to make full use of all labelled data.

In [10]:

```
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')
```

===Remain unchanged===============================

Resnet18 (baseline)

We changes the parameter.

- num_epochs, lr, wd = 100
- lr = 0.03
- wd = 5e-4
- lr_period, lr_decay = 80, 0.1

In [3]:
```python
class Residual(nn.HybridBlock):
    def __init__(self, num_channels, use_1x1conv=False, strides=1, **kwargs):
        super(Residual, self).__init__(**kwargs)
        self.conv1 = nn.Conv2D(num_channels, kernel_size=3, padding=1,
                               strides=strides)
        self.conv2 = nn.Conv2D(num_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.Conv2D(num_channels, kernel_size=1,
                                   strides=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.BatchNorm()
        self.bn2 = nn.BatchNorm()

    def hybrid_forward(self, F, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        return F.relu(Y + X)
```

In [4]:

```
def resnet18(num_classes):
    net = nn.HybridSequential()
    net.add(nn.Conv2D(64, kernel_size=3, strides=1, padding=1),
            nn.BatchNorm(), nn.Activation('relu'))

    def resnet_block(num_channels, num_residuals, first_block=False):
        blk = nn.HybridSequential()
        for i in range(num_residuals):
            if i == 0 and not first_block:
                blk.add(Residual(num_channels, use_1x1conv=True, strides=2))
            else:
                blk.add(Residual(num_channels))
        return blk

    net.add(resnet_block(64, 2, first_block=True),
            resnet_block(128, 2),
            resnet_block(256, 2),
            resnet_block(512, 2))
    net.add(nn.GlobalAvgPool2D(), nn.Dense(num_classes))
    return net
```

In [ ]:

```
def get_net(ctx):
    num_classes = 10
    net = resnet18(num_classes)
    net.initialize(ctx=ctx, init=init.Xavier())
    return net

loss = gloss.SoftmaxCrossEntropyLoss()
```

In [ ]:

```
def train(net, train_iter, valid_iter, num_epochs, lr, wd, ctx, lr_period,
          lr_decay):
    trainer = gluon.Trainer(net.collect_params(), 'sgd',
                            {'learning_rate': lr, 'momentum': 0.9, 'wd': wd})
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n, start = 0.0, 0.0, 0, time.time()
        if epoch > 0 and epoch % lr_period == 0:
            trainer.set_learning_rate(trainer.learning_rate * lr_decay)
        for X, y in train_iter:
            y = y.astype('float32').as_in_context(ctx)
            with autograd.record():
                y_hat = net(X.as_in_context(ctx))
                l = loss(y_hat, y).sum()
            l.backward()
            trainer.step(batch_size)
            train_l_sum += l.asscalar()
            train_acc_sum += (y_hat.argmax(axis=1) == y).sum().asscalar()
            n += y.size
        time_s = "time %.2f sec" % (time.time() - start)
        if valid_iter is not None:
            valid_acc = d2l.evaluate_accuracy(valid_iter, net, ctx)
            epoch_s = ("epoch %d, loss %f, train acc %f, valid acc %f, "
                       % (epoch + 1, train_l_sum / n, train_acc_sum / n,
                       valid_acc))
        else:
            epoch_s = ("epoch %d, loss %f, train acc %f, " %
                       (epoch + 1, train_l_sum / n, train_acc_sum / n))
        print(epoch_s + time_s + ', lr ' + str(trainer.learning_rate))
```

In [ ]:

```
ctx, num_epochs, lr, wd = d2l.try_gpu(), 100, 0.03, 5e-4
lr_period, lr_decay, net = 80, 0.1, get_net(ctx)
net.hybridize()
train(net, train_iter, valid_iter, num_epochs, lr, wd, ctx, lr_period,
      lr_decay)
```

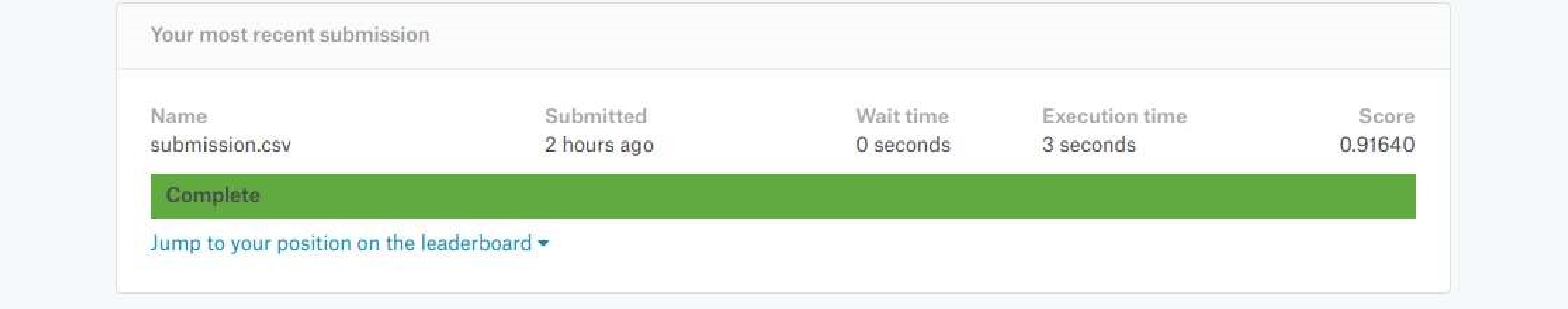
In [ ]:

```
model.save_params('./baseline.params')
```

In [ ]:

```
net = get_net(ctx)
net.hybridize()
net = net.load_parameters('baseline.params', ctx=ctx)
preds = []
train(net, train_valid_iter, None, num_epochs, lr, wd, ctx, lr_period,
      lr_decay)

for X, _ in test_iter:
    y_hat = net(X.as_in_context(ctx))
    preds.extend(y_hat.argmax(axis=1).astype(int).asnumpy())
sorted_ids = list(range(1, len(test_ds) + 1))
sorted_ids.sort(key=lambda x: str(x))
df = pd.DataFrame({'id': sorted_ids, 'label': preds})
df['label'] = df['label'].apply(lambda x: train_valid_ds.synsets[x])
df.to_csv('submission.csv', index=False)
```

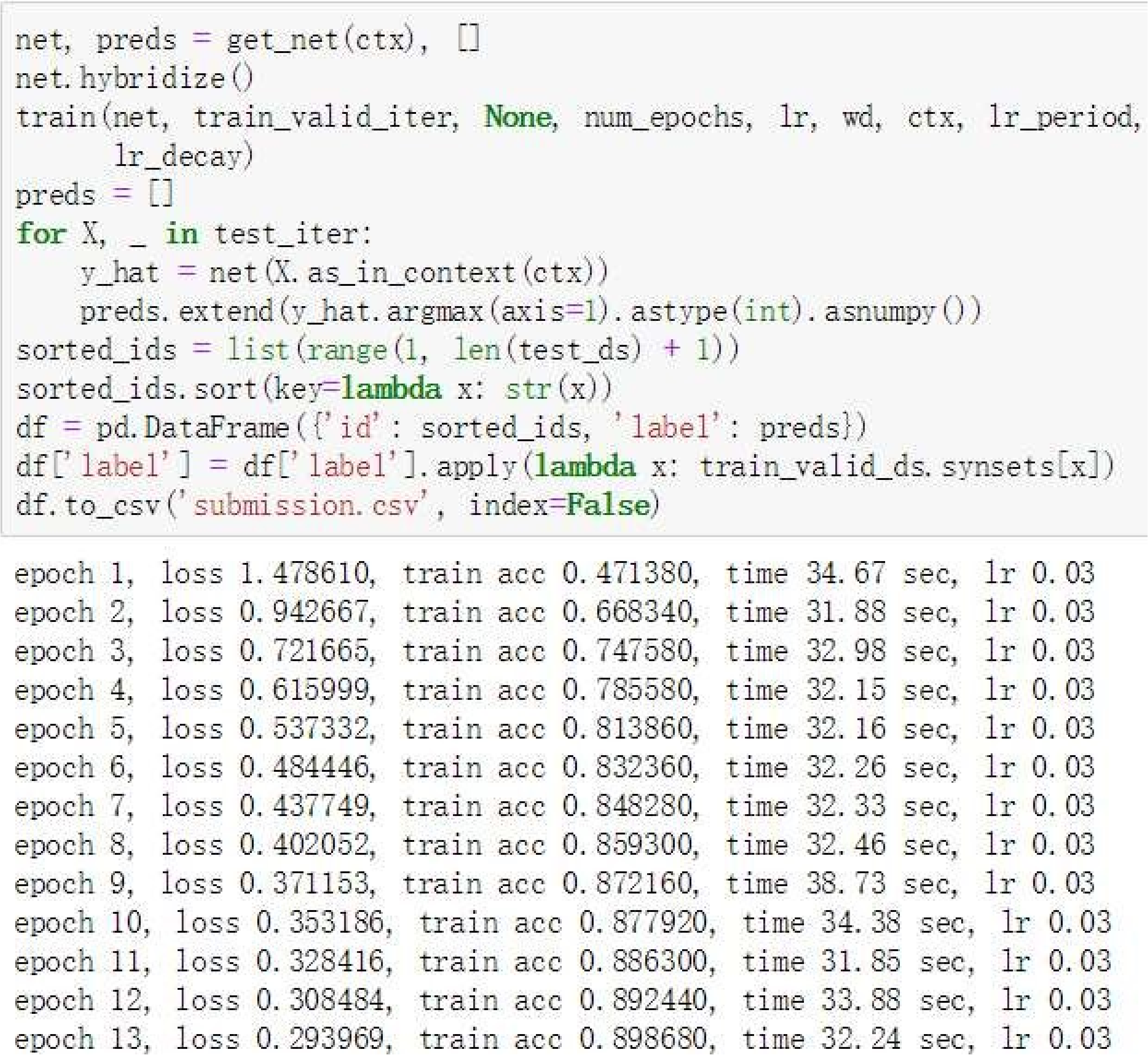


Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	2 hours ago	0 seconds	3 seconds	0.91640

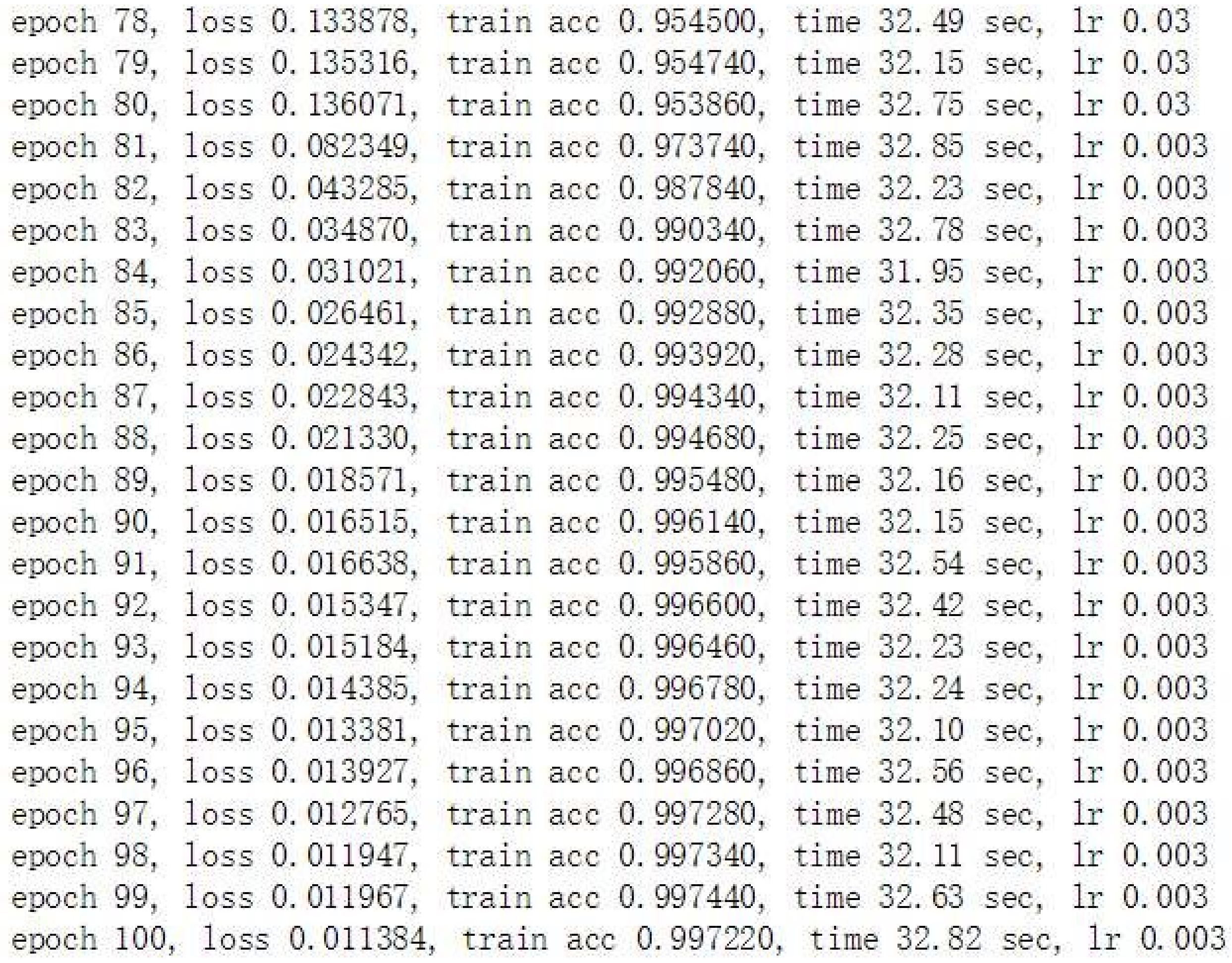
Complete

Jump to your position on the leaderboard ▾



```
net, preds = get_net(ctx), []
net.hybridize()
train(net, train_valid_iter, None, num_epochs, lr, wd, ctx, lr_period,
      lr_decay)
preds = []
for X, _ in test_iter:
    y_hat = net(X.as_in_context(ctx))
    preds.extend(y_hat.argmax(axis=1).astype(int).asnumpy())
sorted_ids = list(range(1, len(test_ds) + 1))
sorted_ids.sort(key=lambda x: str(x))
df = pd.DataFrame({'id': sorted_ids, 'label': preds})
df['label'] = df['label'].apply(lambda x: train_valid_ds.synsets[x])
df.to_csv('submission.csv', index=False)
```

```
epoch 1, loss 1.478610, train acc 0.471380, time 34.67 sec, lr 0.03
epoch 2, loss 0.942667, train acc 0.668340, time 31.88 sec, lr 0.03
epoch 3, loss 0.721665, train acc 0.747580, time 32.98 sec, lr 0.03
epoch 4, loss 0.615999, train acc 0.785580, time 32.15 sec, lr 0.03
epoch 5, loss 0.537332, train acc 0.813860, time 32.16 sec, lr 0.03
epoch 6, loss 0.484446, train acc 0.832360, time 32.26 sec, lr 0.03
epoch 7, loss 0.437749, train acc 0.848280, time 32.33 sec, lr 0.03
epoch 8, loss 0.402052, train acc 0.859300, time 32.46 sec, lr 0.03
epoch 9, loss 0.371153, train acc 0.872160, time 38.73 sec, lr 0.03
epoch 10, loss 0.353186, train acc 0.877920, time 34.38 sec, lr 0.03
epoch 11, loss 0.328416, train acc 0.886300, time 31.85 sec, lr 0.03
epoch 12, loss 0.308484, train acc 0.892440, time 33.88 sec, lr 0.03
epoch 13, loss 0.293969, train acc 0.898680, time 32.24 sec, lr 0.03
```



```
epoch 78, loss 0.133878, train acc 0.954500, time 32.49 sec, lr 0.03
epoch 79, loss 0.135316, train acc 0.954740, time 32.15 sec, lr 0.03
epoch 80, loss 0.136071, train acc 0.953860, time 32.75 sec, lr 0.03
epoch 81, loss 0.082349, train acc 0.973740, time 32.85 sec, lr 0.003
epoch 82, loss 0.043285, train acc 0.987840, time 32.23 sec, lr 0.003
epoch 83, loss 0.034870, train acc 0.990340, time 32.78 sec, lr 0.003
epoch 84, loss 0.031021, train acc 0.992060, time 31.95 sec, lr 0.003
epoch 85, loss 0.026461, train acc 0.992880, time 32.35 sec, lr 0.003
epoch 86, loss 0.024342, train acc 0.993920, time 32.28 sec, lr 0.003
epoch 87, loss 0.022843, train acc 0.994340, time 32.11 sec, lr 0.003
epoch 88, loss 0.021330, train acc 0.994680, time 32.25 sec, lr 0.003
epoch 89, loss 0.018571, train acc 0.995480, time 32.16 sec, lr 0.003
epoch 90, loss 0.016515, train acc 0.996140, time 32.15 sec, lr 0.003
epoch 91, loss 0.016638, train acc 0.995860, time 32.54 sec, lr 0.003
epoch 92, loss 0.015347, train acc 0.996600, time 32.42 sec, lr 0.003
epoch 93, loss 0.015184, train acc 0.996460, time 32.23 sec, lr 0.003
epoch 94, loss 0.014385, train acc 0.996780, time 32.24 sec, lr 0.003
epoch 95, loss 0.013381, train acc 0.997020, time 32.10 sec, lr 0.003
epoch 96, loss 0.013927, train acc 0.996860, time 32.56 sec, lr 0.003
epoch 97, loss 0.012765, train acc 0.997280, time 32.48 sec, lr 0.003
epoch 98, loss 0.011947, train acc 0.997340, time 32.11 sec, lr 0.003
epoch 99, loss 0.011967, train acc 0.997440, time 32.63 sec, lr 0.003
epoch 100, loss 0.011384, train acc 0.997220, time 32.82 sec, lr 0.003
```

Resnet164

Then we used a ResNet164 model(after looking at a bunch of codes and ideas posted in Kaggle and Google). Here are some parameters:

- num_epochs = 200
- learning_rate = 0.1
- weight_decay = 1e-4
- lr_decay = 0.1, happen twice

In [16]:

```
criterion = gluon.loss.SoftmaxCrossEntropyLoss()

from tensorboardX import SummaryWriter
import mxnet as mx
from mxnet.gluon import nn


class Residual_v2_bottleneck(nn.HybridBlock):
    def __init__(self, channels, same_shape=True):
        super(Residual_v2_bottleneck, self).__init__()
        self.same_shape = same_shape
        with self.name_scope():
            strides = 1 if same_shape else 2
            self.bn1 = nn.BatchNorm()
            self.conv1 = nn.Conv2D(channels // 4, 1, use_bias=False)
            self.bn2 = nn.BatchNorm()
            self.conv2 = nn.Conv2D(
                channels // 4, 3, padding=1, strides=strides, use_bias=False)
            self.bn3 = nn.BatchNorm()
            self.conv3 = nn.Conv2D(channels, 1, use_bias=False)
            self.bn4 = nn.BatchNorm()

            if not same_shape:
                self.conv4 = nn.Conv2D(
                    channels, 1, strides=strides, use_bias=False)

    def hybrid_forward(self, F, x):
        out = self.conv1(self.bn1(x))
        out = F.relu(self.bn2(out))
        out = F.relu(self.bn3(self.conv2(out)))
        out = self.bn4(self.conv3(out))
        if not self.same_shape:
            x = self.conv4(x)
        return out + x


class ResNet164_v2(nn.HybridBlock):
    def __init__(self, num_classes, verbose=False):
        super(ResNet164_v2, self).__init__()
        self.verbose = verbose
        with self.name_scope():
            net = self.net = nn.HybridSequential()
            # block 1
            net.add(nn.Conv2D(64, 3, 1, 1, use_bias=False))
            # block 2
            for _ in range(27):
                net.add(Residual_v2_bottleneck(64))
            # block 3
            net.add(Residual_v2_bottleneck(128, same_shape=False))
            for _ in range(26):
                net.add(Residual_v2_bottleneck(128))
            # block 4
            net.add(Residual_v2_bottleneck(256, same_shape=False))
            for _ in range(26):
                net.add(Residual_v2_bottleneck(256))
            # block 5
            net.add(nn.BatchNorm())
            net.add(nn.Activation('relu'))
            net.add(nn.AvgPool2D(8))
            net.add(nn.Dense(num_classes))

    def hybrid_forward(self, F, x):
        out = x
        for i, b in enumerate(self.net):
            out = b(out)
            if self.verbose:
                print('Block %d output: %s' % (i + 1, out.shape))
        return out
```

In [18]:

```
model = ResNet164_v2(10)
model.initialize(ctx=mx.gpu(0), init=mx.initializer.Xavier())
model.hybridize()

import datetime
writer = SummaryWriter()

def get_acc(output, label):
    pred = output.argmax(1, keepdims=True)
    correct = (pred == label).sum()
    return correct.asscalar()

def train(net, train_data, valid_data, num_epochs, lr, wd, ctx, lr_decay):
    trainer = gluon.Trainer(
        net.collect_params(), 'sgd', {'learning_rate': lr, 'momentum': 0.9, 'wd': wd})

    prev_time = datetime.datetime.now()
    for epoch in range(num_epochs):
        train_loss = 0
        correct = 0
        total = 0
        if epoch == 89 or epoch == 139:
            trainer.set_learning_rate(trainer.learning_rate * lr_decay)
        for data, label in train_data:
            bs = data.shape[0]
            data = data.as_in_context(ctx)
            label = label.as_in_context(ctx)
            with autograd.record():
                output = net(data)
                loss = criterion(output, label)
            loss.backward()
            trainer.step(bs)
            train_loss += nd.mean(loss).asscalar()
            correct += get_acc(output, label)
            total += bs
        writer.add_scalars('loss', {'train': train_loss / len(train_data)}, epoch)
        writer.add_scalars('acc', {'train': correct / total}, epoch)
        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)
        if valid_data is not None:
            valid_correct = 0
            valid_total = 0
            valid_loss = 0
            for data, label in valid_data:
                bs = data.shape[0]
                data = data.as_in_context(ctx)
                label = label.as_in_context(ctx)
                output = net(data)
                loss = criterion(output, label)
                valid_loss += nd.mean(loss).asscalar()
                valid_correct += get_acc(output, label)
                valid_total += bs
            valid_acc = valid_correct / valid_total
            writer.add_scalars('loss', {'valid': valid_loss / len(valid_data)}, epoch)
            writer.add_scalars('acc', {'valid': valid_acc}, epoch)
            epoch_str = ("Epoch %d. Train Loss: %f, Train acc %f, Valid Loss: %f, Valid acc %f, "
                         % (epoch, train_loss / len(train_data),
                            correct / total, valid_loss / len(valid_data), valid_acc))
        else:
            epoch_str = ("Epoch %d. Loss: %f, Train acc %f, "
                         % (epoch, train_loss / len(train_data),
                            correct / total))
        prev_time = cur_time
        print(epoch_str + time_str + ', lr ' + str(trainer.learning_rate))
```

In [ ]:

```
ctx = mx.gpu(0)
num_epochs = 200
learning_rate = 0.1
weight_decay = 1e-4
lr_decay = 0.1
train(model, train_valid_iter, None, num_epochs, learning_rate, weight_decay, ctx, lr_decay)
```

In [ ]:

```
model.save_params('./resnet.params')
```

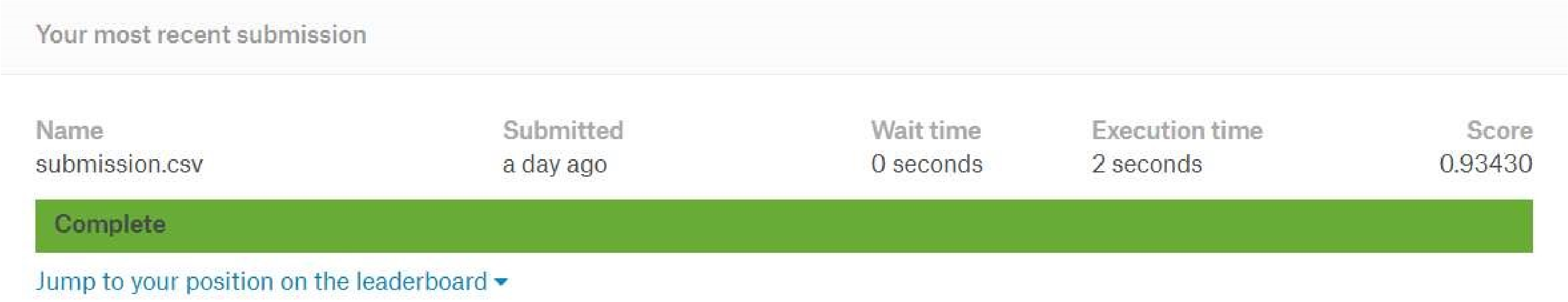
Classify the Testing Set and Submit Results on Kaggle (unchanged)

After obtaining a satisfactory model design and hyper-parameters, we use all training data sets (including validation sets) to retrain the model and classify the testing set.

In [14]:

```
import pandas as pd
preds = []

for X, _ in test_iter:
    y_hat = net(X.as_in_context(ctx))
    preds.extend(y_hat.argmax(axis=1).astype(int).asnumpy())
sorted_ids = list(range(1, len(test_ds) + 1))
sorted_ids.sort(key=lambda x: str(x))
df = pd.DataFrame({'id': sorted_ids, 'label': preds})
df['label'] = df['label'].apply(lambda x: train_valid_ds.synsets[x])
df.to_csv('submission.csv', index=False)
```



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	a day ago	0 seconds	2 seconds	0.93430

Complete

Jump to your position on the leaderboard ▾

Results