Homework 6 - Berkeley STAT 157

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

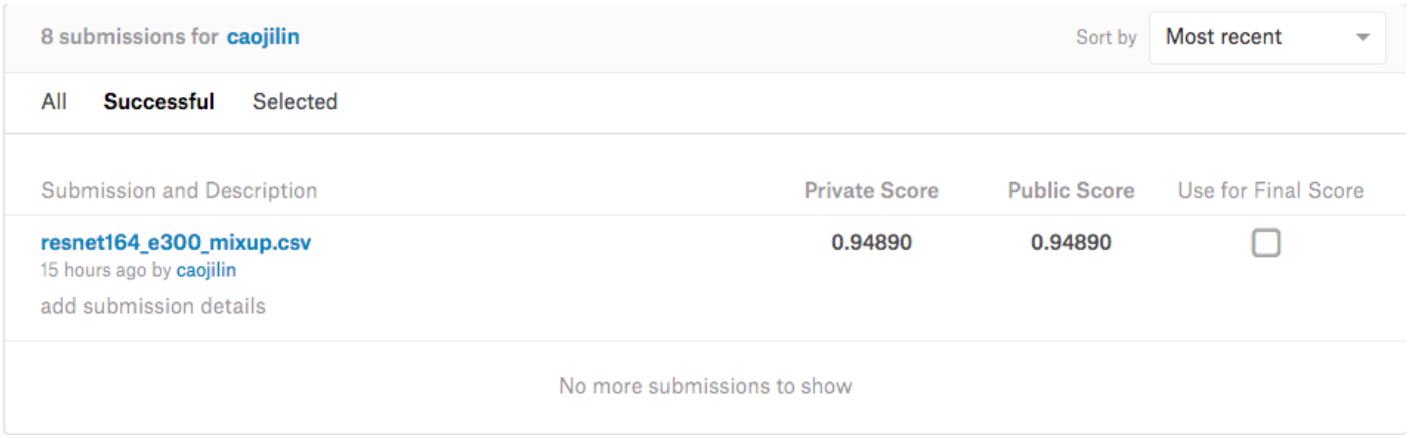
Handout 3/5/2019, due 3/12/2019 by 4pm. Please submit through gradescope.

In this homework, we will train a CNN model on CIFAR-10 and submit the results into Kaggle (https://www.kaggle.com/c/cifar-10). The rule is similar to homework 4:

- 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

The rest of this notebook contains a baseline ResNet-15 model to train on CIFAR-10. Please use it as a starting point. The end of this notebooks has several hints to improve your results.

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



In [1]:
```
import d2l
from mxnet import autograd, gluon, init
from mxnet.gluon import data as gdata, loss as gloss, nn
import os
import pandas as pd
import shutil
import time
```

Obtain and Organize the Data Sets

The competition data is divided into a training set and testing set. The training set contains 50,000 images. The testing set contains 300,000 images, of which 10,000 images are used for scoring, while the other 290,000 non-scoring images are included to prevent the manual labeling of the testing set and the submission of labeling results. The image formats in both data sets are PNG, with heights and widths of 32 pixels and three color channels (RGB). The images cover 10 categories: planes, cars, birds, cats, deer, dogs, frogs, horses, boats, and trucks. The upper-left corner of Figure 9.16 shows some images of planes, cars, and birds in the data set.

Download the Data Set

After logging in to Kaggle, we can click on the "Data" tab on the CIFAR-10 image classification competition webpage shown in Figure 9.16 and download the training data set "train.7z", the testing data set "test.7z", and the training data set labels "trainlabels.csv".

Unzip the Data Set

The training data set "train.7z" and the test data set "test.7z" need to be unzipped after downloading. After unzipping the data sets, store the training data set, test data set, and training data set labels in the following respective paths:

- ../data/kaggle_cifar10/train/[1-50000].png
- ../data/kaggle_cifar10/test/[1-300000].png
- ../data/kaggle_cifar10/trainLabels.csv

To make it easier to get started, we provide a small-scale sample of the data set mentioned above. "train_tiny.zip" contains 100 training examples, while "test_tiny.zip" contains only one test example. Their unzipped folder names are "train_tiny" and "test_tiny", respectively. In addition, unzip the zip file of the training data set labels to obtain the file "trainlabels.csv". If you are going to use the full data set of the Kaggle competition, you will also need to change the following `demo` variable to `False`.

In [2]:
```
demo = True # 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

We need to organize data sets to facilitate model training and testing. The following `read_label_file` function will be used to read the label file for the training data set. The parameter `valid_ratio` in this function is the ratio of the number of examples in the validation set to the number of examples in the original training set.

In [3]:
```
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 [4]:
```
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 [5]:
```
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', lab
el))
        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 [6]:
```
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 [7]:
```
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_rati
o)
    reorg_train_valid(data_dir, train_dir, input_dir, n_train_per_label,
                      idx_label)
    reorg_test(data_dir, test_dir, input_dir)
```

We use only 100 training example and one test example here. The folder names for the training and testing data sets are "train_tiny" and "test_tiny", respectively. Accordingly, we only set the batch size to 1. During actual training and testing, the complete data set of the Kaggle competition should be used and `batch_size` should be set to a larger integer, such as 128. We use 10% of the training examples as the validation set for tuning hyper-parameters.

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 integ
er
    train_dir, test_dir, batch_size = 'train_tiny', 'test_tiny', 1
else:
    train_dir, test_dir, batch_size = 'train', 'test', 64
data_dir, label_file = '../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 will cover image augmentation next week, you can ignore it for this homework.)

To cope with overfitting, we use image augmentation. For example, by adding `transforms.RandomFlipLeftRight()`, the images can be flipped at random. We can also perform normalization for the three RGB channels of color images using `transforms.Normalize()`. Below, we list some of these operations that you can choose to use or modify depending on requirements.

In [9]:
```
transform_train = gdata.vision.transforms.Compose([
    # Magnify the image to a square of 40 pixels in both height and widt
h
    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 [10]:
```
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 [11]:
```
# Read the original image file. Flag=1 indicates that the input image ha
s
# 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 [12]:
```
train_iter = gdata.DataLoader(train_ds.transform_first(transform_train),
                              batch_size, shuffle=True, last_batch='kee
p')
valid_iter = gdata.DataLoader(valid_ds.transform_first(transform_test),
                              batch_size, shuffle=True, last_batch='kee
p')
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='kee
p')
```

Define the Model

(We will cover hybridize next week. It often makes your model run faster, but you can ignore what it means for this homework.)

Here, we build the residual blocks based on the HybridBlock class, which is slightly different than the implementation described in the "Residual networks (ResNet)" (http://d2l.ai/chapter_convolutional-neural-networks/resnet.html) section. This is done to improve execution efficiency.

In [13]:
```
class Residual(nn.HybridBlock):
    def __init__(self, num_channels, use_1x1conv=False, strides=1, **kwa
rgs):
        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)
```

Next, we define the ResNet-18 model.

In [14]:
```
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 [ ]:
```
class Residual_v2_bottleneck(nn.HybridBlock):

    def __init__(self, channels, same_shape=True, **kwargs):
        super(Residual_v2_bottleneck, self).__init__(**kwargs)
        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=channels//4, kernel_size=1,
use_bias=False)
            self.bn2 = nn.BatchNorm()
            self.conv2 = nn.Conv2D(channels=channels//4, kernel_size=3,
strides=strides, padding=1, use_bias=False)
            self.bn3 = nn.BatchNorm()
            self.conv3 = nn.Conv2D(channels=channels, kernel_size=1, use
_bias=False)
            self.bn4 = nn.BatchNorm()   # add this

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

    def hybrid_forward(self, F, x):
        out = self.conv1(self.bn1(x))  # remove relu
        out = self.conv2(F.relu(self.bn2(out)))
        out = self.conv3(F.relu(self.bn3(out)))
        out = self.bn4(out)            # add this

        if not self.same_shape:
            x = self.conv4(x)
        return out + x


class ResNet164_v2(nn.HybridBlock):
    def __init__(self, num_classes, verbose=False, **kwargs):
        super(ResNet164_v2, self).__init__(**kwargs)
        self.verbose = verbose
        with self.name_scope():
            net = self.net = nn.HybridSequential()
            # block 1
            net.add(nn.Conv2D(channels=64, kernel_size=3, padding=1, str
ides=1, use_bias=False))
            # block 2
            for _ in range(27):
                net.add(Residual_v2_bottleneck(channels=64))
            # block 3
            net.add(Residual_v2_bottleneck(128, same_shape=False))
            for _ in range(26):
                net.add(Residual_v2_bottleneck(channels=128))
            # block4
            net.add(Residual_v2_bottleneck(256, same_shape=False))
            for _ in range(26):
                net.add(Residual_v2_bottleneck(channels=256))
            # block 5
            net.add(nn.BatchNorm())
            net.add(nn.Activation(activation='relu'))
```

```
            net.add(nn.AvgPool2D(pool_size=8))
            net.add(nn.Flatten())
            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
```

The CIFAR-10 image classification challenge uses 10 categories. We will perform Xavier random initialization on the model before training begins.

In [15]:
```
def get_net(ctx):
    num_classes = 10
    net = ResNet164_v2(num_classes)
    net.initialize(ctx=ctx, init=init.Xavier())
    return net

loss = gloss.SoftmaxCrossEntropyLoss()
```

Define the Training Functions

We will select the model and tune hyper-parameters according to the model's performance on the validation set. Next, we define the model training function `train`. We record the training time of each epoch, which helps us compare the time costs of different models.

In [16]:
```
def train(net, train_iter, valid_iter, num_epochs, lr, wd, ctx, lr_perio
d,
          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))
```

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 80 and 0.1 respectively, the learning rate of the optimization algorithm will be multiplied by 0.1 after every 80 epochs. For simplicity, we only train one epoch here.

In [ ]:
```
def transform_train(data, label):
    im = data.asnumpy()
    im = np.pad(im, ((4, 4), (4, 4), (0, 0)), mode='constant', constant_
values=0)
    im = nd.array(im, dtype='float32') / 255
    auglist = image.CreateAugmenter(data_shape=(3, 32, 32), resize=0, ra
nd_mirror=True,
                                    rand_crop=True,
                                   mean=np.array([0.4914, 0.4822, 0.4465
]),
                                   std=np.array([0.2023, 0.1994, 0.2010
]))
    for aug in auglist:
        im = aug(im)
    im = nd.transpose(im, (2, 0, 1)) # channel x width x height
    return im, nd.array([label]).astype('float32')
```

In [ ]:
```
train_iter = gdata.DataLoader(train_ds.transform_first(transform_train),
                              batch_size, shuffle=True, last_batch='kee
p')
valid_iter = gdata.DataLoader(valid_ds.transform_first(transform_test),
                              batch_size, shuffle=True, last_batch='kee
p')
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='kee
p')
```

In [9]:
```
ctx, num_epochs, lr, wd = d2l.try_gpu(), 200, 0.1, 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)
```

```
epoch 0, loss 2.16593, train_acc 0.3018, valid_acc 0.3185, Time 00:02:2
4,lr 0.1
epoch 1, loss 1.97320, train_acc 0.3922, valid_acc 0.3997, Time 00:02:4
3,lr 0.1
epoch 2, loss 1.87724, train_acc 0.4802, valid_acc 0.4904, Time 00:02:4
6,lr 0.1
epoch 3, loss 1.77747, train_acc 0.5290, valid_acc 0.5330, Time 00:02:5
1,lr 0.1
epoch 4, loss 1.70237, train_acc 0.6065, valid_acc 0.6088, Time 00:02:4
9,lr 0.1
epoch 5, loss 1.64160, train_acc 0.6250, valid_acc 0.6280, Time 00:02:5
0,lr 0.1
epoch 6, loss 1.59686, train_acc 0.6751, valid_acc 0.6758, Time 00:02:4
9,lr 0.1
epoch 7, loss 1.56121, train_acc 0.6758, valid_acc 0.6765, Time 00:02:5
0,lr 0.1
epoch 8, loss 1.52487, train_acc 0.6424, valid_acc 0.6397, Time 00:02:5
0,lr 0.1
epoch 9, loss 1.49709, train_acc 0.6987, valid_acc 0.6961, Time 00:02:5
0,lr 0.1
epoch 10, loss 1.50484, train_acc 0.7280, valid_acc 0.7212, Time 00:02:
46,lr 0.1
epoch 11, loss 1.47519, train_acc 0.6725, valid_acc 0.6656, Time 00:02:
48,lr 0.1
epoch 12, loss 1.45798, train_acc 0.7600, valid_acc 0.7484, Time 00:02:
52,lr 0.1
epoch 13, loss 1.44291, train_acc 0.6156, valid_acc 0.5938, Time 00:02:
51,lr 0.1
epoch 14, loss 1.45762, train_acc 0.7605, valid_acc 0.7583, Time 00:02:
51,lr 0.1
epoch 15, loss 1.43096, train_acc 0.7232, valid_acc 0.7098, Time 00:02:
51,lr 0.1
epoch 16, loss 1.41539, train_acc 0.7631, valid_acc 0.7487, Time 00:02:
53,lr 0.1
epoch 17, loss 1.41253, train_acc 0.7540, valid_acc 0.7345, Time 00:02:
51,lr 0.1
epoch 18, loss 1.40944, train_acc 0.7716, valid_acc 0.7644, Time 00:02:
49,lr 0.1
epoch 19, loss 1.40919, train_acc 0.7287, valid_acc 0.7108, Time 00:02:
51,lr 0.1
epoch 20, loss 1.39907, train_acc 0.7800, valid_acc 0.7620, Time 00:02:
51,lr 0.1
epoch 21, loss 1.39861, train_acc 0.7807, valid_acc 0.7715, Time 00:02:
52,lr 0.1
epoch 22, loss 1.39499, train_acc 0.7514, valid_acc 0.7349, Time 00:02:
51,lr 0.1
epoch 23, loss 1.37361, train_acc 0.7128, valid_acc 0.6935, Time 00:02:
55,lr 0.1
epoch 24, loss 1.37417, train_acc 0.7443, valid_acc 0.7253, Time 00:02:
53,lr 0.1
epoch 25, loss 1.37380, train_acc 0.7866, valid_acc 0.7695, Time 00:02:
43,lr 0.1
epoch 26, loss 1.37925, train_acc 0.7879, valid_acc 0.7765, Time 00:02:
46,lr 0.1
epoch 27, loss 1.37340, train_acc 0.7796, valid_acc 0.7598, Time 00:03:
16,lr 0.1
epoch 28, loss 1.37557, train_acc 0.7398, valid_acc 0.7125, Time 00:03:
```

```
10,lr 0.1
epoch 29, loss 1.36707, train_acc 0.7860, valid_acc 0.7735, Time 00:03:
15,lr 0.1
epoch 30, loss 1.36420, train_acc 0.7993, valid_acc 0.7810, Time 00:03:
23,lr 0.1
epoch 31, loss 1.35215, train_acc 0.8120, valid_acc 0.7932, Time 00:03:
17,lr 0.1
epoch 32, loss 1.35631, train_acc 0.7857, valid_acc 0.7702, Time 00:03:
14,lr 0.1
epoch 33, loss 1.35603, train_acc 0.7815, valid_acc 0.7667, Time 00:03:
12,lr 0.1
epoch 34, loss 1.34795, train_acc 0.7454, valid_acc 0.7367, Time 00:03:
10,lr 0.1
epoch 35, loss 1.34180, train_acc 0.8172, valid_acc 0.8031, Time 00:03:
14,lr 0.1
epoch 36, loss 1.35993, train_acc 0.7833, valid_acc 0.7733, Time 00:03:
12,lr 0.1
epoch 37, loss 1.33974, train_acc 0.7398, valid_acc 0.7188, Time 00:03:
15,lr 0.1
epoch 38, loss 1.36017, train_acc 0.7912, valid_acc 0.7699, Time 00:03:
15,lr 0.1
epoch 39, loss 1.34915, train_acc 0.7875, valid_acc 0.7749, Time 00:03:
05,lr 0.1
epoch 40, loss 1.32973, train_acc 0.7747, valid_acc 0.7594, Time 00:03:
01,lr 0.1
epoch 41, loss 1.33626, train_acc 0.7625, valid_acc 0.7478, Time 00:03:
03,lr 0.1
epoch 42, loss 1.34808, train_acc 0.7849, valid_acc 0.7636, Time 00:03:
03,lr 0.1
epoch 43, loss 1.34154, train_acc 0.8158, valid_acc 0.8040, Time 00:03:
04,lr 0.1
epoch 44, loss 1.33321, train_acc 0.8199, valid_acc 0.8063, Time 00:03:
04,lr 0.1
epoch 45, loss 1.33215, train_acc 0.7801, valid_acc 0.7579, Time 00:03:
04,lr 0.1
epoch 46, loss 1.33544, train_acc 0.7546, valid_acc 0.7470, Time 00:03:
07,lr 0.1
epoch 47, loss 1.34467, train_acc 0.7803, valid_acc 0.7600, Time 00:03:
03,lr 0.1
epoch 48, loss 1.32607, train_acc 0.8112, valid_acc 0.7971, Time 00:03:
03,lr 0.1
epoch 49, loss 1.33663, train_acc 0.7947, valid_acc 0.7833, Time 00:03:
05,lr 0.1
epoch 50, loss 1.34501, train_acc 0.7905, valid_acc 0.7728, Time 00:03:
07,lr 0.1
epoch 51, loss 1.32826, train_acc 0.7878, valid_acc 0.7675, Time 00:03:
05,lr 0.1
epoch 52, loss 1.32729, train_acc 0.7810, valid_acc 0.7703, Time 00:03:
05,lr 0.1
epoch 53, loss 1.32485, train_acc 0.8157, valid_acc 0.8082, Time 00:03:
11,lr 0.1
epoch 54, loss 1.33577, train_acc 0.7834, valid_acc 0.7568, Time 00:03:
04,lr 0.1
epoch 55, loss 1.32720, train_acc 0.7768, valid_acc 0.7620, Time 00:03:
05,lr 0.1
epoch 56, loss 1.33960, train_acc 0.8235, valid_acc 0.8161, Time 00:05:
52,lr 0.1
```

```
epoch 57, loss 1.31656, train_acc 0.7968, valid_acc 0.7894, Time 00:13:
25,lr 0.1
epoch 58, loss 1.31316, train_acc 0.8177, valid_acc 0.8108, Time 00:17:
19,lr 0.1
epoch 59, loss 1.33070, train_acc 0.8314, valid_acc 0.8126, Time 00:10:
16,lr 0.1
epoch 60, loss 1.32717, train_acc 0.7980, valid_acc 0.7887, Time 00:03:
17,lr 0.1
epoch 61, loss 1.32127, train_acc 0.8070, valid_acc 0.7889, Time 00:03:
09,lr 0.1
epoch 62, loss 1.32567, train_acc 0.7961, valid_acc 0.7795, Time 00:03:
10,lr 0.1
epoch 63, loss 1.33797, train_acc 0.8074, valid_acc 0.8001, Time 00:03:
04,lr 0.1
epoch 64, loss 1.29923, train_acc 0.7971, valid_acc 0.7815, Time 00:03:
12,lr 0.1
epoch 65, loss 1.31300, train_acc 0.8078, valid_acc 0.7979, Time 00:03:
34,lr 0.1
epoch 66, loss 1.32401, train_acc 0.7414, valid_acc 0.7237, Time 00:03:
22,lr 0.1
epoch 67, loss 1.31541, train_acc 0.8264, valid_acc 0.8114, Time 00:03:
07,lr 0.1
epoch 68, loss 1.32304, train_acc 0.8025, valid_acc 0.7884, Time 00:02:
59,lr 0.1
epoch 69, loss 1.33490, train_acc 0.7639, valid_acc 0.7398, Time 00:02:
57,lr 0.1
epoch 70, loss 1.34050, train_acc 0.8351, valid_acc 0.8208, Time 00:03:
01,lr 0.1
epoch 71, loss 1.31406, train_acc 0.7746, valid_acc 0.7570, Time 00:03:
02,lr 0.1
epoch 72, loss 1.29430, train_acc 0.8064, valid_acc 0.7842, Time 00:03:
05,lr 0.1
epoch 73, loss 1.31676, train_acc 0.8037, valid_acc 0.7839, Time 00:03:
06,lr 0.1
epoch 74, loss 1.31748, train_acc 0.7416, valid_acc 0.7223, Time 00:04:
37,lr 0.1
epoch 75, loss 1.30156, train_acc 0.7944, valid_acc 0.7782, Time 00:17:
31,lr 0.1
epoch 76, loss 1.32197, train_acc 0.7859, valid_acc 0.7640, Time 00:17:
31,lr 0.1
epoch 77, loss 1.32253, train_acc 0.8163, valid_acc 0.8001, Time 00:07:
10,lr 0.1
epoch 78, loss 1.33143, train_acc 0.7925, valid_acc 0.7768, Time 00:03:
26,lr 0.1
epoch 79, loss 1.31097, train_acc 0.8077, valid_acc 0.7898, Time 00:03:
04,lr 0.1
epoch 80, loss 1.20304, train_acc 0.8780, valid_acc 0.8592, Time 00:03:
19,lr 0.01
epoch 81, loss 1.14880, train_acc 0.8944, valid_acc 0.8764, Time 00:03:
14,lr 0.01
epoch 82, loss 1.14279, train_acc 0.8997, valid_acc 0.8846, Time 00:03:
06,lr 0.01
epoch 83, loss 1.12559, train_acc 0.8999, valid_acc 0.8835, Time 00:03:
19,lr 0.01
epoch 84, loss 1.14183, train_acc 0.9056, valid_acc 0.8877, Time 00:03:
07,lr 0.01
epoch 85, loss 1.10517, train_acc 0.9109, valid_acc 0.8975, Time 00:03:
```

```
09,lr 0.01
epoch 86, loss 1.12380, train_acc 0.9147, valid_acc 0.8973, Time 00:03:
07,lr 0.01
epoch 87, loss 1.10470, train_acc 0.9093, valid_acc 0.8908, Time 00:03:
09,lr 0.01
epoch 88, loss 1.09398, train_acc 0.9128, valid_acc 0.8927, Time 00:03:
09,lr 0.01
epoch 89, loss 1.09117, train_acc 0.9117, valid_acc 0.8899, Time 00:03:
11,lr 0.01
epoch 90, loss 1.09854, train_acc 0.9201, valid_acc 0.9009, Time 00:03:
06,lr 0.01
epoch 91, loss 1.09604, train_acc 0.9193, valid_acc 0.8959, Time 00:03:
06,lr 0.01
epoch 92, loss 1.07482, train_acc 0.9182, valid_acc 0.8973, Time 00:03:
15,lr 0.01
epoch 93, loss 1.09745, train_acc 0.9165, valid_acc 0.8902, Time 00:06:
27,lr 0.01
epoch 94, loss 1.07607, train_acc 0.9263, valid_acc 0.9030, Time 00:17:
37,lr 0.01
epoch 95, loss 1.07369, train_acc 0.9214, valid_acc 0.8996, Time 00:17:
45,lr 0.01
epoch 96, loss 1.07177, train_acc 0.9225, valid_acc 0.8964, Time 00:05:
32,lr 0.01
epoch 97, loss 1.06274, train_acc 0.9271, valid_acc 0.8991, Time 00:03:
19,lr 0.01
epoch 98, loss 1.09669, train_acc 0.9258, valid_acc 0.8999, Time 00:03:
18,lr 0.01
epoch 99, loss 1.07988, train_acc 0.9306, valid_acc 0.9061, Time 00:06:
53,lr 0.01
epoch 100, loss 1.08015, train_acc 0.9270, valid_acc 0.9012, Time 00:1
7:35,lr 0.01
epoch 101, loss 1.08203, train_acc 0.9295, valid_acc 0.9018, Time 00:1
7:38,lr 0.01
epoch 102, loss 1.08058, train_acc 0.9311, valid_acc 0.9021, Time 00:0
5:31,lr 0.01
epoch 103, loss 1.08665, train_acc 0.9260, valid_acc 0.9001, Time 00:0
3:25,lr 0.01
epoch 104, loss 1.08120, train_acc 0.9300, valid_acc 0.9021, Time 00:0
3:10,lr 0.01
epoch 105, loss 1.05957, train_acc 0.9265, valid_acc 0.8975, Time 00:0
3:10,lr 0.01
epoch 106, loss 1.05478, train_acc 0.9266, valid_acc 0.8982, Time 00:0
3:13,lr 0.01
epoch 107, loss 1.06689, train_acc 0.9312, valid_acc 0.9001, Time 00:0
3:11,lr 0.01
epoch 108, loss 1.06822, train_acc 0.9312, valid_acc 0.9061, Time 00:0
3:13,lr 0.01
epoch 109, loss 1.06758, train_acc 0.9385, valid_acc 0.9105, Time 00:0
3:14,lr 0.01
epoch 110, loss 1.07110, train_acc 0.9270, valid_acc 0.8968, Time 00:0
3:13,lr 0.01
epoch 111, loss 1.06828, train_acc 0.9307, valid_acc 0.8983, Time 00:0
3:13,lr 0.01
epoch 112, loss 1.05636, train_acc 0.9328, valid_acc 0.9036, Time 00:0
3:21,lr 0.01
epoch 113, loss 1.07447, train_acc 0.9359, valid_acc 0.9053, Time 00:0
4:34,lr 0.01
```

```
epoch 114, loss 1.07265, train_acc 0.9332, valid_acc 0.9059, Time 00:1
7:39,lr 0.01
epoch 115, loss 1.05529, train_acc 0.9348, valid_acc 0.9028, Time 00:1
7:39,lr 0.01
epoch 116, loss 1.05023, train_acc 0.9256, valid_acc 0.8970, Time 00:0
7:46,lr 0.01
epoch 117, loss 1.04672, train_acc 0.9373, valid_acc 0.9057, Time 00:0
3:22,lr 0.01
epoch 118, loss 1.03829, train_acc 0.9320, valid_acc 0.9025, Time 00:0
3:32,lr 0.01
epoch 119, loss 1.08419, train_acc 0.9336, valid_acc 0.9012, Time 00:0
3:19,lr 0.01
epoch 120, loss 1.05413, train_acc 0.9351, valid_acc 0.9056, Time 00:1
4:12,lr 0.01
epoch 121, loss 1.04569, train_acc 0.9354, valid_acc 0.9079, Time 00:1
7:39,lr 0.01
epoch 122, loss 1.07001, train_acc 0.9415, valid_acc 0.9089, Time 00:1
2:30,lr 0.01
epoch 123, loss 1.07387, train_acc 0.9355, valid_acc 0.9066, Time 00:0
3:40,lr 0.01
epoch 124, loss 1.05514, train_acc 0.9285, valid_acc 0.8993, Time 00:0
3:26,lr 0.01
epoch 125, loss 1.04990, train_acc 0.9395, valid_acc 0.9066, Time 00:0
3:27,lr 0.01
epoch 126, loss 1.05501, train_acc 0.9334, valid_acc 0.9089, Time 00:0
3:22,lr 0.01
epoch 127, loss 1.06621, train_acc 0.9315, valid_acc 0.9032, Time 00:1
2:21,lr 0.01
epoch 128, loss 1.04811, train_acc 0.9385, valid_acc 0.9048, Time 00:1
8:00,lr 0.01
epoch 129, loss 1.05576, train_acc 0.9301, valid_acc 0.8980, Time 00:1
3:53,lr 0.01
epoch 130, loss 1.07707, train_acc 0.9334, valid_acc 0.9005, Time 00:0
4:08,lr 0.01
epoch 131, loss 1.03549, train_acc 0.9359, valid_acc 0.9057, Time 00:0
8:16,lr 0.01
epoch 132, loss 1.04940, train_acc 0.9345, valid_acc 0.9013, Time 00:1
7:44,lr 0.01
epoch 133, loss 1.05732, train_acc 0.9322, valid_acc 0.9077, Time 00:1
4:53,lr 0.01
epoch 134, loss 1.06246, train_acc 0.9346, valid_acc 0.9062, Time 00:0
3:42,lr 0.01
epoch 135, loss 1.05425, train_acc 0.9290, valid_acc 0.8943, Time 00:0
3:27,lr 0.01
epoch 136, loss 1.04927, train_acc 0.9318, valid_acc 0.8996, Time 00:0
3:25,lr 0.01
epoch 137, loss 1.03976, train_acc 0.9398, valid_acc 0.9097, Time 00:0
3:30,lr 0.01
epoch 138, loss 1.04665, train_acc 0.9309, valid_acc 0.8946, Time 00:0
3:27,lr 0.01
epoch 139, loss 1.04799, train_acc 0.9379, valid_acc 0.9074, Time 00:0
3:24,lr 0.01
epoch 140, loss 1.05712, train_acc 0.9353, valid_acc 0.9042, Time 00:0
3:28,lr 0.01
epoch 141, loss 1.05111, train_acc 0.9382, valid_acc 0.9080, Time 00:0
3:27,lr 0.01
epoch 142, loss 1.03855, train_acc 0.9345, valid_acc 0.9018, Time 00:0
```

```
3:25,lr 0.01
epoch 143, loss 1.02854, train_acc 0.9428, valid_acc 0.9129, Time 00:0
3:27,lr 0.01
epoch 144, loss 1.04087, train_acc 0.9386, valid_acc 0.9081, Time 00:0
3:19,lr 0.01
epoch 145, loss 1.03890, train_acc 0.9356, valid_acc 0.9010, Time 00:0
3:22,lr 0.01
epoch 146, loss 1.03986, train_acc 0.9304, valid_acc 0.9002, Time 00:0
3:26,lr 0.01
epoch 147, loss 1.04848, train_acc 0.9377, valid_acc 0.9056, Time 00:0
3:32,lr 0.01
epoch 148, loss 1.04825, train_acc 0.9367, valid_acc 0.9075, Time 00:0
3:29,lr 0.01
epoch 149, loss 1.04058, train_acc 0.9272, valid_acc 0.8968, Time 00:0
3:26,lr 0.01
epoch 150, loss 1.01519, train_acc 0.9564, valid_acc 0.9250, Time 00:0
3:20,lr 0.001
epoch 151, loss 1.00549, train_acc 0.9589, valid_acc 0.9281, Time 00:0
3:25,lr 0.001
epoch 152, loss 0.97932, train_acc 0.9622, valid_acc 0.9313, Time 00:0
3:29,lr 0.001
epoch 153, loss 0.98294, train_acc 0.9609, valid_acc 0.9287, Time 00:0
3:16,lr 0.001
epoch 154, loss 0.97573, train_acc 0.9608, valid_acc 0.9279, Time 00:0
3:12,lr 0.001
epoch 155, loss 0.95079, train_acc 0.9650, valid_acc 0.9313, Time 00:0
3:08,lr 0.001
epoch 156, loss 0.95617, train_acc 0.9647, valid_acc 0.9294, Time 00:0
3:09,lr 0.001
epoch 157, loss 0.96072, train_acc 0.9658, valid_acc 0.9312, Time 00:0
3:08,lr 0.001
epoch 158, loss 0.96697, train_acc 0.9661, valid_acc 0.9323, Time 00:0
3:10,lr 0.001
epoch 159, loss 0.96537, train_acc 0.9680, valid_acc 0.9333, Time 00:0
3:07,lr 0.001
epoch 160, loss 0.97062, train_acc 0.9683, valid_acc 0.9312, Time 00:0
3:10,lr 0.001
epoch 161, loss 0.95298, train_acc 0.9687, valid_acc 0.9325, Time 00:0
3:08,lr 0.001
epoch 162, loss 0.96177, train_acc 0.9658, valid_acc 0.9306, Time 00:0
3:09,lr 0.001
epoch 163, loss 0.95081, train_acc 0.9690, valid_acc 0.9339, Time 00:0
3:09,lr 0.001
epoch 164, loss 0.93594, train_acc 0.9696, valid_acc 0.9326, Time 00:0
3:11,lr 0.001
epoch 165, loss 0.95492, train_acc 0.9687, valid_acc 0.9345, Time 00:0
3:09,lr 0.001
epoch 166, loss 0.97178, train_acc 0.9704, valid_acc 0.9322, Time 00:0
3:08,lr 0.001
epoch 167, loss 0.95078, train_acc 0.9702, valid_acc 0.9324, Time 00:0
3:10,lr 0.001
epoch 168, loss 0.96071, train_acc 0.9689, valid_acc 0.9326, Time 00:0
3:10,lr 0.001
epoch 169, loss 0.93886, train_acc 0.9710, valid_acc 0.9342, Time 00:0
3:10,lr 0.001
epoch 170, loss 0.94743, train_acc 0.9728, valid_acc 0.9334, Time 00:0
3:11,lr 0.001
```

```
epoch 171, loss 0.94780, train_acc 0.9717, valid_acc 0.9340, Time 00:0
3:12,lr 0.001
epoch 172, loss 0.95980, train_acc 0.9689, valid_acc 0.9309, Time 00:0
3:09,lr 0.001
epoch 173, loss 0.94095, train_acc 0.9727, valid_acc 0.9347, Time 00:0
3:09,lr 0.001
epoch 174, loss 0.92987, train_acc 0.9736, valid_acc 0.9359, Time 00:0
3:09,lr 0.001
epoch 175, loss 0.95549, train_acc 0.9696, valid_acc 0.9308, Time 00:0
3:10,lr 0.001
epoch 176, loss 0.95595, train_acc 0.9726, valid_acc 0.9326, Time 00:0
3:09,lr 0.001
epoch 177, loss 0.92947, train_acc 0.9738, valid_acc 0.9339, Time 00:0
3:10,lr 0.001
epoch 178, loss 0.95370, train_acc 0.9723, valid_acc 0.9349, Time 00:0
3:08,lr 0.001
epoch 179, loss 0.93811, train_acc 0.9701, valid_acc 0.9328, Time 00:0
3:09,lr 0.001
epoch 180, loss 0.95937, train_acc 0.9726, valid_acc 0.9340, Time 00:0
3:11,lr 0.001
epoch 181, loss 0.93950, train_acc 0.9752, valid_acc 0.9367, Time 00:0
3:10,lr 0.001
epoch 182, loss 0.94722, train_acc 0.9721, valid_acc 0.9335, Time 00:0
3:12,lr 0.001
epoch 183, loss 0.93646, train_acc 0.9719, valid_acc 0.9325, Time 00:0
3:10,lr 0.001
epoch 184, loss 0.92570, train_acc 0.9739, valid_acc 0.9344, Time 00:0
3:13,lr 0.001
epoch 185, loss 0.93027, train_acc 0.9747, valid_acc 0.9343, Time 00:0
3:13,lr 0.001
epoch 186, loss 0.94062, train_acc 0.9722, valid_acc 0.9315, Time 00:0
3:12,lr 0.001
epoch 187, loss 0.92805, train_acc 0.9750, valid_acc 0.9347, Time 00:0
3:10,lr 0.001
epoch 188, loss 0.94461, train_acc 0.9739, valid_acc 0.9324, Time 00:0
3:10,lr 0.001
epoch 189, loss 0.94548, train_acc 0.9754, valid_acc 0.9353, Time 00:0
3:15,lr 0.001
epoch 190, loss 0.95523, train_acc 0.9763, valid_acc 0.9357, Time 00:0
3:14,lr 0.001
epoch 191, loss 0.93185, train_acc 0.9758, valid_acc 0.9360, Time 00:0
3:11,lr 0.001
epoch 192, loss 0.92303, train_acc 0.9740, valid_acc 0.9319, Time 00:0
3:13,lr 0.001
epoch 193, loss 0.93946, train_acc 0.9757, valid_acc 0.9315, Time 00:0
3:13,lr 0.001
epoch 194, loss 0.92639, train_acc 0.9743, valid_acc 0.9313, Time 00:0
3:14,lr 0.001
epoch 195, loss 0.94102, train_acc 0.9752, valid_acc 0.9319, Time 00:0
3:13,lr 0.001
epoch 196, loss 0.91991, train_acc 0.9761, valid_acc 0.9337, Time 00:0
3:13,lr 0.001
epoch 197, loss 0.94033, train_acc 0.9772, valid_acc 0.9347, Time 00:0
3:12,lr 0.001
epoch 198, loss 0.94253, train_acc 0.9786, valid_acc 0.9363, Time 00:0
3:12,lr 0.001
```

```
epoch 199, loss 0.92489, train_acc 0.9776, valid_acc 0.9347, Time 00:0
3:13,lr 0.001
```

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 classify the testing set.

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

Hints to Improve Your Results

- You should use the compete CIFAR-10 dataset to get meaningful results.
- You'd better use a GPU machine to run it, otherwise it'll be quite slow. (Please DON'T FORGET to stop or terminate your instance if you are not using it, otherwise AWS will change you)
- Change the `batch_size` and number of epochs `num_epochs` to 128 and 100, respectively. (It will take a while to run.)
- Change to another network, such as ResNet-34 or Inception