5. Model

Loading packages

In [1]:

```
import mxnet as mx
from mxnet import init, gluon, nd, autograd, image
from mxnet.gluon import nn
from mxnet.gluon.data import vision
from mxnet.gluon.model_zoo import vision as models
import numpy as np
import pandas as pd
from tqdm import tqdm
import cv2
import h5py
import os
from glob import glob
import matplotlib.pyplot as plt

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Change the following to mx.cpu() if you don't have GPU in your compute
# To use different GPU, you can try "ctx = mx.gpu(1)", where 1 is the fi
ctx = mx.gpu()
```

Setting parameters

In [2]:

```
data_dir        = 'data'
batch_size      = 128
learning_rate   = 1e-3
epochs          = 150
lr_decay        = 0.95
lr_decay2       = 0.8
lr_period       = 100
submit_fileName = 'pred.csv'
```

Code

Aggregating label

In [3]:

```
synset = list(pd.read_csv(os.path.join('.', data_dir, 'sample_submission
n = len(glob(os.path.join('.', data_dir, 'Images', '*', '*.jpg')))

y = nd.zeros((n,))
for i, file_name in tqdm(enumerate(glob(os.path.join('.', data_dir, 'Ima
    y[i] = synset.index(file_name.split('/')[3][10:].lower())
    nd.waitall()
```

```
100%|██████████| 20580/20580 [00:01<00:00, 10626.80it/s]
```

Loading features of Stanford dogs dataset

In [4]:

```
features = [nd.load(os.path.join(data_dir, 'features_incep.nd'))[0], \
            nd.load(os.path.join(data_dir, 'features_res.nd'))[0]]
features = nd.concat(*features, dim=1)

features.shape
```

Out[4]:

```
(20580, 4096)
```

Loading features of testing dataset

In [5]:

```
models   = ['incep', 'res']
features_test = [nd.load(os.path.join(data_dir, 'features_test_%s.nd') %
features_test = nd.concat(*features_test, dim=1)

print(features_test.shape)
```

```
(10357, 4096)
```

Neural Network

In [6]:

```
def build_model():
    net = nn.Sequential()
    with net.name_scope():
        net.add(nn.BatchNorm())
        net.add(nn.Dense(1024))
        net.add(nn.BatchNorm())
        net.add(nn.Activation('relu'))
#         net.add(nn.Dropout(0.5))
        net.add(nn.Dense(512))
        net.add(nn.BatchNorm())
        net.add(nn.Activation('relu'))
#         net.add(nn.Dropout(0.5))
        net.add(nn.Dense(120))
    net.initialize(ctx=ctx)
    return net


def accuracy(output, labels):
    return nd.mean(nd.argmax(output, axis=1) == labels).asscalar()


def evaluate(net, data_iter):
    loss, acc, n = 0., 0., 0.
    steps = len(data_iter)
    for data, label in data_iter:
        data, label = data.as_in_context(ctx), label.as_in_context(ctx)
        output = net(data)
        acc += accuracy(output, label)
        loss += nd.mean(softmax_cross_entropy(output, label)).asscalar()
    return loss/steps, acc/steps
```

In [7]:

```
data_iter_train       = gluon.data.DataLoader(gluon.data.ArrayDataset(fea
softmax_cross_entropy = gluon.loss.SoftmaxCrossEntropyLoss()
net                   = build_model()
trainer               = gluon.Trainer(net.collect_params(), 'adam', {'lea
```

In [8]:

```
%%time

# https://github.com/yinglang/CIFAR10_mxnet/blob/master/CIFAR10_train.md

for epoch in range(epochs):
    if epoch <= lr_period:
        trainer.set_learning_rate(trainer.learning_rate * lr_decay)
    else:
        trainer.set_learning_rate(trainer.learning_rate * lr_decay2)
    train_loss = 0.
    train_acc = 0.
    steps = len(data_iter_train)
    for data, label in data_iter_train:
        data, label = data.as_in_context(ctx), label.as_in_context(ctx)
        with autograd.record():
            output = net(data)
            loss = softmax_cross_entropy(output, label)
        loss.backward()
        trainer.step(batch_size)
        train_loss += nd.mean(loss).asscalar()
        train_acc += accuracy(output, label)

    val_loss, val_acc = evaluate(net, data_iter_train)

    if epoch % 10 == 0:
        print("Epoch %d. loss: %.4f, acc: %.2f%%, val_loss %.4f, val_acc
            epoch+1, train_loss/steps, train_acc/steps*100, val_loss, va

print("Epoch %d. loss: %.4f, acc: %.2f%%, val_loss %.4f, val_acc %.2f%%"
    epoch+1, train_loss/steps, train_acc/steps*100, val_loss, val_acc*10
```

```
Epoch 1. loss: 0.5165, acc: 87.83%, val_loss 0.1367, val_acc 95.91%
Epoch 11. loss: 0.0140, acc: 99.56%, val_loss 0.0122, val_acc 99.62%
Epoch 21. loss: 0.0059, acc: 99.75%, val_loss 0.0046, val_acc 99.80%
Epoch 31. loss: 0.0043, acc: 99.74%, val_loss 0.0035, val_acc 99.80%
Epoch 41. loss: 0.0038, acc: 99.72%, val_loss 0.0029, val_acc 99.81%
Epoch 51. loss: 0.0034, acc: 99.73%, val_loss 0.0027, val_acc 99.82%
Epoch 61. loss: 0.0031, acc: 99.75%, val_loss 0.0027, val_acc 99.82%
Epoch 71. loss: 0.0029, acc: 99.77%, val_loss 0.0026, val_acc 99.82%
Epoch 81. loss: 0.0028, acc: 99.73%, val_loss 0.0026, val_acc 99.82%
Epoch 91. loss: 0.0027, acc: 99.76%, val_loss 0.0026, val_acc 99.82%
Epoch 101. loss: 0.0026, acc: 99.78%, val_loss 0.0026, val_acc 99.82%
Epoch 111. loss: 0.0026, acc: 99.82%, val_loss 0.0026, val_acc 99.82%
Epoch 121. loss: 0.0026, acc: 99.82%, val_loss 0.0026, val_acc 99.82%
Epoch 131. loss: 0.0026, acc: 99.83%, val_loss 0.0026, val_acc 99.82%
Epoch 141. loss: 0.0026, acc: 99.82%, val_loss 0.0026, val_acc 99.82%
Epoch 150. loss: 0.0026, acc: 99.83%, val_loss 0.0026, val_acc 99.82%
CPU times: user 9min 30s, sys: 1min 9s, total: 10min 39s
Wall time: 6min 32s
```

Applying the trained network on the testing features

In [9]:

```
output = nd.softmax(net(nd.array(features_test).as_in_context(ctx))).asn
```

Outputing submission file

In [10]:

```
df_pred = pd.read_csv(os.path.join('.', data_dir, 'sample_submission.csv

for i, c in enumerate(df_pred.columns[1:]):
    df_pred[c] = output[:,i]

df_pred.to_csv(os.path.join('.', data_dir, submit_fileName), index=None)
```