In [1]:

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
from sklearn.decomposition import PCA
import torch.utils.data
import pickle
from tqdm import tqdm
import time
import gc
import collections
```

In [2]:

```
'''
set parameters
'''
base_path = '.'
train_path = base_path + '/train.npy'
test_path = base_path + '/test.npy'
train_labels_path = base_path + '/train_labels.npy'
dev_labels_path = base_path +  '/dev_labels.npy'
dev_path = base_path + '/dev.npy'

pca_available = True
padding_method = 'self'
# padding_method = 'zero'

device = torch.device('cuda:0')
n_labels = 138
n_features = 40
n_epoch = 10
context_num = 18
```

In [3]:

```
def load_train_data():
    t0 = time.time()
    print("Start loading training data...")
    train = np.load(train_path, allow_pickle=True)
    train_labels = np.load(train_labels_path, allow_pickle=True)
    t1 = time.time()
    elapsed_time = t1 - t0
    print("Done loading training data in {0} minutes...".format(elapsed_time/60)

    return train, train_labels
```

In [4]:

```
def load_validation_data():
    t0 = time.time()
    print("Start loading validation data...")
    val = np.load(dev_path, allow_pickle=True)
    val_labels = np.load(dev_labels_path, allow_pickle=True)
    t1 = time.time()
    elapsed_time = t1 - t0
    print("Done loading validation data in {0} minutes...".format(elapsed_time/6
    
    return val, val_labels
```

In [5]:
```
def load_test_data():
    t0 = time.time()
    print("Start loading test data...")
    test = np.load(test_path, allow_pickle=True)
    t1 = time.time()
    elapsed_time = t1 - t0
    print("Done loading test data in {0} minutes...".format(elapsed_time/60))
    return test
```

In [6]:
```
def load_and_process_data(features, labels, pca, context_num):
    '''
    use the first&last frame of one utterance to pad the empty frame
    '''
    t0 = time.time()
    padding_features = np.concatenate([np.concatenate(( \
                                            np.ones((context_num, pca.n_com
                                            pca.transform(features[i]), \
                                            np.ones((context_num, pca.n_com
                                            for i in range(len(features))])
    padding_features = torch.Tensor(padding_features)
    del features

    '''
    corresponding label for padding frames
    '''
    false_labels = np.array([-1]*context_num)
    padding_labels = np.concatenate([np.concatenate(( \
                                            false_labels, \
                                            labels[i], \
                                            false_labels)) \
                                            for i in range(len(labels))])
    padding_labels = torch.Tensor(padding_labels)
    del labels

    gc.collect()

    return padding_features, padding_labels
```

In [7]:
```
ass ContextDataset(Dataset):

  def __init__(self, context_num, features, targets):

      self.context_num = context_num
      self.features = features
      self.targets = targets

  def __len__(self):

      return len(self.targets)

  def __getitem__(self, index):

      if index-self.context_num >= 0 and index+self.context_num+1 <= len(self.tar
          '''
          no need for padding
          '''
          X = self.features[index-self.context_num: index+self.context_num+1].res
          Y = self.targets[index].long()
      elif index-self.context_num < 0:
          '''
          padding for pre frames, actually doesnt matter since we drop this 'fals
          '''
          X = torch.cat((torch.zeros(self.context_num-index, self.features.shape[
          Y = self.targets[index].long()
      else:
          '''
          padding for post frames, same as before
          '''
          X = torch.cat((self.features[index-self.context_num:], torch.zeros(inde
          Y = self.targets[index].long()

      return index, X, Y
```

In [8]:
```
class SpeechDataset(Dataset):

    def __init__(self, speechdataset):

        self.features = [speechdataset[i][1] for i in range(len(speechdataset)) i
        self.targets = [speechdataset[i][2] for i in range(len(speechdataset)) i

    def __len__(self):

        return len(self.targets)

    def __getitem__(self, index):

        return index, self.features[index], self.targets[index]
```

In [9]:
```
#1000,2048,1024,512,256+2,138
class SpeechNet(nn.Module):

    def __init__(self, context_num):

        super(SpeechNet, self).__init__()
        self.relu1 = nn.ReLU()
        self.relu2 = nn.ReLU()
        self.relu3 = nn.ReLU()
        self.relu4 = nn.ReLU()
        self.relu5 = nn.ReLU()
        self.relu6 = nn.ReLU()
        self.relu7 = nn.ReLU()
        self.relu8 = nn.ReLU()

        self.linear1 = nn.Linear((2*context_num+1)*pca.n_components, 2048)
        self.linear2 = nn.Linear(2048, 1024)
        self.linear3 = nn.Linear(1024, 810)
        self.linear4 = nn.Linear(810, 720)
        self.linear5 = nn.Linear(720, 512)
        self.linear6 = nn.Linear(512, 428)
        self.linear7 = nn.Linear(428, 300)
        self.linear8 = nn.Linear(300, 256)
        self.out = nn.Linear(256+2, 138)

        self.batchnorm1 = nn.BatchNorm1d(2048)
        self.batchnorm2 = nn.BatchNorm1d(1024)
        self.batchnorm3 = nn.BatchNorm1d(810)
        self.batchnorm4 = nn.BatchNorm1d(720)
        self.batchnorm5 = nn.BatchNorm1d(512)
        self.batchnorm6 = nn.BatchNorm1d(428)
        self.batchnorm7 = nn.BatchNorm1d(300)
        self.batchnorm8 = nn.BatchNorm1d(256+2)


        self.dropout1 = nn.Dropout(0.1)
        self.dropout2 = nn.Dropout(0.05)
        self.dropout3 = nn.Dropout(0.1)
        self.dropout4 = nn.Dropout(0.05)
        self.dropout5 = nn.Dropout(0.05)
        self.dropout6 = nn.Dropout(0.05)
        self.dropout7 = nn.Dropout(0.05)

    def forward(self, x):

        x = self.linear1(x)
        x = self.batchnorm1(x)
        x = self.relu1(x)
        #2048
        x = self.dropout1(x)
        x = self.linear2(x)
        x = self.batchnorm2(x)
        x = self.relu2(x)
        #1024
        x = self.dropout2(x)
        x = self.linear3(x)
```

```
        x = self.batchnorm3(x)
        x = self.relu3(x)
        #810
        x = self.dropout3(x)
        x = self.linear4(x)
        x = self.batchnorm4(x)
        x = self.relu4(x)

        #512
        x = self.dropout4(x)
        x = self.linear5(x)
        x = self.batchnorm5(x)
        x = self.relu5(x)

        #512
        x = self.dropout5(x)
        x = self.linear6(x)
        x = self.batchnorm6(x)
        x = self.relu6(x)
        #428
        x = self.dropout6(x)
        x = self.linear7(x)
        x = self.batchnorm7(x)
        x = self.relu7(x)
        #300
        x = self.dropout7(x)
        x = self.linear8(x)
        #300

        avg_pool1 = torch.mean(x, 1, keepdims = True)
        max_pool1,_ = torch.max(x, 1, keepdims = True)

        conc = torch.cat((x, avg_pool1, max_pool1), 1)
        conc = self.batchnorm8(conc)
        output = self.out(conc)

        return output
```

In [10]:
```
def generate_dataset(context_num, features, labels):
    t0 = time.time()
    print("It may takes 20 minutes to generate train dataset...")
    context_dataset = ContextDataset(context_num, features, labels)
    dataset = SpeechDataset(context_dataset)
    t1 = time.time()
    print("Dataset generated. Elapsed time: {0}".format((t1-t0)/60))
    return dataset
```

In [11]:
```
def weights_init(m):
    if isinstance(m, nn.Conv2d):
        xavier(m.weight.data)
        xavier(m.bias.data)
```

In [12]:
```
def scale_cos(x):
    start = 5e-3
    end = 1e-5
    return start + (1 + np.cos(np.pi * (1 - x))) * (end - start) / 2
```

In [13]:
```
def second_scale_cos(x):
    start = 1e-4
    end = 1e-8
    return start + (1 + np.cos(np.pi * (1 - x))) * (end - start) / 2
```

In [14]:
```
class ParamScheduler:

    def __init__(self, optimizer, scale_fn, total_steps):

        self.optimizer = optimizer
        self.scale_fn = scale_fn
        self.total_steps = total_steps
        self.current_iteration = 0

    def batch_step(self):
        for param_group in self.optimizer.param_groups:
            param_group['lr'] = self.scale_fn(self.current_iteration/self.total_

        self.current_iteration += 1
```

In [15]:
```
def train_model(train_dataloader, val_dataloader, n_epochs = 15):

    model = SpeechNet(context_num).to(device)
    model.apply(weights_init)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)

    '''
    set scheduler for decaying learning rate
    '''
    parameter_scheduler = ParamScheduler(optimizer, scale_cos, n_epoch*len(train_
    candidate_model = 1
    print('Start training...')
    for i in range(n_epochs):

        t0 = time.time()

        avg_loss_1000_batch = 0
        val_correct = 0
        val_predicted = 0
        model.train()

        for index, (idx, features, labels) in enumerate(train_dataloader):

            optimizer.zero_grad()
            mask = [i for i in range(len(labels)) if labels[i] != torch.Tensor([
            features = features[mask].cuda()
            labels = labels[mask].cuda()
            '''
            forward and backward
            '''
            output = model(features)
            loss = criterion(output, labels.long())
            avg_loss_1000_batch += loss.item()
            loss.backward()

            parameter_scheduler.batch_step()
            optimizer.step()

            if index % 2000 == 0 and index != 0:

                predictions = torch.max(output.data, 1)[1]
                predicted = len(features)
                correct = int(sum(predictions == labels.to(device)).cpu())
                print("Epoch: {0}/{1} Train batch:{2}/{3}   acc: {4}  loss: {5}"
                                                                             n_
                                                                             in
                                                                             le
                                                                             co
                avg_loss_1000_batch = 0

        for index, (idx, val_features, val_labels) in enumerate(val_dataloader):

            mask = [i for i in range(len(val_labels)) if val_labels[i] != torch.
            val_features = val_features[mask].to(device)
            val_labels = val_labels[mask].to(device)
```

```
            model.eval()
            outputs = model(val_features)
            predictions = torch.max(outputs.data, 1)[1]
            val_predicted += len(val_features)
            val_correct += sum(predictions == val_labels.to(device))

        epoch_acc = int(val_correct.cpu())/val_predicted
        if epoch_acc >= 0.70:
            pickle.dump(model, open("candidate_model_{0}.pkl".format(candidate_mo
            print("Save one candidate model.")
            candidate_model += 1

        t1 = time.time()
        print("Validation Accuracy: {0}. Cost time: {1} minutes".format(int(val_c
        print("===================================================")


    return model
```

In [*]:
```
if __name__ == '__main__':

    train_features, train_labels = load_train_data()
    val_features, val_labels = load_validation_data()
    test_features = load_test_data()

    if pca_available:
        '''
        load local pca file
        '''
        pca = pickle.load(open('pca.pkl', 'rb'))
    else:
        '''
        10 features will be enough
        '''
        pca = PCA(10).fit(np.concatenate(train_features))
        pickle.dump(pca, open('pca_{0}_features.pkl'.format(pca.n_components), 'w

    train_features, train_labels = load_and_process_data(train_features, train_la
    val_features, val_labels = load_and_process_data(val_features, val_labels, pc

    train_context_dataset = ContextDataset(context_num, train_features, train_lab
    val_context_dataset = ContextDataset(context_num, val_features, val_labels)

    '''
    It may takes more than 20 minutes to process since it loops over all 15 milli
    But it could speed up later dataloader process
    And I could save this data by pickle
    The drawback is it's not flexible to feature engineering, like change context
    '''
    '''
    train_dataset = SpeechDataset(train_context_dataset)
    val_dataset = SpeechDataset(val_context_dataset)
    '''
    '''
    train_mask = (train_dataset.targets.numpy() != -1)*1
    train_sampler = WeightedRandomSampler(weights=train_mask, num_samples=int(tra
    val_mask = (val_dataset.targets.numpy() != -1)*1
    val_sampler = WeightedRandomSampler(weights=val_mask, num_samples=int(val_mas
    '''
    '''
    train_dataloader = DataLoader(train_dataset,
                                  shuffle = True,
                                  batch_size = 512,
                                  num_workers = 0,
                                  pin_memory = True)

    val_dataloader = DataLoader(val_dataset,
                                  shuffle = True,
                                  batch_size = 512,
                                  num_workers = 0,
                                  pin_memory = True)
    '''
    '''
    Start train model.
    15 epochs by default.
```

```
    Cost about 1 hour.
    '''

    train_dataloader = DataLoader(train_context_dataset,
                                  shuffle = True,
                                  batch_size = 512,
                                  num_workers = 0,
                                  pin_memory = True)

    val_dataloader = DataLoader(val_context_dataset,
                                shuffle = True,
                                batch_size = 512,
                                num_workers = 0,
                                pin_memory = True)

    model = train_model(train_dataloader, val_dataloader)

    pickle.dump(model, open("submission_model_68.pkl", "wb"))
    #make_submission(model, test_features)
```

```
Start loading training data...
Done loading training data in 0.7967264811197917 minutes...
Start loading validation data...
Done loading validation data in 0.04981781244277954 minutes...
Start loading test data...
Done loading test data in 0.01579108238220215 minutes...
Start training...
Epoch: 1/15 Train batch:2000/31779   acc: 0.42592592592592593  loss: 9.775534
469401464
Epoch: 1/15 Train batch:4000/31779   acc: 0.49269311064718163  loss: 8.109487
411100417
Epoch: 1/15 Train batch:6000/31779   acc: 0.5387840670859538  loss: 7.5979077
54126936
Epoch: 1/15 Train batch:8000/31779   acc: 0.5245901639344263  loss: 7.2745072
58553058
Epoch: 1/15 Train batch:10000/31779   acc: 0.5661157024793388  loss: 7.060203
752014786
Epoch: 1/15 Train batch:12000/31779   acc: 0.5605749486652978  loss: 6.872567
464131862
Epoch: 1/15 Train batch:14000/31779   acc: 0.556935817805383  loss: 6.7424005
856737494
Epoch: 1/15 Train batch:16000/31779   acc: 0.5091649694501018  loss: 6.622533
0552551895
Epoch: 1/15 Train batch:18000/31779   acc: 0.5583333333333333  loss: 6.509914
910653606
Epoch: 1/15 Train batch:20000/31779   acc: 0.5488565488565489  loss: 6.422984
649892896
Epoch: 1/15 Train batch:22000/31779   acc: 0.5854166666666667  loss: 6.343326
085712761
Epoch: 1/15 Train batch:24000/31779   acc: 0.5422680412371134  loss: 6.289215
44062905
Epoch: 1/15 Train batch:26000/31779   acc: 0.5877551020408164  loss: 6.231776
090338826
Epoch: 1/15 Train batch:28000/31779   acc: 0.6270491803278688  loss: 6.169563
454110175
Epoch: 1/15 Train batch:30000/31779   acc: 0.5487804878048781  loss: 6.127935
012802482
Validation Accuracy: 0.6123319858663936. Cost time: 29.898462855815886 minute
```

```
s
====================================================
Epoch: 2/15 Train batch:2000/31779   acc: 0.5463917525773195  loss: 6.0177660
41215509
Epoch: 2/15 Train batch:4000/31779   acc: 0.6326530612244898  loss: 5.9596305
86439744
Epoch: 2/15 Train batch:6000/31779   acc: 0.5859213250517599  loss: 5.9484914
7181958
Epoch: 2/15 Train batch:8000/31779   acc: 0.6008316008316008  loss: 5.9084361
93363741
Epoch: 2/15 Train batch:10000/31779   acc: 0.5873684210526315  loss: 5.887612
983118743
Epoch: 2/15 Train batch:12000/31779   acc: 0.6376518218623481  loss: 5.862514
947075397
Epoch: 2/15 Train batch:14000/31779   acc: 0.6153846153846154  loss: 5.820339
681347832
Epoch: 2/15 Train batch:16000/31779   acc: 0.5941422594142259  loss: 5.798072
6168025285
Epoch: 2/15 Train batch:18000/31779   acc: 0.6137787056367432  loss: 5.779028
256423771
Epoch: 2/15 Train batch:20000/31779   acc: 0.6205450733752621  loss: 5.765704
936115071
Epoch: 2/15 Train batch:22000/31779   acc: 0.6296296296296297  loss: 5.736493
79843846
Epoch: 2/15 Train batch:24000/31779   acc: 0.6112266112266113  loss: 5.712157
631292939
Epoch: 2/15 Train batch:26000/31779   acc: 0.6131687242798354  loss: 5.691869
272151962
Epoch: 2/15 Train batch:28000/31779   acc: 0.5904365904365905  loss: 5.677539
088530466
Epoch: 2/15 Train batch:30000/31779   acc: 0.629399585921325  loss: 5.6451472
66564891
Validation Accuracy: 0.6391254683088796. Cost time: 23.292837572097778 minute
s
====================================================
Epoch: 3/15 Train batch:2000/31779   acc: 0.6145833333333334  loss: 5.5592634
84319672
Epoch: 3/15 Train batch:4000/31779   acc: 0.6155419222903885  loss: 5.5491820
180322975
Epoch: 3/15 Train batch:6000/31779   acc: 0.6114519427402862  loss: 5.5398837
50258014
Epoch: 3/15 Train batch:8000/31779   acc: 0.6354166666666666  loss: 5.5209490
56139216
Epoch: 3/15 Train batch:10000/31779   acc: 0.6088709677419355  loss: 5.514790
925895795
Epoch: 3/15 Train batch:12000/31779   acc: 0.6526315789473685  loss: 5.489247
277611867
Epoch: 3/15 Train batch:14000/31779   acc: 0.5950413223140496  loss: 5.482605
571858585
Epoch: 3/15 Train batch:16000/31779   acc: 0.6350515463917525  loss: 5.471510
59191674
Epoch: 3/15 Train batch:18000/31779   acc: 0.6086956521739131  loss: 5.449926
839210093
Epoch: 3/15 Train batch:20000/31779   acc: 0.5955284552845529  loss: 5.452208
753209561
Epoch: 3/15 Train batch:22000/31779   acc: 0.6464435146443515  loss: 5.437912
923749536
```

```
Epoch: 3/15 Train batch:24000/31779   acc: 0.6638830897703549  loss: 5.425580
8622110635
Epoch: 3/15 Train batch:26000/31779   acc: 0.6523517382413088  loss: 5.404229
0057986975
Epoch: 3/15 Train batch:28000/31779   acc: 0.6028513238289206  loss: 5.386188
2362980396
Epoch: 3/15 Train batch:30000/31779   acc: 0.6157112526539278  loss: 5.392457
937821746
Validation Accuracy: 0.652437869542315. Cost time: 24.991123350461326 minutes
===================================================
Epoch: 4/15 Train batch:2000/31779   acc: 0.6962809917355371  loss: 5.3062039
02699053
Epoch: 4/15 Train batch:4000/31779   acc: 0.6473029045643154  loss: 5.3037068
39447841
Epoch: 4/15 Train batch:6000/31779   acc: 0.6465696465696466  loss: 5.2949280
78074008
Epoch: 4/15 Train batch:8000/31779   acc: 0.6409185803757829  loss: 5.2836531
86634183
Epoch: 4/15 Train batch:10000/31779   acc: 0.6659751037344398  loss: 5.281832
566251978
Epoch: 4/15 Train batch:12000/31779   acc: 0.6452282157676349  loss: 5.256409
40410085
Epoch: 4/15 Train batch:14000/31779   acc: 0.6680497925311203  loss: 5.251310
472376645
Epoch: 4/15 Train batch:16000/31779   acc: 0.6476578411405295  loss: 5.252602
803520858
Epoch: 4/15 Train batch:18000/31779   acc: 0.6273684210526316  loss: 5.243135
431781411
Epoch: 4/15 Train batch:20000/31779   acc: 0.6751054852320675  loss: 5.246580
079197884
Epoch: 4/15 Train batch:22000/31779   acc: 0.6520833333333333  loss: 5.220654
300646856
Epoch: 4/15 Train batch:24000/31779   acc: 0.6556016597510373  loss: 5.222006
601979956
Epoch: 4/15 Train batch:26000/31779   acc: 0.6468172484599589  loss: 5.219035
235699266
Epoch: 4/15 Train batch:28000/31779   acc: 0.620253164556962  loss: 5.1856239
55447227
Epoch: 4/15 Train batch:30000/31779   acc: 0.6836734693877551  loss: 5.192454
103147611
Validation Accuracy: 0.6618336401138738. Cost time: 32.523609634240465 minute
s
===================================================
Epoch: 5/15 Train batch:2000/31779   acc: 0.6625258799171843  loss: 5.1132215
19626677
Epoch: 5/15 Train batch:4000/31779   acc: 0.6756198347107438  loss: 5.1009923
00081998
Epoch: 5/15 Train batch:6000/31779   acc: 0.6551020408163265  loss: 5.1157691
72552973
Epoch: 5/15 Train batch:8000/31779   acc: 0.694672131147541  loss: 5.09827569
5927441
Epoch: 5/15 Train batch:10000/31779   acc: 0.6826722338204593  loss: 5.089762
9777900875
Epoch: 5/15 Train batch:12000/31779   acc: 0.6700819672131147  loss: 5.092070
585116744
Epoch: 5/15 Train batch:14000/31779   acc: 0.6412371134020619  loss: 5.075249
331071973
```

```
Epoch: 5/15 Train batch:16000/31779   acc: 0.6638655462184874  loss: 5.076793
9439974725
Epoch: 5/15 Train batch:18000/31779   acc: 0.6371134020618556  loss: 5.072140
499367379
Epoch: 5/15 Train batch:20000/31779   acc: 0.663135593220339  loss: 5.0696268
0327706
Epoch: 5/15 Train batch:22000/31779   acc: 0.6784968684759917  loss: 5.057284
829672426
Epoch: 5/15 Train batch:24000/31779   acc: 0.7027027027027027  loss: 5.051002
2402741015
Epoch: 5/15 Train batch:26000/31779   acc: 0.6375266524520256  loss: 5.050701
953005046
Epoch: 5/15 Train batch:28000/31779   acc: 0.6659707724425887  loss: 5.041448
8753303885
Epoch: 5/15 Train batch:30000/31779   acc: 0.6155419222903885  loss: 5.019512
769533321
Validation Accuracy: 0.6674311519362686. Cost time: 29.610111657778422 minute
s
===================================================
Epoch: 6/15 Train batch:2000/31779   acc: 0.6673511293634496  loss: 4.9649889
0966177
Epoch: 6/15 Train batch:4000/31779   acc: 0.6199186991869918  loss: 4.9560458
56233686
Epoch: 6/15 Train batch:6000/31779   acc: 0.6503067484662577  loss: 4.9370094
67743337
Epoch: 6/15 Train batch:8000/31779   acc: 0.6776180698151951  loss: 4.9488178
29135805
Epoch: 6/15 Train batch:10000/31779   acc: 0.6434426229508197  loss: 4.938042
079564184
Epoch: 6/15 Train batch:12000/31779   acc: 0.6474226804123712  loss: 4.938233
001390472
Epoch: 6/15 Train batch:14000/31779   acc: 0.6750524109014675  loss: 4.937296
177376993
Epoch: 6/15 Train batch:16000/31779   acc: 0.6406570841889117  loss: 4.921966
823982075

Epoch: 6/15 Train batch:18000/31779   acc: 0.6873706004140787  loss: 4.91727610
3980839
Epoch: 6/15 Train batch:20000/31779   acc: 0.6811594202898551  loss: 4.91769523
2201368
Epoch: 6/15 Train batch:22000/31779   acc: 0.6680327868852459  loss: 4.90579132
8288615
Epoch: 6/15 Train batch:24000/31779   acc: 0.6687116564417178  loss: 4.89853619
8306829
Epoch: 6/15 Train batch:26000/31779   acc: 0.6761710794297352  loss: 4.89721436
8451387
Epoch: 6/15 Train batch:28000/31779   acc: 0.6673596673596673  loss: 4.89238152
3037329
Epoch: 6/15 Train batch:30000/31779   acc: 0.676954732510288  loss: 4.896091582
486406
Validation Accuracy: 0.6729739167490338. Cost time: 24.34170482158661 minutes
===================================================
Epoch: 7/15 Train batch:2000/31779   acc: 0.6639344262295082  loss: 4.816347099
957056
Epoch: 7/15 Train batch:4000/31779   acc: 0.6604938271604939  loss: 4.815884839
161299
Epoch: 7/15 Train batch:6000/31779   acc: 0.6728778467908902  loss: 4.811424470
```

```
8558545
Epoch: 7/15 Train batch:8000/31779   acc: 0.6521739130434783  loss: 4.817386562
121101
Epoch: 7/15 Train batch:10000/31779   acc: 0.6447638603696099  loss: 4.80826326
9718736
Epoch: 7/15 Train batch:12000/31779   acc: 0.691683569979716  loss: 4.808936178
451404
Epoch: 7/15 Train batch:14000/31779   acc: 0.6687370600414079  loss: 4.80394072
5745633
Epoch: 7/15 Train batch:16000/31779   acc: 0.6413934426229508  loss: 4.79524684
0920299
Epoch: 7/15 Train batch:18000/31779   acc: 0.6556701030927835  loss: 4.79816957
7028602
```

In [ ]: