In [1]:

```python
import sys, os, re, json, time

import pandas as pd
import pickle
import h5py

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import plotting
from PIL import Image
from tqdm import tqdm
from utils import imread, img_data_2_mini_batch, imgs2batch

from sklearn import metrics
from sklearn.metrics import accuracy_score

# from naive import EncDec
from fusion_rnn import EncDec
# from attention import EncDec as FuseAttEncDec
# from rnn_att import EncDec
from data_loader import VQADataSet

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data as Data
from torchvision import transforms

%matplotlib inline
%reload_ext autoreload
%autoreload 2
```

In [2]:

```python
N = 5000
dataset_filename = "./data/data_{}.pkl".format(N)
dataset = None
print(dataset_filename)
if (os.path.exists(dataset_filename)):
    with open(dataset_filename, 'rb') as handle:
        print("reading from " + dataset_filename)
        dataset = pickle.load(handle)
else:
    dataset = VQADataSet(Q=N)
    with open(dataset_filename, 'wb') as handle:
        print("writing to " + dataset_filename)
        pickle.dump(dataset, handle)

assert(dataset is not None)
def debug(v,q,a):
    print('\nV: {}\nQ: {}\nA: {}'.format(v.shape, q.shape, a.shape)

```

```
./data/data_5000.pkl
reading from ./data/data_5000.pkl
```

In [7]:

```
embed_size         = 128
hidden_size        = 128
batch_size         = 50
ques_vocab_size    = len(dataset.vocab['question'])
ans_vocab_size     = len(dataset.vocab['answer'])
num_layers         = 1
n_epochs           = 15
learning_rate      = 0.001
momentum           = 0.98
attention_size     = 512
debug              = False
rnn_type           = 'lstm'

print(ques_vocab_size, ans_vocab_size)
```

```
2386 1022
```

In [8]:

```
def eval_model(data_loader, model, criterion, optimizer, batch_size
              epoch = 0, total_loss_over_epochs=[], scores_over_epo
    running_loss = 0.
    final_labels, final_preds = [], []
    scores, losses = [], []
    if data_loader is None:
        return

    run_type = None
    if training:
        run_type = 'train'
        model.train()
    else:
        run_type = 'test'
        model.eval()

    for i, minibatch in enumerate(data_loader):
        # extract minibatch
        t0 = time.time()
        idxs, v, q, a, q_len = minibatch

        # convert torch's DataLoader output to proper format.
        # torch gives a List[Tensor_1, ... ] where tensor has been 
        # batchify transposes back.`
        v = v.to(device)
        q = VQADataSet.batchify_questions(q).to(device)
        a = a.to(device)

        logits = model(v, q, q_len)
        preds = torch.argmax(logits, dim=1)

#         loss = criterion(logits, a)
        loss = F.nll_loss(logits, a)
        running_loss += loss.item()

#         score = metrics.precision_recall_fscore_support(preds.tol
#                                                          a.tolist(
#                                                          average='
        score = metrics.accuracy_score(preds.tolist(),a.tolist())

        scores.append(score)
        losses.append(loss)

        loss_key = '{}_loss'.format(run_type)
        total_loss_over_epochs['{}_loss'.format(run_type)].append(l
        scores_over_epochs['{}_scores'.format(run_type)].append(sco

        if training and optimizer is not None:
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        final_labels += a.tolist()
        final_preds  += preds.tolist()
        if i%10==0:
            score = np.mean(scores)
            print("Epoch {}: {} Loss: {} Score: {} t: {}".format(epo
#             plotting.plot_score_over_n_epochs(scores_over_epochs, 
#             plotting.plot_loss_over_n_epochs(total_loss_over_epoc

    return running_loss, final_labels, final_preds
```

In [9]:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "c
# model = EncDec(embed_size, hidden_size, ques_vocab_size, ans_voca

model = EncDec(embed_size,
               hidden_size,
               ques_vocab_size,
               ans_vocab_size,
               num_layers,
               rnn_type=rnn_type,
               prefix_n=1).to(device)

criterion = nn.CrossEntropyLoss()
# optimizer = torch.optim.SGD(model.get_parameters(), lr=learning_r
optimizer = torch.optim.Adam(model.get_parameters(), lr=learning_ra
# optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate

train_loader = dataset.build_data_loader(train=True, args={'batch_s
test_loader  = dataset.build_data_loader(test=True, args={'batch_si

best_score = 0

train_all_loss, train_all_labels, train_all_preds = [], [], []
print("model built, start training.")
total_loss_over_epochs, scores_over_epochs = plotting.get_empty_stat
total_loss_over_epochs2, scores_over_epochs2 = plotting.get_empty_st
for epoch in tqdm(range(n_epochs)):
    t0= time.time()
    tr_loss, tr_labels, tr_preds = eval_model(data_loader = train_l
                                     model       = model,
                                     criterion   = criterion,
                                     optimizer   = optimizer,
                                     batch_size  = batch_size,
                                     training    = True,
                                     epoch       = epoch,
                                     total_loss_over_epochs = total_
                                     scores_over_epochs     = score

    tr_loss, ts_labels, ts_preds = eval_model(data_loader = test_lo
                                     model       = model,
                                     criterion   = criterion,
                                     optimizer   = None,
                                     batch_size  = batch_size,
                                     training    = False,
                                     epoch       = epoch,
                                     total_loss_over_epochs = total_
                                     scores_over_epochs     = score


    score = metrics.accuracy_score(ts_preds,ts_labels)
#     total_loss_over_epochs['train_loss'].append(tr_loss)
#     scores_over_epochs['train_scores'].append(train_scores)

#     if True:# or epoch%1 == 0:
    print("\n"+"#==#"*7 + "epoch: {}".format(epoch) + "#==#"*7)
    print('TEST ACC: {}'.format(score))
    print("#==#"*7 + "time: {}".format(time.time()-t0) + "#==#"*7 +
#         print(train_scores)
#     plotting.plot_score_over_n_epochs(scores_over_epochs, score_t
#     plotting.plot_loss_over_n_epochs(total_loss_over_epochs, fig_




```

```
  0%|          | 0/10 [00:00<?, ?it/s]

batch_size: 50 shuffle: True
batch_size: 50 shuffle: False
model built, start training.
Epoch 0: train Loss: 6.927465438842773 Score: 0.0 t: 0.56087565422058
1
Epoch 0: train Loss: 6.334043025970459 Score: 0.11272727272727273 t:
0.3291900157928467
Epoch 0: train Loss: 5.424932956695557 Score: 0.140952380952381 t: 0.
31887245178222656
Epoch 0: train Loss: 4.131680965423584 Score: 0.15290322580645163 t:
0.33866071701049805
Epoch 0: train Loss: 4.719060897827148 Score: 0.15707317073170732 t:
0.3201107978820801
Epoch 0: train Loss: 4.946177959442139 Score: 0.16705882352941176 t:
0.33951234817504883
Epoch 0: train Loss: 5.139514923095703 Score: 0.1649180327868852 t:
0.3358619213104248
Epoch 0: train Loss: 4.782313346862793 Score: 0.16816901408450702 t:
```

In [ ]:

```
### Error Analysis
```

In [35]:

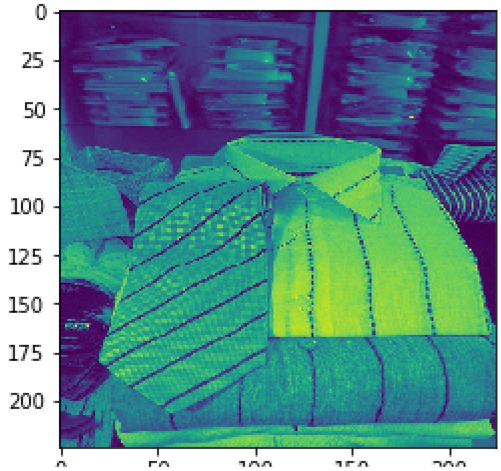
```
# import matplotlib
# import matplotlib.pyplot as plt
# %matplotlib inline
# count = 1
# err_anal_data = []
# for i, minibatch in enumerate(test_loader):
#     # extract minibatch
#     t0 = time.time()
#     idxs, v, q, a, q_len = minibatch

#     v = v.to(device)
#     q = VQADataSet.batchify_questions(q).to(device)
#     a = a.to(device)

#     logits = model(v,q,q_len)
#     preds = torch.argmax(logits, dim=1)

#     for i in range(len(a)):
#         idx = idxs[i]
#         enc_ans = a[i].item()
#         enc_ques = q[i].detach().cpu().numpy()
#         img_v = v[i].detach().cpu().numpy()
#         question = dataset.decode_question(enc_ques)
#         answer_dec = dataset.decode_answer(preds[i])
#         answer = dataset.decode_answer(enc_ans)
# #         img_v = img_v.reshape(224, 224, 3)
#         plt.figure()
#         plt.imshow(img_v[0,:,:], interpolation='nearest')
#         plt.show()
#         question = question.replace("<pad>", "")
#         question = question.replace("<start>", "")
#         question = question.replace("<end>", "").strip()
#         result = answer_dec==answer
#         err_anal_data.append([question, answer_dec, answer])
#         if not result:
#             print("{}. [Q] {} [A] {} [PRED] {}".format(count, que
#             count+=1
# #         print(err_anal_data[-1])
# #         print('question:',  question)
# #         print("[{}] - predicted: {} - ground-truth: {}".format(

#     torch.argmax(a)
```



In [ ]:

```


# for epoch in range(1):
#     ts_loss, ts_labels, ts_preds = eval_model(data_loader = test_
#                                               model       = model,
#                                               criterion   = criterion,
#                                               optimizer   = None,
#                                               batch_size  = batch_size,
#                                               training    = False,
#                                               epoch       = epoch,
#                                               total_loss_over_epochs = tot
#                                               scores_over_epochs     = sco
#     score = metrics.accuracy_score(ts_preds,ts_labels)
#     print("ACC: " + str(score))
```

In [ ]:

```
print(tr_labels[0])
print(tr_preds[0])
```

In [ ]:

```

```