Lab Assignment 4
CSCI 5922, Spring 2022

Varun Manjunath

03/31/2022

1 Code

1.1 Question 1

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from collections import Counter

# Resize the image
from skimage import io
import matplotlib.pyplot as plt
from skimage.transform import resize
%matplotlib inline

def resize_image(image_url):
    image = io.imread(image_url)
    image = resize(image, (448,448))
    return image

# create json file for vocabulary
import json
import os
import nltk
from nltk.stem.snowball import *
from tqdm import *
from collections import Counter, OrderedDict
import string


annotation_dir = "../input/vizwizvarun/Annotations/Annotations/"


# Build the question vocabulary
q_counter = Counter()
n_sample = 0
```

```
maxlen = 0
train_data = json.load(open(annotation_dir + 'train.json'))
for one_data in tqdm(train_data):
    n_sample += 1
    question = one_data['question']
    question = question.lower()
    tokens = nltk.word_tokenize(question)
    token_len = len(tokens)
    maxlen = max([maxlen,token_len])
    q_counter.update(tokens)
print('number of sample = ' + str(n_sample))
print('max len = ' + str(maxlen))
q_word_counts = [x for x in q_counter.items()]
q_word_counts.sort(key=lambda x: x[1], reverse=True)
json.dump(q_word_counts, open('q_word_counts.json', "w"), indent=2)

### build vocabulary based on question
vocab = [x[0] for x in q_word_counts if x[1] >= 0]
unk_word = '<UNK>'
vocab = [unk_word] + vocab
vocab = OrderedDict(zip(vocab,range(len(vocab))))
json.dump(vocab, open('word2vocab_id.json', 'w'), indent=2)

## Build the answer counter that has counts of all answers across the training set.
ans_counter = Counter()

train_data = json.load(open(annotation_dir + 'train.json'))
for annotation in tqdm(train_data):
    for answer in annotation['answers']:
        answer = answer['answer'].lower()
        ans_counter.update([answer]) # don't forget the [], counter.update input a list
ans_counts = [x for x in ans_counter.items()]
ans_counts.sort(key=lambda x: x[1], reverse=True)
json.dump(ans_counts, open('ans_counts.json', "w"), indent=2)

# build the answer vocabulary
output_num = 3000
n_totoal = sum([x[1] for x in ans_counts])
ans_counts = ans_counts[:output_num]
n_cover = sum([x[1] for x in ans_counts])
print("we keep top %d most answers"%len(ans_counts))
print("coverage: %d/%d (%.4f)"%(n_cover, n_totoal, 1.0 * n_cover / n_totoal))
ans_list = [x[0] for x in ans_counts]
ans_list.append('<unk>')
ans_dict = OrderedDict(zip(ans_list,range(len(ans_list))))
json.dump(ans_dict, open('answer2answer_id.json', 'w'), indent=2)

#Get the question vocab size
qvocabsize=0
data=json.load(open("./q_word_counts.json"))
for i in data:
```

```
    qvocabsize+=1

#Resize the images to 448*448 and save the training images
from PIL import Image
from tqdm import *
import numpy as np
from skimage.io import imsave, imread
import os
import json
import sys
image_size = 448
main_dir = "../input/vizwizvarun"
train_dataset = main_dir + '/train/train'
output="images_trainedresized"
images_name = os.listdir(train_dataset)
images_name.sort()
print(len(images_name))
if not os.path.exists(output):
    os.makedirs(output)
for idx, img_info in enumerate(tqdm(images_name)):

    img_path = train_dataset+"/"+img_info

    image = resize_image(img_path)
    imsave(os.path.join(output, img_info),image)

#Resize the images to 448*448 and save the validation images
from tqdm import *
import numpy as np
import os
import json
from PIL import Image
import sys
from skimage.io import imsave, imread
image_size = 448
main_dir = "../input/vizwizvarun"
output="images_resized_val"
dataset_dir = main_dir + '/val/val'
images_name = os.listdir(dataset_dir)
images_name.sort()
print(len(images_name))
if not os.path.exists(output):
    os.makedirs(output)
for idx, img_info in enumerate(tqdm(images_name)):
    img_path = dataset_dir+"/"+img_info
    image = resize_image(img_path)
    imsave(os.path.join(output, img_info),image)


#Resize the images to 448*448 and save the test images
from tqdm import *
```

```
import numpy as np
import os
import json
from PIL import Image
import sys
from skimage.io import imsave, imread
image_size = 448
main_dir = "../input/vizwizvarun"
output="images_testresized"
dataset_dir = main_dir + '/test/test'
all_imgs = os.listdir(dataset_dir)
all_imgs.sort()
print(len(all_imgs))
if not os.path.exists(output):
    os.makedirs(output)
for idx, img_info in enumerate(tqdm(all_imgs[0:1000])):
    img_path = dataset_dir+"/"+img_info
    image = resize_image(img_path)
    imsave(os.path.join(output, img_info),image)


import json
import numpy as np
import nltk
import os
from tqdm import *
from collections import Counter
import re
output_dir="data"

def convert_to_tokens(sentence):
    regex = re.compile(r'(\W+)')
    tokens = regex.split(sentence.lower())
    tokens = [w.strip() for w in tokens if len(w.strip()) > 0]
    return tokens

root_dir = "../input/vizwizvarun/Annotations/Annotations"
#splits = ['train', 'val', 'test']

dirs="../input/vizwizvarun/"
vocab = json.load(open('./word2vocab_id.json'))
answer2answer_id = json.load(open('./answer2answer_id.json'))

#Converts a tokenized question sentence
#into a sequence of numbers based what number each token represents
def sentence_to_vocabid(sentence, vocab):
    unk_word = '<UNK>'
    tokens = nltk.word_tokenize(sentence.lower())
    tokens_id = [vocab.get(x, vocab[unk_word]) for x in tokens]
    return tokens_id
```

```
#the file that preprocesses the input data used to train the mode.
def preprocessingtrain(type1):
        missing_images = []
    ## load annotation file and question file
        dataset = json.load(open(root_dir+'/'+type1+'.json'))
        data = [None]*len(dataset)

        for idx,one_data in enumerate(dataset):
            ans_counter = Counter([x['answer'] for x in one_data['answers']])
            allanswers = [x['answer'] for x in one_data['answers']]
            ans = ans_counter.most_common(1)[0][0]
            a_label = answer2answer_id.get(ans,3000)
            answers=[]
#             if type1 == 'train' and a_label == -1:
#                 continue
            for answer in allanswers :
                answers.append(answer2answer_id.get(answer,3000))
            imagename = one_data['image']
            imagepath = "../input/trainresized/images_trainedresized" "/"+ imagename
            questionsentence=one_data['question']
            questiontokens = nltk.word_tokenize(question)
            questionencoding = sentence_to_vocabid(one_data['question'], vocab)
            if(len(questionencoding)>=37):
                questionencoding=questionencoding[0:37]
            else:
                questionencoding.extend([0]*(37-len(questionencoding)))
            image_info = {'imagename': imagename,
                'imagepath': imagepath,
                'questionsentence': questionsentence,
                'questiontokens': questiontokens
                }
            image_info['questionencoding'] = questionencoding
            image_info['selectedanswer'] = ans
            image_info['answerlabel']=a_label
            image_info['answers']=answers
            data[idx]=image_info
        return data

#The file that preprocesses the validation data used to train the model.
def preprocessingvalidation(type1):
        missing_images = []


    ## load annotation file and question file
        dataset = json.load(open(root_dir+'/'+type1+'.json'))
        data = [None]*len(dataset)

        for idx,one_data in enumerate(dataset):
            ans_counter = Counter([x['answer'] for x in one_data['answers']])
            allanswers = [x['answer'] for x in one_data['answers']]
            ans = ans_counter.most_common(1)[0][0]
```

```
            a_label = answer2answer_id.get(ans,3000)
            answers=[]
#             if type1 == 'val' and a_label == -1:
#                 continue
            for answer in allanswers :
                answers.append(answer2answer_id.get(answer,3000))


            imagename = one_data['image']
            imagepath = "../input/validationresized/images_resized_val" "/"+ imagename
            questionsentence=one_data['question']
            questiontokens = nltk.word_tokenize(question)
            questionencoding = sentence_to_vocabid(one_data['question'], vocab)
            if(len(questionencoding)>=37):
                questionencoding=questionencoding[0:37]
            else:
                questionencoding.extend([0]*(37-len(questionencoding)))

            image_info = {'imagename': imagename,
                'imagepath': imagepath,
                'questionsentence': questionsentence,
                'questiontokens': questiontokens
                        }

            image_info['questionencoding'] = questionencoding
            image_info['selectedanswer'] = ans
            image_info['answerlabel']=a_label
            image_info['answers']=answers
            data[idx]=image_info
        return data

#Sample only a subset of the data
def sampledata(data,num):
    import random
    random.shuffle(data)
    return data[0:num]

#Stores the preprocessed data(train and val)
#in a preprocessed dictionary.
#Taking a sample of 10000 train and 1000 test sets
#to check which of the three models trained below has the highest accuracy.
preprecessed={}
preprecessed['train']=sampledata(preprocessingtrain('train'),10000)
preprecessed['val']=sampledata(preprocessingvalidation('val'),1000)
preprecessed['train-val'] = preprecessed['train'] + preprecessed['val']

#Make the data directory to store the data
os.makedirs("data")

#Stores the train,val,train-val data into files labelled train,val and train-val
for key, value in preprecessed.items():
```

```
        np.save(os.path.join("./data", f'{key}.npy'), np.array(value))


import numpy as np
import os
from torch.utils.data import Dataset
from torchvision import transforms
from torch.utils.data import DataLoader
from PIL import Image

INPUT_DIR = './data'

#The VizWiz dataset  class that is used by the data_loader function
#to load the data.This function is used by the
#data_loader function to get the training and validation data.
#Also used to obtain the test data.
class VizWizdataset(Dataset):

    def __init__(self, input_dir, file, transformimage = None):

        self.input_data = np.load(os.path.join(input_dir, file), allow_pickle=True)
        self.nottest = True if not "test" in file else False
        self.transformimage = transformimage

    def __getitem__(self, idx):
        if(self.input_data[idx] is None):
            path = self.input_data[1]['imagepath']
            img = np.array(Image.open(path))
            questiontokens = self.input_data[1]['questiontokens']
            questiontoid=self.input_data[1]['questionencoding']
            if(len(questiontoid)>=37):
                questiontoid=questiontoid[0:37]
            else:
                questiontoid=questiontoid.extend([0]*(37-len(questiontoid)))
            questiontoid=(np.asarray(self.input_data[1]['questionencoding']))

            dataentry = {'image': img, 'question': questiontoid}
            if(self.nottest):
                dataentry['answers']=np.array(self.input_data[1]['answers'])
            if self.transformimage:
                dataentry['image'] = self.transformimage(dataentry['image'])
                dataentry['imagepath']=self.input_data[1]['imagepath']
            if self.nottest:
                answertoid = self.input_data[1]['answerlabel']
                dataentry['answer'] = answertoid
            return dataentry
        else:
            path = self.input_data[idx]['imagepath']
            #print(path)
            img = np.array(Image.open(path))
            questiontokens = self.input_data[idx]['questiontokens']
```

```
            questiontoid=self.input_data[idx]['questionencoding']
            if(len(questiontoid)>=37):
                questiontoid=questiontoid[0:37]
            else:
                questiontoid=questiontoid.extend([0]*(37-len(questiontoid)))
            questiontoid=(np.asarray(self.input_data[idx]['questionencoding']))
            dataentry = {'image': img, 'question': questiontoid}
            if(self.nottest):
                dataentry['answers']=np.array(self.input_data[idx]['answers'])
            if self.nottest:
                answertoid = self.input_data[idx]['answerlabel']
                dataentry['answer'] = answertoid

            if self.transformimage:
                dataentry['image'] = self.transformimage(dataentry['image'])
                dataentry['imagepath']=self.input_data[idx]['imagepath']

            return dataentry

    def __len__(self):

        return len(self.input_data)

#The data_loader function is used to
#obtain the training and validation datasets
#that is used by the model for training and validation respectively.
def data_loader(input_dir, batch_size, num_worker):
    transformimage = transforms.Compose([
        transforms.ToTensor(),  # convert to a tensor
        transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225],
    )                           # normalize the image so pizel values lie between 0 and 1
    ])

    vizwiz = {
        'train': VizWizdataset(
            input_dir=input_dir,
            file='train.npy',
            transformimage=transformimage),
        'val': VizWizdataset(
            input_dir=input_dir,
            file='val.npy',
            transformimage=transformimage)
    }

    dataloader = {
        'train': DataLoader(
            dataset=vizwiz['train'],
            batch_size=batch_size,
            shuffle=True,
```

```
            num_workers=num_worker
            ),
        'val': DataLoader(
            dataset=vizwiz['val'],
            batch_size=batch_size,
            shuffle=True,
            num_workers=num_worker
            )

    }

    return dataloader


#Model 1: Resnet50 for extracting features from images and a bidirectional LSTM
#to extract encoding from the sentences.
import torch
import torch.nn as nn
import torchvision.models as models
import torch.nn.functional as F


class ImgPreprocess(nn.Module):

    def __init__(self, embed_dim):
        #Loading Resnet50 pretrained model
        super(ImgPreprocess, self).__init__()
        self.model = models.resnet50(pretrained=True)
        in_features = self.model.fc.in_features
        self.model.fc =  nn.Linear(in_features, embed_dim)# remove resnet50 last layer


    def forward(self, image):
        with torch.no_grad():
            imagefeatures = self.model(image) # (batch, channel, height, width)

        l2_norm = F.normalize(imagefeatures, p=2, dim=1).detach()
        return l2_norm

class QuestionPreprocess(nn.Module):
    #Using a single
    def __init__(self, questionvocabularysize, wordembeddingsize,
    hidden_size, num_hidden, sizeofconcatenatedlayer):
                super(QuestionPreprocess, self).__init__()
                self.word_embedding = nn.Embedding(questionvocabularysize, wordembeddingsize)
                self.tanh = nn.Tanh()
                self.hidden_size=hidden_size
                self.num_hidden=num_hidden
                self.lstm=nn.LSTM(wordembeddingsize,hidden_size,
                num_hidden,batch_first=True,bidirectional=True)
                self.fc=nn.Linear(hidden_size*2,sizeofconcatenatedlayer)
```

```
    def forward(self, question):
                qu_embedding = self.word_embedding(question)
                qu_embedding = self.tanh(qu_embedding)
                h0=torch.zeros(self.num_hidden*2,qu_embedding.size(0),self.hidden_size).to('cuda')
                c0=torch.zeros(self.num_hidden*2,qu_embedding.size(0),self.hidden_size).to('cuda')
                output,_=self.lstm(qu_embedding,(h0,c0))
                output=self.fc(output[:,-1,:])
                return output
class VizWizModel(nn.Module):

    def __init__(self, sizeoffeatures, questionvocabularysize,
    answervocabularysize, wordembeddingsize,
    hidden_size, num_hidden):

        super(VizWizModel, self).__init__()
        self.questionpreprocess =
        QuestionPreprocess(questionvocabularysize,
        wordembeddingsize, hidden_size, num_hidden, sizeoffeatures)
        self.imgpreprocess = ImgPreprocess(sizeoffeatures)
        self.tanh = nn.Tanh()
        self.dropout = nn.Dropout(0.5)
        self.layer1 = nn.Linear(sizeoffeatures, answervocabularysize)
        self.layer2 = nn.Linear(answervocabularysize, answervocabularysize)

    def forward(self, image, question):

        imagefeatures = self.imgpreprocess(image)
        questionfeatures = self.questionpreprocess(question)
        dotproductfeatures = imagefeatures * questionfeatures
        dotproductfeatures = self.dropout(dotproductfeatures)
        dotproductfeatures = self.tanh(dotproductfeatures)
        dotproductfeatures = self.layer1(dotproductfeatures)
        dotproductfeatures = self.dropout(dotproductfeatures)
        dotproductfeatures = self.tanh(dotproductfeatures)
        logits = self.layer2(dotproductfeatures)

        return logits

#Training Model 1 for 10 epochs with a batch size of 100.
import os
import time
import torch
import torch.nn as nn

from torch import optim


train_accuracy=[]
valid_accuracy=[]
epochs_total=[]
datadir = './data'
```

```
model_dir = './ckpt'
logs = './log'

workers = 1
batchsize = 100
sizeoffeatures, embeddingsize = 1024, 300
numofhidden, sizeofhiddenlayer = 3, 512

epochs = 10

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#device='cuda'
print(device)
def train():
    dataloader = data_loader(input_dir=datadir, batch_size=batchsize, num_worker=workers)
    questionvocabularysize = qvocabsize+1
    answervocabularysize = 3001

    model = VizWizModel(sizeoffeatures=sizeoffeatures,
    questionvocabularysize=questionvocabularysize,
    answervocabularysize=answervocabularysize,
    wordembeddingsize=embeddingsize,
    hidden_size=sizeofhiddenlayer,
    num_hidden=numofhidden).to(device)


    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)


    print('>> start training')
    start_time = time.time()
    for epoch in range(epochs):
        metrics=[]
        metrics1=[]
        correct1=0
        length1=0
        correct=0
        length=0
        epoch_loss = {key: 0 for key in ['train', 'val']}
        print(epoch)
        model.train()
        for idx, dataentry in enumerate(dataloader['train']):

            image = dataentry['image'].to(device)
            question = dataentry['question'].to(device)
            label = dataentry['answer'].to(device)
            answers=dataentry['answers']
            label1=label.tolist()
```

```
    # forwardpropogation
    logits = model(image, question)

    with torch.no_grad():
        predict = torch.log_softmax(logits, dim=1)
        predict = torch.argmax(predict, dim=1).tolist()
    print("Train set prediction {}".format(Counter(predict)))
    for i in range(len(label1)):
        if(predict[i]==label1[i]):
            correct+=1
    length=length+batchsize
    for i in range(answers.shape[0]):
        count=0
        for j in range(answers.shape[1]):
            if(answers[i][j]==predict[i]):
                count+=1
        metrics.append(min(count//3,1))
    loss = criterion(logits, label)
    epoch_loss['train'] += loss.item()
    # backpropogation
    optimizer.zero_grad()
    loss.backward()
    print(loss.item())
    optimizer.step()
model.eval()
correct = correct/length
print("Train set Accuracy metric is {}".format(np.mean(metrics)))
train_accuracy.append(np.mean(metrics))

for idx, dataentry in enumerate(dataloader['val']):
    image = dataentry['image'].to(device=device)
    question = dataentry['question'].to(device=device)
    label = dataentry['answer'].to(device=device)
    answers=dataentry['answers']
    label1=label
    with torch.no_grad():
        logits = model(image, question)
        loss = criterion(logits, label)
    epoch_loss['val'] += loss.item()
    with torch.no_grad():
        predict = torch.log_softmax(logits, dim=1)
        predict = torch.argmax(predict, dim=1).tolist()
    print("Valid set prediction {}".format(Counter(predict)))
    for i in range(len(predict)):
        if(predict[i]==label1[i]):
            correct1+=1
    length1=length1+batchsize
    for i in range(answers.shape[0]):
        count=0
        for j in range(answers.shape[1]):
            if(answers[i][j]==predict[i]):
```

```
                        count+=1
                metrics1.append(min(count//3,1))
        correct1 = correct1/length1
        print("Validation set Accuracy metric is {}".format(np.mean(metrics1)))
        valid_accuracy.append(np.mean(metrics1))
        epochs_total.append(epoch)
        # statistic
        for phase in ['train', 'val']:
            epoch_loss[phase] /= len(dataloader[phase])
            with open(os.path.join(logs, f'{phase}_log.txt'), 'a') as f:
                f.write(str(epoch+1) + '\t' + str(epoch_loss[phase]) + '\n')
        print('Epoch:{}/{} | Training Loss: {train:6f} |
        Validation Loss: {val:6f}'.format(epoch+1, epochs, **epoch_loss))

        torch.save(model.state_dict(), os.path.join(model_dir, f'model-epoch-{epoch+1}.pth'))

        scheduler.step()
    end_time = time.time()
    training_time = end_time - start_time
    print("Total training time in mins is {}".format(training_time//60))

if not os.path.exists(logs):
    os.makedirs(logs)
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
train()



import matplotlib.pyplot as plt
import numpy as np

# Plotting train and validation accuracy with respect to number of epochs for Model 1.
plt.plot(epochs_total, train_accuracy, label = "train_accuracy")
plt.plot(epochs_total, valid_accuracy, label = "validation_accuracy")
plt.legend()
plt.show()

#Model 2: Resnet50 for extracting features from
images and a bidirectional LSTM to extract encoding
from the sentences.But has different number of hidden layers than Model 1.

import os
import time
import torch
import torch.nn as nn
from torch import optim
train_accuracy=[]
valid_accuracy=[]
epochs_total=[]
datadir = './data'
```

```
model_dir = './ckpt'
logs = './log'
workers = 1
batchsize = 100
sizeoffeatures, embeddingsize = 1024, 1
sizeoffeatures, embeddingsize = 1024, 150
numofhidden, sizeofhiddenlayer = 2, 256
epochs = 10
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#device='cuda'
print(device)
def train():
    dataloader = data_loader(input_dir=datadir, batch_size=batchsize, num_worker=workers)
    questionvocabularysize = qvocabsize+1
    answervocabularysize = 3001

    model = VizWizModel(sizeoffeatures=sizeoffeatures,
    questionvocabularysize=questionvocabularysize,
    answervocabularysize=answervocabularysize,
    wordembeddingsize=embeddingsize,
    hidden_size=sizeofhiddenlayer,
    num_hidden=numofhidden).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
    print('>> start training')
    start_time = time.time()
    for epoch in range(epochs):
        metrics=[]
        metrics1=[]
        correct1=0
        length1=0
        correct=0
        length=0
        epoch_loss = {key: 0 for key in ['train', 'val']}
        print(epoch)
        model.train()
        print(dataloader['train'])
        for idx, dataentry in enumerate(dataloader['train']):

            image = dataentry['image'].to(device)
            question = dataentry['question'].to(device)
            label = dataentry['answer'].to(device)
            answers=dataentry['answers']
            label1=label.tolist()

            # forwardpropogation
            logits = model(image, question)

            with torch.no_grad():
                predict = torch.log_softmax(logits, dim=1)
```

```
        predict = torch.argmax(predict, dim=1).tolist()
    print("Train set prediction {}".format(Counter(predict)))
    for i in range(len(label1)):
        if(predict[i]==label1[i]):
            correct+=1
    length=length+batchsize
    for i in range(answers.shape[0]):
        count=0
        for j in range(answers.shape[1]):
            if(answers[i][j]==predict[i]):
                count+=1
        metrics.append(min(count//3,1))
    loss = criterion(logits, label)
    epoch_loss['train'] += loss.item()
    # backpropogation
    optimizer.zero_grad()
    loss.backward()
    print(loss.item())
    optimizer.step()
model.eval()
correct = correct/length
print("Train set Accuracy metric is {}".format(np.mean(metrics)))
train_accuracy.append(np.mean(metrics))

for idx, dataentry in enumerate(dataloader['val']):
    image = dataentry['image'].to(device=device)
    question = dataentry['question'].to(device=device)
    label = dataentry['answer'].to(device=device)
    answers=dataentry['answers']
    label1=label
    with torch.no_grad():
        logits = model(image, question)
        loss = criterion(logits, label)
    epoch_loss['val'] += loss.item()
    with torch.no_grad():
        predict = torch.log_softmax(logits, dim=1)
        predict = torch.argmax(predict, dim=1).tolist()
    print("Valid set prediction {}".format(Counter(predict)))
    for i in range(len(predict)):
        if(predict[i]==label1[i]):
            correct1+=1
    length1=length1+batchsize
    for i in range(answers.shape[0]):
        count=0
        for j in range(answers.shape[1]):
            if(answers[i][j]==predict[i]):
                count+=1
        metrics1.append(min(count//3,1))
correct1 = correct1/length1
print("Validation set Accuracy metric is {}".format(np.mean(metrics1)))
valid_accuracy.append(np.mean(metrics1))
```

```
        epochs_total.append(epoch)
        # statistic
        for phase in ['train', 'val']:
            epoch_loss[phase] /= len(dataloader[phase])
            with open(os.path.join(logs, f'{phase}_log.txt'), 'a') as f:
                f.write(str(epoch+1) + '\t' + str(epoch_loss[phase]) + '\n')
        print('Epoch:{}/{} | Training Loss: {train:6f} |
        Validation Loss: {val:6f}'.format(epoch+1, epochs, **epoch_loss))

        torch.save(model.state_dict(), os.path.join(model_dir, f'model-epoch-{epoch+1}.pth'))

        scheduler.step()
    end_time = time.time()
    training_time = end_time - start_time
    print("Total training time in mins is {}".format(training_time//60))

if not os.path.exists(logs):
    os.makedirs(logs)
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
train()



import matplotlib.pyplot as plt
import numpy as np

# Plotting train and validation accuracy with respect to number of epochs for Model 2.
plt.plot(epochs_total, train_accuracy, label = "train_accuracy")
plt.plot(epochs_total, valid_accuracy, label = "validation_accuracy")
plt.legend()
plt.show()


import torch
import torch.nn as nn
import torchvision.models as models
import torch.nn.functional as F

#Model 3: using vgg19 pretrained model to extract the image features
and bidirectional LSTM to extract the feature from the question.
class ImgPreprocess(nn.Module):

    def __init__(self, embed_dim):

        super(ImgPreprocess, self).__init__()
        self.model = models.vgg19(pretrained=True)
        in_features = self.model.classifier[-1].in_features
        self.model.classifier[-1] =  nn.Linear(in_features, embed_dim)# remove resnet50 last layer

    def forward(self, image):
```

```
        with torch.no_grad():

            imagefeatures = self.model(image) # (batch, channel, height, width)
        l2_norm = F.normalize(imagefeatures, p=2, dim=1).detach()
        return l2_norm

class QuestionPreprocess(nn.Module):

    def __init__(self, questionvocabularysize, wordembeddingsize,
    hidden_size, num_hidden, sizeofconcatenatedlayer):
                super(QuestionPreprocess, self).__init__()
                self.word_embedding = nn.Embedding(questionvocabularysize, wordembeddingsize)
                self.tanh = nn.Tanh()
                self.hidden_size=hidden_size
                self.num_hidden=num_hidden
                self.lstm=nn.LSTM(wordembeddingsize,hidden_size,num_hidden,
                batch_first=True,bidirectional=True)
                self.fc=nn.Linear(hidden_size*2,sizeofconcatenatedlayer)
    def forward(self, question):
                qu_embedding = self.word_embedding(question)
                qu_embedding = self.tanh(qu_embedding)
                output,_=self.lstm(qu_embedding)
                output=self.fc(output[:,-1,:])
                return output
class VizWizModel(nn.Module):

    def __init__(self, sizeoffeatures, questionvocabularysize,
    answervocabularysize, wordembeddingsize,
    hidden_size, num_hidden):

        super(VizWizModel, self).__init__()
        self.questionpreprocess = QuestionPreprocess(questionvocabularysize,
        wordembeddingsize, hidden_size,
        num_hidden, sizeoffeatures)
        self.imgpreprocess = ImgPreprocess(sizeoffeatures)
        self.tanh = nn.Tanh()
        self.dropout = nn.Dropout(0.5)
        self.layer1 = nn.Linear(sizeoffeatures, answervocabularysize)
        self.layer2 = nn.Linear(answervocabularysize, answervocabularysize)

    def forward(self, image, question):

        imagefeatures = self.imgpreprocess(image)
        queationfeatures = self.questionpreprocess(question)
        dotproductfeatures = imagefeatures * queationfeatures
        dotproductfeatures = self.dropout(dotproductfeatures)
        dotproductfeatures = self.tanh(dotproductfeatures)
        dotproductfeatures = self.layer1(dotproductfeatures)
        dotproductfeatures = self.dropout(dotproductfeatures)
        dotproductfeatures = self.tanh(dotproductfeatures)
```

```
        logits = self.layer2(dotproductfeatures)

        return logits


#Training Model 3
import os
import time
import torch
import torch.nn as nn
from torch import optim
train_accuracy=[]
valid_accuracy=[]
epochs_total=[]
datadir = './data'
model_dir = './ckpt'
logs = './log'
workers = 1
batchsize = 100
sizeoffeatures, embeddingsize = 1024, 300
numofhidden, sizeofhiddenlayer = 3, 512
epochs = 10
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#device='cuda'
print(device)
def train():
    dataloader = data_loader(input_dir=datadir, batch_size=batchsize, num_worker=workers)
    questionvocabularysize = qvocabsize+1
    answervocabularysize = 3001
    model = VizWizModel(sizeoffeatures=sizeoffeatures,
    questionvocabularysize=questionvocabularysize,
    answervocabularysize=answervocabularysize,
    wordembeddingsize=embeddingsize,
    hidden_size=sizeofhiddenlayer,
    num_hidden=numofhidden).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
    print('>> start training')
    start_time = time.time()
    for epoch in range(epochs):
        metrics=[]
        metrics1=[]
        correct1=0
        length1=0
        correct=0
        length=0
        epoch_loss = {key: 0 for key in ['train', 'val']}
        print(epoch)
        model.train()
        print(dataloader['train'])
```

```
for idx, dataentry in enumerate(dataloader['train']):
    image = dataentry['image'].to(device)
    question = dataentry['question'].to(device)
    label = dataentry['answer'].to(device)
    answers=dataentry['answers']
    label1=label.tolist()

    # forwardpropogation
    logits = model(image, question)
    with torch.no_grad():
        predict = torch.log_softmax(logits, dim=1)
        predict = torch.argmax(predict, dim=1).tolist()
    print("Train set prediction {}".format(Counter(predict)))
    for i in range(len(label1)):
        if(predict[i]==label1[i]):
            correct+=1
    length=length+batchsize
    for i in range(answers.shape[0]):
        count=0
        for j in range(answers.shape[1]):
            if(answers[i][j]==predict[i]):
                count+=1
        metrics.append(min(count//3,1))
    loss = criterion(logits, label)
    epoch_loss['train'] += loss.item()
    # backpropogation
    optimizer.zero_grad()
    loss.backward()
    print(loss.item())
    optimizer.step()
model.eval()
correct = correct/length
print("Train set Accuracy is {}".format(correct))
print("Train set Accuracy metric is {}".format(np.mean(metrics)))
train_accuracy.append(np.mean(metrics))

for idx, dataentry in enumerate(dataloader['val']):
    image = dataentry['image'].to(device=device)
    question = dataentry['question'].to(device=device)
    label = dataentry['answer'].to(device=device)
    answers=dataentry['answers']
    label1=label
    with torch.no_grad():
        logits = model(image, question)
        loss = criterion(logits, label)
    epoch_loss['val'] += loss.item()
    with torch.no_grad():
        predict = torch.log_softmax(logits, dim=1)
        predict = torch.argmax(predict, dim=1).tolist()
    print("Valid set prediction {}".format(Counter(predict)))
    for i in range(len(predict)):
```

```
                if(predict[i]==label1[i]):
                    correct1+=1
            length1=length1+batchsize
            for i in range(answers.shape[0]):
                count=0
                for j in range(answers.shape[1]):
                    if(answers[i][j]==predict[i]):
                        count+=1
                metrics1.append(min(count//3,1))
        correct1 = correct1/length1
        print("Validation set Accuracy is {}".format(correct1))
        print("Validation set Accuracy metric is {}".format(np.mean(metrics1)))
        valid_accuracy.append(np.mean(metrics1))
        epochs_total.append(epoch)
        # statistic
        for phase in ['train', 'val']:
            epoch_loss[phase] /= len(dataloader[phase])
            with open(os.path.join(logs, f'{phase}_log.txt'), 'a') as f:
                f.write(str(epoch+1) + '\t' + str(epoch_loss[phase]) + '\n')
        print('Epoch:{}/{} | Training Loss: {train:6f} |
        Validation Loss: {val:6f}'.format(epoch+1, epochs, **epoch_loss))

        torch.save(model.state_dict(), os.path.join(model_dir, f'model-epoch-{epoch+1}.pth'))

        scheduler.step()
    end_time = time.time()
    training_time = end_time - start_time
    print("Total training time in mins is {}".format(training_time//60))

if not os.path.exists(logs):
    os.makedirs(logs)
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
train()


import matplotlib.pyplot as plt
import numpy as np
# Plotting train and validation accuracy with respect to number of epochs for Model 2.
plt.plot(epochs_total, train_accuracy, label = "train_accuracy")
plt.plot(epochs_total, valid_accuracy, label = "validation_accuracy")
plt.legend()
plt.show()

#Final Model:Based on training the above 3 models,
#Model 1 obtained the highest validation accuracy
#and is chosen to train on the entire dataset.
#This model is used to perform prediction on the test set.
import torch
import torch.nn as nn
import torchvision.models as models
```

```
import torch.nn.functional as F


class ImgPreprocess(nn.Module):

    def __init__(self, embed_dim):

        super(ImgPreprocess, self).__init__()
        self.model = models.resnet50(pretrained=True)
        in_features = self.model.fc.in_features
        self.model.fc =  nn.Linear(in_features, embed_dim)# remove resnet50 last layer


    def forward(self, image):
        with torch.no_grad():
            imagefeatures = self.model(image) # (batch, channel, height, width)

        l2_norm = F.normalize(imagefeatures, p=2, dim=1).detach()
        return l2_norm

class QuestionPreprocess(nn.Module):
    def __init__(self, questionvocabularysize,
    wordembeddingsize,
    hidden_size, num_hidden,
    sizeofconcatenatedlayer):
                super(QuestionPreprocess, self).__init__()
                self.word_embedding = nn.Embedding(questionvocabularysize, wordembeddingsize)
                self.tanh = nn.Tanh()
                self.hidden_size=hidden_size
                self.num_hidden=num_hidden
                self.lstm=nn.LSTM(wordembeddingsize,hidden_size,num_hidden,
                batch_first=True,bidirectional=True)
                self.fc=nn.Linear(hidden_size*2,sizeofconcatenatedlayer)
    def forward(self, question):
                qu_embedding = self.word_embedding(question)
                qu_embedding = self.tanh(qu_embedding)
                h0=torch.zeros(self.num_hidden*2,qu_embedding.size(0),self.hidden_size).to('cuda')
                c0=torch.zeros(self.num_hidden*2,qu_embedding.size(0),self.hidden_size).to('cuda')
                output,_=self.lstm(qu_embedding,(h0,c0))
                output=self.fc(output[:,-1,:])
                return output
class VizWizModel(nn.Module):

    def __init__(self, sizeoffeatures,
    questionvocabularysize,
    answervocabularysize,
    wordembeddingsize, hidden_size, num_hidden):

        super(VizWizModel, self).__init__()
        self.questionpreprocess = QuestionPreprocess(questionvocabularysize,
        wordembeddingsize, hidden_size,
```

```
        num_hidden, sizeoffeatures)
        self.imgpreprocess = ImgPreprocess(sizeoffeatures)
        self.tanh = nn.Tanh()
        self.dropout = nn.Dropout(0.5)
        self.layer1 = nn.Linear(sizeoffeatures, answervocabularysize)
        self.layer2 = nn.Linear(answervocabularysize, answervocabularysize)

    def forward(self, image, question):

        imagefeatures = self.imgpreprocess(image)
        queationfeatures = self.questionpreprocess(question)
        dotproductfeatures = imagefeatures * queationfeatures
        dotproductfeatures = self.dropout(dotproductfeatures)
        dotproductfeatures = self.tanh(dotproductfeatures)
        dotproductfeatures = self.layer1(dotproductfeatures)
        dotproductfeatures = self.dropout(dotproductfeatures)
        dotproductfeatures = self.tanh(dotproductfeatures)
        logits = self.layer2(dotproductfeatures)

        return logits

#The entire dataset is preprocessed(training and validation).
preprecessed={}
preprecessed['train']=preprocessingtrain('train')
preprocessed['val']=preprocessing('val')
preprecessed['train-val'] = preprecessed['train'] + preprecessed['val']

#Store the entire dataset results in the files train and val.
for key, value in preprecessed.items():
        np.save(os.path.join("./data", f'{key}.npy'), np.array(value))

#Train the Final chosen model(Model 1) on the entire dataset for 12 epochs.
import os
import time
import torch
import torch.nn as nn
from torch import optim
train_accuracy=[]
valid_accuracy=[]
epochs_total=[]
datadir = './data'
model_dir = './ckpt'
logs = './log'
workers = 1
batchsize = 100
sizeoffeatures, embeddingsize = 1024, 300
numofhidden, sizeofhiddenlayer = 3, 512
epochs = 12
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#device='cuda'
print(device)
```

```
def train():
    dataloader = data_loader(input_dir=datadir, batch_size=batchsize, num_worker=workers)
    questionvocabularysize = qvocabsize+1
    answervocabularysize = 3001

    model = VizWizModel(sizeoffeatures=sizeoffeatures,
    questionvocabularysize=questionvocabularysize,
    answervocabularysize=answervocabularysize,
    wordembeddingsize=embeddingsize,
    hidden_size=sizeofhiddenlayer, num_hidden=numofhidden).to(device)


    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)


    print('>> start training')
    start_time = time.time()
    for epoch in range(epochs):
        metrics=[]
        metrics1=[]
        correct1=0
        length1=0
        correct=0
        length=0
        epoch_loss = {key: 0 for key in ['train', 'val']}
        print(epoch)
        model.train()
        print(dataloader['train'])
        for idx, dataentry in enumerate(dataloader['train']):

            image = dataentry['image'].to(device)
            question = dataentry['question'].to(device)
            label = dataentry['answer'].to(device)
            answers=dataentry['answers']
            label1=label.tolist()

            # forwardpropogation
            logits = model(image, question)

            with torch.no_grad():
                predict = torch.log_softmax(logits, dim=1)
                predict = torch.argmax(predict, dim=1).tolist()
            print("Train set prediction {}".format(Counter(predict)))
            for i in range(len(label1)):
                if(predict[i]==label1[i]):
                    correct+=1
            length=length+batchsize
            for i in range(answers.shape[0]):
                count=0
```

```
        for j in range(answers.shape[1]):
            if(answers[i][j]==predict[i]):
                count+=1
        metrics.append(min(count//3,1))
    loss = criterion(logits, label)
    epoch_loss['train'] += loss.item()
    # backpropogation
    optimizer.zero_grad()
    loss.backward()
    print(loss.item())
    optimizer.step()
model.eval()
correct = correct/length
print("Train set Accuracy is {}".format(correct))
print("Train set Accuracy metric is {}".format(np.mean(metrics)))
train_accuracy.append(np.mean(metrics))

for idx, dataentry in enumerate(dataloader['val']):
    image = dataentry['image'].to(device=device)
    question = dataentry['question'].to(device=device)
    label = dataentry['answer'].to(device=device)
    answers=dataentry['answers']
    label1=label
    with torch.no_grad():
        logits = model(image, question)
        loss = criterion(logits, label)
    epoch_loss['val'] += loss.item()
    with torch.no_grad():
        predict = torch.log_softmax(logits, dim=1)
        predict = torch.argmax(predict, dim=1).tolist()
    print("Valid set prediction {}".format(Counter(predict)))
    for i in range(len(predict)):
        if(predict[i]==label1[i]):
            correct1+=1
    length1=length1+batchsize
    for i in range(answers.shape[0]):
        count=0
        for j in range(answers.shape[1]):
            if(answers[i][j]==predict[i]):
                count+=1
        metrics1.append(min(count//3,1))
correct1 = correct1/length1
print("Validation set Accuracy is {}".format(correct1))
print("Validation set Accuracy metric is {}".format(np.mean(metrics1)))
valid_accuracy.append(np.mean(metrics1))
epochs_total.append(epoch)
# statistic
for phase in ['train', 'val']:
    epoch_loss[phase] /= len(dataloader[phase])
    with open(os.path.join(logs, f'{phase}_log.txt'), 'a') as f:
        f.write(str(epoch+1) + '\t' + str(epoch_loss[phase]) + '\n')
```

```
        print('Epoch:{}/{} | Training Loss: {train:6f} |
        Validation Loss: {val:6f}'.format(epoch+1, epochs, **epoch_loss))

        torch.save(model.state_dict(), os.path.join(model_dir, f'model-epoch-{epoch+1}.pth'))

        scheduler.step()
    end_time = time.time()
    training_time = end_time - start_time
    print("Total training time in mins is {}".format(training_time//60))

if not os.path.exists(logs):
    os.makedirs(logs)
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
train()



import matplotlib.pyplot as plt
import numpy as np
#Plot the training and validation accuracy of the final model(Model 1)
with respect to the number of epochs.
plt.plot(epochs_total, train_accuracy, label = "train_accuracy")
plt.plot(epochs_total, valid_accuracy, label = "validation_accuracy")
plt.legend()
plt.show()


#Preprocess the test dataset
def preprocessingtest(type1):
        missing_images = []
    ## load annotation file and question file
        dataset = json.load(open(root_dir+'/'+type1+'.json'))
        data = [None]*len(dataset)
        for idx,one_data in enumerate(dataset):
            imagename = one_data['image']
            imagepath = "./images_testresized"+"/"+ imagename
            questionsentence=one_data['question']
            questiontokens = nltk.word_tokenize(question)
            questionencoding = sentence_to_vocabid(one_data['question'], vocab)
            if(len(questionencoding)>=37):
                questionencoding=questionencoding[0:37]
            else:
                questionencoding.extend([0]*(37-len(questionencoding)))
            image_info = {'imagename': imagename,
                'imagepath': imagepath,
                'questionsentence': questionsentence,
                'questiontokens': questiontokens
                }
            image_info['questionencoding'] = questionencoding
            data[idx]=image_info
```

```
        return data

#Store the train,val,test and train-val dataset in the dictionary preprocessed.
preprecessed={}
preprecessed['train']=preprocessingtrain('train')
preprecessed['val']=preprocessingvalidation('val')
preprecessed['test']=preprocessingtest('test')
preprecessed['train-val'] = preprecessed['train'] + preprecessed['val']

#Storing the train,val,test and train-val
#dataset in the files named train,val,test and train-val.npy.
for key, value in preprecessed.items():
        np.save(os.path.join("./data", f'{key}.npy'), np.array(value))

#The directory where the final model is saved
ckpt_dir='../input/resizedviz/ckpt/model-epoch-12.pth'
workers = 1
batchsize = 100
sizeoffeatures, embeddingsize = 1024, 300
numofhidden, sizeofhiddenlayer = 3, 512

#reading the answer2answer_id file.
with open('./answer2answer_id.json', 'r') as f:
    answer_data = json.load(f)

#File to swap the keys and values of the answer2answer_id file.
def swap(json):
    ret = {}
    for key in json:
        ret[json[key]] = key
    return ret

#Reverse the answer_data dict keys and values.
idtoanswer=swap(answer_data)


#Get the question vocabulary size which is the number of tokens in the
question + 1 and answer vocab size which is the number of answers
in the answer vocab(30001)
questionvocabularysize = qvocabsize+1
answervocabularysize = 3001

from torchvision import transforms
device='cuda'

#Testing the final model on the test set.The last epoch weights( epoch 12 )
is used to perform prediction on the test set.
The results of the predictions are stored in a file results.csv on the test set.
import csv
predictions=[]
def test(input_dir, data_type, batch_size, num_worker):
```

```
    print("hi")
    transform = transforms.Compose([
        transforms.ToTensor(),  # convert to (C,H,W) and [0,1]
        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))  # mean=0; std=1
    ])
    VizWiz_dataset = VizWizdataset(input_dir, f'{data_type}.npy',
    transformimage= transform)
    dataloader = DataLoader(VizWiz_dataset, batch_size=batch_size,
    shuffle=False, num_workers=num_worker)

    model=VizWizModel(sizeoffeatures=sizeoffeatures,
    questionvocabularysize=questionvocabularysize,
    answervocabularysize=answervocabularysize,
    wordembeddingsize=embeddingsize,
    hidden_size=sizeofhiddenlayer,
    num_hidden=numofhidden).to(device)
    model.load_state_dict(torch.load(ckpt_dir))
    model.eval()
    results = []
    try:
        for idx, dataentry in enumerate(dataloader):
            image = dataentry['image'].to(device)
            question = dataentry['question'].to(device)
            imagepaths=dataentry['imagepath']
            with torch.no_grad():
                logits = model(image, question)
                predict = torch.log_softmax(logits, dim=1)
                predict = torch.argmax(predict, dim=1).tolist()

                predict = [[idtoanswer[idx]] for idx in predict]
                predict_final=predict

                for i in range(len(predict)):
                     if(predict_final[i][0]=='<unk>'):
                            predict_final[i][0]='unanswerable'
                predictions.extend(predict_final)
    except:
        pass
    file = open('results.csv', 'w')
    with file:
            write = csv.writer(file)
            write.writerows(predictions)

#Calling the test function to run the final model on the test set.
test('./data', 'test', batch_size=100, num_worker=1)
```

Neural-Net-4 Writeup

April 2022

1 Writeup

1.1 Preprocessing

The training set comprises of an annotation file and a set of images.In the annotation file we have the image path, the visual question and a set of 10 answers.All the questions from the training set are chosen via the annotation file and then tokenized. These tokenized questions are appended to a file word2vocab_id.json where the question token is mapped to an integer index.For the answer vocabulary we are considering the top 3000 most frequently occurring answers across the training set.These answers are then appended to a file answer2answer_id.json. An additional unknown token 'unk' is added for answers that are not present in top 3000 most frequently occuring answers.The training set images are resized to 448*448 and stored in a separate training set directory.Similarly the validation and test set images are resized to 448*448 and appended to validation and test set directory.Then for the training and validation images we store the imagename(name of the image),imagepath(path of the image),the question sentence(the question) and the questiontokens(the tokenized version of the sentence).The questionencoding is the numbered version of questiontokens(Every token will have a number from the word2vocab_id.json file) and the questionencoding is padded with zeros with a padding length 37, the selectedanswer(the answer that occurs most frequently among the 10 answers to questions),the answerlabel which is integer label to the answer selected from the answer2answer_id.json file, answers are the set of 10 answers for a question.We store these attributes in a list.The training and validation sets have separate dictionaries.For the training set we have 'train' as the key in the dictionary and for the validation set we have 'val' as the key.The values for the key 'train' is the training set list and for the validation set the validation set list.Similarly for the test set we store the rest of the attributes except the answers,answerlabel,selectedanswer as the test set doesn't have answers.The test set list is stored in the preprocess dictionary where is key is 'test' and the value is the test set list.When the final model (in our case model 1) makes a prediction on the test set then the 'unk' token predicted by the model is substituted with 'unanswerable'.

1.2 Analysis on the validation set

1.3 Models

There were three models that were trained on the training set and the performance of these models were evaluated on the validation set.

Model 1 architecture:For the first model we are utilizing Resnet50 pretrained model to extract features from the images.The last layer of Resnet50 model is removed and replaced with a fully connected layer of size 1024.These extracted features from the images are L2 normalized. We use a bidirectional LSTM where the number of recurrent layers is 3(3 bidirectional LSTM's stacked on top of each other) to extract the features from the question.A vector of dimension 1024 is obtained.The first layer of the bidirectional LSTM is an embedding layer that uses embedding dimension of size 300 , the number of embeddings is the question vocab size which is 3906.Then we add a tanh activation function layer, an lstm layer which has hidden state h0 and c0 as the input.We then pass a fully connected layer and the input to this layer is output of the last LSTM state. We then take the element wise product of the two vectors(one from Resnet50 and the other from bidirectional LSTM) and then obtain a final vector of length 1024.Then we add a dropout layer of 0.5,tanh activation function after the dropout layer followed by a linear layer, dropout layer,tanh layer and then a linear layer.The output layer dimension is 3001 as we have 3001 answers in the answer vocab. Softmax activation function is used with Categorical Cross entropy loss function.The hidden layer size is 512.

Model 2 architecture: The second model is identical to the first model except that we set the number of recurrent layers in the bidirectional LSTM to 2,the size of each hidden layer is 256 and the embedding layer size is 150.

Model 3 architecture: In the third model we use vgg19 pretrained model to extract features from the images.The last layer of the vgg19 model is removed and replaced with a fully connected layer of size 1024.These extracted features from the images are L2 normalized.The rest of the architecture remains the same with the same number of recurrent layers(3) and the same dimension of the hidden state(512).

1.4 Optimization Schemes

The optimizer used is Adam with a learning rate of 0.001.A step learning rate that decays the learning rate of the model by a factor of 0.1(gamma factor).The model is trained for 10 epochs with a batch size of 100.The number of examples the model is trained on is 10000 and the validation set size is 1000.For all the three models the same optimization scheme is used.

1.5 Hardware

All the three models use Tesla P100 GPU for training and are trained on Kaggle.

2 Performance on validation set

Model 1 achieved an accuracy of 0.442 on the validation set and a validation loss of 5.046.The reason for such a low validation accuracy is that the training set and validation set images along with questions may not be representative of each other.Hence when a model trains on the training set it achieves a low validation set accuracy, although the training set accuracy is higher(0.5118).Dropout regularization tends to reduce overfitting but only to some extent as training set and validation set are quite different from each other.Similarly for Model 2 we achieve a validation set accuracy of 0.414 and a validation loss of 4.66.Again a similar reason to achieve such low validation accuracy is similar to the reason described for Model 1.For Model 3 we achieve a validation set accuracy of 0.391 and a validation loss of 5.045.The reason for achieving such low validation accuracy is that the training accuracy is low 0.3689 and the training and validation sets may not be representative of each other.

3 Analysis of different tested models

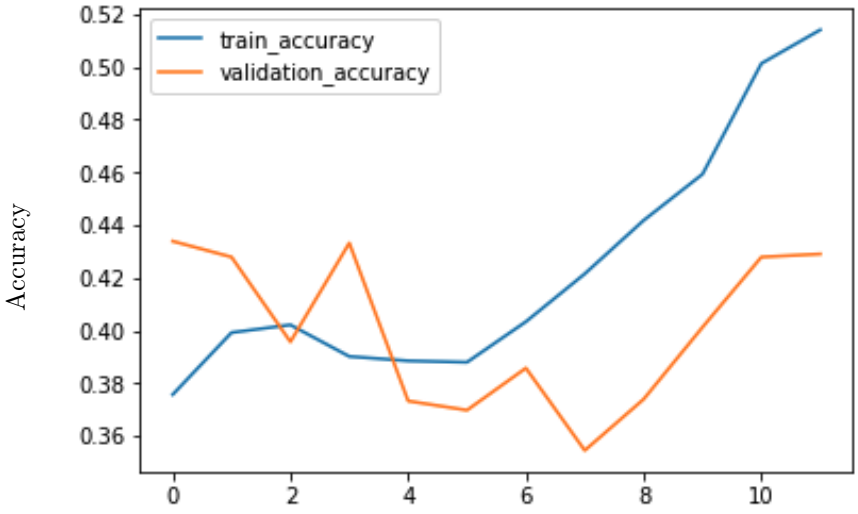
One general trend observed is that all the three models predict a lot of unanswerable and unsuitable answers on the training set and the validation set.This is because the frequency of unanswerable and unsuitable tokens across all the answers in the training set is extremely high,much higher compared to the the frequency of other answers in the training set.A class imbalance results which leads to the models making these predictions a lot.Also while training all the three models it was observed that initially the predictions made by all the three models on the test set were all unanswerables and unsuitables and then the models predicted other classes of answers.This trend may occur as the model initially gets biased to these classes(unanswerables and unsuitables) but as the training keeps on progressing it tends to train on other less frequent classes(answers) like yes and no.Model 1 and Model 2 that used resnet50 pretrained architecture to extract the features from images achieved better results that when we use vgg19 pretrained architecture in terms of accuracy on the validation set.This may be because Resnet50 has skip connections that improves the performance of the model by propogating a linear component.This relieves non-linearity which is usually makes it difficult for the model to converge to a global optimum.Hence convergence is faster in case of skip connections.Also resnet has more layers than vgg which enables it to perform better.

4 Final model used

Model 1 is the final model used to make a prediction on the test set as it achieves the highest accuracy of 0.442 on the validation set.The model has a Resnet50 pretrained model to extract features from the images.The last layer of Resnet50 model is removed and replaced with a fully connected layer of size 1024.These extracted features from the images are L2 normalized. We use a bidirectional LSTM where the number of recurrent layers is 3(3 bidirectional LSTM's stacked on top of each other) to extract the features from the question.A vector of dimension 1024 is obtained.The first layer of the bidirectional LSTM is an embedding layer that uses embedding dimension of size 300 , the number of embeddings is the question vocab size which is 3906.Then we add a tanh activation function layer, an lstm layer which has hidden state h0 and c0 as the input.We then pass a fully connected layer and the input to this layer is output of the last LSTM state. We then take the element wise product of the two vectors(one from Resnet50 and the other from bidirectional LSTM) and then obtain a final vector of length 1024.Then we add a dropout layer of 0.5,tanh activation function after the dropout layer followed by a linear layer, dropout layer,tanh layer and then a linear layer.The output layer dimension is 3001 as we have 3001 answers in the answer vocab. Softmax activation function is used with Categorical Cross entropy loss function.The hidden layer size is 512.The model is trained on 100 epochs and uses Adam optimizer with a learning rate of 0.001.A step learning rate that decays the learning rate of the model by a factor of 0.1(gamma factor).The model is trained for 12 epochs with a batch size of 100.This model is trained on the entire training set and tested on the entire validation set.

5 Loss curve for final prediction

Training vs Validation accuracy for the final model(Model 1) vs the number of epochs.



For the training set curve we can observe that as the number of epochs increases the accuracy of the model also increases.This is because initially the model predicts unanswerable and unsuitable answers as prediction but as the number of epochs increases the model starts predicting other labels apart from unanswerable and unsuitable which results in the training set accuracy increasing.But for the validation set the accuracy of the model kind of decreases till epoch 7 and then increases till epoch 12.This may be the case as initially the model overfits the training data until epoch 7 and then the dropout regularization that is used for the final model enables the model to generalize well on the validation set.The training set accuracy doesn't seem to converge till 12 epochs but for the validation set the accuracy seems to stagnate after 10 epochs.

6 References

https://github.com/liqing-ustc/VizWiz_LSTM_CNN_Attention